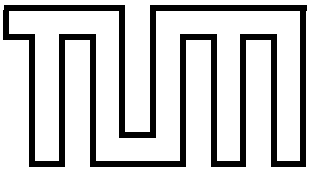


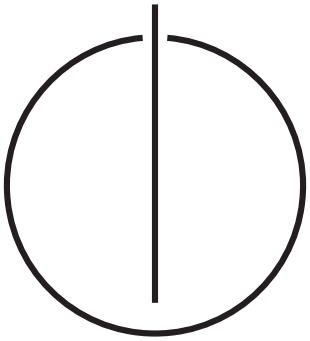
FAKULTÄT FÜR INFORMATIK

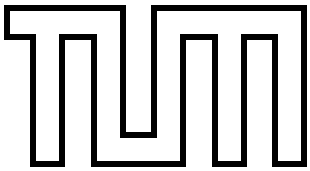
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatik

Monocular Visual Inertial Odometry on a Mobile Device

Michael Andrew Shelley





FAKULTÄT FÜR INFORMATIK

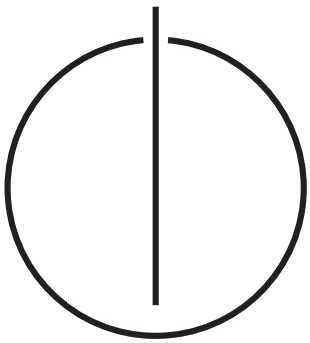
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatik

Monocular Visual Inertial Odometry on a Mobile Device

Monokulare visuell-inertiale Odometrie für mobile Geräte

Author:	Michael Andrew Shelley
Supervisor:	Prof. Dr. Daniel Cremers
Advisor:	Jakob Engel
Date:	August 18, 2014



Ich versichere, dass ich diese Masterarbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

I assure the single handed composition of this master's thesis, only supported by declared resources.

München, den 18. August 2014
Munich, August 18, 2014 Michael Andrew Shelley

Contents

Abstract

We present a complete system for single camera visual odometry with the help of inertial sensors. Recent research in visual inertial odometry has produced high quality results, but is largely inaccessible outside of the scientific community. We focus on the implementation and explanation of the Multi-State Constraint Kalman Filter using the low cost and commonly available hardware of a mobile device, such as a smart phone. Inertial sensors and cameras found in these devices are often in need of heavy calibration before use in odometry. For example, inertial sensors suffer from misalignment and scale factor errors, and rolling shutter cameras suffer from distortion, especially while in motion. We show how these sensors can be used with only basic knowledge of the hardware by calibrating all the necessary parameters online instead of using complex and time consuming offline calibration techniques. Our results show the algorithm is consistent and good quality calibration can be performed in seconds. We demonstrate the filter on simulated and real data, with results comparable to the state of the art.

1. Introduction

As robots become more common, they also become more accessible and easier to develop. With cheaper hardware, faster processors and the internet for sharing ideas, the possibilities for a hobbyist are greater than ever. Modern advances in research have lead to high quality semi-autonomous vehicles, such as Google's driverless car or the Mars Rovers. However, these examples are outliers. The majority of robots still lack even the most basic forms of autonomy.

Every autonomous mobile robot must be able to answer three questions: Where am I? How did I get here? and How do I get *there*? The answer to these questions can be found, respectively, through localization, odometry, and control.

- **Localization** tells the robot where it is in a map and is often directly combined with the mapping process. In order to localize, one needs a map. In order to create a map, one must localize.
- **Odometry** is the process of incrementally estimating the pose (position and orientation) of the robot using motion sensors. Pure odometry can be independent from mapping and localization, giving the pose relative to, for example, the starting position.
- **Control** is the method for getting from one place to the other. Control theory is a huge part of robotics, dominated by research on topics such as grasping, walking and even flight.

In this thesis we focus on *odometry*. Specifically, we develop a system for self-contained odometry without a map. Odometry, also known as *pose estimation*, is widely studied and deeply important in robotics and computer science. Nearly every kind of robot, from flying drones to automated vacuum cleaners, need to know precisely where they are in order to function. When a map is not available and building one is not feasible, it is impossible to know a robot's exact location, so a best guess pose estimate can be maintained along with a measure of uncertainty.

There are many well known examples of modern technology that use odometry. Any car with a GPS navigation system is using a simple form of odometry. Google's self-driving car [43] and the Mars Rovers [45] use much more sophisticated techniques for more precise measurements. These systems use expensive hardware such as 3D scanners and multiple cameras. A more cost effective odometry is that performed by the Roomba vacuum cleaner [44]. It uses only infrared sensors and bump sensors to navigate and clean a room, along with encoders that tell it how the wheels move and acoustic sensors to search for dirty spots on the floor.

All of these examples are wheeled robots that can make use of motor control commands and wheel encoders to gain immediate feedback about their poses. We wish to seperate

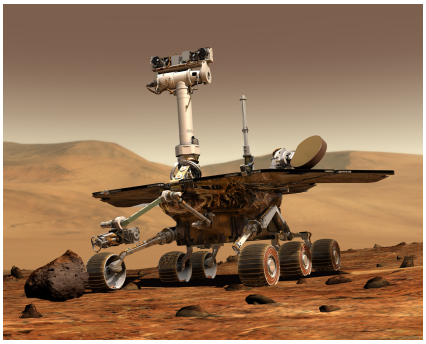


Figure 1.1. NASA's Mars rover [45] and Google's driverless car [43] both perform odometry with expensive high quality sensors

the control component completely and consider the robot as an observer, or a passenger, with no control over its motion. By separating control from odometry, the algorithm generalizes well and can be integrated into virtually any system with the required sensor configuration.

1.1. Problem Statement

There are a plethora of techniques for odometry and many are specific to a certain type of hardware or are limited to a particular environment. Some methods use prohibitely expensive hardware, require complex calibration, or do not perform in real time without powerful parallel processing. In order to make odometry more accessible, it is important that the sensors used be low cost with simple calibration requirements, and that the algorithm can run in real time on a single core CPU.

In this thesis we are interested in estimating the pose of a body in motion with minimal low cost sensors. More specifically, we wish to develop a method of odometry that meets the following goals.

- **Quality**: The estimate must be close to the actual value.
- **Consistency**: The method must correctly report the uncertainty of the pose estimate.
- **Scalability**: The method must work for long time periods over long distances.
- **Accessibility**: The method must use commonly found low cost hardware without overly complex calibration techniques.
- **Efficiency**: The method must run in real time, preferably on the device itself.
- **Robustness**: The method must be robust enough to work in different environments (i.e. indoors and outdoors) and in the presence of common environmental disturbances (e.g. pedestrians and cars).

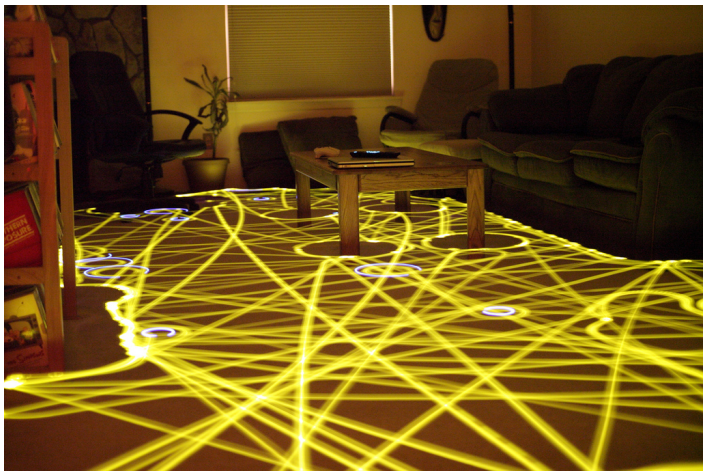


Figure 1.2. A time-lapse image of the Roomba [44] as it moves around a room. Odometry helps prevent cleaning the same positions repeatedly.

1.2. Proposed Solution

In recent years, the camera has become a small, low cost and ubiquitious sensor, present in smart phones, quadrotors [1], and many other vehicles. A single video camera can provide a huge amount of data over time. Processing that data into usable structural information about the environment is a complex process and a highly active field of research. Popular computer vision techniques (for example [35][6]) focus on finding features in the image, usually corners or edges, with properties that make them easy to reidentify if seen again. By matching features between images, along with having either multiple cameras or a moving camera, 3D information to be obtained via triangulation [11]. Triangulated points provide structural information about the camera's surroundings which in turn yield information about the location of the camera itself.

Stereo camera systems are larger, more expensive, and more difficult to calibrate than a single camera, but they are able to give depth information rapidly and accurately [12]. How well a stereo system estimates depth depends on the baseline, the distance between the cameras, and the quality of calibration. The main advantage over monocular systems is that no motion is required for depth estimation. Moving cameras can provide depth information as well, but they suffer from an inherent lack of scale information. If a camera is the only sensor used, the system has no way of knowing scale. This is a result of the way a camera projects the 3D world into a 2D image. This well known problem has been addressed by adding an initialization procedure or additional sensors [29].

There are few small, low power and low cost sensors that can give scale information. Ultrasound sensors can give metric distances and are useful with, for example, quadrocopters for determining how high they are from the ground. For a general system with any theoretical orientation, however, ultrasound is not feasable. Barometers can give precise air pressure levels which can be translated over time into scaled height differences, but they are highly sensitive to weather changes and do not work well indoors. The Inertial Measurement Unit (IMU) found in modern smart phones and quadrocopters provides

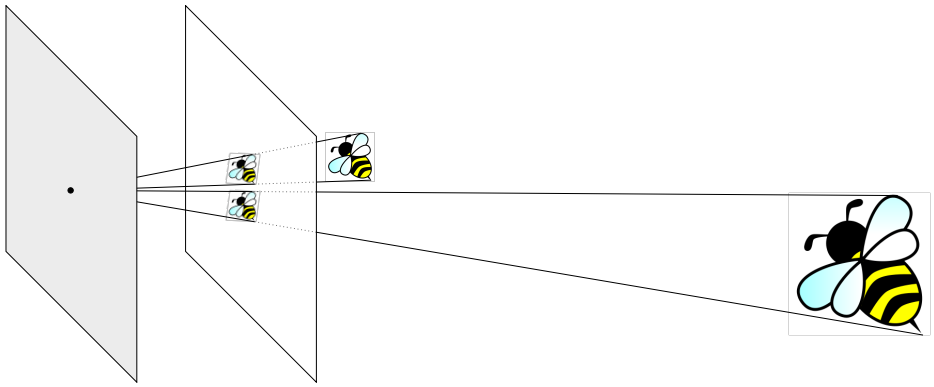


Figure 1.3. The projective model of the camera creates a scale ambiguity. It is impossible with only one image to tell the scale of an object. Here both bees appear to be the same size in the image plane because the larger bee is simply farther away.

precise high frequency measurements of acceleration and rotational velocity. Although the sensors measurements are contaminated by noise, and obtaining position from acceleration requires integrating the signal twice, this inertial data can be used effectively to provide short term scale information.

Combining the camera with the IMU for odometry is called **Visual Inertial Odometry**, or VIO. This thesis explores VIO and its implementation on a mobile device while meeting all the requirements listed above.

2. Visual Inertial Odometry

Since its introduction in 1986, Simultaneous Localization and Mapping (SLAM) has been a popular and an important step toward autonomy in robots. In the SLAM paradigm [9], landmarks are detected and compared with a map to localize the robot while at the same time newly detected landmarks are added to that map. Landmark detection can be done with a variety of sensors including laser scanners and cameras. Early SLAM systems suffered from a number of shortcomings.

- A map is neccessary to localize, but one must localize to create a map [17]. This chicken and egg problem results in a variety of often complex initialization procedures, depending on the hardware involved.
- Single (monocular) camera systems have an inherent inability to measure scale. Initialization patterns of known size or movements of known length were common early solutions [13]. Additional sensors have recently been a more popular choice [29].
- Running in real time was initially impossible. Over the years much research has been devoted to this problem. Modern solutions can often run on mobile smart phones [14][18].
- Traditional SLAM tightly couples mapping, localization and odometry making it difficult to scale to large environments [8]. Later systems have moved toward decoupled systems for scalability, encapsulation, and efficiency.

In the rest of this chapter we explain important historical solutions to the above problems and how they have lead to VIO. We describe the solution we have chosen and how it relates to the state of the art.

2.1. Filter Based SLAM Methods

Early SLAM approaches represented the robot's state, as well as the 3D landmarks, probabilistically by using the weighted average of noisy measurements in a process called filtering. The Extended Kalman Filter is a widely used filtering method and was the first popular choice for SLAM. In EKF-SLAM, the uncertainty correlations between the landmarks and the robot's pose are maintained in a large covariance matrix. When the number of landmarks grows, the computational complexity of EKF-SLAM grows quadratically, essentially making it impossible for real-time applications.

Davison [8] first introduced real-time monocular SLAM (MonoSLAM) using an Extended Kalman Filter in 2003. Before that, state of the art SLAM systems were only capable of off-line batch processing. A real-time requirement meant a slight reduction

in accuracy in exchange for constant time computation. In MonoSLAM, great care is given to only adding landmarks to the map that contribute important location information and removing features that are no longer useful. While MonoSLAM can run in real time, it is still limited to a small area because the complexity remains quadratic in the number of landmarks. Additionally, MonoSLAM parameterizes detected landmarks with bearing vectors and an initially unknown depth estimate. Small errors in measurements can result in huge non-linear errors in the depth, causing inconsistency in the filter. To address this, Montiel et al. [7] introduced the unified *inverse depth* representation for improved consistency. Expressing landmarks by their inverse depth allows for a more accurate uncertainty representation because of the high degree of linearity and it makes it possible to represent features at great distances.

There have been many proposed solutions for addressing the real-time constraints of SLAM in large areas. SLAM shortcut methods such as [15] aim to postpone evaluation of certain information until absolutely necessary, but it achieves the best results when remaining in a small repeated area. Another method, FastSLAM [25], uses a Particle Filter instead of the Extended Kalman Filter for estimating the locations of landmarks. This reduces the complexity to $O(k \log n)$, where k is the number of particles (on the order of 1000 and n is the number of landmarks. While this is an improvement over EKF-SLAM's quadratic complexity, it still does not scale indefinitely.

2.2. Key-Frame Based Methods

In order to achieve real-time performance with a monocular SLAM system, Klein and Murray [13] developed Parallel Tracking and Mapping (PTAM) in 2007. This was one of a now wide range of algorithms that are key-frame based instead of using only filters. While exploring a map, key-frame based methods maintain a sparse set of important images, along with the landmarks detected and the camera's position. One advantage of key-frames is the retention of old information for explicit use instead of marginalizing out old landmarks and camera poses, as in EKF-SLAM. Although the original PTAM was still limited to a relatively small work area, it increased the number of usable landmarks significantly as compared to EKF-SLAM. PTAM also separated mapping from tracking to run on seperate threads in parallel. This uncoupling was an important step that opened up the field of research in applying graph optimization to the task of mapping. Each key-frame could be represented as a node in a graph with local euclidean constraints. There have been multiple proposed methods for optimizing this graph for real-time large scale SLAM [36][16].

2.3. Dense Methods

Camera based methods such as those previously mentioned almost exclusively rely on the detection and triangulation of landmarks. Detecting salient repeatable patches in an image results in a sparse representation of the surrounding environment. Dense methods, also called direct methods, make use of the entire image for frame to frame tracking. Kinect Fusion [12] and Dense Tracking and Mapping (DTAM) [28] in 2011 were the first major

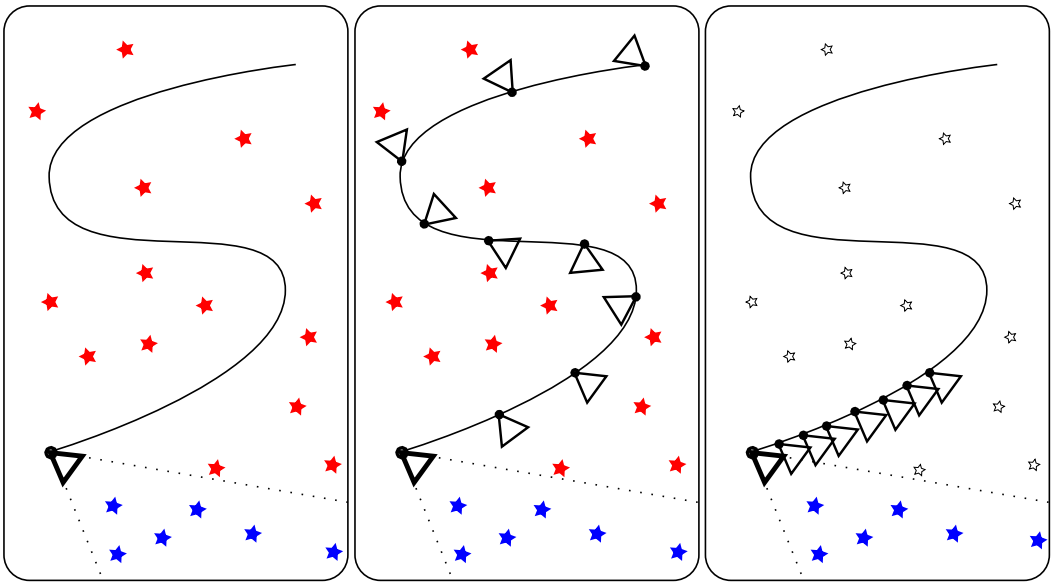


Figure 2.1. A simple illustration of different methods. On the left is EKF-SLAM, with old (red) points maintained in the map and only the latest camera pose. In the middle is a key-frame example with sparse poses and their landmark associations. On the right is the MSCKF with a sliding window of camera poses to track shared landmarks. Old poses and landmarks are discarded.

real-time advances in dense tracking and mapping. REMODE [31] is another example, using probabilistic methods for 3D monocular reconstruction. However, these early methods required high levels of parallel processing, which were done on a GPU. In 2013 Engel et al. [10] introduced semi-dense visual odometry for a monocular camera. This system ran in real-time on multiple cores and was later successfully ported to a modern smart phone.

2.4. Inertial Aided Methods

As mentioned previously, a common problem among monocular odometry and SLAM systems is a lack of scale. Additionally, *scale drift* can occur when the scale is estimated during initialization only [37]. The projective model of a single camera prevents it from being able to distinguish scale in an image. Many SLAM algorithms, such as PTAM, have an inialization sequence to aquire scale or they make use of other sensors. Unfortunately, initializing the scale at the beginning of an odometry sequence causes the scale estimate to drift over time as error accumulates. As explained in the introduction, the ideal sensor configuration for a generalized system with unknown motor capabilities is the Inertial Measurement Unit (IMU) together with a camera. Inertial aided EKF-SLAM [30] has shown success in alleviating the scale problem without a complex initialization proceedure. However, the inertial aided EKF-SLAM algorithm still suffers from all the normal EKF-SLAM problems.

Recently, Weiss et. al. [41] have used the IMU and monocular camera for odometry by using the camera as a black box 6 degree of freedom sensor, which is then loosely coupled with a Kalman Filter using IMU measurements for state estimation. One disadvantage

of this system is that valuable probabilistic information maintained in the filter is not used by the visual odometry module for outlier detection. Outliers occur when the visual odometry module makes a mistake when tracking objects or it tracks moving objects that it thinks are static.

2.5. MSCKF

Mourikis and Roumeliotis introduced the MSCKF [26][27], the **Multi-State Constraint Kalman Filter**, in 2006 to address a number of problems with inertial aided EKF-SLAM. The MSCKF is a pure odometry method. Because it does not build a map, its complexity is linear in the number of features and it does not suffer from assumptions about the nature of the landmarks. Instead of estimating the positions of landmarks in the filter, it maintains a sliding window of camera poses from which landmarks are accurately triangulated using all available data. The landmarks are then used as constraints on the window of camera poses. As a result, the filter achieves better results than EKF-SLAM.

Since the introduction of the MSCKF, a number of proposed improvements have been published. M. Li and Mourikis improved the accuracy [19], made it work on a resource constrained system such as a mobile phone [20], and designed a novel method for rolling shutter camera compensation [18]. Online calibration is also possible for many parameters of the system. The transformation between the camera and IMU is not always precisely known, but can be estimated online [21]. During the writing of this thesis, the same author published a method for online calibration of all relavent parameters [24], including camera intrinsics (Chapter 5), IMU intrinsics (Chapter 4), and camera-IMU relative transformations. We have chosen the MSCKF as our solution for many of the reasons listed above and have incorporated the latest techniques for online calibration.

2.6. Outline

The rest of this thesis is dedicated to explaining the details, how they can be implemented, and what the results look like. As described in the algorithm, the state and the uncertainty must be *propagated* using IMU measurements and then *updated* using camera measurements. This two-step process makes up the Extended Kalman Filter and is explained in **Chapter 3**. The propagation step in particular uses the IMU measurements to estimate the way the state changes at a high frequency. The nature of these measurements, the IMU noise characteristics, and the IMU calibration parameters are described in **Chapter 4**. Images are detected less frequently than IMU measurements are available. These images contain important information which is detected and compared with previous images. The details of computer vision relevant to this thesis are described in detail in **Chapter 5**.

The algorithm detailed here is on a high level and leaves out a significant amount of important details. These details are described in the main chapter of this thesis, **Chapter 6**. We have implemented the MSCKF to work with Apple's iOS devices. Details of the implementaion and problems we encountered are described in **Chapter 7**. Finally, the results of our experiments, both in simulation and with real data, are laid out in **Chapter 8**.

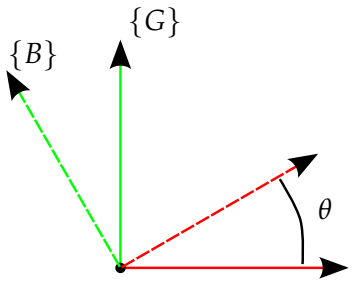
2.7. Notation

This thesis will use the following notation.

Basic Notation	**Example**
Scalars are lower case	x
Vectors are lower case bold	$\mathbf{x}$
Matrices are upper case bold	$\mathbf{X}$
A dot implies a continous time derivative	$\dot{\mathbf{v}} = \mathbf{a}$
A hat implies an estimated value	$\hat{\mathbf{x}}$
A bar implies a unit vector or unit quaternion	$\bar{\mathbf{q}}$
Coordinate frames are upper case	B or $\{B\}$
The coordinate frame of a point resides in the upper left	${}^{G}\mathbf{p}$
Rotation from frame A to B	${}^{B}_{A}\mathbf{R}$

Error Notation	**Example**
Δ represents the differential error of a vector	$\Delta\mathbf{p} = \mathbf{p} - \hat{\mathbf{p}}$
δ represends the orientation error	$\delta\boldsymbol{\theta}$ or $\delta\mathbf{q}$
A tilde implies error in general	$\tilde{\mathbf{x}} = \mathbf{x} - \hat{\mathbf{x}}$ with $\mathbf{x} = (\mathbf{p}, \boldsymbol{\theta})^T$

Common coordinate frames used throughout this thesis are G, the global or world frame, B, the inertial body frame, and C, the camera frame.



$$ {}^{B}_{G}\mathbf{R} = \begin{bmatrix} cos(\theta) & sin(\theta) & 0 \\ -sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{B}_{G}\mathbf{q} = \begin{bmatrix} 0 \\ 0 \\ sin(\theta/2) \\ cos(\theta/2) \end{bmatrix} $$

Figure 2.2. A simple example of the rotation and quaternion convention from [39]

3. Probabilistic State Estimation

Filter based approaches for state estimation make heavy use of probability theory. When tracking the location of a robot, we wish to consider its location probabilisitically, rather than explicitly, so we can keep track of the uncertainty in our estimate. In this chapter we explain the basics of probability theory insofar as they apply to state estimation. Specifically, we explain what it means to estimate the state and maintain a measure of the uncertainty. We then introduce linear filtering with the Kalman filter, and finally non-linear filtering with the Extended Kalman Filter.

3.1. Probability Theory Basics

We model the true unknown state as a random variable, x, with a probability distribution. This distribution has a **probability density function** (pdf): a continous function, p, that describes the relative likelihood of x. The pdf is not the same thing as the probability. Unlike the probability, a pdf can have values higher than 1. It is important, however, that the total area under the curve is equal to 1. The probability that x is within a certain range is equal to the integral of p in that range.

$$Pr(a \leq x \leq b) = \int_a^b p(x)\, dx \tag{3.1}$$

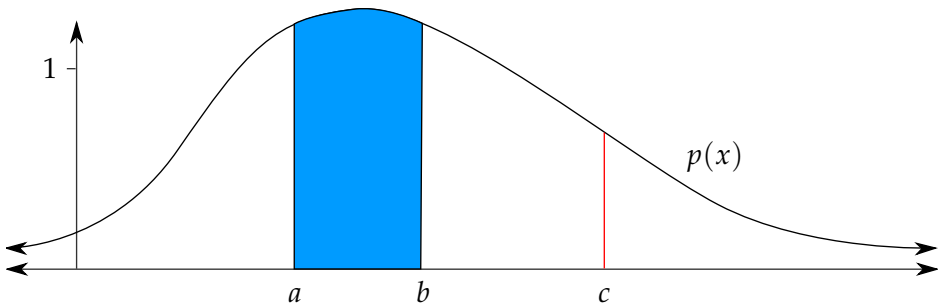


Figure 3.1. An example probability density function. The probability $Pr(a \leq x \leq b)$ is the blue area while $Pr(x = c) = 0$. The total area under the curve must equal 1.

The weighted average of all possible values of x is the *expected value* of x, or $E[x]$. The expected value is determined by multiplying every possible value of x by $p(x)$, which acts as a weight for that value, and summing the results. In some cases, notably the **normal distribution**, the expected value is the same as the maximum value of $p(x)$. The *variance* of x, $\mathrm{Var}[x]$, is the average squared difference between each value of x and the expected

value. High values for the variance mean that x is distributed far from the mean. If x is a sensor measurement, it could mean the sensor is heavily currupted by noise.

$$E[x] = \int x\, p(x)\, dx \tag{3.2}$$

$$\mathrm{Var}[x] = E[(x - E[x])^2] = \sigma^2 \tag{3.3}$$

We represent the standard deviation of x as σ, which is the square root of the variance.

3.2. The Normal Distribution

The most useful and common class of probability density functions is the normal, or Gaussian, distribution. The normal distribution has a number of advantages.

- Sensor noise is usually distributed normally, making it a natural choice for modeling sensor data. See figure 3.2. This is called Gaussian white noise.
- The expected value, $E[x]$, is also the value that maximizes the pdf and is the mean of the Gaussian.
- The mean and variance are the parameters for the Gaussian distribution, making it intuitive and allowing for easy analysis.
- A linear transformation of a Gaussian random variable remains Gaussian.
- The intersection, or joint probabilty, of two Gaussian random variables also remains Gaussian.

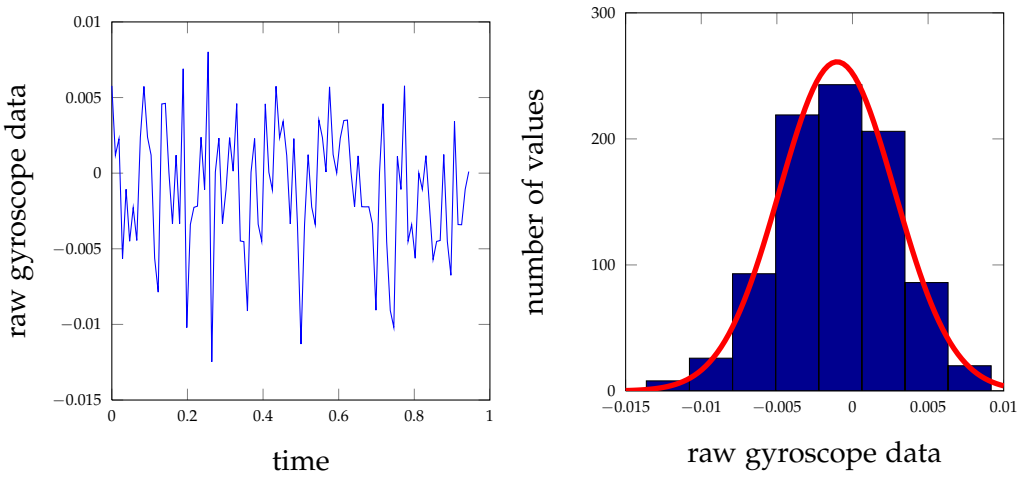


Figure 3.2. Gyroscope data taken from an iPhone 4S. Although the phone is stationary, the sensor data is currupted by noise. This noise follows the normal distribution.

3.2.1. One-Dimensional Case

In one dimension, the formula for the probability density function (*pdf*) of the normal distribution is as follows.

$$p(x;\mu,\sigma) = \underbrace{\frac{1}{\sqrt{2\pi\sigma^2}}}_{\text{normalization term}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \tag{3.4}$$

The mean, μ, and variance, σ^2, are the parameters that determine the shape of the Gaussian. The normalization term ensures that the total probability (the area under the curve) is equal to 1. Figure 3.3 shows the effect of different values for the mean and variance in the one dimensional case. Low values of the variance create a narrow and high peak around the mean, resulting in a high probability that x is near μ. This is why the variance is often refered to as the uncertainty.

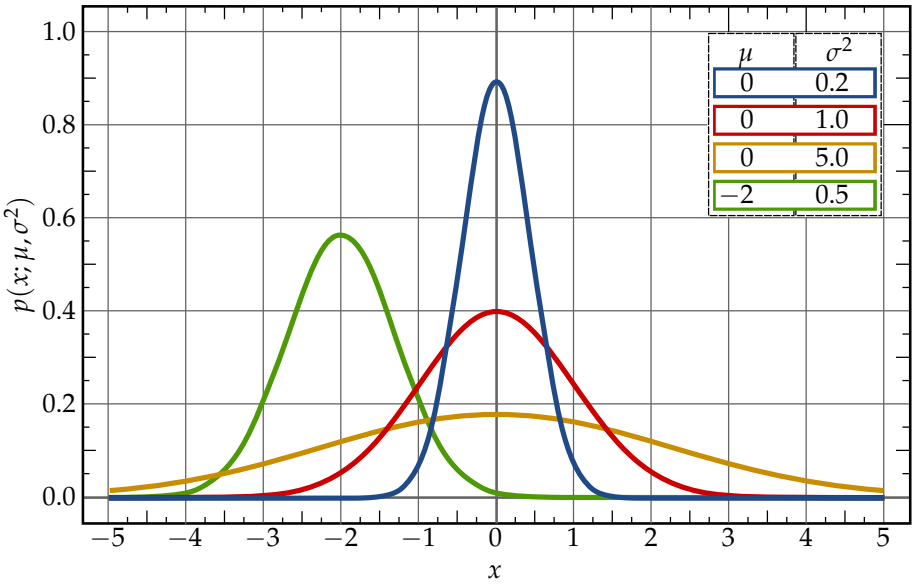


Figure 3.3. The normal distribution is a bell shaped curve centered at the mean, μ. High values of σ, the standard deviation, make the curve wide while low values result in a narrow shape.

3.2.2. Multidimensional Case

In pose estimation, we model the true unknown state, $\mathbf{x}$, as a random multidimensional variable with a multivariate Gaussian distribution, written as $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. Here $\mathbf{x}$ and $\boldsymbol{\mu}$ are $n \times 1$ vectors while $\boldsymbol{\Sigma}$ is an $n \times n$ matrix. Now instead of a single value for the uncertainty (σ^2) we have a full **covariance matrix** $\boldsymbol{\Sigma}$. The diagonal entries of $\boldsymbol{\Sigma}$ are simply the variances of each element of $\mathbf{x}$: $(\sigma^2_{x_1}, \sigma^2_{x_2}, \ldots \sigma^2_{x_n})$. The non-diagonal entries are $\rho_{(x_a,x_b)}\sigma_{x_a}\sigma_{x_b}$ where ρ is the **correlation**.

$$\mathrm{Cov}(x, y) = E[(x - E[x])(y - E(y))] \tag{3.5}$$

$$\mathrm{Cov}\left(\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}\right) = \begin{bmatrix} \sigma_{x_1}^2 & \rho_{(x_1,x_2)}\sigma_{x_1}\sigma_{x_2} & \cdots & \rho_{(x_1,x_n)}\sigma_{x_1}\sigma_{x_n} \\ \rho_{(x_2,x_1)}\sigma_{x_2}\sigma_{x_1} & \sigma_{x_2}^2 & \cdots & \rho_{(x_2,x_n)}\sigma_{x_2}\sigma_{x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \rho_{(x_n,x_1)}\sigma_{x_n}\sigma_{x_1} & \rho_{(x_n,x_2)}\sigma_{x_n}\sigma_{x_2} & \cdots & \sigma_{x_n}^2 \end{bmatrix} \tag{3.6}$$

What follows is the n-dimensional multivariate Gaussian probability density function. Note that in practice, the Gaussian pdf is not explicitly evaluated. This is because we are only interested in the mean and covariance of $\mathbf{x}$, and as we shall see, the nature of the Gaussian distribution allows for us to maintain and updated these values during the filtering process.

$$p_n(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^n |\boldsymbol{\Sigma}|}} \exp(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu})) \tag{3.7}$$

where $|\boldsymbol{\Sigma}|$ is the determinant of $\boldsymbol{\Sigma}$

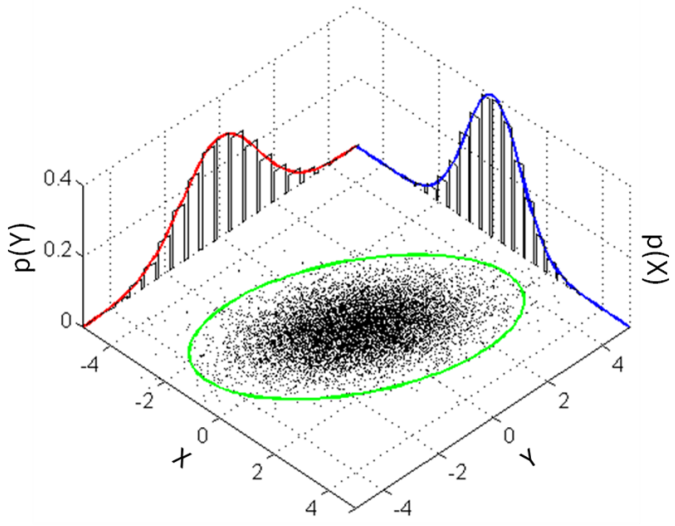


Figure 3.4. A two-dimensional normal distribution

3.2.3. Operations on a Gaussian

In this subsection we discuss important tools that are useful when working with Gaussian random variables. The special properties of the normal distribution allow certain operations on Gaussians to be calculated in closed form (i.e. without estimation) and with only the knowledge of the mean and covariance. Those operations are intersection[1], marginal-

[1]Intersection of a random variable with a linear combination of that variable, but not intersection in general.

ization, and conditioning.

Consider a two-dimensional Gaussian random variable $(x, y)^T \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. If we rewrite this by partitioning the mean and covariance matrix, we get the following joint probability distribution.

$$\begin{bmatrix} x \\ y \end{bmatrix} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \mathcal{N}\left(\begin{bmatrix} \mu_x \\ \mu_y \end{bmatrix}, \begin{bmatrix} \Sigma_{xx} \Sigma_{xy} \\ \Sigma_{yx} \Sigma_{yy} \end{bmatrix}\right) \tag{3.8}$$

We write the partitions of $\boldsymbol{\Sigma}$ in capital letters despite being 1×1 matrices to show that the following operations scale to multiple dimensions for x and y. In the two-dimensional case, $\Sigma_{xx} = \sigma_x^2$. The probability density function will have the following form (from equation 3.7).

$$p\left(\begin{bmatrix} x \\ y \end{bmatrix}; \boldsymbol{\mu}, \boldsymbol{\Sigma}\right) = \eta \exp\left(-\frac{1}{2}\left(\begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} \mu_x \\ \mu_y \end{bmatrix}\right)^T \begin{bmatrix} \Sigma_{xx} & \Sigma_{xy} \\ \Sigma_{yx} & \Sigma_{yy} \end{bmatrix}^{-1} \left(\begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} \mu_x \\ \mu_y \end{bmatrix}\right)\right)$$

$$\text{with } \eta = \frac{1}{2\pi|\boldsymbol{\Sigma}|^{1/2}} \text{ and } \Sigma_{xy} = \Sigma_{yx}^T$$

Marginalization

It is a common circumstance that we know the joint pdf of x and y but we only are interested in that of x. If we wish to find the pdf of x alone, we must *marginalize* out y. Here we see an advantage of the normal distribution as $p(x)$ is simple to determine with knowledge of $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$. We omit the proof for brevity.

$$p(x) = \int p(x, y)\, dy \tag{3.9}$$

$$= \int p(x|y)p(y)\, dy \tag{3.10}$$

$$= \mathcal{N}(\mu_x, \Sigma_{xx}) \tag{3.11}$$

Conditioning

Another common situation is that we know the joint pdf of x and y, and are given a specific value $y = y_0$. In this case, we can find the pdf of x through *conditioning*. The main point is that conditioning and marginalization can be done in closed form with the mean and covariance of the multivariate Gaussian.

$$x|_{y=y_0} \sim \mathcal{N}(\mu_x + \underbrace{\Sigma_{xy}\Sigma_{yy}^{-1}(y_0 - \mu_y)}_{\text{mean offset}}, \Sigma_{xx} - \underbrace{\Sigma_{xy}\Sigma_{yy}^{-1}\Sigma_{yx}}_{\text{covariance offset}}) \tag{3.12}$$

Note that the new mean for x has been shifted based on the the correlation between x and y and how closely y_0 was to the mean μ_y. In other words, if $y_0 = \mu_y$ then the mean μ_x remains the same. Similarly, if x and y are uncorrelated, then the off-diagonal entries of $\boldsymbol{\Sigma}$, Σ_{xy}, equal 0, making the new updated covariance simply Σ_{xx}. If they are correlated, however, then knowing the value of y will intuitively decrease the uncertainty of x.

Intersection

If we know $x \sim \mathcal{N}(\mu_x, \Sigma_{xx})$ and y is a linear combination of x, we can find the joint probability $p(x, y)$, also called the intersection. Let $y = Ax + b$ where A is a constant and $b \sim \mathcal{N}(0, Q)$.

$$\begin{aligned} p(\begin{bmatrix} x \\ y \end{bmatrix}) &= \mathcal{N}(\begin{bmatrix} \mu_x \\ \mu_y \end{bmatrix}, \begin{bmatrix} \Sigma_{xx} & \Sigma_{xy} \\ \Sigma_{yx} & \Sigma_{yy} \end{bmatrix}) \\ &= \mathcal{N}(\begin{bmatrix} \mu_x \\ A\mu_x + b \end{bmatrix}, \begin{bmatrix} \Sigma_{xx} & \Sigma_{xx}A^T \\ A\Sigma_{xx} & A\Sigma_{xx}A^T + Q \end{bmatrix}) \end{aligned}$$

3.3. The Kalman Filter

Here we introduce the Kalman filter and explain how it can be derived using the previously explained operations on Gaussians. Kalman Filters are extremely useful for, among other things, pose estimation of a linear system. For this purpose, we assume a robot's pose changes linearly according to some motion model. For example, a train car on a level plane with no known forces acting upon it will eventually slow down and stop. With some knowledge about the dynamics of the train car and some classical physics, we can calculate the probability density of the next state *given* the current state. We define the linear relationship between the next state and the current state as $\mathbf{x}_{t+1} = \mathbf{A}_t\mathbf{x}_t$. We can use this information to propagate the probability density function. If $\mathbf{x}_t \sim \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$ and $\mathbf{x}_{t+1} = \mathbf{A}_t\mathbf{x}_t$ then through intersection, $\mathbf{x}_{t+1} \sim \mathcal{N}(\mathbf{A}_t\boldsymbol{\mu}_t, \mathbf{A}_t\boldsymbol{\Sigma}_t\mathbf{A}_t^T)$.

If there is some more information about how $\mathbf{x}$ changes, such as an action or a measurement which is *independent* of the current state, we can also use this information to propagate the probability. If the variable $\mathbf{u}$ represents the velocity of a train car, it is both independent of the current state (if the state is the position) and it affects the next state. In many applications of the Kalman filter, $\mathbf{u}$ is called the control input and describes things like movement commands sent to a robot. It is entirely possible, however, that $\mathbf{u}$ represents a sensor measurement, as long as it is both independent of $\mathbf{x}$ and has a linear relationship with it. In either case, there is a some amount of noise inherent in the model $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{Q})$, a small unknown value which will increase the uncertainty in the pose estimate.

$$\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t) \tag{3.13}$$

$$\mathbf{x}_{t+1} = \mathbf{A}_t\mathbf{x}_t + \mathbf{B}_t\mathbf{u}_t + \boldsymbol{\epsilon}_t \tag{3.14}$$

In order to find $p(\mathbf{x}_{t+1})$, we can first use intersection to find $p(\mathbf{x}_{t+1}, \mathbf{x}_t)$. Then we can marginalize out $\mathbf{x}_t$.

$$p(\mathbf{x}_{t+1}, \mathbf{x}_t) = \mathcal{N}(\begin{bmatrix} \boldsymbol{\mu}_t \\ \mathbf{A}_t\boldsymbol{\mu}_t + \mathbf{B}_t\mathbf{u}_t \end{bmatrix}, \begin{bmatrix} \boldsymbol{\Sigma}_t & \boldsymbol{\Sigma}_t\mathbf{A}_t^T \\ \mathbf{A}_t\boldsymbol{\Sigma}_t & \mathbf{A}_t\boldsymbol{\Sigma}_t\mathbf{A}_t^T + \mathbf{Q}_t \end{bmatrix}) \tag{3.15}$$

$$\mathbf{x}_{t+1} \sim \mathcal{N}(\mathbf{A}_t\boldsymbol{\mu}_t + \mathbf{B}_t\mathbf{u}_t, \mathbf{A}_t\boldsymbol{\Sigma}_t\mathbf{A}_t^T + \mathbf{Q}_t) \tag{3.16}$$

$$\mathbf{x}_{t+1} \sim \mathcal{N}(\boldsymbol{\mu}_{t+1}, \boldsymbol{\Sigma}_{t+1}) \tag{3.17}$$

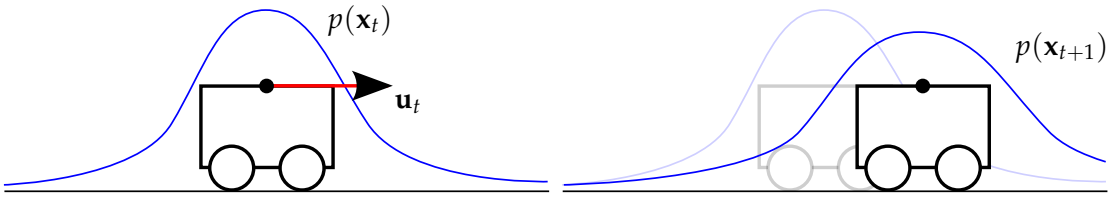


Figure 3.5. The train car's position pdf changes based on a simple motion model. The pdf expands horizontally and shrinks vertically because uncertainty in the motion model is directly translated into uncertainty in the state.

Over time, the motion model will increase the uncertainty of the state estimate indefinitely. To reduce the uncertainty, direct measurements of the state are required. Such a measurement, $\mathbf{z}$, can be used to update the pdf of $\mathbf{x}$, assuming we know the linear relationship of the true values $\mathbf{z} = \mathbf{C}\mathbf{x} + \delta$, where $\delta \sim \mathcal{N}(\mathbf{0}, \mathbf{R})$ is the measurement noise. Here we find the intersection $p(\mathbf{x}_t, \mathbf{z}_t)$ and then use conditioning (equation 3.12) to find the updated mean and covariance of $\mathbf{x}_t$.

$$p(\mathbf{x}_t, \mathbf{z}_t) = \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu}_t \\ \mathbf{C}_t\boldsymbol{\mu}_t \end{bmatrix}, \begin{bmatrix} \boldsymbol{\Sigma}_t & \boldsymbol{\Sigma}_t\mathbf{C}_t^T \\ \mathbf{C}_t\boldsymbol{\Sigma}_t & \mathbf{C}_t\boldsymbol{\Sigma}_t\mathbf{C}_t^T + \mathbf{R}_t \end{bmatrix}\right) \tag{3.18}$$

$$\mathbf{x}_t|_{\mathbf{z}_t=\mathbf{z}_o} \sim \mathcal{N}(\boldsymbol{\mu}_t + \mathbf{K}_t(\mathbf{z}_o - \mathbf{C}_t\boldsymbol{\mu}_t),\ \boldsymbol{\Sigma}_t - \mathbf{K}_t\mathbf{C}\boldsymbol{\Sigma}_t) \tag{3.19}$$

$$\mathbf{K}_t = \boldsymbol{\Sigma}_t\mathbf{C}_t^T(\mathbf{C}_t\boldsymbol{\Sigma}_t\mathbf{C}_t^T + \mathbf{R}_t)^{-1} \tag{3.20}$$

An important assumption in pose estimation is that each state is modeled as being only dependent on the previous state and the control input, $\mathbf{u}$. Similarly, observations are only dependent on the current state. Assuming independence from older states is known as the **Markov Assumption** and is a key element in filtering algorithms.

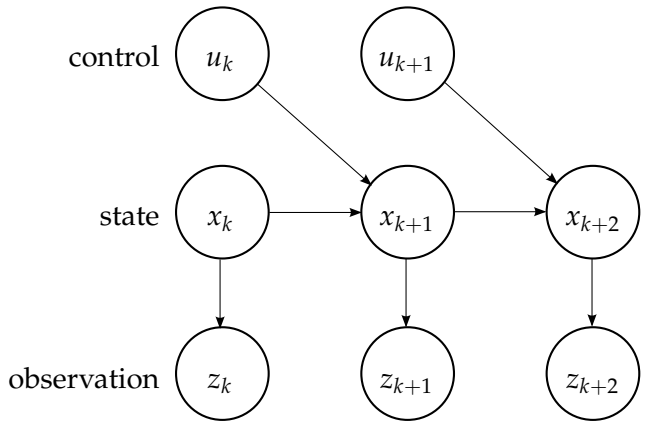


Figure 3.6. The Markov assumption simplifies recursive calculations by asserting independence from old states and measurements.

3.3.1. Kalman Filter Summary

We now use a slightly different notation to remain consistent with the rest of this thesis. Specifically, instead of $\boldsymbol{\mu}$ for the mean of the state pdf, we use the estimate $\hat{\mathbf{x}}$. Furthermore, we now use the notation $\hat{\mathbf{x}}_{t|t-1}$, which is read as "the state estimate at time t given measurements from time 0 through $t-1$".

Initial Estimate

$$\mathbf{x}_0 \sim \mathcal{N}(\hat{\mathbf{x}}_{0|0},\, \boldsymbol{\Sigma}_{0|0})$$

Prediction Step

$$\begin{aligned}
\textbf{Given: } \mathbf{x}_{t+1} &= \mathbf{A}_t\mathbf{x}_t + \mathbf{B}_t\mathbf{u}_t + \boldsymbol{\epsilon}_t \text{ where } \boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0},\, \mathbf{Q}_t) \\
\hat{\mathbf{x}}_{t+1|t} &= \mathbf{A}_t\hat{\mathbf{x}}_{t|t} + \mathbf{B}_t\mathbf{u}_t \\
\boldsymbol{\Sigma}_{t+1|t} &= \mathbf{A}_t\boldsymbol{\Sigma}_{t|t}\mathbf{A}_t^T + \mathbf{Q}_t
\end{aligned}$$

Update Step

$$\begin{aligned}
\textbf{Given: } \mathbf{z}_t &= \mathbf{C}_t\mathbf{x}_t + \boldsymbol{\delta}_t \text{ where } \boldsymbol{\delta}_t \sim \mathcal{N}(\mathbf{0},\, \mathbf{R}_t) \\
\hat{\mathbf{x}}_{t|t} &= \hat{\mathbf{x}}_{t|t-1} + \mathbf{K}_t(\mathbf{z}_t - \mathbf{C}_t\hat{\mathbf{x}}_{t|t-1}) \\
\boldsymbol{\Sigma}_{t|t} &= \boldsymbol{\Sigma}_{t|t-1} - \mathbf{K}_t\mathbf{C}_t\boldsymbol{\Sigma}_{t|t-1} \\
\mathbf{K}_t &= \boldsymbol{\Sigma}_{t|t-1}\mathbf{C}_t^T(\mathbf{C}_t\boldsymbol{\Sigma}_{t|t-1}\mathbf{C}_t^T + \mathbf{R}_t)^{-1}
\end{aligned}$$

3.4. Example

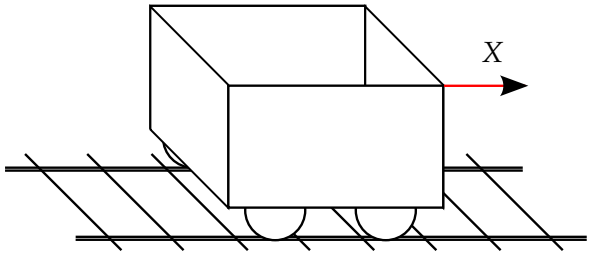


Figure 3.7. A simple car that can only move on one axis.

A simple train car can move forward and backward in only one dimension (Figure 3.7. We wish to estimate its position and velocity with the use of some basic sensors. First we choose the state representation. $\mathbf{x} = (p, v)^T$, where p is the position and v is the velocity. Our initial state is $\mathbf{x}_0 = \mathbf{0}_{2\times 1}$.

The car is equipped with a single-axis accelerometer working at 100Hz, which measures the linear acceleration, a, of the device. We will use this measurement for the prediction step of the Kalman Filter. For the purposes of this example, the accelerometer measurements are one dimensional, bias free, and corrupted by noise $n_a \sim \mathcal{N}(0, \sigma_a^2)$. First we

identity the system dynamics.

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{p} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} v \\ a \end{bmatrix}$$
$$p_{t+1} = p_t + \int_t^{t+1} v_\tau d\tau \simeq p_t + v_t \Delta t$$
$$v_{t+1} = v_t + \int_t^{t+1} a_\tau d\tau \simeq v_t + a_t \Delta t$$

Here $\dot{\mathbf{x}}$ is the state's continuous time derivative. It is clear that the system is linear and the prediction step can proceed as follows.

$$\hat{\mathbf{x}}_{t+1|t} = \mathbf{A}_t \hat{\mathbf{x}}_{t|t} + \mathbf{B}_t \mathbf{u}_t$$
$$\begin{bmatrix} \hat{p}_{t+1|t} \\ \hat{v}_{t+1|t} \end{bmatrix} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \hat{p}_{t|t} \\ \hat{v}_{t|t} \end{bmatrix} + \begin{bmatrix} 0 \\ \Delta t \end{bmatrix} a_{m_t}$$
$$\mathbf{\Sigma}_{t+1|t} = \mathbf{A}_t \mathbf{\Sigma}_{t|t} \mathbf{A}_t^T + \mathbf{Q}_t$$
$$\mathbf{Q}_t = \begin{bmatrix} 0 & 0 \\ 0 & \sigma_a^2 \end{bmatrix}$$

Less frequently we are able to measure the velocity of the device with an speedometer. Because the velocity is part of the state vector, this measurement must be used in the update step of the Kalman filter. Let $\mathbf{z}_t$ be the measured velocity at time t currupted by noise with variance σ_v^2. The matrix $\mathbf{C}$ defines the linear relationship between $\mathbf{z}$ and $\mathbf{x}$, which, because the velocity is part of the state vector, is rather simple.

$$\mathbf{z}_t = \mathbf{C}_t \mathbf{x} + \delta$$
$$\text{where } \delta \sim \mathcal{N}(\mathbf{0}, \mathbf{R}) \text{ and } \mathbf{R}_t = \sigma_v^2$$
$$\mathbf{z}_t = \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} p_t \\ v_t \end{bmatrix} + \delta_t$$

The Kalman update compares the measured value $\mathbf{z}_t$ with the value it would expect based on the state estimate, $\mathbf{C}_t \hat{\mathbf{x}}_t$. The Kalman filter update then proceeds as follows.

$$\hat{\mathbf{x}}_{t|t} = \hat{\mathbf{x}}_{t|t-1} + \mathbf{K}_t(\mathbf{z}_t - \mathbf{C}_t \hat{\mathbf{x}}_{t|t-1})$$
$$\mathbf{\Sigma}_{t|t} = \mathbf{\Sigma}_{t|t-1} - \mathbf{K}_t \mathbf{C}_t \mathbf{\Sigma}_{t|t-1}$$
$$\text{with } \mathbf{K}_t = \mathbf{\Sigma}_{t|t-1} \mathbf{C}_t^T (\mathbf{C}_t \mathbf{\Sigma}_{t|t-1} \mathbf{C}_t^T + \mathbf{R}_t)^{-1}$$

3.4.1. Example Simulation

We present a simple simulation of the train car example. The car moves back and forth along a simple sinusoidal wave pattern while its pose is estimated with the Kalman filter.

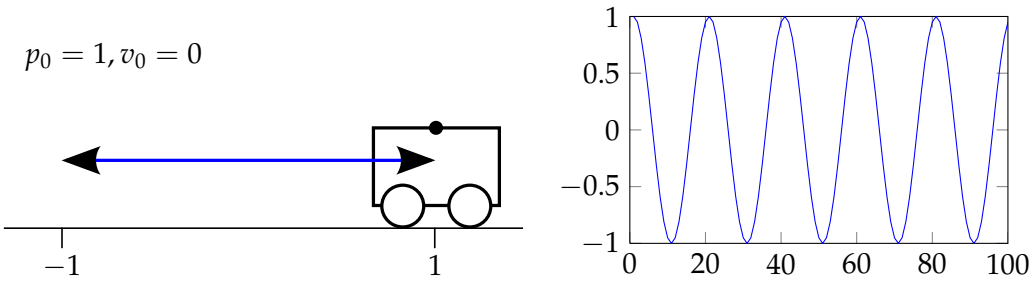


Figure 3.8. The train car's position over time.

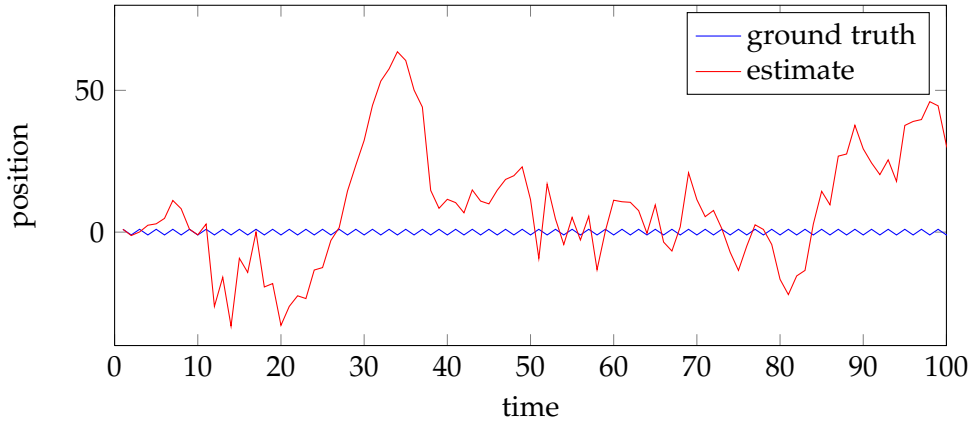


Figure 3.9. The position error is large and grows even larger over time. This is because the position is not observable.

The resulting pose estimate after a long period appears very wrong. However, if we observe the error compared to the maintained uncertainty (Figure 3.10), we can see that it remains well bounded in 3σ, where σ is the square root of the corresponding diagonal entry in $\mathbf{\Sigma}$. Because the velocity is being directly observered, the uncertainty is maintained at a consistently low level. The position's uncertainty, however, is unbounded. This is because the position in this example is unobservable.

3.5. Observability

If it is possible to determine the behavior of an entire system given only the system's outputs, then the system is *observable*. In terms of a Kalman Filter, the measurement vector $\mathbf{z}$ is considered the output of the system. The system is then observable if these outputs are enough to determine the true state. It can also be that the system is only partially

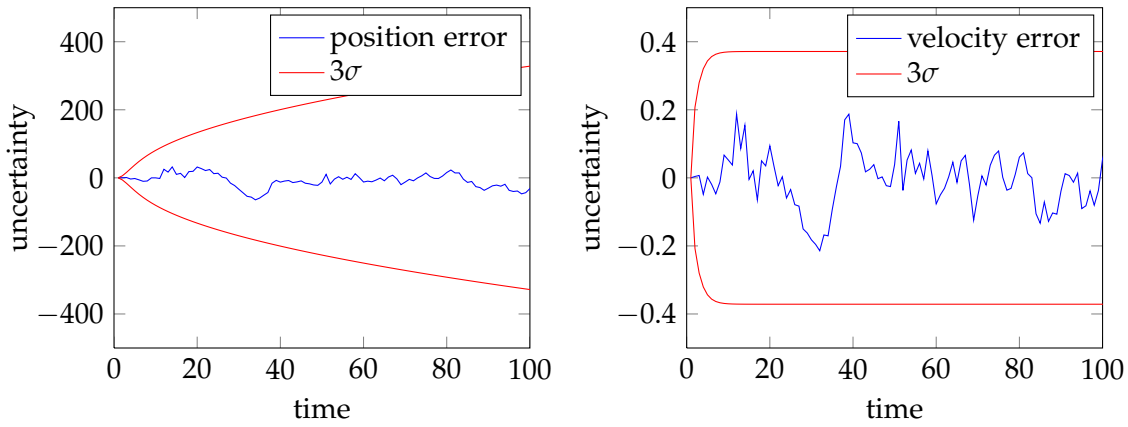


Figure 3.10. The error in the position and velocity of the train car remain within the bounds of three standard deviations. The velocity is observable and the uncertainty remains bounded, but the position is not observable, causing the uncertainty to increase over time.

observable. More formally, the system is observable at time t_k if the state vector x_k can be determined with measurements $\mathbf{z} = \{z_0, z_1, \ldots, z_k\}$, with $t_0 < t_k < \infty$.

To find if a system is observable, we construct and analyze the *observability matrix*, $\mathcal{O}_k$. Then, if $\mathcal{O}_k^T \mathcal{O}_k$ has full rank, the system is observable.

$$\mathcal{O}_k = \begin{bmatrix} \mathbf{C}_0 \\ \mathbf{C}_1 \mathbf{A}_0 \\ \mathbf{C}_2 \mathbf{A}_1 \mathbf{A}_2 \\ \vdots \\ \mathbf{C}_k \mathbf{A}_{k-1} \ldots \mathbf{A}_0 \end{bmatrix} \tag{3.21}$$

$$\mathcal{O}_k^T \mathcal{O}_k = \mathbf{C}_0^T \mathbf{C}_0 + \sum_{i=1}^{k} \mathbf{A}_{i-1,0}^T \mathbf{C}_i^T \mathbf{C}_i \mathbf{A}_{i-1,0} \tag{3.22}$$

It is easily shown that the train car example above is an unobservable system, because the rank of $\mathcal{O}_k^T \mathcal{O}_k$ for any k is 1. It is also possible to show that the velocity *is* observable while the position is not. To do this, we can simply examine the columns of the observability matrix seperately. For a particular row i of $\mathcal{O}$,

$$\mathcal{O}^{(i)} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = 0$$

Because this holds for all values of i, we can see that the position is unobservable because the it belongs to the null space of $\mathcal{O}$. On the other hand, the same does not hold for the velocity.

$$\mathcal{O}^{(i)} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = 1$$

Figure 3.10 shows the clear difference between observable and unobservable parameters in a system. Although the position can be estimated, the uncertainty is unbounded and will grow indefinitely. In contrast the velocity is observable, which we would expect given direct velocity readings, but direct readings are not required for observability. Additionally, observability is necessary but not sufficient to ensure a bounded uncertainty. For example, high values for the noise matrices $\mathbf{Q}$ and $\mathbf{R}$ can cause the uncertainty to increase towards infinity.

3.6. Extended Kalman Filter

Ultimately we are interested in pose estimation of a device with a full possible range of motion. This means 3 degrees of freedom in position, and 3 in the orientation. Estimating this pose with the Kalman filter would be impossible because the dynamics are non-linear. To account for this, we now introduce the non-linear Extended Kalman Filter (EKF). The two main differences are in the assumptions about the dynamical motion model and the sensor model.

$$\mathbf{x}_{t+1} = \mathbf{f}(\mathbf{x}_t, \mathbf{u}_t, \mathbf{w}_t) \qquad \mathbf{w}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_t) \tag{3.23}$$

$$\mathbf{z}_t = \mathbf{h}(\mathbf{x}_t, \mathbf{v}_t) \qquad \mathbf{v}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_t) \tag{3.24}$$

The functions $\mathbf{f}$ and $\mathbf{h}$ are non-linear functions. Due to this non-linearity, we can no longer simply find the intersection of $\mathbf{x}_{t+1}$ and $\mathbf{x}_t$ as in the Kalman filter. The EKF works around this problem by linearizing the functions $\mathbf{f}$ and $\mathbf{h}$ around the current estimate $\hat{\mathbf{x}}_t$. To accomplish this, we make use of the **Taylor Series Expansion**: any differentiable function $f(x)$ can be approximated at x with a series as follows:

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)(x-a)^2}{2!} + \frac{f'''(a)(x-a)^3}{3!} + \dots \tag{3.25}$$

The terms in the series decrease in size and importance at higher orders and for an approximation can be ignored. We use only the terms up to the first order, making the EKF a first order estimator.

3.6.1. Prediction Step

Consider the function $\mathbf{f}(\mathbf{x}, \mathbf{u}, \mathbf{w})$ at time t. The Taylor series expansion gives

$$\mathbf{f}(\mathbf{x}_t, \mathbf{u}_t, \mathbf{w}_t) \approx \mathbf{f}(\hat{\mathbf{x}}_t, \mathbf{u}_t) + \mathbf{f}'(\hat{\mathbf{x}}_t, \mathbf{u}_t)(\mathbf{x}_t - \hat{\mathbf{x}}_t) + \dots \tag{3.26}$$

$$\approx \mathbf{f}(\hat{\mathbf{x}}_t, \mathbf{u}_t) + \mathbf{F}(\hat{\mathbf{x}}_t, \mathbf{u}_t)\tilde{\mathbf{x}}_t \tag{3.27}$$

Here $\tilde{\mathbf{x}}$ is the state error, the difference between the true state and the state estimate, while $\mathbf{F}$ is the Jacobian matrix for $\mathbf{f}$. If $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_n(\mathbf{x}))^T$ and $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$

$$\mathbf{F} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix} \tag{3.28}$$

We can now propagate the state estimate for the EKF prediction step. By comparing equation 3.23 with 3.27, we see that the next state can be estimated.

$$\hat{\mathbf{x}}_{t+1} = \mathbf{f}(\hat{\mathbf{x}}_t, \mathbf{u}_t) \tag{3.29}$$

The propagated error can now be represented as a linear combination of the previous error and the noise. We write $\mathbf{F}(\hat{\mathbf{x}}_t, \mathbf{u}_t)$ as $\mathbf{F}_t$ for readability.

$$\tilde{\mathbf{x}}_{t+1} = \mathbf{F}_t \tilde{\mathbf{x}}_t + \mathbf{G}_t \mathbf{w}_t \tag{3.30}$$

$$\text{with } \tilde{\mathbf{x}}_{t+1} = \mathbf{x}_{t+1} - \hat{\mathbf{x}}_{t+1} \tag{3.31}$$

The matrix $\mathbf{G}$ is the Jacobian of $\mathbf{f}$ with respect to the noise. It also serves to transform the noise vector into the same dimensionality as the state error. Making use of the previously mentioned definition of the covariance, $\text{Cov}(x, y) = E[(x - E[x])(y - E(y))]$, we can find the propagated covariance matrix at time $t + 1$.

$$\begin{aligned} \mathbf{\Sigma}_{t+1} &= E[(\mathbf{x}_{t+1} - \hat{\mathbf{x}}_{t+1})(\mathbf{x}_{t+1} - \hat{\mathbf{x}}_{t+1})^T] && (3.32)\\ &= E[\tilde{\mathbf{x}}_{t+1}\, \tilde{\mathbf{x}}_{t+1}^T] && (3.33)\\ &= E[\mathbf{F}_t\, \tilde{\mathbf{x}}_t\, \tilde{\mathbf{x}}_t^T\, \mathbf{F}_t^T] + E[\mathbf{G}_t\, \mathbf{w}_t\, \mathbf{w}_t^T\, \mathbf{G}_t^T] && (3.34)\\ &= \mathbf{F}_t\, E[\tilde{\mathbf{x}}_t\, \tilde{\mathbf{x}}_t^T]\, \mathbf{F}_t^T + \mathbf{G}_t\, E[\mathbf{w}_t\, \mathbf{w}_t^T]\, \mathbf{G}_t^T && (3.35)\\ &= \mathbf{F}_t\, \mathbf{\Sigma}_t\, \mathbf{F}_t^T + \mathbf{G}_t\, \mathbf{Q}_t\, \mathbf{G}_t^T && (3.36) \end{aligned}$$

3.6.2. Discrete vs. Continuous Filtering

Until now we have discussed continuous time filtering, but in real applications we are given discrete non-linear time sensitive measurements. The method for discretization varies in the literature depending on application specifics. Assumptions made, if any, during discretization of the propagated state estimate should not have any effect on the uncertainty. Therefore, whenever possible, it is best to find an analytical closed form solution. If that is impossible, numerical integration is a possible alternative. In any case, we wish to find the matrix $\mathbf{\Phi}$ that satisfies

$$\tilde{\mathbf{x}}_{t+1} = \mathbf{\Phi}_t\, \tilde{\mathbf{x}}_t + \mathbf{w}_{d_t} \tag{3.37}$$

with discrete noise $\mathbf{w}_{d_t} \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_d)$. This can be accomplished by solving the differential equation $\dot{\mathbf{\Phi}}(t, t_i) = \mathbf{F}(t)\, \mathbf{\Phi}(t, t_i)$ from $t \in [t, t + 1]$ with initial condition $\mathbf{\Phi}_t = \mathbf{I}$.

The noise covariance also needs to be discretized as follows, where $\mathbf{Q}_c$ is the covariance

matrix of the continuous noise vector.

$$\mathbf{Q}_d = \int_t^{t+1} \mathbf{\Phi}(t+1,\tau)\,\mathbf{G}(\tau)\,\mathbf{Q}_c\,\mathbf{G}(\tau)^T\,\mathbf{\Phi}(t+1,\tau)^T\,d\tau \tag{3.38}$$

3.6.3. Update Step

Given a measurement $\mathbf{z}_t = \mathbf{h}(\mathbf{x}_t) + \mathbf{v}_t$, we can once again use the Taylor expansion to find $\mathbf{h}(\mathbf{x}_t)$ with the current estimate.

$$\mathbf{h}(\mathbf{x}_t) = \mathbf{h}(\hat{\mathbf{x}}_t) + \mathbf{H}(\hat{\mathbf{x}}_t)(\mathbf{x}_t - \hat{\mathbf{x}}_t) + \dots \tag{3.39}$$
$$\mathbf{h}(\mathbf{x}_t) \approx \mathbf{h}(\hat{\mathbf{x}}_t) + \mathbf{H}(\hat{\mathbf{x}}_t)\tilde{\mathbf{x}}_t \tag{3.40}$$

where $\mathbf{H}$ is the Jacobian of $\mathbf{h}$ with respect to $\mathbf{x}$ (see equation 3.28). The Jacobian $\mathbf{H}$ can then be used in the EKF update equations. It follows from equation 3.40 that

$$\mathbf{h}(\mathbf{x}_t) - \mathbf{h}(\hat{\mathbf{x}}_t) \approx \mathbf{H}_t\tilde{\mathbf{x}} \tag{3.41}$$
$$\mathbf{z}_t - \mathbf{h}(\hat{\mathbf{x}}_t) \approx \mathbf{H}_t\tilde{\mathbf{x}} - \mathbf{v}_t \tag{3.42}$$
$$\mathbf{r}_t = \mathbf{H}_t\tilde{\mathbf{x}}_t + \mathbf{n}_t \tag{3.43}$$

where $\mathbf{r}$ is the **residual** vector, the difference between the measurement and hypothesis, and $\mathbf{n}$ is noise. This is an important equation that shows that the residual must be a linear combination of the state error and noise. The update now proceeds in a similar manner to that of the normal Kalman filter.

$$\hat{\mathbf{x}}_{t|t} = \hat{\mathbf{x}}_{t|t-1} + \mathbf{K}_t(\mathbf{z}_t - \mathbf{h}(\hat{\mathbf{x}}_{t|t-1})) \tag{3.44}$$
$$\mathbf{\Sigma}_{t|t} = \mathbf{\Sigma}_{t|t-1} - \mathbf{K}_t\mathbf{H}_t\mathbf{\Sigma}_{t|t-1} \tag{3.45}$$
$$\mathbf{K}_t = \mathbf{\Sigma}_{t|t-1}\mathbf{H}_t^T(\mathbf{H}_t\mathbf{\Sigma}_{t|t-1}\mathbf{H}_t^T + \mathbf{R}_t)^{-1} \tag{3.46}$$

The derivation of the EKF update step is similar in principle to the prediction step. However, because it is complex and not directly relevant to this thesis, we omit it entirely. Interested readers should refer to Thrun's Probabilistic Robotics [38] for more information.

3.6.4. EKF Summary

The final EKF equations look very similar to the normal Kalman filter. In fact, if the functions $\mathbf{f}$ and $\mathbf{h}$ are linear, then the Extended Kalman Filter becomes a regular Kalman

filter.

Initial Estimate

$$\mathbf{x}_0 \sim \mathcal{N}(\hat{\mathbf{x}}_{0|0},\, \boldsymbol{\Sigma}_{0|0})$$

Prediction Step

$$\begin{aligned}
&\textbf{Given: } \mathbf{x}_{t+1} = \mathbf{f}(\mathbf{x}_t, \mathbf{u}_t) + \mathbf{w}_t \text{ where } \mathbf{w}_t \sim \mathcal{N}(\mathbf{0},\, \mathbf{Q}_t) \\
&\hat{\mathbf{x}}_{t+1|t} = \mathbf{f}(\hat{\mathbf{x}}_{t|t}, \mathbf{u}_t) \\
&\mathbf{F}_t = \frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x}_t, \mathbf{u}_t) \\
&\boldsymbol{\Sigma}_{t+1|t} = \mathbf{F}_t \boldsymbol{\Sigma}_{t|t} \mathbf{F}_t^T + \mathbf{Q}_t
\end{aligned}$$

Update Step

$$\begin{aligned}
&\textbf{Given: } \mathbf{z}_t = \mathbf{h}(\mathbf{x}_t) + \mathbf{v}_t \text{ where } \mathbf{v}_t \sim \mathcal{N}(\mathbf{0},\, \mathbf{R}_t) \\
&\hat{\mathbf{x}}_{t|t} = \hat{\mathbf{x}}_{t|t-1} + \mathbf{K}_t(\mathbf{z}_t - \mathbf{h}(\hat{\mathbf{x}}_{t|t-1})) \\
&\mathbf{H}_t = \frac{\partial \mathbf{h}}{\partial \mathbf{x}}(\mathbf{x}_t, \mathbf{u}_t) \\
&\boldsymbol{\Sigma}_{t|t} = \boldsymbol{\Sigma}_{t|t-1} - \mathbf{K}_t \mathbf{H}_t \boldsymbol{\Sigma}_{t|t-1} \\
&\mathbf{K}_t = \boldsymbol{\Sigma}_{t|t-1} \mathbf{H}_t^T (\mathbf{H}_t \boldsymbol{\Sigma}_{t|t-1} \mathbf{H}_t^T + \mathbf{R}_t)^{-1}
\end{aligned}$$

4. Inertial Measurement Unit

The Inertial Measurement Unit (IMU) is a commonly used electronic device usually comprised of accelerometers, gyroscopes and sometimes magnetometers. Modern IMUs are cheap to produce and are prevalent in, among other things, smartphones and aerial vehicles. We are primarily interested in the accelerometer and gyroscope because of their noise characteristics. The magnetometer (3-axis compass) is heavily influenced by nearby magnetic fields that cannot be easily modeled.

Figure 4.1. The iPhone 4S, AR Drone, and Oculus Rift all contain low cost MEMS IMUs.

4.1. Accelerometer

An accelerometer measures proper acceleration: its acceleration relative to a state of free-fall. Gravity does not cause proper acceleration. Therefore, an accelerometer in free-fall measures zero acceleration, while an accelerometer at rest on the surface of the earth measures one g, approximately $9.81\text{m}/\text{s}^2$, directly upwards.

Internally, accelerometers work like a spring-mass system as described in figure 4.2. Sometimes, the measurements given by the accelerometer are actually the direction that the mass is being pulled, rather than the acceleration. In this case, we negate the measured value to get the proper acceleration. Each accelerometer measures acceleration on only one axis, and are therefore usually found in groups of three orthogonal devices on a single low cost microelectromechanical systems (MEMS) chip. These 3-axis devices are found in many modern devices, e.g. smartphones, quadrocopters, and virtual reality headsets. An ideal accelerometer would give the following measurement.

$$ {}^{B}\mathbf{a}_m = {}^{B}_{G}\mathbf{R}({}^{G}\mathbf{a} - {}^{G}\mathbf{g}) \tag{4.1} $$

Here ${}^{G}\mathbf{a}$ is the true acceleration of the IMU in the global frame and $\{B\}$ represents the inertial body (IMU) frame. Our global frame of reference sets the z-axis upwards away from the earth so that $\mathbf{g} = (0, 0, -1)^T$. The subscript m of $\mathbf{a}_m$ denotes that it is the measurement. The accelerometer's unit of measurement is g. Low cost accelerometers, like

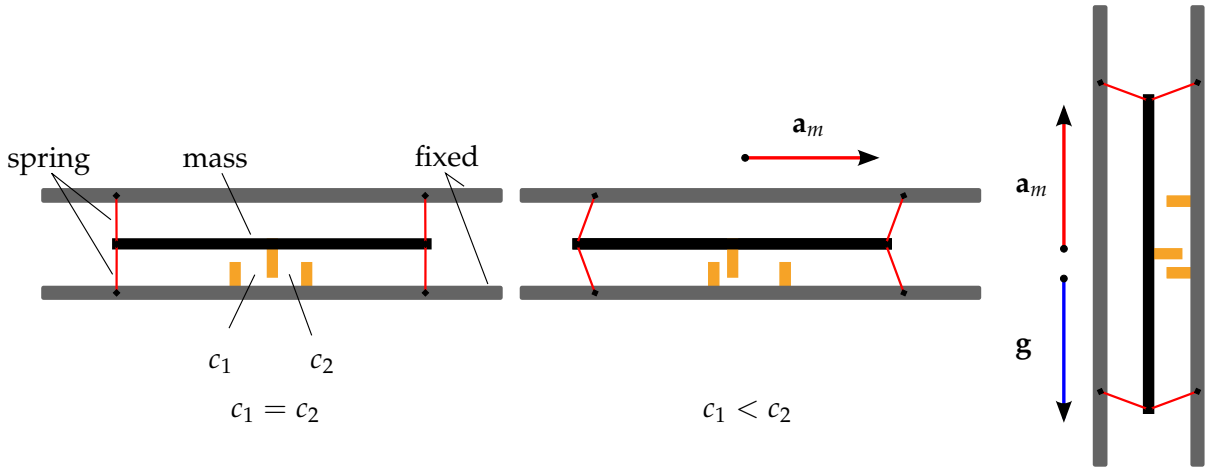


Figure 4.2. The accelerometer works like a spring-mass system. As the body accelerates, the mass drags behind, causing a difference in the measured capacitances c_1 and c_2. This illustrates why an accelerometer at rest measures positive acceleration upwards due to gravity.

those in smartphones, are far from ideal and give measurements corrupted by noise and bias.

$$ {}^{B}\mathbf{a}_m = {}^{B}_{G}\mathbf{R}({}^{G}\mathbf{a} - {}^{G}\mathbf{g}) + \mathbf{n}_a + \mathbf{b}_a \tag{4.2} $$

The noise $\mathbf{n}_a$ is a zero mean normally distributed random variable, $\mathbf{n}_a \sim \mathcal{N}(0, \mathbf{N}_a)$. The bias, $\mathbf{b_a}$, changes over time and is modeled as a random walk process driven by its own noise vector $\mathbf{n}_{wa} \sim \mathcal{N}(0, \mathbf{N}_{wa})$. Accelerometers can also suffer from misalignment and scale errors.

$$ {}^{B}\mathbf{a}_m = \mathbf{T}_a\, {}^{B}_{G}\mathbf{R}({}^{G}\mathbf{a} - {}^{G}\mathbf{g}) + \mathbf{n}_a + \mathbf{b}_a \tag{4.3} $$

Here $\mathbf{T}_a$ is the shape matrix causing both misalignment and scale errors in the accelerometer measurements. Scale errors can be made of static components or temperature related components and must be determined during calibration (see section **??**).

4.2. Gyroscope

The gyroscope measures rotational velocity, $\boldsymbol{\omega}$, of the IMU. Like the accelerometer, it suffers from noise, bias, misalignment errors and scale errors.

$$ {}^{B}\boldsymbol{\omega}_m = \mathbf{T}_g\, {}^{B}\boldsymbol{\omega} + \mathbf{n}_g + \mathbf{b}_g \tag{4.4} $$

The noise $\mathbf{n}_g$ is Gaussian white noise such that $\mathbf{n}_g \sim \mathcal{N}(0, \mathbf{N}_g)$. The bias, $\mathbf{b}_g$, can be modeled as a random walk process driven by the noise vector $\mathbf{n}_{wg} \sim \mathcal{N}(0, \mathbf{N}_{wg})$.

Gyroscopes are also often influenced by acceleration in what is called g-sensitivity. The magnitude of this influence is considered negligible if it is within the range of the additive white noise, but in some hardware, specifically low cost MEMS gyroscopes, it is more

significant and can be modeled.

$$ {}^{B}\boldsymbol{\omega}_m = \mathbf{T}_g \, {}^{B}\boldsymbol{\omega} + \mathbf{T}_s \, {}^{B}\mathbf{a} + \mathbf{n}_g + \mathbf{b}_g \tag{4.5} $$

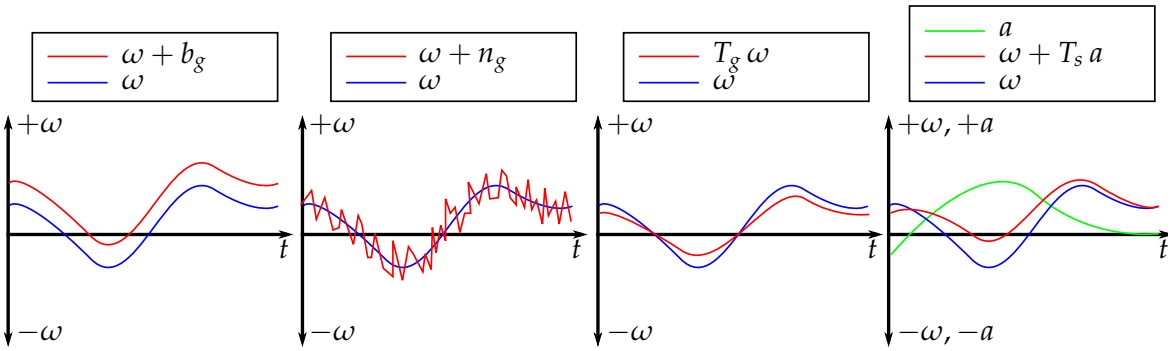


Figure 4.3. Illustrative examples of some different types of noise exhibited on an MEMS gyroscope. From left to right: bias, white noise, scale factor, and g-sensitivity. Note the units for ω are $\frac{\text{rad}}{\text{s}}$, while the units for a are $\frac{\text{m}}{\text{s}^2}$.

4.3. Noise and Bias Characteristics

Before we can use the values from the accelerometer and gyroscope, we must calculate the noise variances ($\mathbf{N}_a$, $\mathbf{N}_g$) and random walk variances ($\mathbf{N}_{wa}$, $\mathbf{N}_{wg}$). The following focuses on a single axis of the accelerometer, but applies to all accelerometer axes and to the gyroscope.

The accelerometer noise variance can be determined by calculating the standard deviation a successive group of measurements while the sensor is at rest for a short time period, t. If the time interval is too long, the changing bias might currupt the measurements and artificially inflate the result. The measured standard deviation is *discrete*, σ_{a_d}, and is not the value we need for the EKF prediction step (see equation 3.38). To obtain the continuous values, we follow [39] and multiply by the sample noise.

$$ \sigma_{a_c} = \sigma_{a_d} \sqrt{\Delta t} \tag{4.6} $$

Similarly the random walk bias standard deviation must also be converted to a continuous value.

$$ \sigma_{wa_c} = \frac{\sigma_{wa_d}}{\sqrt{\Delta t}} \tag{4.7} $$

These continuous values can now be used in the noise matrix for the EKF prediction step.

5. Computer Vision Basics

Computer vision is a rapidly expanding field on the cutting edge of computer science and robotics. As cameras improve in quality, they also decrease in size and price, which, along with the availability of a number of open source tools, has made the field of computer vision accessible and feasable for a wider array of applications. This is why using a camera as a sensor for odometry is such a widely researched topic when compared to more accurate and expensive sensors. In this chapter, we present our choice of camera model and how we use the camera to give us reliable structural information about the surrounding scene.

5.1. Pinhole Camera Model

The pinhole camera model describes the camera as a single point in space. Light travels from the scene through this hole onto an image plane where an image is formed and captured. This point is called the **focal point**. Figure 5.1 shows that the captured image is an inverted projection of the 3D surroundings. A more intuitive model is to consider the uninverted virtual image plane which rests between the focal point and the real scene.

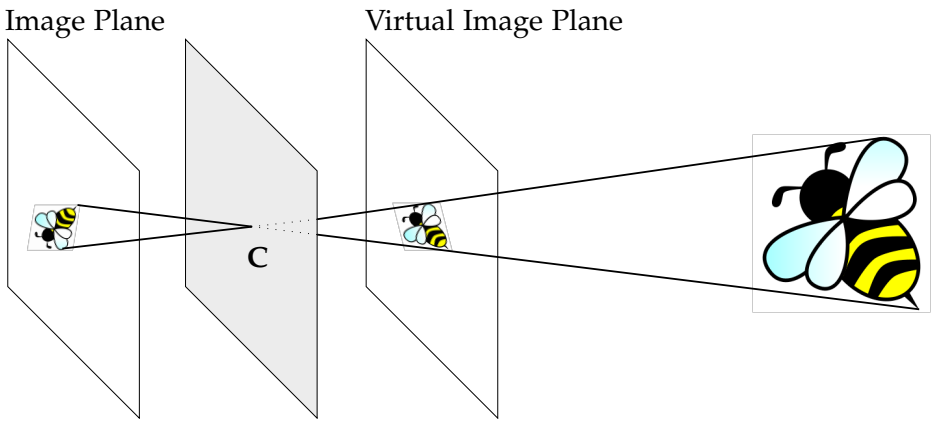


Figure 5.1. The pinhole camera projects an inverted image onto the image plane. It is useful to consider the uninverted virtual image plane instead.

While the pinhole camera model is useful, it is not entirely realistic. In reality there are usually one or more lenses between the camera and the scene, and the aperture, the size of the pinhole, can vary in size in order to let in more or less light. These properties, along with other minor imperfections, can cause blur and distortion in the image which can be accounted for with proper camera calibration.

5.1.1. Camera Projection

The goal in this section is to find the function that projects a known 3D point into the image plane at a certain pixel location. This function is simple in the case of the pinhole camera model and becomes slightly more complex when taking into account lense distortion. The shortest distance from the focal point to the image plane is known as the focal length f. To find the location of an object on the image plane, we use the geometry of similar triangles (see figure 5.2). An object appears on the image plane of a camera with focal length in meters f at image coordinates $\mathbf{x}_{img} = (x, y)^T$ while its position in space is ${}^C\mathbf{X} = (X, Y, Z)^T$.

$$x = f\frac{X}{Z} \qquad\qquad y = f\frac{Y}{Z} \tag{5.1}$$

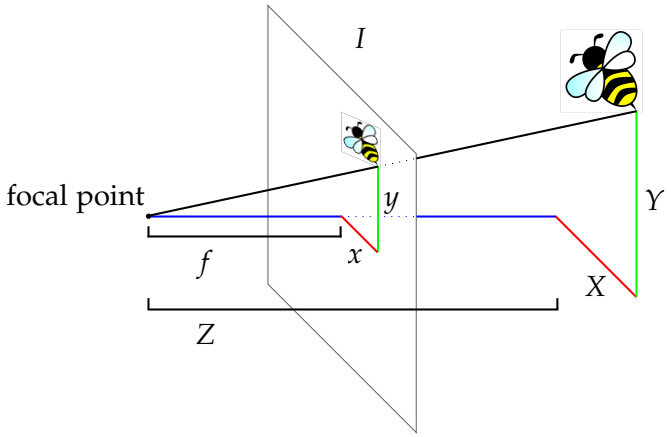


Figure 5.2. 3D to 2D projection

Although $\mathbf{x}_{img}$ is the location in the image plane, it has the wrong units (e.g. meters as opposed to pixels). To convert the units to pixels we need the size of a pixel in meters. Additionally, the top left corner of an image has pixel coordinates $(0, 0)$. To convert from image coordinates to pixel coordinates, we divide by d, the size of a pixel in meters, and add the pixel coordinate of the center point, o. Note that the camera sensor is often not square, meaning d_x and d_y will be different values.

$$x = f_x\frac{X}{Z} + o_x \qquad\qquad y = f_y\frac{Y}{Z} + o_y \tag{5.2}$$

For simplicity, we define $f_x = \frac{f}{d_x}$ and $f_y = \frac{f}{d_y}$. The units of o, f_x, and f_y are all pixels. We now have the basic pinhole model camera projection function.

$$\mathbf{h}\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{bmatrix} o_x \\ o_y \end{bmatrix} + \begin{bmatrix} f_x & 0 \\ 0 & f_y \end{bmatrix} \begin{bmatrix} \frac{X}{Z} \\ \frac{Y}{Z} \end{bmatrix} \tag{5.3}$$

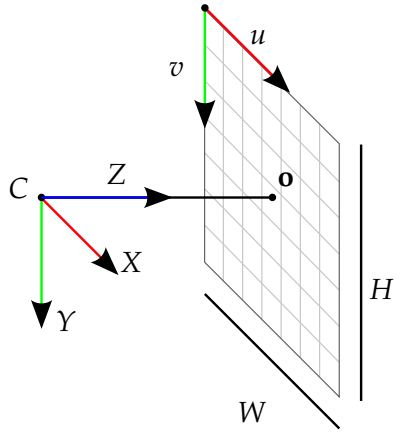


Figure 5.3. The camera and pixel coordinate frames.

5.1.2. Image Distortion

In many cameras with limited distortion, such as the camera on a smart phone or tablet, a simple polynomial approximation is adequate for modeling the radial and tangential distortions. The radial distortion d_r can be written as an infinite series mapping the undistorted coordinates to the distorted ones, while the tangential distortion d_t is additive. The resulting projection function requires five distortion parameters: $(k_1, k_2, k_3, t_1, t_2)$.

$$\mathbf{h}\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{bmatrix} o_x \\ o_y \end{bmatrix} + \begin{bmatrix} f_x & 0 \\ 0 & f_y \end{bmatrix} \left(d_r \begin{bmatrix} u \\ v \end{bmatrix} + d_t \right) \tag{5.4}$$

$$d_r = (1 + k_1 r + k_2 r^2 + k_3 r^3) \tag{5.5}$$

$$d_t = \begin{bmatrix} 2uvt_1 + (r + 2u^2)t_2 \\ 2uvt_2 + (r + 2v^2)t_1 \end{bmatrix} \tag{5.6}$$

$$\text{with } u = \frac{X}{Z},\ v = \frac{Y}{Z},\ r = u^2 + v^2 \tag{5.7}$$

5.1.3. Frame Transformation

If a point is expressed in the global frame instead of the camera frame, it must be transformed before applying the above equations. A 3D point, $\mathbf{x}$, in the global frame, $\{G\}$, can be transformed to the camera frame, $\{C\}$, in the following manner.

$${}^{C}\mathbf{x} = {}^{C}_{G}\mathbf{R}\left({}^{G}\mathbf{x} - {}^{G}\mathbf{p}_{C}\right) \tag{5.8}$$

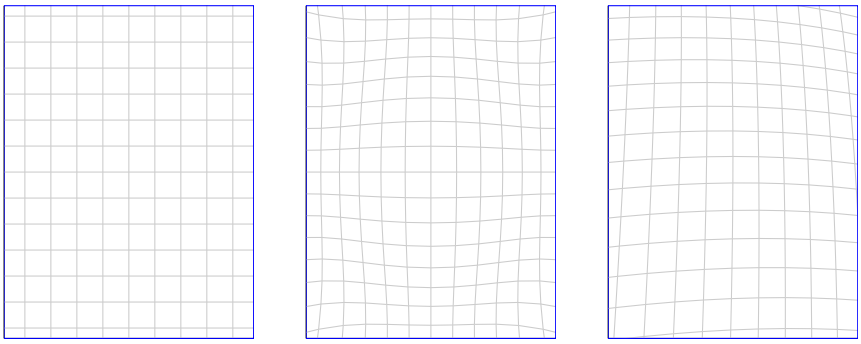


Figure 5.4. Examples of distortion. Left: no distortion, middle: radial distortion, right: tangential distortion.

Here ${}^{G}\mathbf{p}_{C}$ is the position of the camera in the global frame. The final equation for transforming a 3D point in the *global frame*, ${}^{G}\mathbf{x}$, to pixel coordinates is as follows.

$$\begin{bmatrix} u \\ v \end{bmatrix} = \mathbf{h}\left({}^{C}_{G}\mathbf{R}\left({}^{G}\mathbf{x} - {}^{G}\mathbf{p}_{C}\right)\right) \tag{5.9}$$

5.2. Triangulation

The previous section discusses the projection from a 3D point to the image plane. In this section, we use 2D pixel measurements to find 3D positions. Consider two different cameras at known locations that capture images of the same 3D point. In an ideal world, finding the 3D position of the point is a simple problem of solving a system of equations. In reality, however, error is introduced in a number of ways causing the system to have no solution.

Our goal is to find the best guess for the location of the point. We wish to formulate the problem as a minimization problem that can be solved with the Gauss-Newton algorithm, a method used to solve non-linear lease square problems [42]. In order to accomplish this, we begin by formulating the error function $\mathbf{f}$.

$$\mathbf{f}_i(\boldsymbol{\theta}) = \mathbf{z}_i - \mathbf{h}(\boldsymbol{\theta}) \tag{5.10}$$

In this general form, $\boldsymbol{\theta}$ is the *parameter*, $\mathbf{z}_i$ is the *measurement*, and the function $\mathbf{h}$ projects the parameter to the measurement space.

A feature point, ${}^{G}\mathbf{p}_{f}$, is tracked by a camera from n locations (Figure 5.6). For a given camera frame, $\{C_i\}$, the feature position in that frame is ${}^{C_i}\mathbf{p}_{f} = ({}^{C_i}X, {}^{C_i}Y, {}^{C_i}Z)^T$, but the camera only measures the pixel values $(u_i, v_i)^T$. We define our measurement as $\mathbf{z}_i = \mathbf{h}({}^{C_i}X, {}^{C_i}Y, {}^{C_i}Z)$, where $\mathbf{h}$ is the projection function from the previous section and $\mathbf{z}_i$ is in pixel coordinates. The frame $\{C_0\}$ is the camera frame from which the point was first

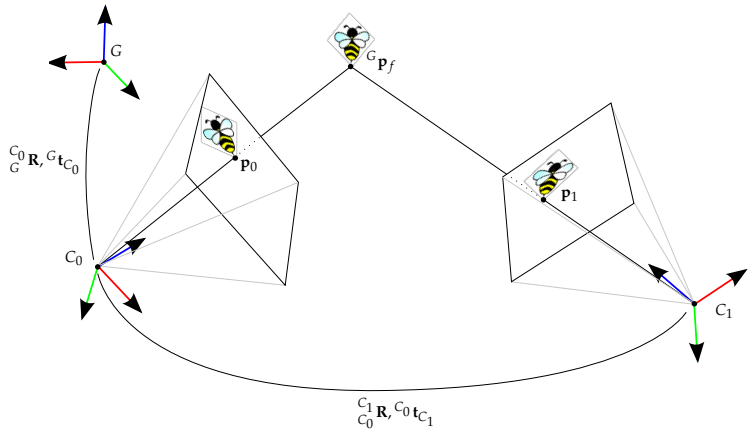


Figure 5.5. Ideal triangulation with two points in two images. In reality, it is extremely unlikely that the rays will intersect.

observed, and the position of the point in the ith camera frame is as follows.

$$ {}^{C_i}\mathbf{p}_f = {}^{C_i}_{C_0}\mathbf{R}({}^{C_0}\mathbf{p}_f - {}^{C_0}\mathbf{p}_{C_i}) \tag{5.11} $$

$$ {}^{C_i}\mathbf{p}_f = {}^{C_i}_{C_0}\mathbf{R}\,{}^{C_0}\mathbf{p}_f + {}^{C_i}\mathbf{p}_{C_0} \tag{5.12} $$

This can be rewritten with the *inverse depth parameterization* [7] in order to improve numerical stability and help to avoid local minima.

$$ {}^{C_i}\mathbf{p}_f = {}^{C_i}_{C_0}\mathbf{R} \begin{bmatrix} {}^{C_n}X \\ {}^{C_n}Y \\ {}^{C_n}Z \end{bmatrix} + {}^{C_i}\mathbf{p}_{C_0} \tag{5.13} $$

$$ = {}^{C_0}Z \left({}^{C_i}_{C_0}\mathbf{R} \begin{bmatrix} {}^{C_0}X/{}^{C_0}Z \\ {}^{C_0}Y/{}^{C_0}Z \\ 1 \end{bmatrix} + \frac{1}{{}^{C_0}Z} {}^{C_i}\mathbf{p}_{C_0} \right) \tag{5.14} $$

$$ = {}^{C_0}Z \left({}^{C_i}_{C_0}\mathbf{R} \begin{bmatrix} \alpha \\ \beta \\ 1 \end{bmatrix} + \rho\, {}^{C_i}\mathbf{p}_{C_0} \right) \tag{5.15} $$

$$ = {}^{C_0}Z\, \mathbf{g}_i \begin{pmatrix} \alpha \\ \beta \\ \rho \end{pmatrix} \tag{5.16} $$

The function $\mathbf{g}_i$ is a three dimensional function of α, β, and ρ.

$$ \alpha = \frac{{}^{C_0}X}{{}^{C_0}Z} \qquad \beta = \frac{{}^{C_0}Y}{{}^{C_0}Z} \qquad \rho = \frac{1}{{}^{C_0}Z} \tag{5.17} $$

Now we can rewrite the general error function (equation 5.10) for our specific case. The

meaning of this new equation is simple: the 3D point, parameterized by $\boldsymbol{\theta}$ in frame C_0, is transformed into frame C_i by $\mathbf{g}_i$ and projected into pixel coordinates by the function $\mathbf{h}$ where it is compared with the pixel measurement in that frame. The goal in the next section will be to minimize this error for all measurements by iteratively updating the parameters.

$$\mathbf{f}_i(\boldsymbol{\theta}) = \mathbf{z}_i - \mathbf{h}(\mathbf{g}_i(\boldsymbol{\theta})) \text{ with } \boldsymbol{\theta} = \begin{pmatrix} \alpha \\ \beta \\ \rho \end{pmatrix} \tag{5.18}$$

5.2.1. Gauss-Newton Minimization

The Gauss-Newton method is an efficient iterative method for finding the parameters that minimize the sum of squares of a function. This function is often (as in our case) a measure of the error between observations and a model.

$$S(\boldsymbol{\theta}) = \sum_{i=1}^{n} \mathbf{f}_i(\boldsymbol{\theta})^2 \tag{5.19}$$

To find the parameters that minimize S, we begin with an initial guess, $\boldsymbol{\theta}^{(0)}$. Each iteration of the Gauss-Newton method then proceeds by improving the current best guess by calculating the **Jacobian** (the first-order partial derivatives of $\mathbf{f}$ with respect to the parameters) and approximating the function quadratically, which makes it easy to minimize. The parameters are moved to this minimum and the process repeats. To calculate the Jacobian of $\mathbf{f}$, $\mathbf{J}_\mathbf{f}$, we first use chain rule and differentiate the projection function.

$$\mathbf{J}_\mathbf{f} = \frac{\partial \mathbf{f}}{\partial \boldsymbol{\theta}} = \frac{\partial \mathbf{h}}{\partial \mathbf{g}} \frac{\partial \mathbf{g}}{\partial \boldsymbol{\theta}}$$

$$\frac{\partial \mathbf{h}}{\partial \mathbf{g}} = \begin{bmatrix} f_x & 0 \\ 0 & f_y \end{bmatrix} \begin{bmatrix} \frac{d_r}{z} + \frac{\partial d_r}{\partial x}u + \frac{\partial d_t}{\partial x}, & \frac{\partial d_r}{\partial y}u + \frac{\partial d_t}{\partial y}, & -\frac{d_r}{z}u + \frac{\partial d_r}{\partial z}u + \frac{\partial d_t}{\partial z} \\ \frac{\partial d_r}{\partial x}v + \frac{\partial d_t}{\partial x}, & \frac{d_r}{z} + \frac{\partial d_r}{\partial y}v + \frac{\partial d_t}{\partial y}, & -\frac{d_r}{z}v + \frac{\partial d_r}{\partial z}v + \frac{\partial d_t}{\partial z} \end{bmatrix} \tag{5.20}$$

We use the shorthand $u = x/z$ and $v = y/z$ along with $(x, y, z)^T$ as the output of the function $\mathbf{g}$. The inner partial derivatives of d_r and d_t are straight forward to calculate and ommitted for brevity. We now rewrite the definition of $\mathbf{g}$ before calculating its Jacobian.

$$\begin{aligned} \mathbf{g}_i(\boldsymbol{\theta}) &= {}^{C_i}_{C_0}\mathbf{R} \begin{bmatrix} \alpha \\ \beta \\ 1 \end{bmatrix} + \rho \, {}^{C_i}\mathbf{p}_{C_0} \\ \frac{\partial \mathbf{g}_i}{\partial \boldsymbol{\theta}} &= \begin{bmatrix} \frac{\partial \mathbf{g}_i}{\partial \alpha}, & \frac{\partial \mathbf{g}_i}{\partial \beta}, & \frac{\partial \mathbf{g}_i}{\partial \rho} \end{bmatrix} \\ &= \begin{bmatrix} {}^{C_i}_{C_0}\mathbf{R} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, & {}^{C_i}_{C_0}\mathbf{R} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, & {}^{C_i}\mathbf{p}_{C_0} \end{bmatrix} \end{aligned}$$

The function $\mathbf{f}_i$ and the corresponding Jacobian $\mathbf{J}_{\mathbf{f}i}$ are calculated for all n camera poses and stacked to create the $2n \times 1$ vector $\mathbf{f}$ and the $2n \times 3$ matrix $\mathbf{J}_\mathbf{f}$. These are then used to update the parameters.

$$\boldsymbol{\theta}^{(s+1)} = \boldsymbol{\theta}^{(s)} - \left({\mathbf{J}_\mathbf{f}}^T \mathbf{J}_\mathbf{f}\right)^{-1} {\mathbf{J}_\mathbf{f}}^T \mathbf{f}(\boldsymbol{\theta}^{(s)}) \tag{5.21}$$

This process continues until either a maximum number of iterations is reached or the error function f descreases below a small threshold. Finally, the global point position is calculated from the estimated parameters as follows.

$${}^G\hat{\mathbf{p}}_f = \frac{1}{\hat{\rho}} {}^G_{C_0}\mathbf{R} \begin{bmatrix} \hat{\alpha} \\ \hat{\beta} \\ 1 \end{bmatrix} + {}^G\mathbf{p}_{C_0} \tag{5.22}$$

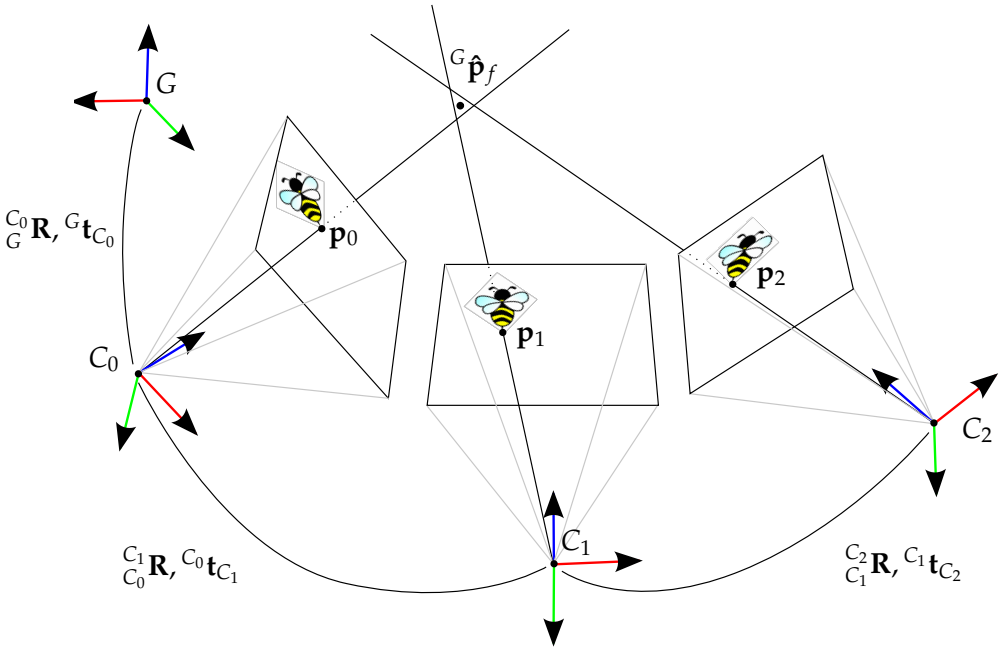


Figure 5.6. Real world triangulation has non-intersecting lines. Finding the best estimate becomes a minimization problem.

5.3. Rolling Shutter

Most small, low cost cameras, especially those in mobile phones, have rolling shutters. Instead of capturing the entire image simultaneously, each column (or row, depending on the sensor) is captured sequentially over a certain time period called the *read time*. Long read times can cause significant image distortion and can lead to problems with triangulation. Many odometry methods operate under the assumption of a global shutter [18] because they are often developed in labs where such cameras are present.

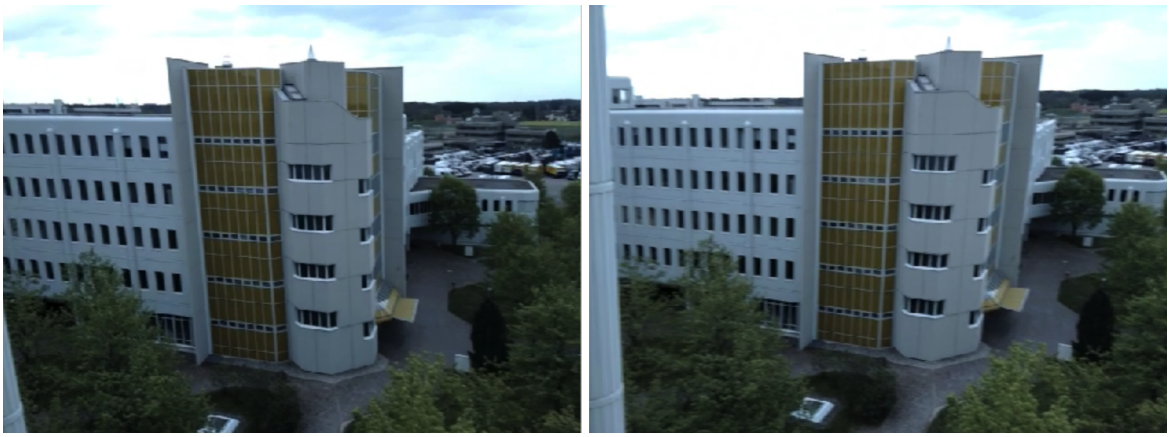


Figure 5.7. The rolling shutter can cause visible distortion when the camera rotates quickly. These examples were taken by the iPhone 4S camera while moving in opposite directions.

When an image is captured by a rolling shutter camera, there are two temporal values that need to be known to accurately use the image. The first is the offset t_d between the image's timestamp and the time that the center column was captured. This value is usually negative, meaning that the timestamp is late, but in cases where video data and IMU data is completely unsynchronized and only the framerate is known, t_d could in theory be a positive value.

The second important calibration constant is the camera's read time, t_r. If this value is negative, it implies the rolling shutter moves from right to left, which is the case for the iPhone 4S. Our experiments gave values of around -20ms for t_r. The method followed in this thesis finds both of these temporal calibration factors by estimating them in the EKF, as explained in chapter 6.

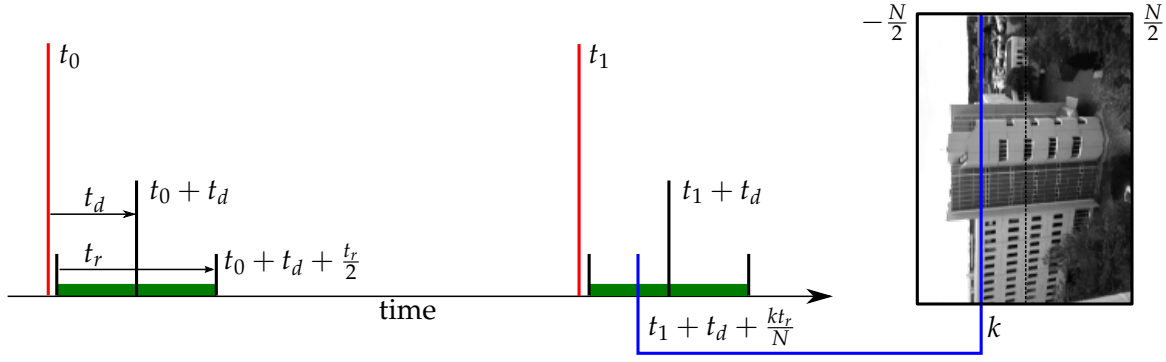


Figure 5.8. When an image is captured, a timestamp is provided ($t_0, t_1, \ldots$). The middle column of the image was captured at some time $t + t_d$. The entire image of N columns has a read time of t_r, meaning that column k was captured at $t + t_d + \frac{kt_r}{N}$ where $k \in [-\frac{N}{2}, \frac{N}{2}]$.

5.4. Feature Points

In the previous section we discussed how to triangulate a point captured in images at different known camera positions. Here we explain how these points are found, what they represent, and how they are matched between images. We are interested in finding areas of interest in an image that are associated with repeatably identifiable 3D entities. There has been a huge amount of research in the area of feature detection, mostly focusing on "corners" or lines. Algorithms that find repeatable features in an image are known as *feature detectors* and they are quite numerous, each having its own pros and cons. A full explanation of how each feature detector works is beyond the scope of this thesis. Instead, we describe and justify only the chosen algorithm. For a survey of different feature detectors, we refer the reader to [40].

5.4.1. The FAST Feature Detector

The FAST feature detector [33] is widely used in odometry because of its speed and high feature output. Each gray scale pixel value is compared to neighbors in a circle (Figure 5.9). If a certain combination of pixels is higher and lower in value to the center, the central pixel is determined to be a corner point. The exact combinations that lead to corner points were found through an offline machine learning using a large dataset with ground truth data. The result is a number of open source implementations that run on many devices. Finding all the FAST features in a camera image takes on the order of 5ms on a single core processor of a laptop computer.

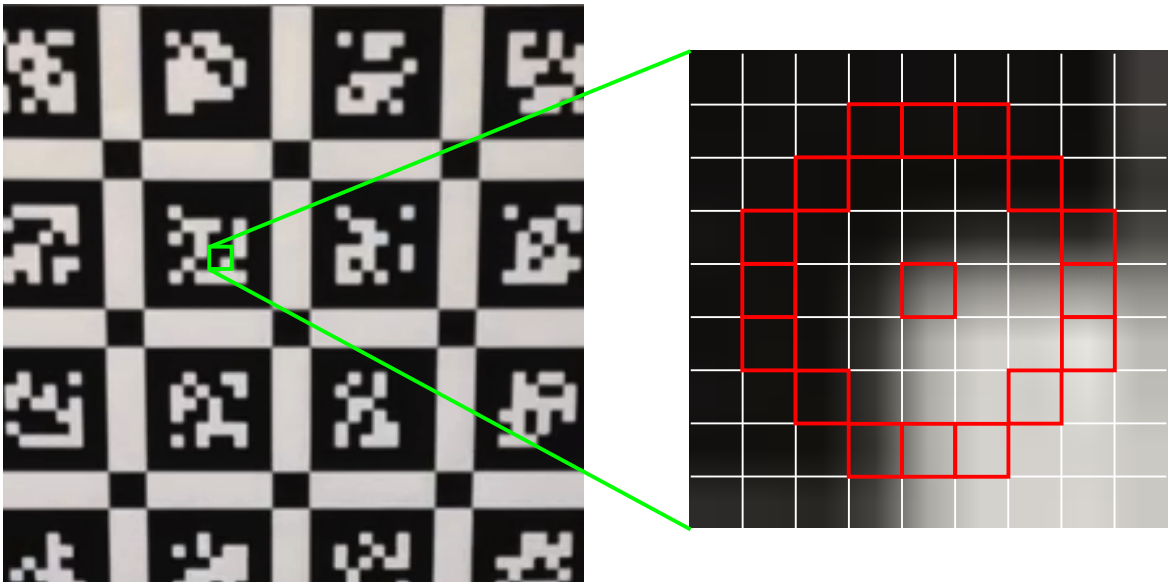


Figure 5.9. The FAST corner detector compares image intensities in a circle around a central pixel to determine if it is a corner.

The number of features returned depends on a threshold value, the difference in intensity between the central and outer pixels required to trigger the detector. Using a static

value for this threshold can lead to too many or too few features in an image, depending largly on motion blur and the type of scene. Dynamically varying the threshold gives better results, forcing the output to give a certain number of features. In order to ensure an even distribution of features throughout the image, we used Grid-FAST from the OpenCV library [3]. This algorithm splits the image into a grid, each with a unique dynamic threshold to guarantee that the total number of features is within a certain range.

The FAST feature detector has its share of disadvantages. Like any feature detector that finds corner points, it relies on textured surfaces or cluttered scenes. Blank walls and floors often have no visible features, making visual odometry difficult. Motion blur, especially on the iPhone, causes the FAST detector to find very few features. This is in contrast to more computationally expensive feature detectors which compute image gradients at multiple scales. Nevertheless, the improved speed from the FAST detector makes up for this shortcoming.

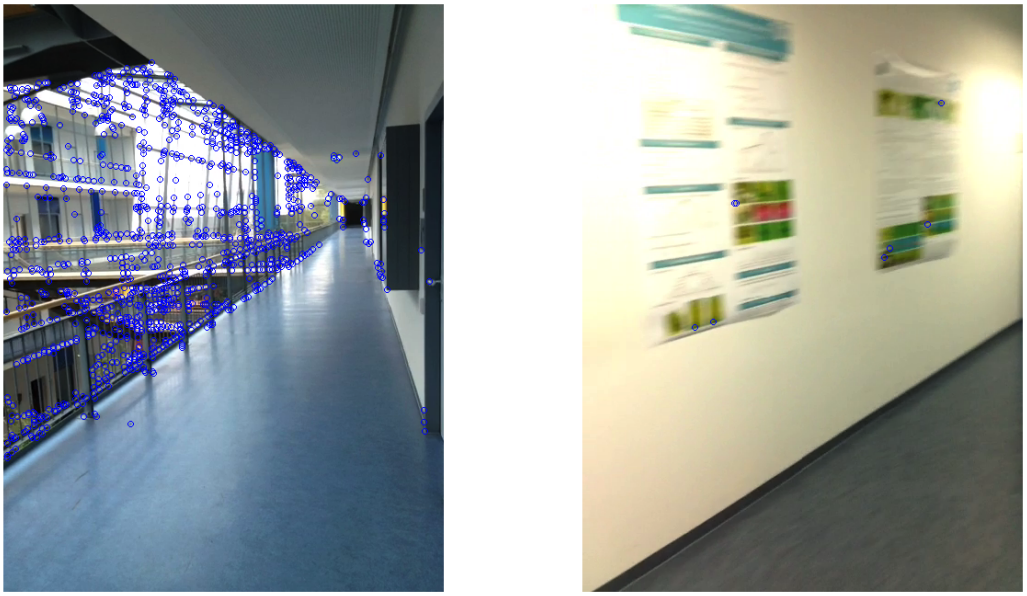


Figure 5.10. The FAST feature detector is, of course, fast, but it is not without disadvantages. Featureless areas like the floor and ceiling in the left image, and motion blur like in the right image can cause significant problems with odometry.

5.4.2. Feature Matching

Once features are detected in sequential images, they must be matched with each other quickly and accurately. One strategy for figuring out if two image regions in two different images describe the same physical location, is the compare abstract descriptions of those regions. There are many algorithms for extracting detail rich descriptions of image patches in order to create robust and repeatable matches. SIFT [22] and SURF [2] are two popular examples, with invariance to minor rotations, lighting changes and so on, but they are

both far too computationally expensive for our purposes. They require calculation of the image gradient, which is a highly parallelizable process, but is nevertheless slow on mobile devices. The large dimension of the descriptors (64 floating point values) also makes matching descriptors slow.

More recently, binary descriptors such as BRIEF [5] have been developed. These descriptors work in a similar way to the FAST corner detector, by comparing image intensity values around the central feature point. The result of the comparison is a single binary digit indicating a higher or lower intensity value. All the comparisons are put together into a vector. The resulting binary vector is relatively small and can be quickly compared with the Hamming distance[1]. The ORB [34] descriptor takes BRIEF to the next level by adding orientation invariance, which BRIEF lacks by design. In our experiments, ORB works well for real time purposes on a laptop's single core CPU, but an even faster solution exists, making direct use of the provided odometry.

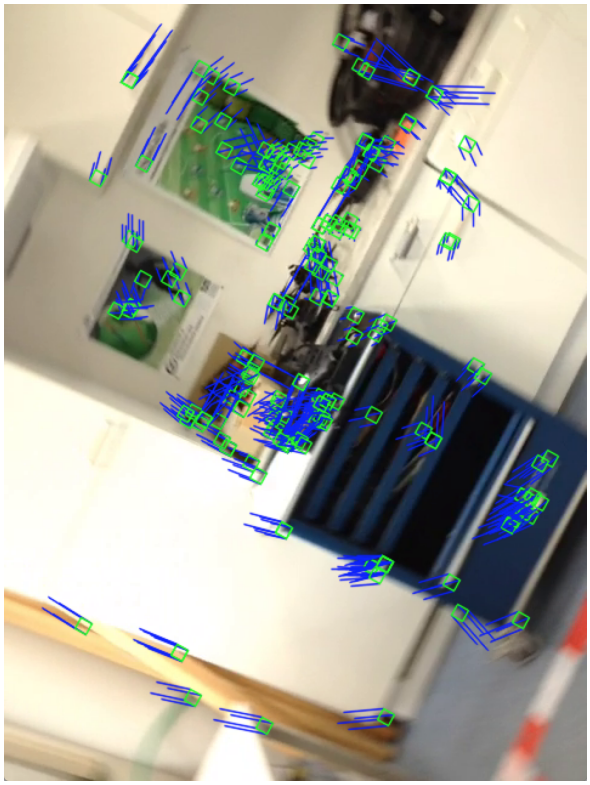


Figure 5.11. An example of rapid rotation while tracking patches. The green boxes represent the patches which are rotated to a canonical orientation (upright according to gravity). The blue lines show how the patches have moved since the previous image. While some outliers are present, the majority of tracked patches correctly convey the type of rotation in progress.

Most feature descriptors are designed for general matching without prior knowledge of the way in which the images were captured. While performing odometry, we know

[1]The Hamming distance of two binary numbers is equivalent to the XOR operator followed by adding up the number of 1's (the Hamming weight) in the result.

roughly the orientation from which the image was captured. We extract a square image patch around each feature point, but before extraction, we rotate the square region to a canonical orientation. When comparing with future image patches, they will also be rotated to the same global orientation before comparison, improving match quality and making the system invariant to rotation around the camera's principal axis.

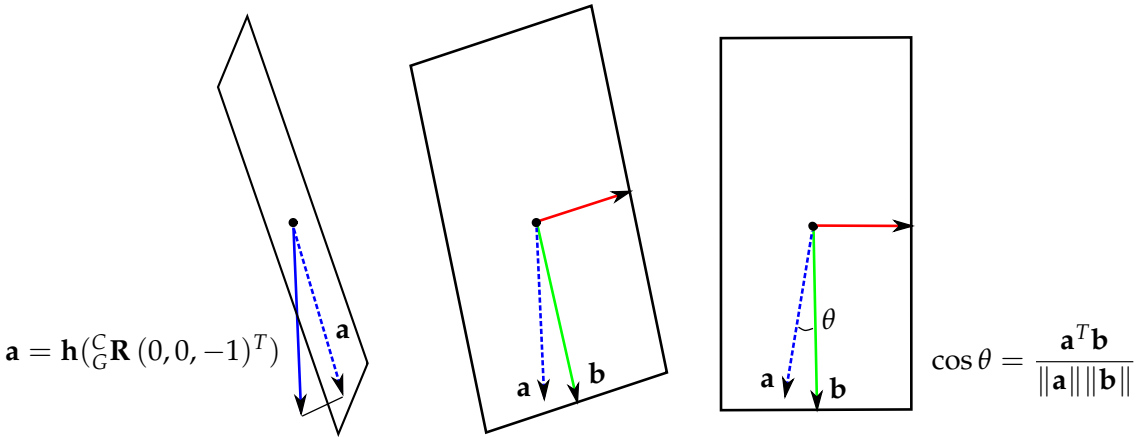


Figure 5.12. The angle θ needed to rotate patches can be found with the estimated camera rotation ${}^{C}_{G}\mathbf{R} = {}^{C}_{B}\mathbf{R}\,{}^{B}_{G}\mathbf{R}$. The global vector $(0,0,-1)^T$ is rotated into the camera's frame and projected into the image plane. There it is normalized and directly compared with the vertical y-axis of the image with a dot product.

Patches are extracted using bilinear interpolation. After extraction, we subtract the mean value of the patch from all the values and normalize the entire patch for a total value of 1. This normalization process greatly improves the performance when matching. In order to compare patches, we simply take the sum of squared differences (SSD) of the patch values and normalize this by result with the size of the patch. This process is also often referred to as normalized cross correlation, or NCC.

When determining if two patches match, we compare both the NCC result and the pixel distance to some thresholds. If 3D information was available, we would be able to estimate a possible location for each correspondence by back projecting the point into the image and searching in that nearby area. In our case, because we do not estimate the depth of features, this is not possible. We do, however, have access to the dynamics of the system, specifically the rotational and translational velocities. These could in theory be used to help narrow the search for point correspondences to reduce potentially wrong matches. We consider this as future work.

6. Multi-State Constraint Kalman Filter

6.1. Overview

The MSCKF is, like inertial aided EKF-SLAM, an Extended Kalman Filter based estimator that tracks the pose of the body $\{B\}$ over time using a camera and an IMU. Both algorithms use the IMU to propagate the state and covariance matrix in the EKF prediction step, and both algorithms track feature points between images. In EKF-SLAM, the feature point is added to the state vector and the covariance matrix. Using the inverse depth representation, the feature can be added after being detected in just one camera frame. In subsequent frames in which it is tracked, it is used to constrain the camera pose while also updating its own inverse depth estimate. A significant disadvantage of this approach is that the inverse depth be given an initial estimate and variance for the inverse depth, which is unknowable. As a result, if features do not match the chosen prior distribution, the filter will become inconsistent. Even worse, outliers which are detected have already had irreversible effects on the filter, because they are used in updates at every stage.

The MSCKF does not add feature points to the state vector at all. Instead, after each camera image, the current pose of the body is appended to the state vector. After m poses have been appended, the oldest pose is dropped from the state vector in a First In First Out (FIFO) manner. In other words, the state vector maintains a sliding window of poses. Using this sliding window, a feature point can be triangulated after it falls out of view or it appears m times, yielding a very accurate estimate of the 3D feature position. This and other 3D features are then used to impose constraints on the sliding window of poses in the EKF update state.

As discussed in previous chapters, a number of parameters need to be known in order to accurately use the IMU and camera measurements.

- The camera calibration and distortion parameters.
- The IMU shape matrices $\mathbf{T}_g$ and $\mathbf{T}_a$ as well as the g-sensitivity matrix $\mathbf{T}_s$.
- The IMU-camera position offset, ${}^{C}\mathbf{p}_{B}$
- The IMU random walk bias values, $\mathbf{b}_a$ and $\mathbf{b}_g$.
- The Camera to IMU time delay, t_d.
- The Camera rolling shutter read time, t_r.

Following [24], which was published during the final months of this thesis, we show that calibration for *all* of these parameters can be done online with the filter itself, given sufficiently exciting and trackable motion. It is important to note that we do not estimate the rotation between the camera and IMU, but instead use a best guess ${}^{C}_{B}\mathbf{R}$, which is

treated as a constant. Minor errors in the rotation will be absorbed into the accelerometer and gyroscope shape matrices.

6.2. State Representation

6.2.1. Rotation Representation

To minimally describe a rotation, we only need three degrees of freedom. Euler angles, often called *roll*, *pitch*, and *yaw*, are a common way to express orientation, but they suffer from singularities. These occur when the angles are not uniquely determined, resulting in what is called gimbal lock. To prevent these kinds of ambiguities, the rotations of the body state and the body-camera rotation are represented by quaternions.

A quaternion is a four dimensional orientation representation that does not suffer from singularities. One common way to think of a quaternion is the axis-angle representation, in which three of the values represent an axis in 3D while the fourth value is a rotation about that axis, as in equation (6.1).

$$\bar{\mathbf{q}} = \begin{bmatrix} \vec{\mathbf{q}} \\ q_4 \end{bmatrix} = \begin{bmatrix} k_x \sin(\theta/2) \\ k_y \sin(\theta/2) \\ k_z \sin(\theta/2) \\ \cos(\theta/2) \end{bmatrix} \tag{6.1}$$

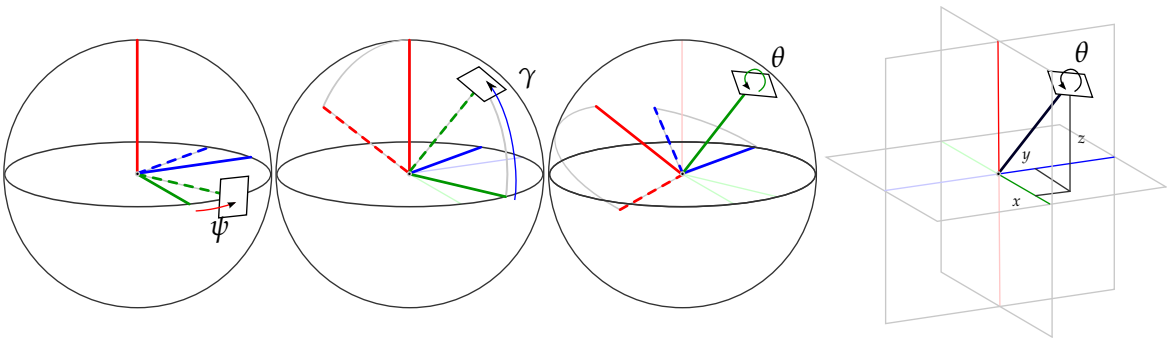


Figure 6.1. This shows how Euler Angles and a quaternion can be used to represent a particular rotation. Euler angles can be applied in different orders to achieve the same result.

Throughout this chapter, quaternions are considered unit quaternions unless otherwise noted. Rotation matrices such as ${}^{C}_{B}\mathbf{R}$ are actually computed from corresponding quaternions, written as ${}^{C}_{B}\mathbf{q}$. For more details on quaternions we refer the reader to Appendix A.

6.2.2. Coordinate Frames

We define the coordinate frames by starting with the camera frame C. We assume this frame is exactly the location of the camera, both in terms of position and rotation. The

body frame B has the exact position of the accelerometer, with rotation[1] relative to the camera of ${}^{C}_{B}\mathbf{R}$, which is deduced by examining the sensor outputs and attempting a best guess estimate of the relative rotation. This rotation is assumed to be correct, and any small errors can be absorbed into the shape matrix $\mathbf{T}_a$, which is estimated online. The position of the body in the camera frame ${}^{C}\mathbf{p}_B$ is guessed as well and will be estimated online. Small errors in the relative rotation of the body frame and the gyroscope will be corrected by $\mathbf{T}_g$, and translational errors are not important, because the rotational velocity at any point of the rigid body is the same. Finally, the filter will estimate the rotation, position and velocity of the body in the global frame.

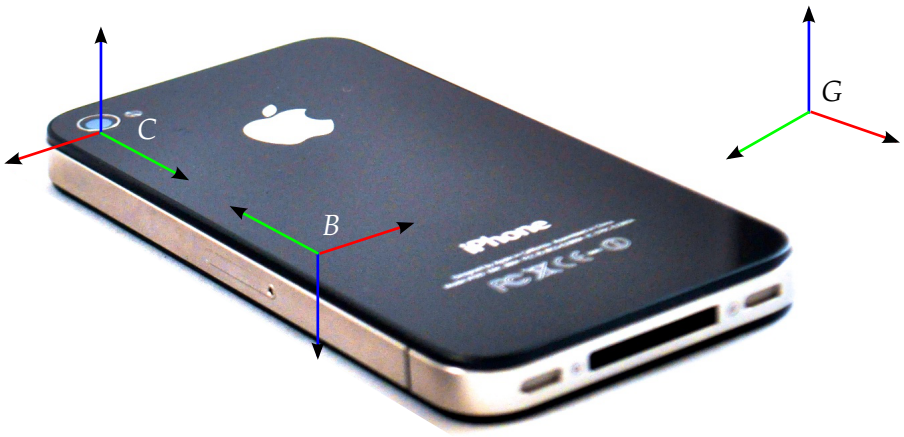


Figure 6.2. The iPhone4S coordinate frames. The camera frame is defined with the Z axis going outwards and the Y axis down. The body frame, B, is located at the accelerometer, while the global frame G is at an arbitrary static location.

6.2.3. Full State Representation

The full state representation can be partitioned into four parts. The first is the evolving body state, the second is the IMU calibration matrices, the third is the remaining calibration parameters, and the fourth is the sliding window of body poses.

$$\mathbf{x}_k = \left[\mathbf{x}_B^T \,|\, \mathbf{x}_{imu}^T \,|\, \mathbf{x}_c^T \,|\, \boldsymbol{\pi}_{B_{N-m}}^T \;\ldots\; \boldsymbol{\pi}_{B_{N-1}}^T\right]^T$$

The changing state estimate of the body frame is made up of the orientation (as a quaternion), position, velocity, and the IMU biases. These properties evolve over time and must be propagated during the EKF prediction step.

$$\mathbf{x}_B = \left[{}^{G}\mathbf{q}_B^T,\ {}^{G}\mathbf{p}_B^T,\ {}^{G}\mathbf{v}_B^T,\ \mathbf{b}_g^T,\ \mathbf{b}_a^T\right]^T$$

[1] Rotation matrices and quaternions are used interchangably in this chapter. For the method of conversion between them, see Appendix A.

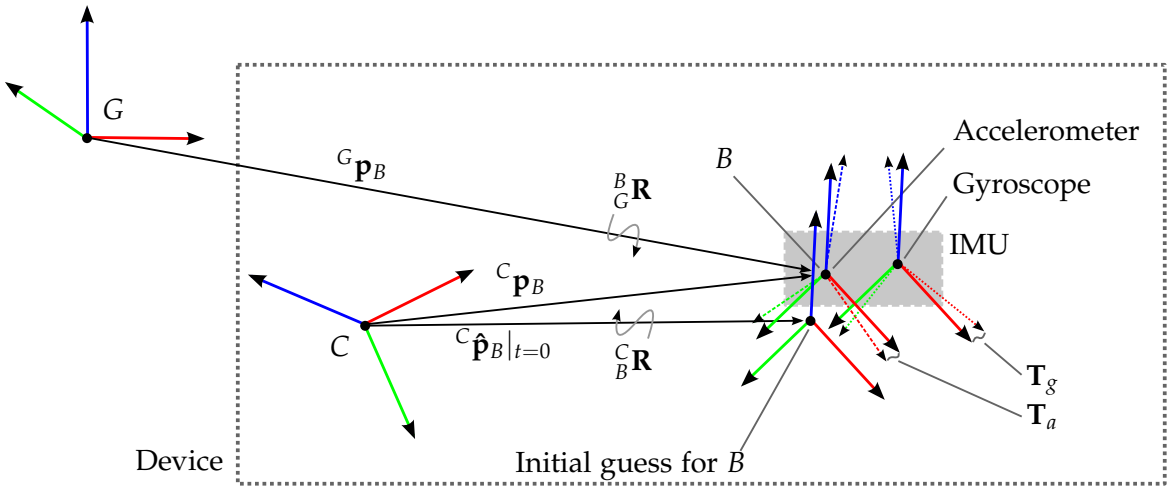


Figure 6.3. This figure shows the three important frames and their relationship to the three sensors. We define the body frame B as the location of the accelerometer with known rotation relative to the camera frame C of ${}^{C}_{B}\mathbf{R}$. Small errors in this rotation and scale will be compensated for by $\hat{\mathbf{T}}_a$, while other errors in the gyroscope rotation and scale will be compensated for by $\hat{\mathbf{T}}_g$. The translation from IMU to camera is estimated online starting with a close approximation, ${}^{C}\hat{\mathbf{p}}_{B}$.

The IMU calibration component of the state vector is 27×1, comprised of three vectorized matrices. As explained in chapter 4, the matrices $\mathbf{T}_g$ and $\mathbf{T}_a$ correct scale and misalignment of the gyroscope and accelerometer, respectively, and the matrix $\mathbf{T}_s$ corrects for g-sensitivity.

$$\mathbf{x}_{imu} = \left[\vec{\mathbf{T}}_g^T, \vec{\mathbf{T}}_s^T, \vec{\mathbf{T}}_a^T, \right]^T$$

The $\mathbf{T}$ matrices are vectorized as follows

$$\mathbf{T}_g = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \qquad \vec{\mathbf{T}}_g = [a, b, c, d, e, f, g, h, i]^T$$

The calibration parameters are constants which are estimated in the filter and have some small initial uncertainty.

$$\mathbf{x}_c = \left[{}^{C}\mathbf{p}_B^T, \mathbf{x}_{\text{cam}}^T, t_d, t_r\right]^T$$
$$\mathbf{x}_{\text{cam}} = \left[f_x, f_y, o_x, o_y, k_1, k_2, k_3, t_1, t_2\right]^T$$

Finally, each body pose in the sliding window contains the quaternion, position and velocity. The velocity will be used when estimating the temporal calibration parameters. Note that we do not need to add the angular velocity to the sliding window because we have access to the IMU readings directly.

$$\boldsymbol{\pi}_{B_i} = \left[{}^{G}\mathbf{q}_{B_i}^T, {}^{G}\mathbf{p}_{B_i}^T, {}^{G}\mathbf{v}_{B_i}^T\right]^T$$

The total size of the state vector is $(16 + 27 + 14 + 10m) \times 1 = (57 + 10m) \times 1$, where m is the number of poses in the sliding window.

6.2.4. Error Definition

With vectors such as the position and velocity, or scalars such as calibration parameters, we can model the error as simply the difference between the true state and the estimate, e.g. $\tilde{\mathbf{p}} = \mathbf{p} - \hat{\mathbf{p}}$. This does not work with the quaternion, however, because of the constraint that it must maintain unit length, $||\mathbf{q}|| = 1$. Instead, the error is modeled as the small difference in angles in the three dimensions, $\delta\mathbf{q} = (\frac{1}{2}\delta\boldsymbol{\theta}^T, 1)^T$. This has the advantage of being both a minimal error representation (3 degrees of freedom) and avoiding singularities found in Euler angles, because the error angles are always small. Further explanation of the quaternion representation and the Euler equivalent can be found in Appendix A. The affected error state representation is defined as follows.

$$\tilde{\mathbf{x}}_{B_k} = \left[{}^G\delta\boldsymbol{\theta}_{B_k}^T,\ {}^G\tilde{\mathbf{p}}_{B_k}^T,\ {}^G\tilde{\mathbf{v}}_{B_k}^T,\ \tilde{\mathbf{b}}_g^T,\ \tilde{\mathbf{b}}_a^T \right]^T$$

$$\tilde{\boldsymbol{\pi}}_{B_i} = \left[{}^G\delta\boldsymbol{\theta}_{B_i}^T,\ {}^G\tilde{\mathbf{p}}_{B_i}^T,\ {}^G\tilde{\mathbf{v}}_{B_i}^T \right]^T$$

The covariance matrix $\boldsymbol{\Sigma}$ has size $(56 + 9m) \times (56 + 9m)$.

6.3. Propagation

State and covariance propagation occurs after new IMU data is available. The IMU provides the current rotational velocity and linear acceleration. In the following sections we break down how the motion is described with the IMU data both continuously and discretely, and how the discrete error is propagated. For the purposes of this chapter, we assume that both the gyroscope and accelerometer measurements are available at the same time at a fixed interval, Δt. In reality they might have different periods and interpolation will be required.

6.3.1. Continuous Dynamics

Only the values in the evolving body partition of the state change over time and must be modeled dynamically. The rest are kept constant during the EKF prediction step.

$$ {}^B_G\dot{\mathbf{q}}(t) = \frac{1}{2}\boldsymbol{\Omega}({}^B\boldsymbol{\omega}(t))\, {}^B_G\mathbf{q}(t) \tag{6.2}$$

$$\text{with } \boldsymbol{\Omega}(\boldsymbol{\omega}) = \begin{bmatrix} -\lfloor\boldsymbol{\omega}\times\rfloor & \boldsymbol{\omega} \\ -\boldsymbol{\omega}^T & 0 \end{bmatrix} \tag{6.3}$$

$$ {}^G\dot{\mathbf{p}}(t) = {}^G\mathbf{v}(t) \tag{6.4}$$

$$ {}^G\dot{\mathbf{v}}(t) = {}^G\mathbf{a}(t) \tag{6.5}$$

$$ \dot{\mathbf{b}}_g(t) = \mathbf{n}_{w_g}(t) \tag{6.6}$$

$$ \dot{\mathbf{b}}_a(t) = \mathbf{n}_{w_a}(t) \tag{6.7}$$

6.3.2. Gyroscope Measurements

As discussed in Chapter 4, the IMU measurements are currupted by a number of factors. During propagation, we use the current estimates of the calibration parameters to correct for these measurement errors. The gyroscope gives the rotational velocity in the body frame after correcting for misalignments, scale factors, g-sensitivity, and bias.

$$ {}^{B}\hat{\boldsymbol{\omega}}(t) = \hat{\mathbf{T}}_g^{-1}\left({}^{B}\boldsymbol{\omega}_m(t) - \hat{\mathbf{T}}_s\,{}^{B}\hat{\mathbf{a}}(t) - \hat{\mathbf{b}}_g(t)\right) \tag{6.8} $$

We denote the measured rotational velocity as ${}^{B}\boldsymbol{\omega}_m$, and the estimate as ${}^{B}\hat{\boldsymbol{\omega}}$. This estimated rotational velocity is used to propagate the current orientation estimate, but we cannot integrate it directly because the orientation is parameterized as a quaternion. In order to find the new orientation, we first require the small quaternion ${}^{B_{\ell+1}}_{B_\ell}\hat{\mathbf{q}}$ that represents the rotation from B_ℓ to $B_{\ell+1}$.

$$ {}^{B_{\ell+1}}_{G}\hat{\mathbf{q}} = {}^{B_{\ell+1}}_{B_\ell}\hat{\mathbf{q}} \otimes {}^{B_\ell}_{G}\hat{\mathbf{q}} \tag{6.9} $$

In order to find this small quaternion, we integrate the differential equation

$$ {}^{B_t}_{B_\ell}\dot{\hat{\mathbf{q}}} = \frac{1}{2}\boldsymbol{\Omega}({}^{B}\hat{\boldsymbol{\omega}}(t))\,{}^{B_t}_{B_\ell}\hat{\mathbf{q}} \qquad\qquad t \in [t_\ell, t_{\ell+1}] \tag{6.10} $$

With discrete measurements and time difference Δt, we integrate this using the fourth order Runge-Kutta method. We use the shorthand $\hat{\boldsymbol{\omega}}_\ell = {}^{B}\hat{\boldsymbol{\omega}}(t_\ell)$.

$$ {}^{B_{\ell+1}}_{B_\ell}\hat{\mathbf{q}} = \bar{\mathbf{q}}_0 + \frac{\Delta t}{6}\left(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4\right) \tag{6.11} $$

$$ \bar{\mathbf{q}}_0 = [0, 0, 0, 1]^T \tag{6.12} $$

The terms $\mathbf{k}_1$ through $\mathbf{k_4}$ are defined as follows, with $\boldsymbol{\Omega}(\hat{\boldsymbol{\omega}})$ from equation (6.3).

$$ \mathbf{k}_1 = \frac{1}{2}\boldsymbol{\Omega}(\hat{\boldsymbol{\omega}}_\ell)\bar{\mathbf{q}}_0 \tag{6.13} $$

$$ \mathbf{k}_2 = \frac{1}{2}\boldsymbol{\Omega}(\frac{\hat{\boldsymbol{\omega}}_\ell + \hat{\boldsymbol{\omega}}_{\ell+1}}{2})(\bar{\mathbf{q}}_0 + \frac{\Delta t}{2}\mathbf{k}_1) \tag{6.14} $$

$$ \mathbf{k}_3 = \frac{1}{2}\boldsymbol{\Omega}(\frac{\hat{\boldsymbol{\omega}}_\ell + \hat{\boldsymbol{\omega}}_{\ell+1}}{2})(\bar{\mathbf{q}}_0 + \frac{\Delta t}{2}\mathbf{k}_2) \tag{6.15} $$

$$ \mathbf{k}_4 = \frac{1}{2}\boldsymbol{\Omega}(\hat{\boldsymbol{\omega}}_{\ell+1})(\bar{\mathbf{q}}_0 + \Delta t\mathbf{k}_3) \tag{6.16} $$

The disadvantage to this numerical integration is that the resulting quaternion may not be a unit quaternion. To account for this, we must normalize it to the unit sphere.

$$ {}^{B_{\ell+1}}_{B_\ell}\hat{\bar{\mathbf{q}}} \leftarrow \frac{{}^{B_{\ell+1}}_{B_\ell}\hat{\mathbf{q}}}{||{}^{B_{\ell+1}}_{B_\ell}\hat{\mathbf{q}}||} \tag{6.17} $$

6.3.3. Accelerometer Measurements

Like the gyroscope, the accelerometer measurements must be corrected with the current IMU calibration parameter estimates in order to get the local proper acceleration. This can then be rotated to the global frame where gravity can be removed. Recall that gravity is $(0, 0, -1)^T$ and an accelerometer at rest reads an acceleration of 1 in the positive global z direction. This is why we must *add* gravity to remove it from the global acceleration.

$$
\begin{aligned}
{}^{B}\hat{\mathbf{a}}(t) &= \hat{\mathbf{T}}_a^{-1}(\mathbf{a}_m(t) - \hat{\mathbf{b}}_a) && (6.18)\\
{}^{G}\hat{\mathbf{a}}(t) &= {}^{G}_{B}\hat{\mathbf{R}}(t)\,{}^{B}\hat{\mathbf{a}}(t) + {}^{G}\mathbf{g} && (6.19)
\end{aligned}
$$

Integration of the acceleration gives the propagated velocity in the global frame.

$$
\begin{aligned}
{}^{G}\hat{\mathbf{v}}_{\ell+1} &= {}^{G}\hat{\mathbf{v}}_\ell + \int_{t_\ell}^{t_{\ell+1}} {}^{G}\hat{\mathbf{a}}(\tau)\, d\tau && (6.20)\\
&= {}^{G}\hat{\mathbf{v}}_\ell + \int_{t_\ell}^{t_{\ell+1}} \left({}^{G}_{B_\tau}\hat{\mathbf{R}}\,{}^{B}\hat{\mathbf{a}}(\tau) + {}^{G}\mathbf{g} \right) d\tau && (6.21)\\
&= {}^{G}\hat{\mathbf{v}}_\ell + \int_{t_\ell}^{t_{\ell+1}} {}^{G}_{B_\tau}\hat{\mathbf{R}}\,{}^{B}\hat{\mathbf{a}}(\tau)\, d\tau + {}^{G}\mathbf{g}\,\Delta t && (6.22)\\
&= {}^{G}\hat{\mathbf{v}}_\ell + {}^{G}_{B_\ell}\hat{\mathbf{R}} \int_{t_\ell}^{t_{\ell+1}} {}^{B_\ell}_{B_\tau}\hat{\mathbf{R}}\,{}^{B}\hat{\mathbf{a}}(\tau)\, d\tau + {}^{G}\mathbf{g}\,\Delta t && (6.23)\\
&= {}^{G}\hat{\mathbf{v}}_\ell + {}^{G}_{B_\ell}\hat{\mathbf{R}}\hat{\mathbf{s}}_\ell + {}^{G}\mathbf{g}\,\Delta t && (6.24)
\end{aligned}
$$

Integrating the velocity gives the propagated position.

$$
\begin{aligned}
{}^{G}\hat{\mathbf{p}}_{\ell+1} &= {}^{G}\hat{\mathbf{p}}_\ell + \int_{t_\ell}^{t_{\ell+1}} {}^{G}\hat{\mathbf{v}}(\tau)\, d\tau && (6.25)\\
&= {}^{G}\hat{\mathbf{p}}_\ell + {}^{G}\hat{\mathbf{v}}_\ell\,\Delta t + {}^{G}_{B_\ell}\hat{\mathbf{R}}\,\hat{\mathbf{y}}_\ell + \frac{1}{2}{}^{G}\mathbf{g}\,\Delta t^2 && (6.26)
\end{aligned}
$$

where $\Delta t = t_{\ell+1} - t_\ell$, ${}^{G}_{B_\ell}\hat{\mathbf{R}} = {}^{G}_{B}\hat{\mathbf{R}}(t_\ell)$, and the lever arm components are as follows

$$
\begin{aligned}
\hat{\mathbf{s}} &= \int_{t_\ell}^{t_{\ell+1}} {}^{B_\ell}_{B_\tau}\hat{\mathbf{R}}\,{}^{B}\hat{\mathbf{a}}(\tau)\, d\tau && (6.27)\\
\hat{\mathbf{y}} &= \int_{t_\ell}^{t_{\ell+1}} \int_{t_\ell}^{s} {}^{B_\ell}_{B_\tau}\hat{\mathbf{R}}\,{}^{B}\hat{\mathbf{a}}(\tau)\, d\tau\, ds && (6.28)
\end{aligned}
$$

In practice, the integrals are evaluated using trapezoidal integration[2].

$$
\begin{aligned}
\hat{\mathbf{s}} &\simeq \frac{\Delta t}{2}\left({}^{B_\ell}_{B_{\ell+1}}\hat{\mathbf{R}}\,\mathbf{T}_a^{-1}\left(\mathbf{a}_m(t_{\ell+1}) - \hat{\mathbf{b}}_a\right) + \mathbf{T}_a^{-1}\left(\mathbf{a}_m(t_\ell) - \hat{\mathbf{b}}_a\right)\right) && (6.29)\\
\hat{\mathbf{y}} &\simeq \frac{\Delta t}{2}\hat{\mathbf{s}} && (6.30)
\end{aligned}
$$

[2]Trapezoidal integration is simply defined as follows: $\int_a^b f(x)dx \simeq \frac{b-a}{2}(f(a) + f(b))$

6.3.4. Error Propagation

In the Extended Kalman Filter prediction step, we need to linearize the dynamics to propagate the covariance. This equates to finding the matrix that describes how the error evolves over time. Intuitively, we know that the error after propagation will be the previous error plus some new updated value and it should increase over time. In reality, because of rotations, we might see error decrease in one dimension and increase in another, but the overall error will increase during propagation. The goal of this section is to find the matrix $\mathbf{\Phi}$ that describes how the error evolves over time. Specifically, we want to find how each error term can be described in terms of previous state errors and the IMU inputs. First we turn towards the orientation error, where we use the estimates,

$$
\begin{aligned}
{}^{B}_{G}\mathbf{R} &\simeq {}^{B}_{G}\hat{\mathbf{R}}\left(\mathbf{I}_3 - \lfloor {}^{G}\tilde{\boldsymbol{\theta}}\times \rfloor\right) && (6.31)\\
{}^{B_{\ell+1}}_{B_\ell}\mathbf{R} &\simeq {}^{B_{\ell+1}}_{B_\ell}\hat{\mathbf{R}}\left(\mathbf{I}_3 - \lfloor {}^{B}\tilde{\boldsymbol{\theta}}_{\Delta\ell}\times \rfloor\right) && (6.32)
\end{aligned}
$$

Through substitution and ignoring the product of error terms, we get the orientation error propagation.

$$
{}^{G}\tilde{\boldsymbol{\theta}}_{\ell+1} \simeq {}^{G}\tilde{\boldsymbol{\theta}}_{\ell} + {}^{G}_{B_\ell}\hat{\mathbf{R}}\,{}^{B_\ell}\tilde{\boldsymbol{\theta}}_{\Delta\ell} \tag{6.33}
$$

Similarly, we linearize the velocity and position propagation equations above to find the error propagation, using the identities $\lfloor \mathbf{a}\times \rfloor \mathbf{b} = -\lfloor \mathbf{b}\times \rfloor \mathbf{a}$, and $\lfloor \mathbf{a}\times \rfloor^T = -\lfloor \mathbf{a}\times \rfloor$.

$$
\begin{aligned}
{}^{G}\tilde{\mathbf{v}}_{\ell+1} &\simeq -\lfloor {}^{B_\ell}_{G}\hat{\mathbf{R}}^T\,\hat{\mathbf{s}}_\ell\times \rfloor\,{}^{G}\tilde{\boldsymbol{\theta}}_{\ell} + {}^{G}\tilde{\mathbf{v}}_{\ell} + {}^{B_\ell}_{G}\hat{\mathbf{R}}^T\,\tilde{\mathbf{s}}_\ell && (6.34)\\
{}^{G}\tilde{\mathbf{p}}_{\ell+1} &\simeq -\lfloor {}^{B_\ell}_{G}\hat{\mathbf{R}}^T\,\hat{\mathbf{y}}_\ell\times \rfloor\,{}^{G}\tilde{\boldsymbol{\theta}}_{\ell} + {}^{G}\tilde{\mathbf{v}}_{\ell}\Delta t + {}^{G}\tilde{\mathbf{p}}_{\ell} + {}^{B_\ell}_{G}\hat{\mathbf{R}}^T\,\tilde{\mathbf{y}}_\ell && (6.35)\\
& && (6.36)
\end{aligned}
$$

By combining these error terms, we produce the error state transition matrix.

$$
\begin{aligned}
\begin{bmatrix} {}^{G}\tilde{\boldsymbol{\theta}}_{\ell+1} \\ {}^{G}\tilde{\mathbf{p}}_{\ell+1} \\ {}^{G}\tilde{\mathbf{v}}_{\ell+1} \end{bmatrix} &= \begin{bmatrix} \mathbf{I}_3 & \mathbf{0}_3 & \mathbf{0}_3 \\ -\lfloor {}^{B_\ell}_{G}\hat{\mathbf{R}}^T\,\hat{\mathbf{y}}_\ell\times \rfloor & \mathbf{I}_3 & \mathbf{I}_3\Delta t \\ -\lfloor {}^{B_\ell}_{G}\hat{\mathbf{R}}^T\,\hat{\mathbf{s}}_\ell\times \rfloor & \mathbf{0}_3 & \mathbf{I}_3 \end{bmatrix} \begin{bmatrix} {}^{G}\tilde{\boldsymbol{\theta}}_{\ell} \\ {}^{G}\tilde{\mathbf{p}}_{\ell} \\ {}^{G}\tilde{\mathbf{v}}_{\ell} \end{bmatrix} + \begin{bmatrix} {}^{B_\ell}_{G}\hat{\mathbf{R}}^T\,{}^{B_\ell}\tilde{\boldsymbol{\theta}}_{\Delta\ell} \\ {}^{B_\ell}_{G}\hat{\mathbf{R}}^T\,\tilde{\mathbf{y}}_{\Delta\ell} \\ {}^{B_\ell}_{G}\hat{\mathbf{R}}^T\,\tilde{\mathbf{s}}_{\Delta\ell} \end{bmatrix} && (6.37)\\
&= \begin{bmatrix} \mathbf{I}_3 & \mathbf{0}_3 & \mathbf{0}_3 \\ \mathbf{\Phi}_{pq} & \mathbf{I}_3 & \mathbf{I}_3\Delta t \\ \mathbf{\Phi}_{vq} & \mathbf{0}_3 & \mathbf{I}_3 \end{bmatrix} \begin{bmatrix} {}^{G}\tilde{\boldsymbol{\theta}}_{\ell} \\ {}^{G}\tilde{\mathbf{p}}_{\ell} \\ {}^{G}\tilde{\mathbf{v}}_{\ell} \end{bmatrix} + \begin{bmatrix} {}^{B_\ell}_{G}\hat{\mathbf{R}}^T\,{}^{B_\ell}\tilde{\boldsymbol{\theta}}_{\Delta\ell} \\ {}^{B_\ell}_{G}\hat{\mathbf{R}}^T\,\tilde{\mathbf{y}}_{\Delta\ell} \\ {}^{B_\ell}_{G}\hat{\mathbf{R}}^T\,\tilde{\mathbf{s}}_{\Delta\ell} \end{bmatrix} && (6.38)
\end{aligned}
$$

The right most component is made up of the bias, IMU calibration constants, and noise elements. The sparse nature of the error state transition matrix becomes clear when we examine the full covariance propagation from timestep t_ℓ to $t_{\ell+1}$.

$$
\mathbf{\Sigma}_{\ell+1} = \begin{bmatrix} \mathbf{\Phi}_{B_\ell} & \mathbf{\Gamma}_{imu_\ell} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix} \mathbf{\Sigma}_{\ell} \begin{bmatrix} \mathbf{\Phi}_{B_\ell} & \mathbf{\Gamma}_{imu_\ell} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix}^T + \begin{bmatrix} \mathbf{Q}_{\ell} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix} \tag{6.39}
$$

The full $\mathbf{\Phi}_{B_\ell}$ matrix is 15×15 while $\mathbf{\Gamma}_{imu_\ell}$ is 15×27. With the random walk biases and estimated IMU calibration constants included, the two error state transition matrices have

the following structure.

$$\mathbf{\Phi}_{B_\ell} = \begin{bmatrix} \mathbf{\Phi}_{qq} & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{\Phi}_{qb_g} & \mathbf{\Phi}_{qb_a} \\ \mathbf{\Phi}_{pq} & \mathbf{I}_3 & \mathbf{I}_3\Delta t & \mathbf{\Phi}_{pb_g} & \mathbf{\Phi}_{pb_a} \\ \mathbf{\Phi}_{vq} & \mathbf{0}_3 & \mathbf{I}_3 & \mathbf{\Phi}_{vb_g} & \mathbf{\Phi}_{vb_a} \\ \mathbf{0_3} & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{I}_3 & \mathbf{0}_3 \\ \mathbf{0_3} & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{I}_3 \end{bmatrix} \qquad \mathbf{\Gamma}_{imu_\ell} = \begin{bmatrix} \mathbf{\Gamma}_{q\vec{\mathbf{T}}_g} & \mathbf{\Gamma}_{q\vec{\mathbf{T}}_s} & \mathbf{\Gamma}_{q\vec{\mathbf{T}}_a} \\ \mathbf{\Gamma}_{p\vec{\mathbf{T}}_g} & \mathbf{\Gamma}_{p\vec{\mathbf{T}}_s} & \mathbf{\Gamma}_{p\vec{\mathbf{T}}_a} \\ \mathbf{\Gamma}_{v\vec{\mathbf{T}}_g} & \mathbf{\Gamma}_{v\vec{\mathbf{T}}_s} & \mathbf{\Gamma}_{v\vec{\mathbf{T}}_a} \\ \mathbf{0_{3\times 9}} & \mathbf{0_{3\times 9}} & \mathbf{0_{3\times 9}} \\ \mathbf{0_{3\times 9}} & \mathbf{0_{3\times 9}} & \mathbf{0_{3\times 9}} \end{bmatrix} \tag{6.40}$$

For the full error transition matrix and its derivation, we refer the reader to Appendix B.

The Noise Matrix

Using the continuous time variances of the IMU noise characteristics, we have the noise matrix $\mathbf{Q}_c$ with the accelerometer and gyroscope continuous noise and bias values. For simplification we use a single value for all three axis.

$$\mathbf{Q_c} = \begin{bmatrix} \sigma_{g_c}^2\mathbf{I}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \sigma_{a_c}^2\mathbf{I}_3 & \mathbf{0}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & \sigma_{wg_c}^2\mathbf{I}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \sigma_{wa_c}^2\mathbf{I}_3 \end{bmatrix} \tag{6.41}$$

The mapping between these variances and the error state is done by the matrix $\mathbf{G}_c$. This is the Jacobian of the body error state, $\tilde{\mathbf{x}}_B$, with respect to the noise vector $(\mathbf{n}_g^T, \mathbf{n}_a^T, \mathbf{n}_{wg}^T, \mathbf{n}_{wa}^T)^T$.

$$\mathbf{G_c} = \begin{bmatrix} -\mathbf{I}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & -\hat{\mathbf{R}}_\ell^T & \mathbf{0}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{I}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{I}_3 \end{bmatrix} \tag{6.42}$$

The discrete noise matrix is computed through trapezoidal integration.

$$\mathbf{Q}_d = \int_{t_\ell}^{t_{\ell+1}} \mathbf{\Phi}(t_{\ell+1}, \tau)\, \underbrace{\mathbf{G}_c(\tau)\, \mathbf{Q}_c\, \mathbf{G}_c(\tau)^T}_{\mathbf{N_c}}\, \mathbf{\Phi}(t_{\ell+1}, \tau)^T\, d\tau \tag{6.43}$$

$$\mathbf{Q}_d(t+\Delta t, t) \simeq \frac{\Delta t}{2}\mathbf{\Phi}(t+\Delta t, t)\, \mathbf{N}_c\, \mathbf{\Phi}(t+\Delta t, t)^T + \mathbf{N}_c \tag{6.44}$$

Note that the matrix $\mathbf{N}_c$ is constant because the term $\hat{\mathbf{R}}_\ell^T\, \sigma_{a_c}^2\mathbf{I}_3\, \hat{\mathbf{R}}_\ell = \sigma_{a_c}^2\mathbf{I}_3$.

6.4. Augmentation

When an image is captured, the current body quaternion, position and velocity are added to the state vector and the covariance is augmented accordingly.

$$\mathbf{\Sigma}_{k|k-1} \leftarrow \begin{bmatrix} \mathbf{\Sigma}_{k|k-1} & \mathbf{\Sigma}_{k|k-1}\mathbf{J}_\pi^T \\ \mathbf{J}_\pi\mathbf{\Sigma}_{k|k-1} & \mathbf{J}_\pi\mathbf{\Sigma}_{k|k-1}\mathbf{J}_\pi^T \end{bmatrix}^T \tag{6.45}$$

$$\text{with } \mathbf{J}_\pi = \begin{bmatrix} \mathbf{I}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \dots \\ \mathbf{0}_3 & \mathbf{I}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \dots \\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{I}_3 & \mathbf{0}_3 & \dots \end{bmatrix} \tag{6.46}$$

The Jacobian $\mathbf{J}_\pi$ is the derivative of the camera pose with respect to the state vector.

6.5. Update Step

An EKF update requires a measurement, $\mathbf{z}$, that depends on and can be estimated from the current state. When compared, the difference between the measurement and the estimate is the residual vector.

$$\mathbf{r} = \mathbf{z} - \hat{\mathbf{z}} \tag{6.47}$$

$$\simeq \mathbf{H}\tilde{\mathbf{x}} + \mathbf{n} \tag{6.48}$$

Here $\mathbf{H}$ is the Jacobian of the measurement with respect to the state vector, $\tilde{\mathbf{x}}$ is the state error vector and $\mathbf{n}$ is the measurement noise.

6.5.1. Calculating the Residuals

Every time an image is captured, it is accompanied by a timestamp, t. The IMU measurements are used to propagate the state vector up to time $t + \hat{t}_d$ and the body pose is appended to the state vector as described above. The features in the image are detected and matched with features in the previous frame. If the ith feature is lost, meaning it was detected for n frames ($n \geq 3$) but not in the most recent frame, then it is triangulated with the method described in section 5.2 to give ${}^G\mathbf{p}_{f_i}$. This triangulated point yields the pixel measurement $\mathbf{z}_i$ at time-step j in which it appeared.

$$\mathbf{z}_{i,j} = \mathbf{h}({}^{C_j}\mathbf{p}_{f_i}) + \mathbf{n}_{i,j} \tag{6.49}$$

$${}^{C_j}\mathbf{p}_{f_i} = {}^C_B\mathbf{R}\,{}^{B_j}_G\mathbf{R}(t_n)\left({}^G\mathbf{p}_{f_i} - {}^G\mathbf{p}_{B_j}(t_n)\right) + {}^C\mathbf{p}_B \tag{6.50}$$

The function $\mathbf{h}$ is the camera projection function as described in equation (5.4), and $\mathbf{n}_{i,j}$ is the camera measurement noise vector. The rotation and position of the body depend on the time t_n, which is the time that the column[3] containing the pixel measurement was captured. The middle column of the image is captured at $t + t_d$ while row k is captured at $t + t_d + \frac{kt_r}{N}$, $k \in [-\frac{N}{2}, \frac{N}{2}]$. In order to compute the estimate $\hat{\mathbf{z}}_{i,j}$, the buffered IMU

[3]Some rolling shutter cameras are row-wise and some are column-wise. In this section we simply assume the latter.

measurements are used to integrate forwards (or backwards for $kt_r < 0$) to time t_n using the equations from the EKF prediction step in the previous section.

$$\hat{\mathbf{z}}_{i,j} = \mathbf{h}\left({}^{C}_{B}\mathbf{R}\,{}^{B_j}_{G}\hat{\mathbf{R}}(t+\hat{t}_d+\frac{k\hat{t}_r}{N})\left({}^{G}\hat{\mathbf{p}}_{f_i} - {}^{G}\hat{\mathbf{p}}_{B_j}(t+\hat{t}_d+\frac{k\hat{t}_r}{N})\right) + {}^{C}\hat{\mathbf{p}}_{B}\right) \tag{6.51}$$

6.5.2. Error Representation

In this section, we determine the measurement jacobian needed by the EKF for the update step. While integration of IMU measurements is a good way to estimate the measurement at time t_n, as described above, the covariance cannot be propagated in this way. Instead, linearizing the measurement equation with a Taylor expansion and using only low order terms gives a good enough bound for the measurement error. To understand this further, we first examine the residual for feature i and camera measurement j. Just as before, $\mathbf{z}$ is the actual measured value (the pixel coordinates of the feature), while $\hat{\mathbf{z}}$ is from equation (6.51).

$$\mathbf{r}_{i,j} = \mathbf{z}_{i,j} - \hat{\mathbf{z}}_{i,j} \tag{6.52}$$

$$\simeq \mathbf{H}_{\mathbf{x}}\,\tilde{\mathbf{x}} + \mathbf{H}_{\mathbf{f}}\,{}^{G}\tilde{\mathbf{p}}_{f_i} + \mathbf{n}_{i,j} \tag{6.53}$$

The matrices $\mathbf{H}_{\mathbf{x}}$ and $\mathbf{H}_{\mathbf{f}}$ are the Jacobians of the measurement with respect to the state and the feature position, respectively. The vector $\mathbf{n}_{i,j}$ is the pixel noise. $\mathbf{H}_{\mathbf{x}}$ is a sparse matrix with nonzero entries only corresponding to pose j and the camera calibration parameters. We can break $\mathbf{H}_{\mathbf{x}}$ apart into those partitions to better understand its structure.

$$\mathbf{r} \simeq \mathbf{H}_{\boldsymbol{\theta}}\,\tilde{\boldsymbol{\theta}}_{B_j}(\hat{t}_n) + \mathbf{H}_{\mathbf{p}}\,{}^{G}\tilde{\mathbf{p}}_{B_j}(\hat{t}_n) + \mathbf{H}_{c}\,\tilde{\mathbf{x}}_{c} + \mathbf{H}_{\mathbf{f}}\,{}^{G}\tilde{\mathbf{p}}_{f_i} + \mathbf{n} \tag{6.54}$$

This residual depends on t_n, which is not a part of the state vector and therefore cannot be used in the EKF update. We can express the errors at $\hat{t}_n$ using $\hat{t}_d$ and $\hat{t}_r$, which *are* in the state vector. The position and orientation errors at time $\hat{t}_n$ can be written using the Taylor expansion as follows

$${}^{G}\tilde{\mathbf{p}}_{B}(\hat{t}_n) = {}^{G}\tilde{\mathbf{p}}_{B}(t+\hat{t}_d) + \frac{k\hat{t}_r}{N}{}^{G}\tilde{\mathbf{v}}_{B}(t+\hat{t}_d) + \frac{(k\hat{t}_r)^2}{2N^2}{}^{G}\tilde{\mathbf{a}}_{B}(t+\hat{t}_d) + \dots \tag{6.55}$$

$$\tilde{\boldsymbol{\theta}}_{B}(\hat{t}_n) = \tilde{\boldsymbol{\theta}}_{B}(t+\hat{t}_d) + \frac{k\hat{t}_r}{N}\tilde{\boldsymbol{\omega}}_{B}(t+\hat{t}_d) + \dots \tag{6.56}$$

We model the position error using only the first two terms, and the orientation with only the first term, with minimal loss of accuracy [24]. The new residual without the higher order terms is as follows

$$\mathbf{r} \simeq \mathbf{H}_{\boldsymbol{\theta}}\,\tilde{\boldsymbol{\theta}}_{B_j}(t+\hat{t}_d) + \mathbf{H}_{\mathbf{p}}\,{}^{G}\tilde{\mathbf{p}}_{B_j}(t+\hat{t}_d) + \frac{k\hat{t}_r}{N}\mathbf{H}_{\mathbf{p}}\,{}^{G}\tilde{\mathbf{v}}_{B_j}(t+\hat{t}_d) + \mathbf{H}_{c}\,\tilde{\mathbf{x}}_{c} + \mathbf{H}_{\mathbf{f}}\,{}^{G}\tilde{\mathbf{p}}_{f_i} + \mathbf{n} \tag{6.57}$$

Jacobian Definitions

We seek now to find the value of $\mathbf{H}_{\mathbf{x}}$ and $\mathbf{H}_{\mathbf{f}}$. The Jacobian of the body pose B_j (the orientation, position and velocity) where measurement i was taken can be calculated by

making use of the chain rule.

$$\mathbf{H}_{\mathbf{x}B_j} = \frac{\partial \mathbf{z}}{\partial \mathbf{x}_{B_j}} = \frac{\partial \mathbf{h}}{\partial {}^{C_j}\mathbf{p}_{f_i}} \frac{\partial {}^{C_j}\mathbf{p}_{f_i}}{\partial \mathbf{x}_{B_j}} = \mathbf{J_h} \frac{\partial {}^{C_j}\mathbf{p}_{f_i}}{\partial \mathbf{x}_{B_j}} \tag{6.58}$$

The matrix $\mathbf{J_h}$ is the projection Jacobian from equation 5.20. Finding the Jacobian of the feature position can be done by formulating the error ${}^{C}\tilde{\mathbf{p}}_f$ in terms of the errors in the state vector. For clarity, we omit the indices i and j and we use $t_n = t + t_d + \frac{kt_r}{N}$.

$${}^{C}\tilde{\mathbf{p}}_f = {}^{C}\mathbf{p}_f - {}^{C}\hat{\mathbf{p}}_f \tag{6.59}$$

We now substite the actual and estimated values for the feature position with the goal of writing the error ${}^{C}\tilde{\mathbf{p}}_f$ in terms of the state errors and ${}^{G}\tilde{\mathbf{p}}_f$.

$$\begin{aligned} {}^{C}\tilde{\mathbf{p}}_f &= \left({}^{C}_{B}\mathbf{R}\, {}^{B}_{G}\mathbf{R}(t_n)\left({}^{G}\mathbf{p}_f - {}^{G}\mathbf{p}_B(t_n)\right) + {}^{C}\mathbf{p}_B\right) - \left({}^{C}_{B}\mathbf{R}\, {}^{B}_{G}\hat{\mathbf{R}}(\hat{t}_n)\left({}^{G}\hat{\mathbf{p}}_f - {}^{G}\hat{\mathbf{p}}_B(\hat{t}_n)\right) + {}^{C}\hat{\mathbf{p}}_B\right) \\ &= {}^{C}_{B}\mathbf{R}\left({}^{B}_{G}\mathbf{R}(t_n)\left({}^{G}\mathbf{p}_f - {}^{G}\mathbf{p}_B(t_n)\right) - {}^{B}_{G}\hat{\mathbf{R}}(\hat{t}_n)\left({}^{G}\hat{\mathbf{p}}_f - {}^{G}\hat{\mathbf{p}}_B(\hat{t}_n)\right)\right) + {}^{C}\tilde{\mathbf{p}}_B \end{aligned} \tag{6.60}$$

We now expand the rotation term with the first order taylor series expansion and the approximation ${}^{B}_{G}\mathbf{R} \simeq {}^{B}_{G}\hat{\mathbf{R}}(\mathbf{I}_3 - \lfloor {}^{G}\tilde{\boldsymbol{\theta}} \times \rfloor)$.

$${}^{B}_{G}\mathbf{R}(t_n) \simeq {}^{B}_{G}\mathbf{R}(\hat{t}_n) + \frac{\partial {}^{B}_{G}\mathbf{R}}{\partial t_n}(\hat{t}_n)\tilde{t}_n \tag{6.61}$$

$$\simeq {}^{B}_{G}\hat{\mathbf{R}}(\hat{t}_n)(\mathbf{I}_3 - \lfloor {}^{G}\tilde{\boldsymbol{\theta}}(\hat{t}_n) \times \rfloor) - \lfloor {}^{B}\boldsymbol{\omega}(\hat{t}_n) \times \rfloor {}^{B}_{G}\hat{\mathbf{R}}\, \tilde{t}_n \tag{6.62}$$

$${}^{G}\mathbf{p}_B(t_n) \simeq {}^{G}\mathbf{p}_B(\hat{t}_n) + {}^{G}\mathbf{v}_B(\hat{t}_n)\, \tilde{t}_n \tag{6.63}$$

Substitution and rearranging gives the following

$$\begin{aligned} {}^{C}\tilde{\mathbf{p}}_f = {}&{}^{C}_{B}\mathbf{R}\, {}^{B}_{G}\hat{\mathbf{R}}(\hat{t}_n)\left(({}^{G}\tilde{\mathbf{p}}_f - {}^{G}\tilde{\mathbf{p}}_B(\hat{t}_n)) - \lfloor {}^{G}\tilde{\boldsymbol{\theta}}(\hat{t}_n) \times \rfloor ({}^{G}\mathbf{p}_f - {}^{G}\mathbf{p}_B(t_n)) - {}^{G}\mathbf{v}_B(t_n)\, \tilde{t}_n\right) \\ &- {}^{C}_{B}\mathbf{R} \lfloor {}^{B}\boldsymbol{\omega}(\hat{t}_n) \times \rfloor {}^{B}_{G}\hat{\mathbf{R}}(\hat{t}_n)({}^{G}\mathbf{p}_f - {}^{G}\mathbf{p}_B(t_n))\tilde{t}_n + {}^{C}\tilde{\mathbf{p}}_B \end{aligned} \tag{6.64}$$

And finally, after further simplification and with the first two terms of equation (6.55), we can add the velocity error.

$$\begin{aligned} {}^{C}\tilde{\mathbf{p}}_f = {}&{}^{C}_{B}\mathbf{R}\, {}^{B}_{G}\hat{\mathbf{R}}(\hat{t}_n)\, {}^{G}\tilde{\mathbf{p}}_f \\ &- {}^{C}_{B}\mathbf{R}\, {}^{B}_{G}\hat{\mathbf{R}}(\hat{t}_n)\, {}^{G}\tilde{\mathbf{p}}_B(t + \hat{t}_d) \\ &- \frac{k\hat{t}_r}{N}\, {}^{C}_{B}\mathbf{R}\, {}^{B}_{G}\hat{\mathbf{R}}(\hat{t}_n)\, {}^{G}\tilde{\mathbf{v}}_B(t + \hat{t}_d) \\ &+ {}^{C}_{B}\mathbf{R}\, {}^{B}_{G}\hat{\mathbf{R}}(\hat{t}_n) \lfloor {}^{G}\hat{\mathbf{p}}_f - {}^{G}\hat{\mathbf{p}}_B(\hat{t}_n) \times \rfloor\, {}^{G}\tilde{\boldsymbol{\theta}}(t + \hat{t}_d) \\ &- {}^{C}_{B}\mathbf{R}\left({}^{B}_{G}\hat{\mathbf{R}}(\hat{t}_n)\, {}^{G}\hat{\mathbf{v}}_B - \lfloor {}^{B}_{G}\hat{\mathbf{R}}(\hat{t}_n)({}^{G}\hat{\mathbf{p}}_f - {}^{G}\hat{\mathbf{p}}_B(\hat{t}_n))) \times \rfloor\, {}^{B}\boldsymbol{\omega}(\hat{t}_n)\right)\tilde{t}_n \\ &+ {}^{C}\tilde{\mathbf{p}}_B \end{aligned} \tag{6.65}$$

Note that ${}^{B}\boldsymbol{\omega}(\hat{t}_n)$ is the rotational velocity at time $\hat{t}_n$. This can be calculated as in the prediction step using the buffered IMU measurements. We now have all of the sources of error in the feature measurement. Returning to the Jacobian of just the body pose j, we take the above terms for ${}^{G}\tilde{\boldsymbol{\theta}}$, ${}^{G}\tilde{\mathbf{p}}_B$, and ${}^{G}\tilde{\mathbf{v}}_B$ as the orientation, position and velocity Jacobians, respectively.

$$\mathbf{H}_{\mathbf{x}B_j} = \mathbf{M}_{i,j}\begin{bmatrix} \lfloor {}^{G}\hat{\mathbf{p}}_{f_i} - {}^{G}\hat{\mathbf{p}}_{B_j}(t+\hat{t}_d+\frac{k\hat{t}_r}{N})\times\rfloor & -\mathbf{I}_3 & -\frac{k\hat{t}_r}{N}\mathbf{I}_3 \end{bmatrix} \tag{6.66}$$

$$\mathbf{M}_{i,j} = \mathbf{J}_{\mathbf{h}}\,{}^{C}_{B}\mathbf{R}\,{}^{B_j}_{G}\hat{\mathbf{R}}(t+\hat{t}_d+\frac{k\hat{t}_r}{N}) \tag{6.67}$$

We can also see from equation (6.65) that the Jacobians with respect to the spacial and temporal calibration parameters are fairly straight forward with $\tilde{t_n} = \tilde{t}_d + \frac{k}{N}\tilde{t}_r$.

$$\frac{\partial \mathbf{z}}{\partial {}^{C}\mathbf{p}_B} = \mathbf{J}_{\mathbf{h}} \tag{6.68}$$

$$\frac{\partial \mathbf{z}}{\partial t_d} = \mathbf{J}_{\mathbf{h}}\,{}^{C}_{B}\mathbf{R}\left(\lfloor {}^{B}_{G}\hat{\mathbf{R}}({}^{G}\hat{\mathbf{p}}_{f_i} - {}^{G}\mathbf{p}_{B_j})\times\rfloor {}^{B}\boldsymbol{\omega} - {}^{B}_{G}\hat{\mathbf{R}}\,{}^{G}\mathbf{v}_{B_j}\right) \tag{6.69}$$

$$\frac{\partial \mathbf{z}}{\partial t_r} = \frac{k}{N}\frac{\partial \mathbf{z}}{\partial t_d} \tag{6.70}$$

The Jacobian of the remaining camera calibration parameters is a simple matter of differentiation of equation (5.4) with respect to each of the intrinsic parameters.

$$\mathbf{H}_{\mathbf{c}} = \begin{bmatrix} \frac{\partial \mathbf{z}}{\partial {}^{C}\mathbf{p}_B} & \frac{\partial \mathbf{z}}{\partial \mathbf{x}_{cam}} & \frac{\partial \mathbf{z}}{\partial t_d} & \frac{\partial \mathbf{z}}{\partial t_r} \end{bmatrix} \tag{6.71}$$

$$\frac{\partial \mathbf{z}}{\partial \mathbf{x}_{cam}} = \begin{bmatrix} \frac{\partial h}{\partial f_x} & \frac{\partial h}{\partial f_y} & \frac{\partial h}{\partial o_x} & \frac{\partial h}{\partial o_y} & \frac{\partial h}{\partial k_1} & \frac{\partial h}{\partial k_2} & \frac{\partial h}{\partial k_3} & \frac{\partial h}{\partial t_1} & \frac{\partial h}{\partial t_2} \end{bmatrix} \tag{6.72}$$

Finally, the Jacobian of the feature position can be found from equation (6.65).

$$\mathbf{H}_{\mathbf{f}} = \mathbf{J}_{\mathbf{h}}\,{}^{C}_{B}\mathbf{R}\,{}^{B_j}_{G}\hat{\mathbf{R}}(t+\hat{t}_d+\frac{k\hat{t}_r}{N}) \tag{6.73}$$

After calculating all the residuals and Jacobians for feature i, they are stacked to create $\mathbf{r}_i$, $\mathbf{H}_{\mathbf{x}i}$, and $\mathbf{H}_{\mathbf{f}i}$. If the first measurement of feature i was at time α_0 and it was measured n times, it will be processed at time-step α_{n+1} and the resulting stacked residual will be a $2n \times 1$ vector.

$$\mathbf{r}_i = \begin{bmatrix} \mathbf{r}_{i,\alpha_0}^T & \dots & \mathbf{r}_{i,\alpha_n}^T \end{bmatrix}^T \tag{6.74}$$

$$\simeq \mathbf{H}_{\mathbf{x}i}\,\tilde{\mathbf{x}} + \mathbf{H}_{\mathbf{f}i}\,{}^{G}\tilde{\mathbf{p}}_{f_i} + \mathbf{n}_i \tag{6.75}$$

6.5.3. Feature Error Marginalization

The form of the above equation is different from the required form of equation 6.48 because the feature is not part of the state vector, but it is computed from and therefore correlated to the state. In order to fix this, we follow the strategy of [27] and eliminate the feature

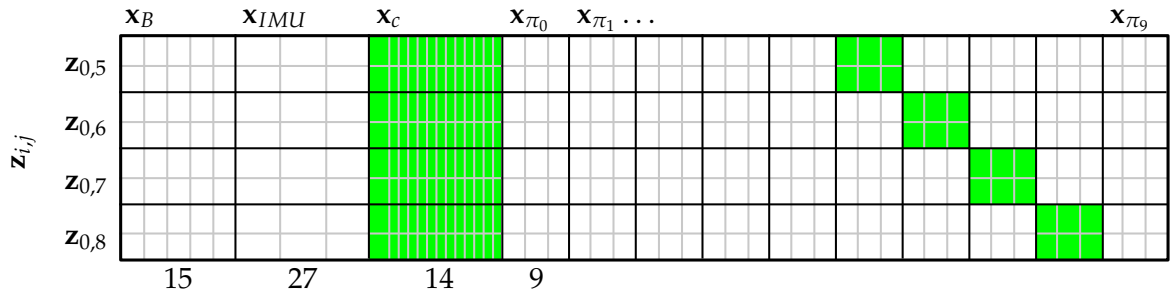


Figure 6.4. The stacked Jacobian $\mathbf{H_x}$ for feature i has nonzero entries (marked in green) at the poses in which it was detected and at the camera calibration indices. Note that all the features should include the second to last pose in the sliding window, which means they were tracked up until the most recent image.

from the residual. Because $\mathbf{H_{f}}_i$ is a $2n \times 3$ matrix with full column rank, its left nullspace[4] has dimension $2n-3$. If $\mathbf{A}_i$ is a unitary matrix whose columns form the basis of the left nullspace of $\mathbf{H_{f}}_i$, then $\mathbf{A}_i^T \mathbf{H_{f}}_i = \mathbf{0}_{2n-3\times 1}$ and $\mathbf{A}_i \mathbf{A}_i^T = \mathbf{I}_{2n-3}$. Left multiplying equation 6.75 by $\mathbf{A}_i^T$ marginalizes out the feature error.

$$\mathbf{A}_i^T \mathbf{r}_i \simeq \mathbf{A}_i^T \mathbf{H_x}_i \tilde{\mathbf{x}} + \mathbf{A}_i^T \mathbf{H_f}_i + \mathbf{A}_i^T \mathbf{n}_i \tag{6.76}$$

$$\mathbf{A}_i^T \mathbf{r}_i \simeq \mathbf{A}_i^T \mathbf{H_x}_i \tilde{\mathbf{x}} + \mathbf{A}_i^T \mathbf{n}_i \tag{6.77}$$

$$\mathbf{r^o}_i \simeq \mathbf{H^o}_i \tilde{\mathbf{x}} + \mathbf{n^o}_i \tag{6.78}$$

The new residual has the required form of equation (6.48) and is independent of the feature position error. It now has dimension $2n-3 \times 1$. The noise vector $\mathbf{n^o}_i$ has covariance matrix $\sigma_{im}^2 \mathbf{I}_{2n-3}$ where σ_{im}^2 is the image noise variance calculated during camera calibration.

6.5.4. Outlier Detection

Before using $\mathbf{r^o}_i$ and $\mathbf{H^o}_i$ in an EKF update, we apply a statistical test to seperate inliers from outliers. Outliers occur when the feature tracker fails and matches features incorrectly or when the tracked feature is not static, such as on a car or pedestrian. The entire residual vector from a feature is designated an inlier or an outlier and all outliers are discarded.

The Chi-square test is a commonly used test to see if a hypothesis fits an observation. The test yields a single value, γ, by summing the square differences between an observation and a hypothesis, divided by the variance.

$$\gamma = \sum_{i=1}^{k} \left(\frac{X_i - \mu_i}{\sigma_i} \right)^2 \tag{6.79}$$

In our case, the residual $\mathbf{r^o}_i$ is already the difference between the observation and hypoth-

[4]The left nullspace of A is the nullspace of A^T. If A is $n \times m$, with $m \leq n$, and has *full column rank*, ie. rank m, then A^T is $m \times n$ and also has rank m because $rank(A) = rank(A^T)$. This means that the nullspace of A^T dimension $n-m$.

esis, and the residual covariance is given by $\mathbf{H^o}_i \, \mathbf{\Sigma} \, \mathbf{H^o}_i^T$.

$$\gamma_i = \mathbf{r^o}_i \left(\mathbf{H^o}_i \, \mathbf{\Sigma} \, \mathbf{H^o}_i^T \right)^{-1} \mathbf{r^o}_i^T \tag{6.80}$$

If γ is large, then it is clear that some or all of the observations are farther from the hypothesis than predicted by the uncertainty. We compare γ to X such that the Chi-squared cumulative density function for X and k degrees of freedom is 0.95. In other words, the value X that yields 0.95 from the Chi-square cdf is the upper bound for statistical significance, so if γ_i is *less than* X, then the ith feature is an inlier. The Chi-squared (χ^2) cdf is as follows. Note that k is the number of degrees of freedom, which for $\mathbf{r^o}_i$ is $2n - 3$.

$$\text{cfd}_{\chi^2}(X, k) = \frac{\gamma(\frac{k}{2}, \frac{x}{2})}{\Gamma(\frac{k}{2}, \frac{k}{2})} \tag{6.81}$$

$$\Gamma(s, x) = \int_x^\infty t^{s-1} \, e^{-t} \, dt \tag{6.82}$$

$$\gamma(s, x) = \int_0^x t^{s-1} \, e^{-t} \, dt \tag{6.83}$$

In practice, the Chi-square cdf is not evaluated, but a table of possible values for X and k that give 0.95 is used for efficient comparisons.

6.5.5. EKF Update

After the outliers are discarded, the remaining residual vectors from all L evaluated inlier features are stacked together.

$$\begin{bmatrix} \mathbf{r^o}_1 \\ \vdots \\ \mathbf{r^o}_L \end{bmatrix} = \begin{bmatrix} \mathbf{H^o}_1 \\ \vdots \\ \mathbf{H^o}_L \end{bmatrix} \tilde{\mathbf{x}} + \begin{bmatrix} \mathbf{n^o}_1 \\ \vdots \\ \mathbf{n^o}_L \end{bmatrix} \tag{6.84}$$

$$\mathbf{r} = \mathbf{H} \, \tilde{\mathbf{x}} + \mathbf{n} \tag{6.85}$$

The dimension of $\mathbf{r}$ is now $d \times 1$ with $d = \sum_{j=1}^{L} (2n_j - 3)$. Because the noise vectors for each feature are uncorrelated, the covariance matrix of $\mathbf{n}$ is $\mathbf{R}_n = \sigma_{im}^2 \mathbf{I}_d$. The EKF Update proceeds by calculating the Kalman Gain.

$$\mathbf{K} = \mathbf{\Sigma} \, \mathbf{H}^T (\mathbf{H} \, \mathbf{\Sigma} \, \mathbf{H}^T + \mathbf{R}_n)^{-1} \tag{6.86}$$

For large values of d, the complexity of the above equation grows very large. For example, if 11 features are each seen in 11 images, then $d = 209$. To reduce the complexity, we examine the matrix $\mathbf{H}$. The QR decomposition of $\mathbf{H}$ reveals that much of the information is noise and can be disregarded.

$$\mathbf{H} = [\mathbf{Q}_1 \; \mathbf{Q}_2] \begin{bmatrix} \mathbf{T}_H \\ \mathbf{0} \end{bmatrix} \tag{6.87}$$

The matrices $\mathbf{Q}_1$ and $\mathbf{Q}_2$ are unitary and have columns that form the bases for the range and nullspace of H, respectively. The R part of the QR decomposition is divided into an upper triangular matrix $\mathbf{T}_H$ and zeros. Because $\mathbf{Q}_1$ and $\mathbf{Q}_2$ are unitary, we can rewrite equation 6.85 as follows

$$\mathbf{r} = [\mathbf{Q}_1 \ \mathbf{Q}_2] \begin{bmatrix} \mathbf{T}_H \\ \mathbf{0} \end{bmatrix} \tilde{\mathbf{x}} + \mathbf{n} \tag{6.88}$$

$$[\mathbf{Q}_1 \ \mathbf{Q}_2]^T \mathbf{r} = \begin{bmatrix} \mathbf{T}_H \\ \mathbf{0} \end{bmatrix} \tilde{\mathbf{x}} + [\mathbf{Q}_1 \ \mathbf{Q}_2]^T \mathbf{n} \tag{6.89}$$

$$\begin{bmatrix} \mathbf{Q}_1^T \mathbf{r} \\ \mathbf{Q}_2^T \mathbf{r} \end{bmatrix} = \begin{bmatrix} \mathbf{T}_H \\ \mathbf{0} \end{bmatrix} \tilde{\mathbf{x}} + \begin{bmatrix} \mathbf{Q}_1^T \mathbf{n} \\ \mathbf{Q}_2^T \mathbf{n} \end{bmatrix} \tag{6.90}$$

It is now clear that $\mathbf{Q}_2^T \mathbf{r}$ is only noise and can be discarded giving a smaller dimensional residual and Jacobian. The new noise vector, $\mathbf{Q}_1^T \mathbf{n}$, has covariance $\mathbf{R}_q = \mathbf{Q}_1^T \mathbf{R}_n \mathbf{Q}_1 = \sigma_{im}^2 \mathbf{I}_r$ (because $\mathbf{Q}_1$ is unitary) where r is the number of columns in $\mathbf{Q_1}$ and the new dimension of the residual.

$$\mathbf{Q}_1^T \mathbf{r} = \mathbf{T}_H \tilde{\mathbf{x}} + \mathbf{Q}_1^T \mathbf{n} \tag{6.91}$$

$$\mathbf{r}_q = \mathbf{T}_H \tilde{\mathbf{x}} + \mathbf{n}_q \tag{6.92}$$

Finally, the EKF update can take place with the new residual and Jacobian matrix.

$$\mathbf{K} = \mathbf{\Sigma}_{k+1|k} \mathbf{T}_H^T (\mathbf{T}_H \mathbf{\Sigma}_{k+1|k} \mathbf{T}_H^T + \mathbf{R}_q)^{-1} \tag{6.93}$$

$$\mathbf{\Sigma}_{k+1|k+1} = (\mathbf{I}_\beta - \mathbf{K} \mathbf{T}_H) \mathbf{\Sigma}_{k+1|k} (\mathbf{I}_\beta - \mathbf{K} \mathbf{T}_H)^T + \mathbf{K} \mathbf{R}_q \mathbf{K}^T \tag{6.94}$$

$$\Delta \mathbf{x} = \mathbf{K} \mathbf{r}_q \tag{6.95}$$

$$\Delta \mathbf{x} = \begin{bmatrix} \underbrace{\delta \boldsymbol{\theta}_B^T, \Delta \mathbf{p}_B^T, \Delta \mathbf{v}_B^T, \Delta \mathbf{b}_g^T, \Delta \mathbf{b}_a^T}_{\text{Body Pose Update}}, \ \ldots \ \underbrace{\delta \boldsymbol{\theta}_{B_i}^T, \Delta \mathbf{p}_{B_j}^T, \Delta \mathbf{v}_{B_j}^T}_{\text{Pose } j \text{ Update}}, \ \ldots \end{bmatrix}^T \tag{6.96}$$

The identity matrix $\mathbf{I}_\beta$ has the dimensions of the covariance matrix: $9m + 56$ for m poses in the sliding window. The vector $\Delta \mathbf{x}$ is now used to update the current state estimate.

$$\hat{\mathbf{x}}_{k+1|k+1} = \begin{bmatrix} {}^B_G\hat{\mathbf{q}} \otimes \delta \mathbf{q}_B \\ {}^G\hat{\mathbf{p}}_B + \Delta \mathbf{p}_B \\ {}^G\hat{\mathbf{v}}_B + \Delta \mathbf{v}_B \\ \hat{\mathbf{b}}_g + \Delta \mathbf{b}_g \\ \hat{\mathbf{b}}_a + \Delta \mathbf{b}_a \\ \vdots \\ {}^{B_i}_G\hat{\mathbf{q}} \otimes \delta \mathbf{q}_{B_i} \\ {}^G\hat{\mathbf{p}}_{B_i} + \Delta \mathbf{p}_{B_i} \\ {}^G\hat{\mathbf{v}}_{B_i} + \Delta \mathbf{p}_{B_i} \end{bmatrix} \qquad \text{with } \delta \mathbf{q} = \begin{bmatrix} \frac{1}{2} \delta \boldsymbol{\theta} \\ 1 \end{bmatrix} \tag{6.97}$$

6.6. Post EKF Update

After the update takes place, if there are too many poses in the state vector, the oldest pose is removed and the corresponding rows and columns of the covariance matrix are cut out. Other old poses which also no longer have tracked features can be removed at this point as well.

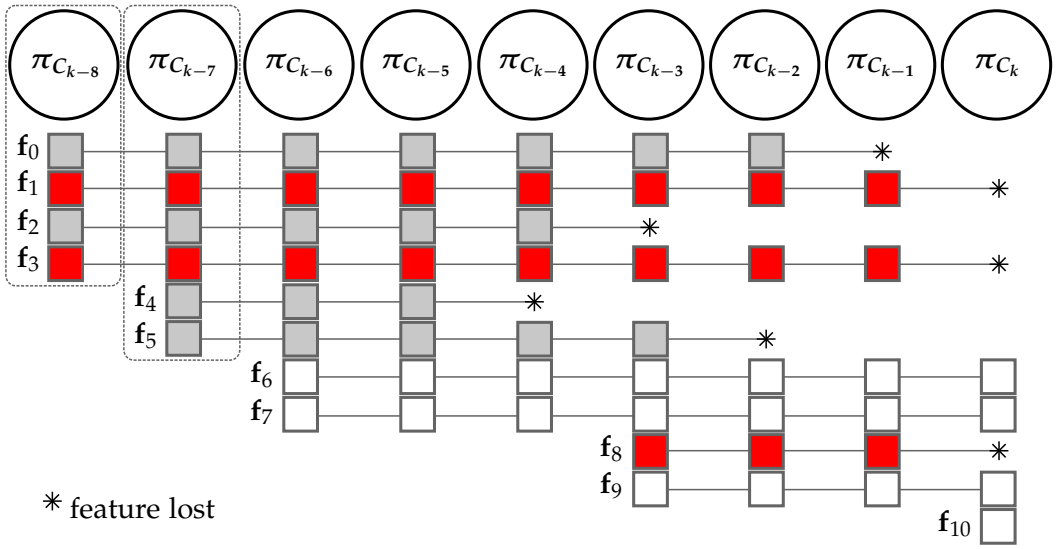


Figure 6.5. The sliding window of camera poses at time k and the corresponding features. For features that are lost in the current frame (visualized in red), an update is performed using their triangulated positions as constraints. After the update, the oldest camera poses with no remaining tracked features will be discarded (in this example, the oldest two camera poses). The grey boxes are features that have already been used for an update.

7. Implementation

7.1. Simulation Generation

7.1.1. Motion Generation

The Bézier curve is a method of interpolation between points using additional control points to smooth the transitions. It is a particularly useful method for generating simulation data. The most important advantage of the Bézier curve is that it is trivial to calculate the first and second derivative at any point. A fifth order curve is particularly useful because we can set the initial velocity to zero and assure that the acceleration is continuous throughout the entire motion. We implement the method from [4] to generate two Bézier curves through a series of predefined waypoints. One curve describes the position, velocity and acceleration while the second curve describes the rotation and rotational velocity with quaternions.

7.1.2. Feature Generation

Given the simulation's camera parameters, we force each image to have a number of features between two thresholds, f_{min} and f_{max}. When not enough features exist, they are generated by choosing random pixel locations to form a ray from the camera's focal point. A random depth is applied and the feature is initialized with a global 3D location. The depth is initialized with a normally distributed random inverse depth value $\rho \sim \mathcal{N}(\frac{1}{4}, \frac{1}{4})$.

Each frame the 3D features are shuffled randomly and one by one they are projected into the current frame with random gaussian noise added to the pixel value. If the feature is within the bounds of the image, it must then pass a strength test. A uniform random number is compared with a threshold t_{detect} and if lower, the feature is visible. Finally, if the feature was detected in the previous frame, another random threshold t_{match} must be passed for it to be matched with the previous frame. If it is not matched, it is considered a new feature.

Additionally, outlier features and feature tracks are generated by following the same process as above with the exception that the feature's position is generated randomly each time it is detected.

7.2. Application Details

7.2.1. iOS Implementation

Because of work flow requirements and time constraints, we have created an iOS application for data gathering only. The app records a video at 30Hz, acceleration and gyroscope readings at their maximum frequencies of 100Hz, compass and GPS data. The compass

and GPS data are used purely for comparison. The video is saved in the application's documents folder with a data file containing all the measurements with time stamps. The two files can then be used directly in our desktop application, written in XCode on the Mac.

Apple's iOS in general suffers from a few limitations. The camera's focal length cannot be set programmatically, but can only be locked once recording has started. This means that the focal length must be independently estimated with high initial uncertainty in each trial run. Additionally, we found that on the resource constrained iPhone 4S, recorded video tends to drop frames on occassion while the iPad 3 did not exhibit this problem. This causes problems when the video frame timestamps are not exactly aligned with the video frames, for example when an extra frame is present at the beginning of the video file. Manual inspection of each file and timestamp set was the only way to correct for possible misalignments in practice. Running the algorithm live on the device would be the best way to circumvent this problem.

7.2.2. Desktop Implementation

The algorithm was implemented first in Matlab, but was ported to C++ to be more efficient and to eventually be ported to the iOS device or ROS, the Robot Operating System [32]. The desktop app is mostly implemented in C++, but user interface components are coded in Apple's Objective-C. We used OpenCV for feature detection and matching and the Eigen library for all matrix mathematics.

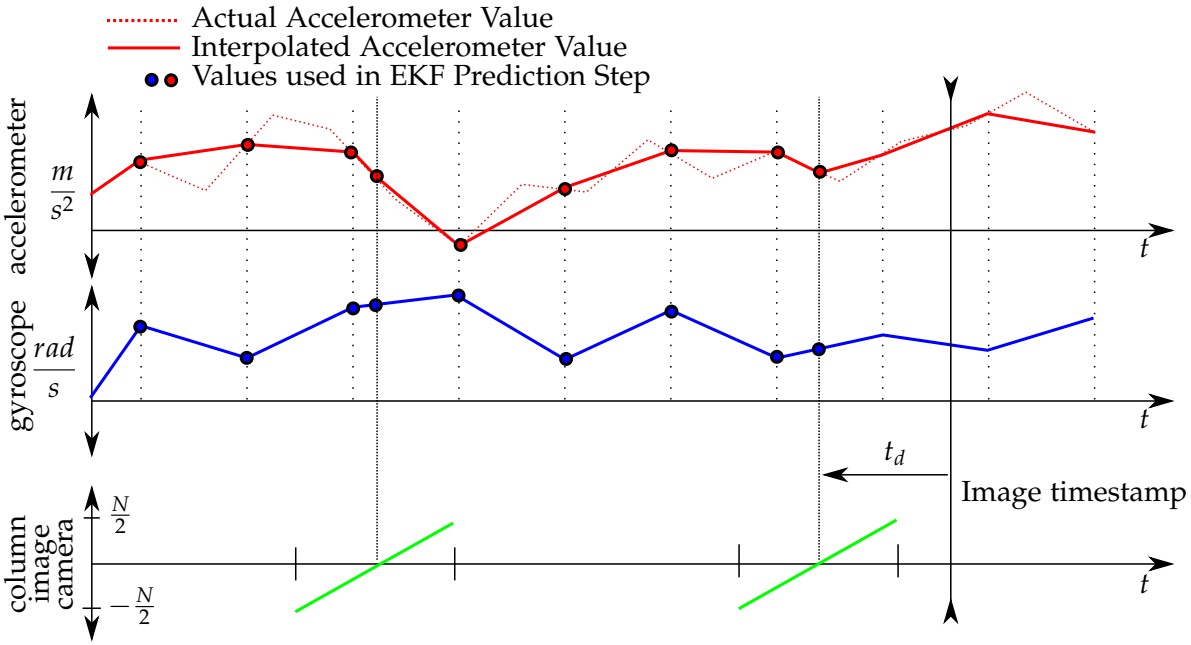


Figure 7.1. IMU data is interpolated and used in the EKF prediction step up until the time of the center column of the image, $t + t_d$.

Although the accelerometer and gyroscope readings were requested at 100Hz, in practice, they arrive at slightly different frequencies and need to be interpolated. In our imple-

mentation, IMU data is buffered at the slower of the two frequencies, using interpolated values for the higher frequency input (Figure 7.1). When an image with timestamp t is available, IMU data is interpolated up to time $t + \hat{t}_d$ and multiple EKF prediction steps occur. Image processing then takes over by tracking and matching features, and finally an update takes place.

8. Results

Here we present the results of the MSCKF both in simulation and with data collected from the Apple iPhone 4S and the iPad Retina (which we simply refer to as the iPhone and the iPad, respectively). Simulations were generated as described in chapter 7 by choosing a series of waypoints through which a virtual camera moves while tracking virtual features. When presenting the results, we show the error, $\tilde{\mathbf{x}}$ as compared to $\pm 3\sigma$, where σ is the square root of the corresponding diagonal entry in the covariance matrix $\mathbf{\Sigma}$. If the error is within 3σ, it is within the 99.7 percentile of the gaussian distribution, as seen in Figure 8.1 below. When errors are within this range, it is clear that the filter is correctly reporting uncertainty and is consistent.

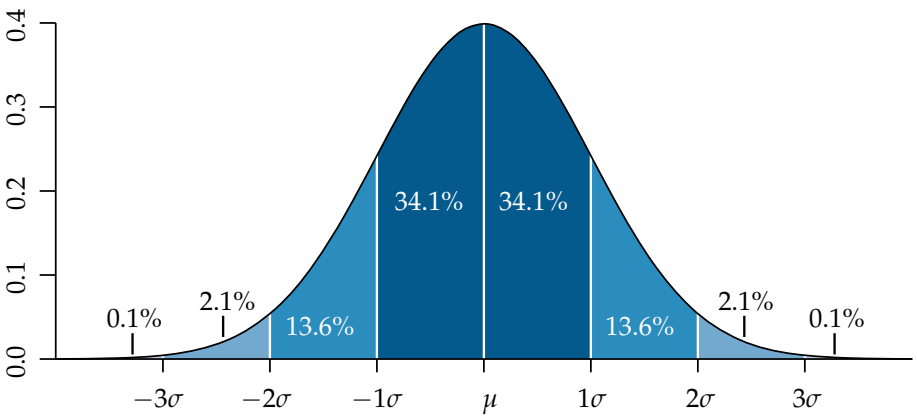


Figure 8.1. The normal probability density function. The probability of a normal random variable being within $\pm 3\sigma$ from the mean is 99.7 percent.

8.1. Speed vs. Quality

One of the goals of this thesis is real-time performance on the mobile device. Unfortunately, due to time constraints, a system running on a smart phone was not realized. However, it is possible for our implementation to run on a single core of a PC at over 30 frames per second with the right parameters. Particularly, the maximum size of the sliding window and the number of features detected in each image govern the speed of the algorithm.

When a feature is lost, all of that feature's measurements are used in an EKF update. Similarly, if a feature is tracked for the maximum amount of images in the sliding window,

it must be considered lost in order to keep the algorithm bounded. When this happens, the latest measurement of the feature is considered a new feature and all the previous ones are used in an update. Unfortunately, this could be throwing away important information, but some trade off must occur, especially considering that a feature could theoretically be tracked indefinitely. Figure 8.2 shows that using a maximum window size of around 16 typically captures over 99% of features in their entirety.

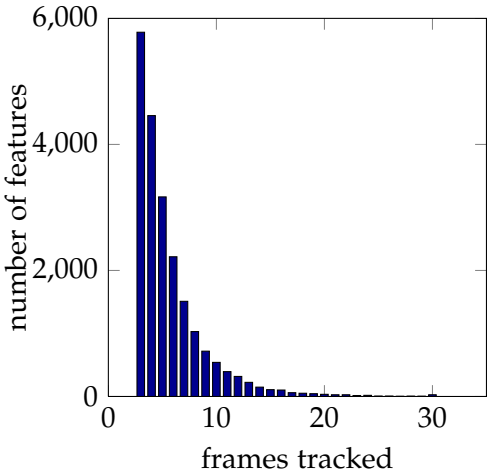


Figure 8.2. The majority of features are tracked for only a few frames with a few rare long feature tracks. The small increase at 30 is because it was the maximum window size in this example.

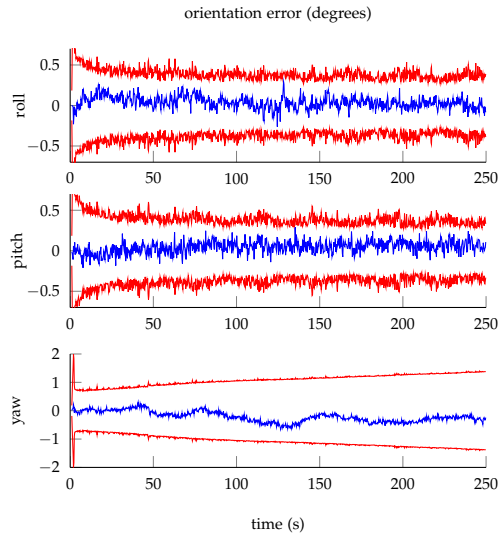
The other important factor in runtime speed is the number of features to be detected and matched in each frame. In our experiments, detecting more than 300 features per frame resulted in no significant improvement.

8.2. Simulation Tests

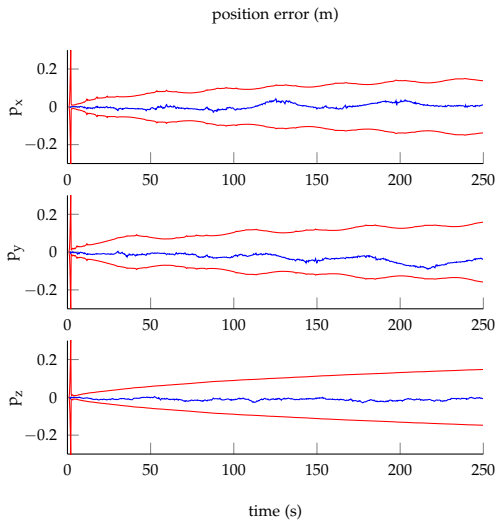
Simulations were performed using waypoints and curves as described in Chapter 7. The waypoints were arranged in a circle with normally distributed random deviations in height. The yaw orientation was facing in the direction of travel in the x-y plane while there were small random deviations in roll and pitch. The results shown in Figure 8.3 reveal that the error remains bounded by $\pm 3\sigma$, meaning that the filter is consistent. Because the velocity and the roll and pitch are observable, the error bounds (red) do not increase indefinitely. On the other hand, those for the position and yaw do continuously increase because they are not observable [19].

8.2.1. Calibration Results

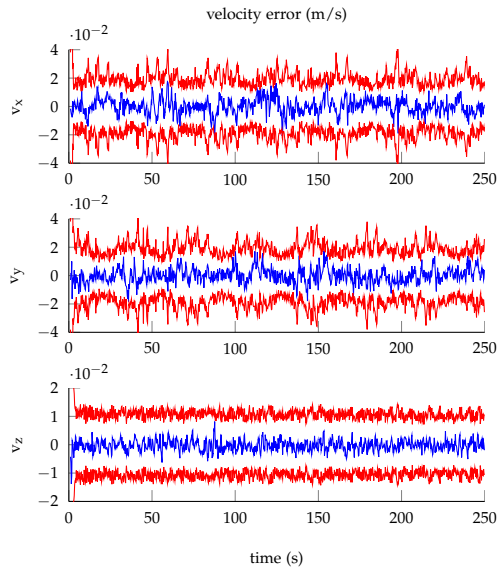
The ground truth calibration parameters were set to initial values similar to those of a real world device. In each trial, a small amount of random error was introduced for each parameter. While the initial state estimate was the same in each trial, the ground



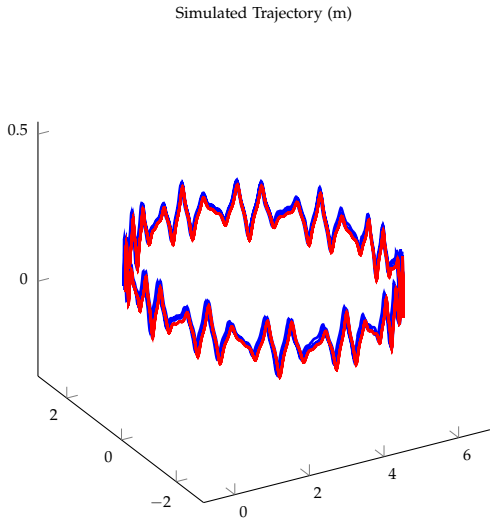
(a) Orientation Error



(b) Position Error



(c) Velocity Error



(d) Simulated Trajectory

Figure 8.3. The ground truth simulated trajectory (d) is red, while the result of the filter is blue. (a)-(c) show plots of the body state error (blue) as compared to $\pm 3\sigma$ (red), where σ is the square root of the corresponding diagonal entry in the covariance matrix.

Parameter	Units	Initial uncertainty (σ_0)	Final error ($\tilde{x}$)	Final uncertainty (σ)
$\mathbf{b}_g$	rad/s	0.001	0.3192E-3	0.3796E-3
$\mathbf{b}_a$	m/s^2	0.01	0.7782E-4	0.1824E-3
$\mathbf{T}_g$	unitless	0.02	0.0014	0.0018
$\mathbf{T}_s$	$\frac{\text{s}\cdot\text{rad}}{\text{m}}$	0.001	0.3888E-5	0.1434E-4
$\mathbf{T}_a$	unitless	0.02	0.0011	0.3768E-3
$\mathbf{f}$	pixels	9	0.4783	0.2074
$\mathbf{o}$	pixels	11	0.6335	0.2887
$\mathbf{k}$	unitless	0.04	0.0134	0.0093
$\mathbf{t}$	unitless	0.02	0.0005	0.0001
t_r	s	0.01	0.3663E-4	0.1041E-3
t_d	s	0.01	0.9841E-4	0.4222E-4
${}^B\mathbf{p}_C$	m	0.001	0.0012	0.8726E-3

Figure 8.4. Results for simulation parameter calibration show that the uncertainty decreases for all values. For values with multiple dimensions, the dimension with the highest error is shown. Notice that in all cases, the error is within the bounds of 3σ.

truth parameters varied randomly with standard deviations listed in Table 8.4 equal to the initial uncertainty.

Results with local instead of global orientation error representation.

8.3. Real-World Experiments

Video and IMU data was recorded using a custom app for the iPhone written in Objective-C and C++. Videos were effectively recorded at 20 frames per second, and IMU data at approximately 100Hz. In order to determine the noise values for the IMU, we followed the procedure described in Chapter 4, but we found that added noise from holding the device in hand must be taken into account. The result is a slightly higher noise value than what would be seen by a robotic system without the handheld shaking. For the continuous random walk values, we applied grid search to find the parameters that worked best in practice, giving good results. Values higher or lower than these by a single order of magnitude failed to track the biases and diverged. This analysis of the IMU determined the continuous noise values to be as follows.

$$\sigma_g = 0.001 \frac{\text{rad}}{\text{s}} \tag{8.1}$$

$$\sigma_a = 0.008 \frac{\text{m}}{\text{s}^2} \tag{8.2}$$

$$\sigma_{wg} = 0.0001 \frac{\text{rad}}{\text{s}^2} \tag{8.3}$$

$$\sigma_{wa} = 0.0001 \frac{\text{m}}{\text{s}^3} \tag{8.4}$$

Initial calibration parameter values were set as follows. The accelerometer bias and g-sensitivity matrix, $\mathbf{T}_s$, were set to all zeros, the matrices $\mathbf{T}_a$ and $\mathbf{T}_g$ were set to the identity matrix, and the gyroscope bias was set to the initial gyroscope reading while the device was not in motion. The camera-IMU position offset, ${}^B\mathbf{p}_C$, the camera-IMU time offset and camera read time (t_d, t_r), as well as the camera distortion parameters $(k_1, k_2, k_3, t_1, t_2)$ were all set to zero. The principal point, $\mathbf{o}$, was set to the center of the image, $(240, 320)$ while the focal lengths, $\mathbf{f}$, were set to 600 pixels, which is a near estimate of different calibration results. As mentioned in Chapter 7, the iPhone's focal length is different in each recording and cannot be set in the software. Initial uncertainties for all of the parameters can be seen in Table 8.4.

8.3.1. Scale Drift

As mentioned in the introduction, monocular visual odometry methods often suffer from a lack of scale or a drift in scale from an initial estimate. In order to judge the scale estimation performance of the MSCKF, we performed an experiment in which the handheld device was moved along a one meter edge in three dimensions many times over a long period (see Figure 8.6). Although the absolute position tends to drift, because it is unobservable, the results in Figure 8.5 show that the MSCKF consistently and correctly measures a motion of roughly one meter.

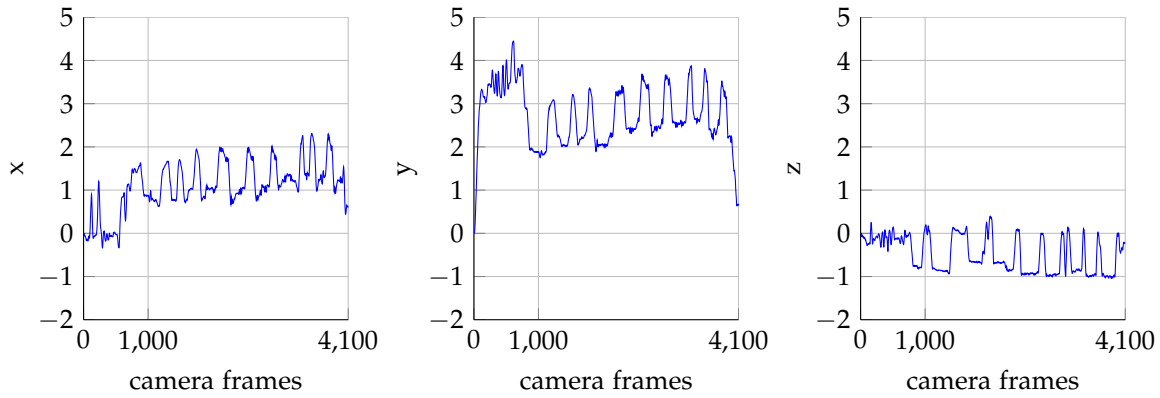


Figure 8.5. In this experiment, the iPhone 4S was moved repeatedly along the edges of a one meter squared horizontal board and a one meter vertical measuring stick. The results show an consistently accurate scale, despite small amounts of position drift. Note that the first 1000 frames of the trial is made up of exciting motions for parameter calibration.

8.3.2. Device Calibration Results

Because ground truth data was unavailable for the iPhone calibration parameters, we present the results over many trials to show that they are consistent. The results of 10 different calibrations, all with initial parameters as described above, are averaged to find the mean values and their standard deviations.

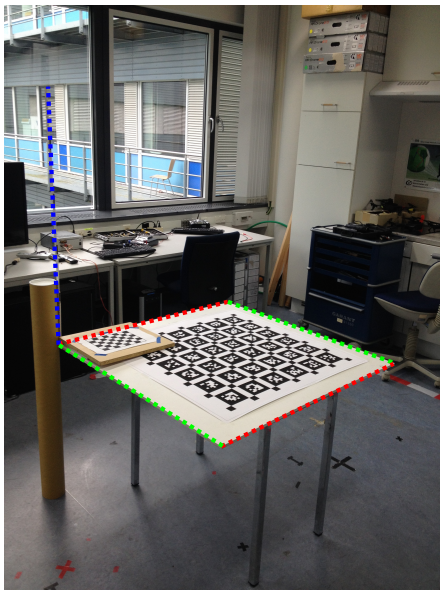


Figure 8.6. The setup for the scale experiment. The center board is one meter square and the vertical measuring stick is one meter tall.

Camera-IMU Spatial Calibration

The positional offset between the IMU and camera of the iPhone yields results similar to those expected when examining images of the internal components (Figure 8.7).

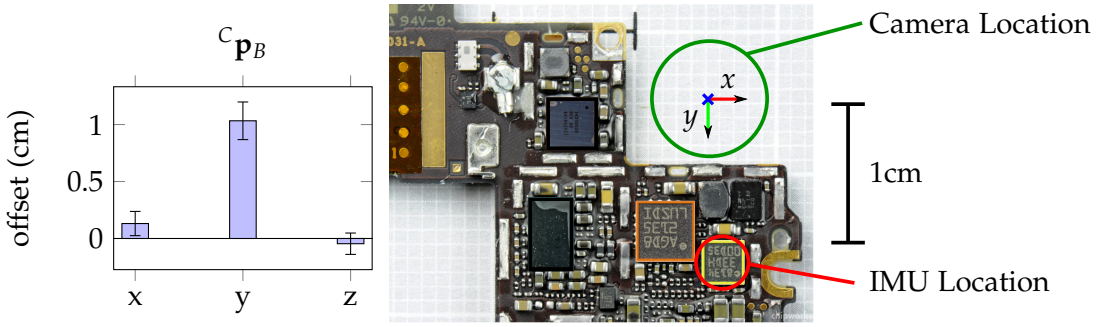
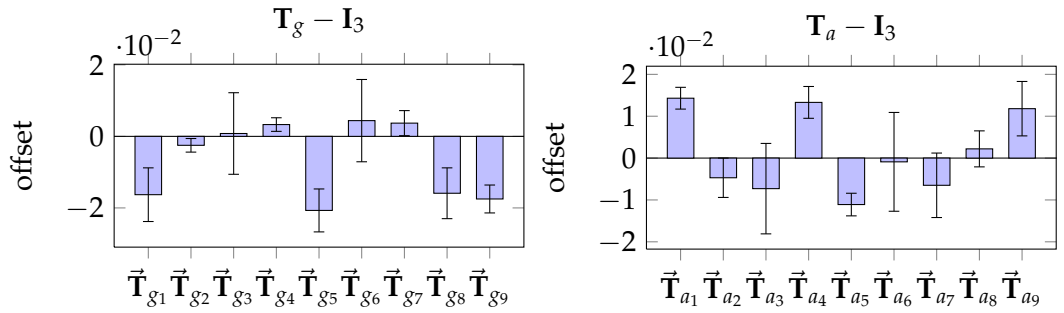


Figure 8.7. The iPhone Camera-IMU Offset results are consistent with images of the phone's internals. While the gyroscope (orange) and the accelerometer (yellow) both comprise the IMU, the location of the body frame is centered on the accelerometer.

IMU Calibration

The gyroscope and accelerometer scale and misalignment matrices show minimal misalignment but a small amount of scale distortion. This can be seen by observing that entries 1, 5, and 9 in the linearized matrix $\vec{\mathbf{T}}_g$ are the diagonal of the 3×3 matrix $\mathbf{T}_g$. The largest scale effects are as much as 2%, having some significant impact on long term

trajectories. It is important to note that the high standard deviations on these values imply no statistical significance[1] for many, but not all, of the values.



The g-sensitivity matrix shows no statistically significant deviation from zero. This implies that either there is no significant g-sensitivity or, more likely, the iPhone comes factory calibrated.

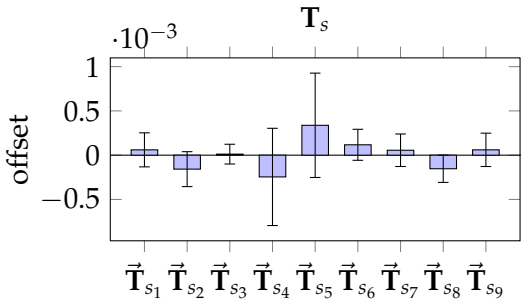


Figure 8.8. The iPhone g-sensitivity matrix shows no significant deviation from zero, implying that no calibration is necessary.

IMU calibration results for the iPad reveal even better factory calibration. In fact, the two main differences between the iPhone and iPad are the available resources and speed at which they heat up. While recording video data, the iPhone becomes very warm to the touch within just one minute, while the iPad can remain cool for a much longer period. Our results suggest that the iPhone is in fact as well calibrated as the iPad. Unmodeled thermal effects on the IMU are the likely cause of the scale factors found in the iPhone $\mathbf{T}_a$ and $\mathbf{T}_g$ matrices. Without access to internal temperature readings, these findings will have to remain speculation.

Temporal Calibration

Timing calibration reveals that the timestamp of each image on the iPhone has a high variance when compared to the actual time of the center image column $t + t_d$. This could be caused by non-deterministic delays in the iPhone software or hardware, such as limited resources. In stark contrast, the iPad has very little variance in t_d and it is clear that the

[1] Using a p-value of 0.05, if the null-hypothesis is included in the 2σ range, the result is not statistically significant.

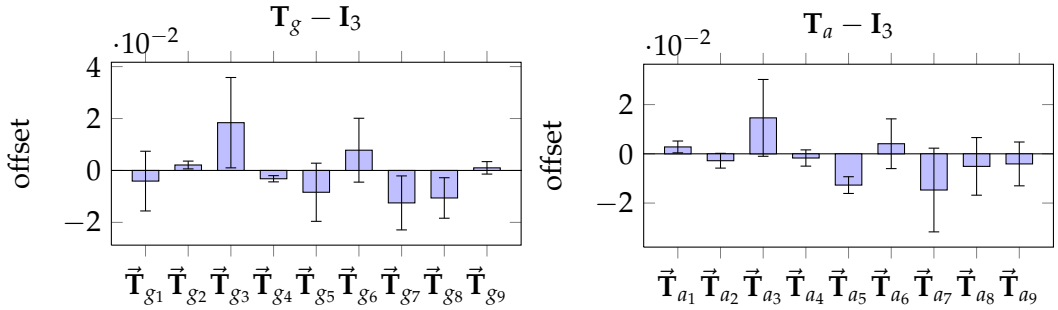
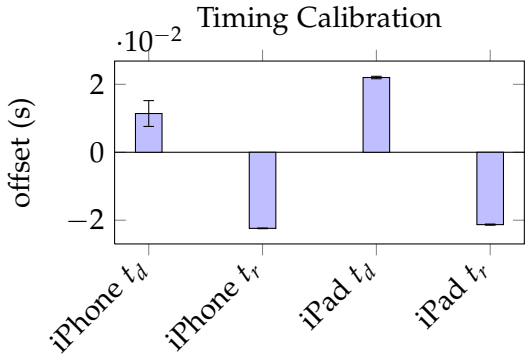


Figure 8.9. IMU Calibration results for the iPad Retina with 1σ error bars after 10 trials. Results indicate that there is no statistically significant calibration required.

image is time-stamped well before being read by the sensor. The image read time, for both the iPhone and iPad, was determined with extreme precision, giving a standard deviation of 0.0001 seconds.



	t (σ)
iPhone t_d	0.0114s (0.0038s)
iPad t_d	0.0220s (0.00036s)
iPhone t_r	-0.0224s (0.0001s)
iPad t_r	-0.0213s (0.00017s)

Figure 8.10. The iPhone 4S appears to have a less deterministic temporal offset than the newer iPad. This may be an internal software issue or a result of less available hardware resources during recording.

8.3.3. Camera Calibration Results

Because iOS prevents setting of the focal length programatically, we locked the camera settings to the same values and recorded 10 different calibration trials. Results show consistent camera calibration results for the iPad (Figure 8.11).

8.3.4. Experimental Results

Long trajectories in the range of about 7-10 minutes have shown good results for both the iPhone and the iPad. Because ground truth data was not available, we can only assess the final positional error by ending the trajectory at the starting point or superimposing it on a map for scale. GPS data gave poor results which we were unable to use in any meaningful way (Figure 8.12).

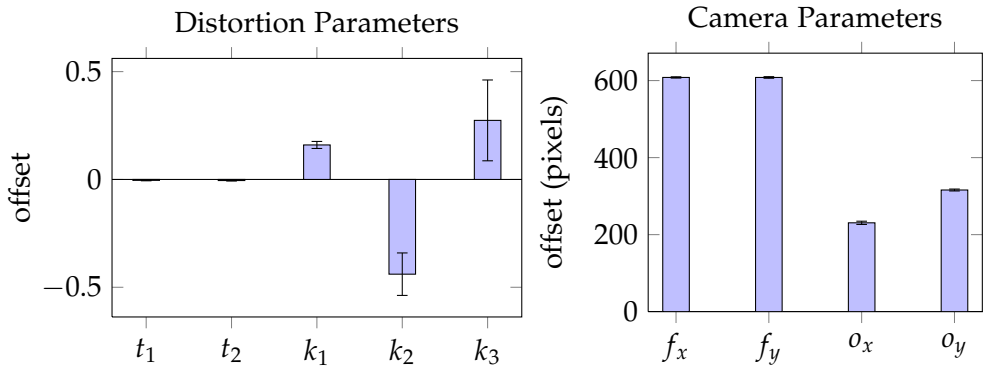


Figure 8.11. iPad camera calibration results after 10 trials. Focal length, exposure and white balance were locked in each trial at the same values.

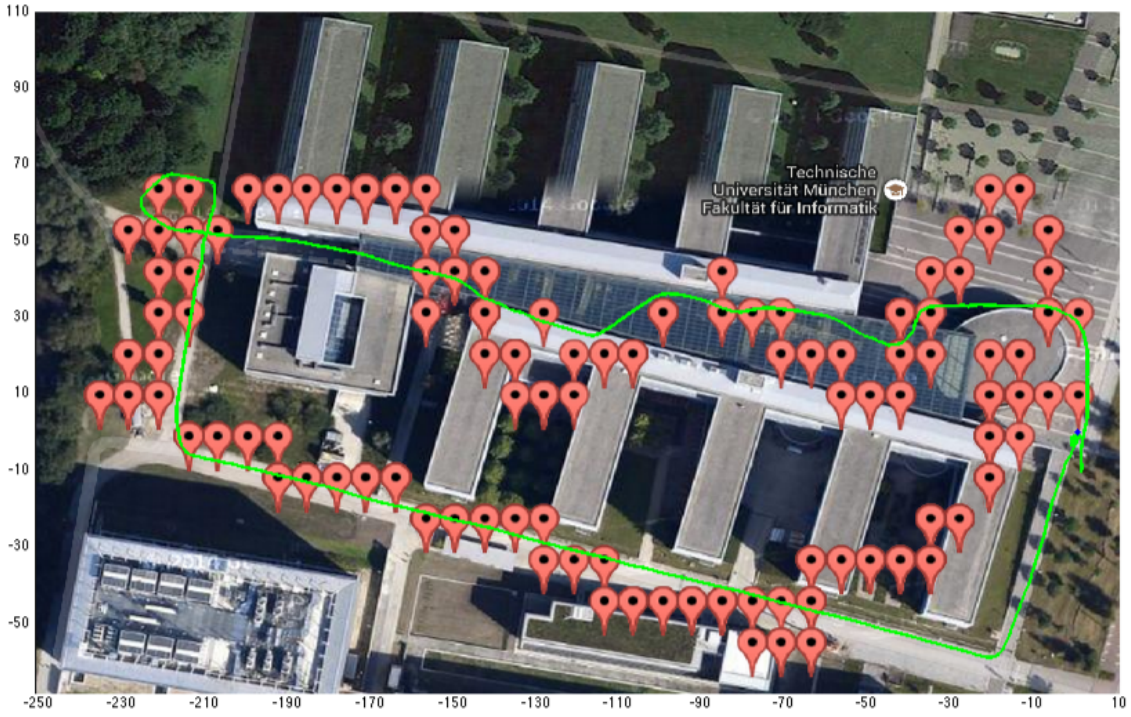


Figure 8.12. Results of a trial taken with the handheld iPhone. The highly inaccurate GPS is included for reference. Although no ground truth data is available, the actual final position was the same as the starting position. The initial starting position and orientation (yaw) were aligned with the map manually.

Figure 8.13. Sample images from the iPhone 4S taken in a long trajectory. The recording took place at the Technical University of Munich in Garching, Germany.

The final positional error in this trajectory was 20 meters, or 1% of the total distance traveled. The error is larger than the reported translational uncertainty (3σ) of 8 meters. This inconsistency could be caused by some unmodeled sources of error, such as IMU temperature or the previously mentioned nondeterministic nature of t_d.

Long distance results for the iPad gave better results. We recorded a long 7.4 minute trajectory both outdoors and indoors traveling on four floors and using two staircases. The video was recorded at 10Hz and the IMU at 100Hz. Final error was 3.1 meters (0.4% of the distance traveled) with reported uncertainty of 7.6 meters. These good results are in spite of sparse feature regions, pedestrians, automatic doors, and severe white balance issues. The entire 7.4 minute trajectory took 2.7 minutes to process on a Retina Macbook Pro and the resulting video can be seen on Youtube [46].

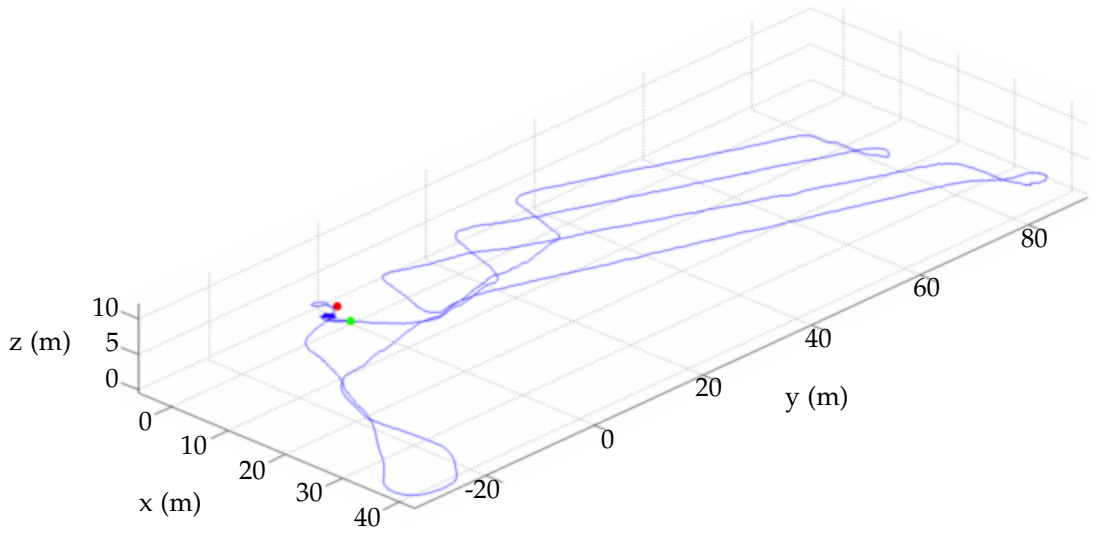


Figure 8.14. This trajectory, recorded with the iPad, went along 4 floors of the Mathematics and Informatics building of TUM.

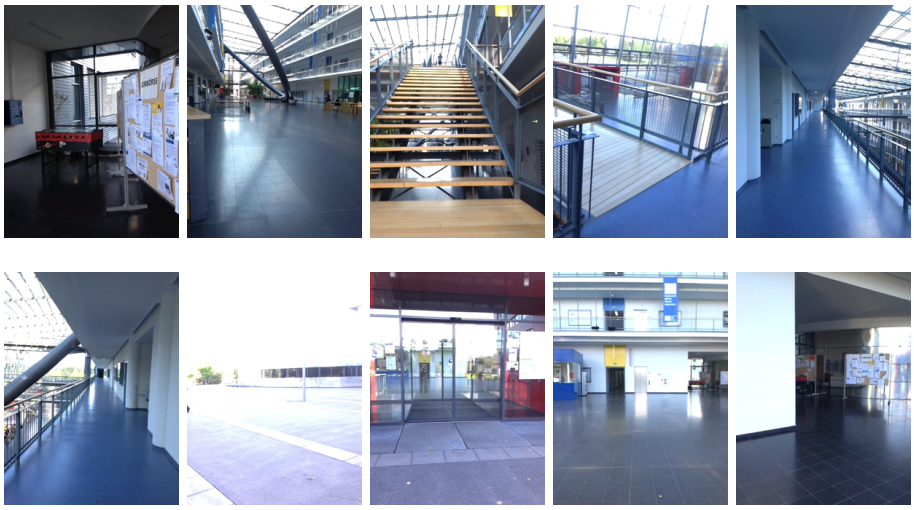


Figure 8.15. Sample images from the iPad trajectory. The floors and white walls are almost entirely featureless. Because the white balance was locked to prevent miscalibration of the camera, images taken outside are poor quality.

9. Conclusion

In this thesis, we presented the results of the Multi-State Constraint Kalman Filter for visual inertial odometry with online self calibration. The MSCKF has been known to out perform the traditional EKF SLAM in both accuracy and runtime. The high accuracy is a result of two strategies: Removing feature outliers before they are used in the EKF, and waiting until all the feature measurements are available before estimating the feature depth. The runtime speed of the MSCKF is constant and faster than EKF SLAM because features are not added to the state vector. Instead, a sliding window of camera poses is maintained. The disadvantage of this in our implementation is the requirement of constant motion.

Using detailed models of the IMU, a rolling shutter camera with lense distortion, and camera-IMU spacial and temporal offsets, we show that joint online calibration of these parameters can improve odometry performance. Simulation tests demonstrate that all of the calibration parameters appear to be observable with sufficiently exciting motion. Real world experiments show that the Apple iPhone 4S and the iPad 3 Retina both come with well calibrated IMUs and only suffer from minor scale offsets. We speculate that the temperature of the IMU also has a noticable effect. The camera-IMU spacial and temporal calibration parameters, as well as the rolling shutter read time and camera distortion parameters all converged to plausable and consistent values over multiple trials.

On the devices tested, the MSKCF successfully tracked motion over long periods with errors lower than 1% of the distance traveled. In some cases, we have shown the algorithm to be robust to rapid motion and rotation. The addition of the IMU to visual odometry has proven to be useful not only for scale estimation, but also for robust tracking of highly exciting motion. Although we did not yet attempt to run the algorithm on the mobile device itself, we have shown that it can work in real-time on a single core CPU of a standard laptop computer. Overall, our results indicate that the MSCKF is an accurate and robust tool for real-time pose estimation using a hand held device.

Appendix

A. Quaternions

A.1. Quaternion Definition

A quaternion is defined as follows

$$\mathbf{q} = \mathbf{i}q_1 + \mathbf{j}q_2 + \mathbf{k}q_3 + q_4 \tag{A.1}$$

The numbers $\mathbf{i}$, $\mathbf{j}$, and $\mathbf{k}$ are *hyperimaginary* numbers.

$$\mathbf{i}^2 = -1 \qquad \mathbf{j}^2 = -1 \qquad \mathbf{k}^2 = -1 \tag{A.2}$$

$$-\mathbf{ij} = \mathbf{ji} = \mathbf{k} \qquad -\mathbf{jk} = \mathbf{kj} = \mathbf{i} \qquad -\mathbf{ki} = \mathbf{ik} = \mathbf{j} \tag{A.3}$$

This convention is different form the original quaternion convention from Hamilton. The main motivation for this convention is the resulting natural and intuitive multiplication order for quaternions.

$${}^B_G\bar{\mathbf{q}} = {}^B_A\bar{\mathbf{q}} \otimes {}^A_G\bar{\mathbf{q}} \tag{A.4}$$

The $\otimes$ operator describes quaternion multiplication.

$$\bar{\mathbf{q}} \otimes \bar{\mathbf{p}} = (q_4 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k})(p_4 + p_1\mathbf{i} + p_2\mathbf{j} + p_3\mathbf{k}) \tag{A.5}$$

$$= \begin{bmatrix} q_4p_1 + q_3p_2 - q_2p_3 + q_1p_4 \\ -q_3p_1 + q_4p_2 + q_1p_3 + q_2p_4 \\ q_2p_1 - q_1p_2 + q_4p_3 + q_3p_4 \\ -q_1p_1 - q_2p_2 - q_3p_3 + q_4p_4 \end{bmatrix} \tag{A.6}$$

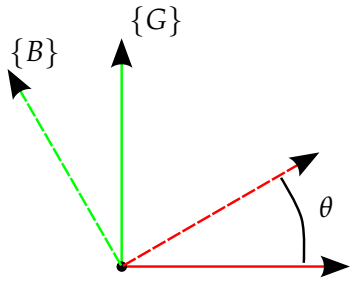
The function $\mathbf{C}(\bar{\mathbf{q}})$ transforms the unit quaternion into a rotation matrix.

$${}^B_G\mathbf{C}(\bar{\mathbf{q}}) = \begin{bmatrix} q_1^2 - q_2^2 - q_3^2 + q_4^2 & 2(q_1q_2 + q_3q_4) & 2(q_1q_3 - q_2q_4) \\ 2(q_1q_2 - q_3q_4) & -q_1^2 + q_2^2 - q_3^2 + q_4^2 & 2(q_2q_3 + q_1q_4) \\ 2(q_1q_3 + q_2q_4) & 2(q_2q_3 - q_1q_4) & -q_1^2 - q_2^2 + q_3^2 + q_4^2 \end{bmatrix} \tag{A.7}$$

A.1.1. Axis Angle Representation

A unit quaternion can be represented by a three dimensional unit vector, $\mathbf{k}$ and an angle of rotation, θ around this vector.

$$\bar{\mathbf{q}} = \begin{bmatrix} k_x \sin(\theta/2) \\ k_y \sin(\theta/2) \\ k_z \sin(\theta/2) \\ \cos(\theta/2) \end{bmatrix} \tag{A.8}$$



$$ {}^{B}_{G}\mathbf{R} = \begin{bmatrix} cos(\theta) & sin(\theta) & 0 \\ -sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{B}_{G}\mathbf{q} = \begin{bmatrix} 0 \\ 0 \\ sin(\theta/2) \\ cos(\theta/2) \end{bmatrix} $$

Figure A.1. A simple example of the rotation and quaternion convention from [39]

A.2. Small Rotations

Consider a very small quaternion $\delta\bar{\mathbf{q}}$. If we use the axis angle represenation, this can be written as the following

$$ \delta\bar{\mathbf{q}} = \begin{bmatrix} \mathbf{k}\ \sin(\delta\theta/2) \\ \cos(\delta\theta/2) \end{bmatrix} \tag{A.9} $$

$$ \approx \begin{bmatrix} \frac{1}{2}\delta\boldsymbol{\theta} \\ 1 \end{bmatrix} \tag{A.10} $$

In the above expression, the bold term $\delta\boldsymbol{\theta}$ is the minimal 3DoF euler angle error. The approximation results in a convenient approximation for the associated rotation matrix.

$$ {}^{B}_{G}\mathbf{C}(\delta\bar{\mathbf{q}}) \approx \mathbf{I}_3 - \lfloor\delta\boldsymbol{\theta}\times\rfloor \tag{A.11} $$

B. Error-State Transition Matrix

In this section we derive the full error-state transition matrix including bias terms and IMU calibration terms. We base our derivation on that of the error-state transition matrix without the IMU calibration terms found in [23]. The covariance propagation during the prediction step from timestep t_ℓ to $t_{\ell+1}$ has the following sparse form

$$\mathbf{\Sigma}_{\ell+1} = \begin{bmatrix} \mathbf{\Phi}_{B_\ell} & \mathbf{\Gamma}_{imu_\ell} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix} \mathbf{\Sigma}_\ell \begin{bmatrix} \mathbf{\Phi}_{B_\ell} & \mathbf{\Gamma}_{imu_\ell} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix}^T + \begin{bmatrix} \mathbf{Q}_\ell & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix} \tag{B.1}$$

Closer inspection of $\mathbf{\Phi}$ and $\mathbf{\Gamma}$ reveal that the first 9 rows contain all the important information.

$$\mathbf{\Phi}_{B_\ell} = \begin{bmatrix} \mathbf{I}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{\Phi}_{qb_g} & \mathbf{\Phi}_{qb_a} \\ \mathbf{\Phi}_{pq} & \mathbf{I}_3 & \mathbf{I}_3\Delta t & \mathbf{\Phi}_{pb_g} & \mathbf{\Phi}_{pb_a} \\ \mathbf{\Phi}_{vq} & \mathbf{0}_3 & \mathbf{I}_3 & \mathbf{\Phi}_{vb_g} & \mathbf{\Phi}_{vb_a} \\ \mathbf{0_3} & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{I}_3 & \mathbf{0}_3 \\ \mathbf{0_3} & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{I}_3 \end{bmatrix} \qquad \mathbf{\Gamma}_{imu_\ell} = \begin{bmatrix} \mathbf{\Gamma}_{q\vec{\mathbf{T}}_g} & \mathbf{\Gamma}_{q\vec{\mathbf{T}}_s} & \mathbf{\Gamma}_{q\vec{\mathbf{T}}_a} \\ \mathbf{\Gamma}_{p\vec{\mathbf{T}}_g} & \mathbf{\Gamma}_{p\vec{\mathbf{T}}_s} & \mathbf{\Gamma}_{p\vec{\mathbf{T}}_a} \\ \mathbf{\Gamma}_{v\vec{\mathbf{T}}_g} & \mathbf{\Gamma}_{v\vec{\mathbf{T}}_s} & \mathbf{\Gamma}_{v\vec{\mathbf{T}}_a} \\ \mathbf{0_{3\times 9}} & \mathbf{0_{3\times 9}} & \mathbf{0_{3\times 9}} \\ \mathbf{0_{3\times 9}} & \mathbf{0_{3\times 9}} & \mathbf{0_{3\times 9}} \end{bmatrix} \tag{B.2}$$

We can write the relevant terms of the error transition in three equations. These describe how the state error evolves during propagation. In order to improve readability, we write ${}^{B_{\ell|\ell}}_{G}\hat{\mathbf{R}}$ as $\hat{\mathbf{R}}_\ell$ and ${}^{B_{\ell+1|\ell}}_{G}\hat{\mathbf{R}}$ as $\hat{\mathbf{R}}_{\ell+1}$. We also write ${}^G\tilde{\mathbf{p}}_{B_\ell}$ as ${}^G\tilde{\mathbf{p}}_\ell$.

$${}^G\tilde{\boldsymbol{\theta}}_{\ell+1} = {}^G\tilde{\boldsymbol{\theta}}_\ell + \mathbf{\Phi}_{qb_g}\tilde{\mathbf{b}}_{g_\ell} + \mathbf{\Phi}_{qb_a}\tilde{\mathbf{b}}_{a_\ell} + \mathbf{\Gamma}_{q\vec{\mathbf{T}}_g}\tilde{\mathbf{T}}_{g_\ell} + \mathbf{\Gamma}_{q\vec{\mathbf{T}}_s}\tilde{\mathbf{T}}_{s_\ell} + \mathbf{\Gamma}_{q\vec{\mathbf{T}}_a}\tilde{\mathbf{T}}_{a_\ell} \tag{B.3}$$

$${}^G\tilde{\mathbf{p}}_{\ell+1} = \mathbf{\Phi}_{pq}{}^G\tilde{\boldsymbol{\theta}}_\ell + {}^G\tilde{\mathbf{p}}_\ell + {}^G\tilde{\mathbf{v}}_\ell\Delta t + \mathbf{\Phi}_{pb_g}\tilde{\mathbf{b}}_{g_\ell} + \mathbf{\Phi}_{pb_a}\tilde{\mathbf{b}}_{a_\ell} + \mathbf{\Gamma}_{p\vec{\mathbf{T}}_g}\tilde{\mathbf{T}}_{g_\ell} + \mathbf{\Gamma}_{p\vec{\mathbf{T}}_s}\tilde{\mathbf{T}}_{s_\ell} + \mathbf{\Gamma}_{p\vec{\mathbf{T}}_a}\tilde{\mathbf{T}}_{a_\ell} \tag{B.4}$$

$${}^G\tilde{\mathbf{v}}_{\ell+1} = \mathbf{\Phi}_{vq}{}^G\tilde{\boldsymbol{\theta}}_\ell + {}^G\tilde{\mathbf{v}}_\ell + \mathbf{\Phi}_{vb_g}\tilde{\mathbf{b}}_{g_\ell} + \mathbf{\Phi}_{vb_a}\tilde{\mathbf{b}}_{a_\ell} + \mathbf{\Gamma}_{v\vec{\mathbf{T}}_g}\tilde{\mathbf{T}}_{g_\ell} + \mathbf{\Gamma}_{v\vec{\mathbf{T}}_s}\tilde{\mathbf{T}}_{s_\ell} + \mathbf{\Gamma}_{v\vec{\mathbf{T}}_a}\tilde{\mathbf{T}}_{a_\ell} \tag{B.5}$$

To find these error terms, we begin with equation (6.38) which we write here again.

$$\begin{bmatrix} {}^G\tilde{\boldsymbol{\theta}}_{\ell+1} \\ {}^G\tilde{\mathbf{p}}_{\ell+1} \\ {}^G\tilde{\mathbf{v}}_{\ell+1} \end{bmatrix} = \begin{bmatrix} \mathbf{I}_3 & \mathbf{0}_3 & \mathbf{0}_3 \\ \mathbf{\Phi}_{pq} & \mathbf{I}_3 & \mathbf{I}_3\Delta t \\ \mathbf{\Phi}_{vq} & \mathbf{0}_3 & \mathbf{I}_3 \end{bmatrix} \begin{bmatrix} {}^G\tilde{\boldsymbol{\theta}}_\ell \\ {}^G\tilde{\mathbf{p}}_\ell \\ {}^G\tilde{\mathbf{v}}_\ell \end{bmatrix} + \begin{bmatrix} \hat{\mathbf{R}}_\ell^T\,{}^{B_\ell}\tilde{\boldsymbol{\theta}}_{\Delta\ell} \\ \hat{\mathbf{R}}_\ell^T\,\tilde{\mathbf{y}}_{\Delta\ell} \\ \hat{\mathbf{R}}_\ell^T\,\tilde{\mathbf{s}}_{\Delta\ell} \end{bmatrix} \tag{B.6}$$

B.1. Orientation Components

From [23] we have the definition of ${}^G\tilde{\boldsymbol{\theta}}_{\Delta\ell}$.

$${}^G\tilde{\boldsymbol{\theta}}_{\Delta\ell} \simeq \hat{\mathbf{R}}_\ell^T \int_{t_\ell}^{t_{\ell+1}} {}^{B_\ell}_{B_\tau}\mathbf{R}\,{}^{B_\tau}\tilde{\boldsymbol{\omega}}\,d\tau \tag{B.7}$$

The true rotational velocity of the body, ${}^{B}\boldsymbol{\omega}$, in terms of the measured value and the true IMU calibration parameters, is as follows.

$$ {}^{B}\boldsymbol{\omega} = \mathbf{T}_g^{-1} \left({}^{B}\boldsymbol{\omega}_m - \mathbf{T}_s \, {}^{B}\mathbf{a} - \mathbf{b}_g - \mathbf{n}_g \right) \tag{B.8} $$

Here $\mathbf{b}_g$ and $\mathbf{n}_g$ are the gyroscope bias and noise, respectively, and ${}^{B}\mathbf{a}$ is the acceleration in the body frame.

$$ {}^{B}\mathbf{a} = \mathbf{T}_a^{-1} \left({}^{B}\mathbf{a}_m - \mathbf{b}_a - \mathbf{n}_a \right) \tag{B.9} $$

We can write the rotational velocity error in terms of the state error using the Taylor expansion.

$$ {}^{B}\tilde{\boldsymbol{\omega}} = {}^{B}\boldsymbol{\omega} - {}^{B}\hat{\boldsymbol{\omega}} \tag{B.10} $$

$$ {}^{B}\boldsymbol{\omega} \simeq {}^{B}\hat{\boldsymbol{\omega}} + \frac{\partial {}^{B}\boldsymbol{\omega}}{\partial \mathbf{x}} \tilde{\mathbf{x}} \tag{B.11} $$

$$ {}^{B}\tilde{\boldsymbol{\omega}} \simeq \frac{\partial {}^{B}\boldsymbol{\omega}}{\partial \mathbf{x}} \tilde{\mathbf{x}} \tag{B.12} $$

Substituting this into equation (B.7) gives us

$$ {}^{G}\tilde{\boldsymbol{\theta}}_{\Delta\ell} \simeq \hat{\mathbf{R}}_{\ell+1} \hat{\mathbf{R}}_{\ell}^{T} \int_{t_\ell}^{t_{\ell+1}} {}^{B_\ell}_{B_\tau}\mathbf{R} \, \frac{\partial {}^{B}\boldsymbol{\omega}(\tau)}{\partial \mathbf{x}} \tilde{\mathbf{x}} \, d\tau \tag{B.13} $$

We proceed by calculating the partial derivatives of ${}^{B}\boldsymbol{\omega}$ with respect to the state vector. Because the three $\mathbf{T}$ matrices are stored in vectorized format, they must be differentiated element-wise. For example, the first column of the 3×9 matrix $\boldsymbol{\Gamma}_{q\vec{\mathbf{T}}_g}$ will use the first element of $\frac{\partial \mathbf{T}_g}{\partial \vec{\mathbf{T}}_g}$ shown here.

$$ \frac{\partial \mathbf{T}_g}{\partial \vec{\mathbf{T}}_g} = \begin{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} & \dots & \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{bmatrix} \tag{B.14} $$

These terms therefore must be evaluated separately for each of the 9 columns. This is because there is no matrix multiplication operation that can transform a 9×1 vector into a 3×3 matrix.

$$ \frac{\partial {}^{B}\boldsymbol{\omega}}{\partial \mathbf{b}_g} = -\mathbf{T}_g^{-1} \tag{B.15} $$

$$ \frac{\partial {}^{B}\boldsymbol{\omega}}{\partial \mathbf{b}_a} = \mathbf{T}_g^{-1} \, \mathbf{T}_s \, \mathbf{T}_a^{-1} \tag{B.16} $$

$$ \frac{\partial {}^{B}\boldsymbol{\omega}}{\partial \mathbf{T}_g} = -\mathbf{T}_g^{-1} \, \frac{\partial \mathbf{T}}{\partial \vec{\mathbf{T}}_g} \, \mathbf{T}_g^{-1} \left({}^{B}\boldsymbol{\omega}_m - \mathbf{T}_s \, {}^{B}\mathbf{a} - \mathbf{b}_g \right) \tag{B.17} $$

$$ \frac{\partial {}^{B}\boldsymbol{\omega}}{\partial \mathbf{T}_s} = -\mathbf{T}_g^{-1} \, \frac{\partial \mathbf{T}}{\partial \vec{\mathbf{T}}_s} \, {}^{B}\mathbf{a} \tag{B.18} $$

$$ \frac{\partial {}^{B}\boldsymbol{\omega}}{\partial \mathbf{T}_a} = -\mathbf{T}_g^{-1} \, \mathbf{T}_s \, \mathbf{T}_a^{-1} \frac{\partial \mathbf{T}}{\partial \vec{\mathbf{T}}_a} \, \mathbf{T}_a^{-1} \left({}^{B}\mathbf{a}_m - \mathbf{b}_a \right) \tag{B.19} $$

Combining these with equations (B.13) and (6.38) give the error terms. When using these in practice, we employ trapezoidal integration.

$$\mathbf{\Phi}_{qb_g} = -\hat{\mathbf{R}}_\ell^T \int_{t_\tau}^{t_{\ell+1}} {}^{B_\ell}_{B_\tau}\hat{\mathbf{R}}\, \hat{\mathbf{T}}_g^{-1}\, d\tau \tag{B.20}$$

$$\mathbf{\Phi}_{qb_a} = \hat{\mathbf{R}}_\ell^T \int_{t_\tau}^{t_{\ell+1}} {}^{B_\ell}_{B_\tau}\hat{\mathbf{R}}\, \hat{\mathbf{T}}_g^{-1}\, \hat{\mathbf{T}}_s\, \hat{\mathbf{T}}_a^{-1}\, d\tau \tag{B.21}$$

$$\mathbf{\Gamma}_{q\vec{\mathbf{T}}_g} = -\hat{\mathbf{R}}_\ell^T \int_{t_\tau}^{t_{\ell+1}} {}^{B_\ell}_{B_\tau}\hat{\mathbf{R}}\, \hat{\mathbf{T}}_g^{-1}\, \frac{\partial \mathbf{T}}{\partial \vec{\mathbf{T}}_g}\, \hat{\mathbf{T}}_g^{-1}\, \left({}^B\boldsymbol{\omega}_m(\tau) - \hat{\mathbf{T}}_s\, {}^B\hat{\mathbf{a}} - \hat{\mathbf{b}}_g \right) d\tau \tag{B.22}$$

$$\mathbf{\Gamma}_{q\vec{\mathbf{T}}_s} = -\hat{\mathbf{R}}_\ell^T \int_{t_\tau}^{t_{\ell+1}} {}^{B_\ell}_{B_\tau}\hat{\mathbf{R}}\, \hat{\mathbf{T}}_g^{-1}\, \frac{\partial \mathbf{T}}{\partial \vec{\mathbf{T}}_s}\, {}^B\hat{\mathbf{a}}(\tau)\, d\tau \tag{B.23}$$

$$\mathbf{\Gamma}_{q\vec{\mathbf{T}}_a} = \hat{\mathbf{R}}_\ell^T \int_{t_\tau}^{t_{\ell+1}} {}^{B_\ell}_{B_\tau}\hat{\mathbf{R}}\, \hat{\mathbf{T}}_g^{-1}\, \hat{\mathbf{T}}_s\, \hat{\mathbf{T}}_a^{-1}\, \frac{\partial \mathbf{T}}{\partial \vec{\mathbf{T}}_a}\, \hat{\mathbf{T}}_a^{-1} \left({}^B\mathbf{a}_m(\tau) - \hat{\mathbf{b}}_a \right) d\tau \tag{B.24}$$

$$\tag{B.25}$$

B.2. Velocity Components

We turn now to the velocity components of the error-state transition matrix. Using equation (6.38) we begin with the definition of $\tilde{\mathbf{s}}$.

$$\tilde{\mathbf{s}}_{\Delta\ell} = \int_{t_\ell}^{t_{\ell+1}} {}^{B_\ell}_{B_\tau}\mathbf{R}\, {}^{B_\tau}\mathbf{a}\, d\tau - \int_{t_\ell}^{t_{\ell+1}} {}^{B_\ell}_{B_\tau}\hat{\mathbf{R}}\, {}^{B_\tau}\hat{\mathbf{a}}\, d\tau \tag{B.26}$$

Using the approximation ${}^{B_\ell}_{B_\tau}\mathbf{R} \simeq \left(\mathbf{I}_3 - \lfloor {}^B\tilde{\boldsymbol{\theta}}_{\Delta\tau}\times \rfloor\right) {}^{B_\ell}_{B_\tau}\hat{\mathbf{R}}$, we get the following.

$$\tilde{\mathbf{s}}_{\Delta\ell} \simeq \int_{t_\ell}^{t_{\ell+1}} \left(\mathbf{I}_3 - \lfloor {}^B\tilde{\boldsymbol{\theta}}_{\Delta\tau}\times \rfloor\right) {}^{B_\ell}_{B_\tau}\hat{\mathbf{R}}\, {}^{B_\tau}\mathbf{a}\, d\tau - \int_{t_\ell}^{t_{\ell+1}} {}^{B_\ell}_{B_\tau}\hat{\mathbf{R}}\, {}^{B_\tau}\hat{\mathbf{a}}\, d\tau \tag{B.27}$$

$$= \int_{t_\ell}^{t_{\ell+1}} {}^{B_\ell}_{B_\tau}\hat{\mathbf{R}}\, {}^{B_\tau}\tilde{\mathbf{a}}\, d\tau - \int_{t_\ell}^{t_{\ell+1}} \lfloor {}^B\tilde{\boldsymbol{\theta}}_{\Delta\tau}\times \rfloor {}^{B_\ell}_{B_\tau}\hat{\mathbf{R}}\, {}^{B_\tau}\hat{\mathbf{a}}\, d\tau \tag{B.28}$$

$$= \int_{t_\ell}^{t_{\ell+1}} {}^{B_\ell}_{B_\tau}\hat{\mathbf{R}}\, {}^{B_\tau}\tilde{\mathbf{a}}\, d\tau + \int_{t_\ell}^{t_{\ell+1}} \lfloor {}^{B_\ell}_{B_\tau}\hat{\mathbf{R}}\, {}^{B_\tau}\hat{\mathbf{a}}\times \rfloor\, {}^B\tilde{\boldsymbol{\theta}}_{\Delta\tau}\, d\tau \tag{B.29}$$

$$= \int_{t_\ell}^{t_{\ell+1}} {}^{B_\ell}_{B_\tau}\hat{\mathbf{R}}\, {}^{B_\tau}\tilde{\mathbf{a}}\, d\tau + \int_{t_\ell}^{t_{\ell+1}} \hat{\mathbf{R}}_\ell \lfloor {}^G\hat{\mathbf{a}}(\tau)\times \rfloor \hat{\mathbf{R}}_\ell^T\, {}^B\tilde{\boldsymbol{\theta}}_{\Delta\tau}\, d\tau \tag{B.30}$$

Just as with the rotational velocity, we make use of the Taylor expansion to subsitute the acceleration error term with the state error.

$$\tilde{\mathbf{s}} = \int_{t_\ell}^{t_{\ell+1}} \overset{B_\ell}{B_\tau}\hat{\mathbf{R}} \frac{\partial {}^B\mathbf{a}(\tau)}{\partial \mathbf{x}} \tilde{\mathbf{x}}\, d\tau + \int_{t_\ell}^{t_{\ell+1}} \hat{\mathbf{R}}_\ell \lfloor {}^G\hat{\mathbf{a}}(\tau)\times \rfloor \hat{\mathbf{R}}_\ell^T \, {}^B\tilde{\boldsymbol{\theta}}_{\Delta\tau}\, d\tau \tag{B.31}$$

We proceed by differentiating ${}^B\mathbf{a}$ according to the state vector.

$$\frac{\partial {}^B\mathbf{a}}{\partial \mathbf{b}_g} = \mathbf{0} \tag{B.32}$$

$$\frac{\partial {}^B\mathbf{a}}{\partial \mathbf{b}_a} = -\mathbf{T}_a^{-1} \tag{B.33}$$

$$\frac{\partial {}^B\mathbf{a}}{\partial \mathbf{T}_g} = \mathbf{0} \tag{B.34}$$

$$\frac{\partial {}^B\mathbf{a}}{\partial \mathbf{T}_s} = \mathbf{0} \tag{B.35}$$

$$\frac{\partial {}^B\mathbf{a}}{\partial \mathbf{T}_a} = -\mathbf{T}_a^{-1} \frac{\partial \mathbf{T}}{\partial \vec{\mathbf{T}}_a} \mathbf{T}_a^{-1} \left({}^B\mathbf{a}_m - \mathbf{b}_a \right) \tag{B.36}$$

Substitution with equation (B.31) and (6.38), along with ${}^B\tilde{\boldsymbol{\theta}}_{\Delta\tau}$ and the orientation results above gives us the error-state transition components.

$$\boldsymbol{\Phi}_{vb_g} = -\int_{t_\ell}^{t_{\ell+1}} \lfloor {}^G\hat{\mathbf{a}}(\tau)\times \rfloor \hat{\mathbf{R}}_\ell^T \int_{t_\ell}^{\tau} \overset{B_\ell}{B_s}\hat{\mathbf{R}}\, \hat{\mathbf{T}}_g^{-1}\, ds\, d\tau \tag{B.37}$$

$$\boldsymbol{\Phi}_{vb_a} = -\mathbf{R}_\ell^T \int_{t_\ell}^{t_{\ell+1}} \overset{B_\ell}{B_\tau}\hat{\mathbf{R}}\, \hat{\mathbf{T}}_a^{-1}\, d\tau + \int_{t_\ell}^{t_{\ell+1}} \lfloor {}^G\hat{\mathbf{a}}(\tau)\times \rfloor \hat{\mathbf{R}}_\ell^T \int_{t_\ell}^{\tau} \overset{B_\ell}{B_s}\hat{\mathbf{R}}\, \hat{\mathbf{T}}_g^{-1}\, \hat{\mathbf{T}}_s\, \hat{\mathbf{T}}_a^{-1}\, ds\, d\tau \tag{B.38}$$

$$\boldsymbol{\Gamma}_{v\vec{\mathbf{T}}_g} = -\mathbf{R}_\ell^T \int_{t_\ell}^{t_{\ell+1}} \hat{\mathbf{R}}_\ell \lfloor {}^G\hat{\mathbf{a}}(\tau)\times \rfloor \hat{\mathbf{R}}_\ell^T \int_{t_\ell}^{t_\tau} \overset{B_\ell}{B_s}\hat{\mathbf{R}}\, \hat{\mathbf{T}}_g^{-1} \frac{\partial \mathbf{T}}{\partial \vec{\mathbf{T}}_g}\, {}^B\hat{\boldsymbol{\omega}}(s)\, ds\, d\tau \tag{B.39}$$

$$\boldsymbol{\Gamma}_{v\vec{\mathbf{T}}_s} = -\int_{t_\ell}^{t_{\ell+1}} \lfloor {}^G\hat{\mathbf{a}}(\tau)\times \rfloor \hat{\mathbf{R}}_\ell^T \int_{t_\ell}^{t_\tau} \overset{B_\ell}{B_s}\hat{\mathbf{R}}\, \hat{\mathbf{T}}_g^{-1} \frac{\partial \mathbf{T}}{\partial \vec{\mathbf{T}}_s}\, {}^B\hat{\mathbf{a}}(s)\, ds\, d\tau \tag{B.40}$$

$$\begin{aligned}\boldsymbol{\Gamma}_{v\vec{\mathbf{T}}_a} = &-\mathbf{R}_\ell^T \int_{t_\ell}^{t_{\ell+1}} \overset{B_\ell}{B_\tau}\hat{\mathbf{R}}\mathbf{T}_a^{-1} \frac{\partial \mathbf{T}}{\partial \vec{\mathbf{T}}_a} \mathbf{T}_a^{-1} \left({}^B\mathbf{a}_m(\tau) - \mathbf{b}_a \right) d\tau \\ &+ \int_{t_\ell}^{t_{\ell+1}} \lfloor {}^G\hat{\mathbf{a}}(\tau)\times \rfloor \hat{\mathbf{R}}_\ell^T \int_{t_\ell}^{t_\tau} \overset{B_\ell}{B_s}\hat{\mathbf{R}}\, \hat{\mathbf{T}}_g^{-1}\, \hat{\mathbf{T}}_s\, \hat{\mathbf{T}}_a^{-1} \frac{\partial \mathbf{T}}{\partial \vec{\mathbf{T}}_a} \hat{\mathbf{T}}_a^{-1} \left({}^B\mathbf{a}_m(s) - \hat{\mathbf{b}}_a \right) ds\, d\tau\end{aligned} \tag{B.41}$$

With ${}^G\hat{\mathbf{a}} = {}^G\hat{\dot{\mathbf{v}}} - {}^G\mathbf{g}$ for gravity vector ${}^G\mathbf{g}$.

B.3. Position Components

Here we show the bias and IMU calibration terms of the error-state transition matrix relating to the position. Using equation (6.38) we begin with the definition of $\hat{\mathbf{y}}$.

$$\hat{\mathbf{y}} = \int_{t_\ell}^{t_{\ell+1}} \int_{t_\ell}^{s} {}^{B_\ell}_{B_\tau}\hat{\mathbf{R}}\, {}^{B_\tau}\hat{\mathbf{a}}\, d\tau\, ds \tag{B.42}$$

Because $\mathbf{y}$ is the integral of $\mathbf{s}$ it is clear that $\tilde{\mathbf{y}}$ has the following form.

$$\tilde{\mathbf{y}}_{\Delta\ell} = \int_{t_\ell}^{t_{\ell+1}} \tilde{\mathbf{s}}(\tau)\, d\tau \tag{B.43}$$

With this definition, we can use the previously calculated velocity components and integrate them one more time. This is intuitively clear because the position is the integral of the velocity.

$$\mathbf{\Phi}_{pb_g} = -\int_{t_\ell}^{t_{\ell+1}} \int_{t_\ell}^{w} \lfloor {}^{G}\hat{\mathbf{a}}(\tau)\times \rfloor \hat{\mathbf{R}}_\ell^T \int_{t_\ell}^{\tau} {}^{B_\ell}_{B_s}\hat{\mathbf{R}}\, \hat{\mathbf{T}}_g^{-1}\, ds\, d\tau\, dw \tag{B.44}$$

$$\mathbf{\Phi}_{pb_a} = -\mathbf{R}_\ell^T \int_{t_\ell}^{t_{\ell+1}} \int_{t_\ell}^{w} {}^{B_\ell}_{B_\tau}\hat{\mathbf{R}}\, \hat{\mathbf{T}}_a^{-1}\, d\tau\, dw + \int_{t_\ell}^{t_{\ell+1}} \int_{t_\ell}^{w} \lfloor {}^{G}\hat{\mathbf{a}}(\tau)\times \rfloor \hat{\mathbf{R}}_\ell^T \int_{t_\ell}^{\tau} {}^{B_\ell}_{B_s}\hat{\mathbf{R}}\, \hat{\mathbf{T}}_g^{-1}\, \hat{\mathbf{T}}_s\, \hat{\mathbf{T}}_a^{-1}\, ds\, d\tau\, dw \tag{B.45}$$

$$\mathbf{\Gamma}_{p\vec{\mathbf{T}}_g} = -\mathbf{R}_\ell^T \int_{t_\ell}^{t_{\ell+1}} \int_{t_\ell}^{w} \hat{\mathbf{R}}_\ell \lfloor {}^{G}\hat{\mathbf{a}}(\tau)\times \rfloor \hat{\mathbf{R}}_\ell^T \int_{t_\ell}^{t_\tau} {}^{B_\ell}_{B_s}\hat{\mathbf{R}}\, \hat{\mathbf{T}}_g^{-1}\, \frac{\partial \mathbf{T}}{\partial \vec{\mathbf{T}}_g}\, {}^{B}\hat{\boldsymbol{\omega}}(s)\, ds\, d\tau\, dw \tag{B.46}$$

$$\mathbf{\Gamma}_{p\vec{\mathbf{T}}_s} = -\int_{t_\ell}^{t_{\ell+1}} \int_{t_\ell}^{w} \lfloor {}^{G}\hat{\mathbf{a}}(\tau)\times \rfloor \hat{\mathbf{R}}_\ell^T \int_{t_\ell}^{t_\tau} {}^{B_\ell}_{B_s}\hat{\mathbf{R}}\, \hat{\mathbf{T}}_g^{-1}\, \frac{\partial \mathbf{T}}{\partial \vec{\mathbf{T}}_s}\, {}^{B}\hat{\mathbf{a}}(s)\, ds\, d\tau\, dw \tag{B.47}$$

$$\begin{aligned}\mathbf{\Gamma}_{p\vec{\mathbf{T}}_a} = &-\mathbf{R}_\ell^T \int_{t_\ell}^{t_{\ell+1}} \int_{t_\ell}^{w} {}^{B_\ell}_{B_\tau}\hat{\mathbf{R}}\mathbf{T}_a^{-1} \frac{\partial \mathbf{T}}{\partial \vec{\mathbf{T}}_a}\, \mathbf{T}_a^{-1} \left({}^{B}\mathbf{a}_m(\tau) - \mathbf{b}_a\right) d\tau\, dw \\ &+ \int_{t_\ell}^{t_{\ell+1}} \int_{t_\ell}^{w} \lfloor {}^{G}\hat{\mathbf{a}}(\tau)\times \rfloor \hat{\mathbf{R}}_\ell^T \int_{t_\ell}^{t_\tau} {}^{B_\ell}_{B_s}\hat{\mathbf{R}}\, \hat{\mathbf{T}}_g^{-1}\, \hat{\mathbf{T}}_s\, \hat{\mathbf{T}}_a^{-1} \frac{\partial \mathbf{T}}{\partial \vec{\mathbf{T}}_a}\, \hat{\mathbf{T}}_a^{-1} \left({}^{B}\mathbf{a}_m(s) - \hat{\mathbf{b}}_a\right) ds\, d\tau\, dw\end{aligned} \tag{B.48}$$

B.4. Discrete Implementation

To use the error-state transition matrix in practice, we use trapezoidal integration with the following approximate results.

$$\mathbf{\Phi}_{qb_g} \simeq -\frac{\Delta t}{2}\left(\hat{\mathbf{R}}_{\ell+1}^T + \hat{\mathbf{R}}_{\ell}^T\right)\hat{\mathbf{T}}_g^{-1} \tag{B.49}$$

$$\mathbf{\Phi}_{vb_g} \simeq \frac{\Delta t^2}{4}\left(\lfloor({}^G\mathbf{a}_{\ell+1} - {}^G\mathbf{g})\times\rfloor)(\hat{\mathbf{R}}_{\ell+1}^T + \hat{\mathbf{R}}_{\ell}^T)\right)\hat{\mathbf{T}}_g^{-1} \tag{B.50}$$

$$\mathbf{\Phi}_{pb_g} \simeq \frac{\Delta t}{2}\mathbf{\Phi}_{vb_g} \tag{B.51}$$

$$\mathbf{\Phi}_{qb_a} \simeq \frac{\Delta t}{2}\left(\hat{\mathbf{R}}_{\ell+1}^T + \hat{\mathbf{R}}_{\ell}^T\right)\hat{\mathbf{T}}_g^{-1}\hat{\mathbf{T}}_s\hat{\mathbf{T}}_a^{-1} \tag{B.52}$$

$$\mathbf{\Phi}_{vb_a} \simeq -\frac{\Delta t}{2}\left(\hat{\mathbf{R}}_{\ell+1}^T + \hat{\mathbf{R}}_{\ell}^T\right)\hat{\mathbf{T}}_a^{-1} + \frac{\Delta t^2}{4}\lfloor{}^G\mathbf{a}_{\ell+1} - {}^G\mathbf{g}\times\rfloor\left(\hat{\mathbf{R}}_{\ell+1}^T + \hat{\mathbf{R}}_{\ell}^T\right)\hat{\mathbf{T}}_g^{-1}\hat{\mathbf{T}}_s\hat{\mathbf{T}}_a^{-1} \tag{B.53}$$

$$\mathbf{\Phi}_{pb_a} \simeq \frac{\Delta t}{2}\mathbf{\Phi}_{vb_a} \tag{B.54}$$

$$\mathbf{\Gamma}_{q\vec{\mathbf{T}}_g} \simeq -\frac{\Delta t}{2}\left(\hat{\mathbf{R}}_{\ell+1}^T + \hat{\mathbf{R}}_{\ell}^T\right)\hat{\mathbf{T}}_g^{-1}\frac{\partial \mathbf{T}_g}{\partial \vec{\mathbf{T}}_g}\,{}^B\hat{\boldsymbol{\omega}}_{\ell+1} \tag{B.55}$$

$$\mathbf{\Gamma}_{v\vec{\mathbf{T}}_g} \simeq -\frac{\Delta t^2}{4}\lfloor{}^G\mathbf{a}_{\ell+1} - {}^G\mathbf{g}\times\rfloor\left(\hat{\mathbf{R}}_{\ell+1}^T + \hat{\mathbf{R}}_{\ell}^T\right)\hat{\mathbf{T}}_g^{-1}\frac{\partial \mathbf{T}_g}{\partial \vec{\mathbf{T}}_g}\,{}^B\hat{\boldsymbol{\omega}}_{\ell+1} \tag{B.56}$$

$$\mathbf{\Gamma}_{p\vec{\mathbf{T}}_g} \simeq \frac{\Delta t}{2}\mathbf{\Gamma}_{v\vec{\mathbf{T}}_g} \tag{B.57}$$

$$\mathbf{\Gamma}_{q\vec{\mathbf{T}}_s} \simeq -\frac{\Delta t}{2}\left(\hat{\mathbf{R}}_{\ell+1}^T + \hat{\mathbf{R}}_{\ell}^T\right)\hat{\mathbf{T}}_g^{-1}\frac{\partial \mathbf{T}_s}{\partial \vec{\mathbf{T}}_s}\,{}^B\hat{\mathbf{a}}_{\ell+1} \tag{B.58}$$

$$\mathbf{\Gamma}_{v\vec{\mathbf{T}}_s} \simeq -\frac{\Delta t^2}{4}\lfloor{}^G\mathbf{a}_{\ell+1} - {}^G\mathbf{g}\times\rfloor\left(\hat{\mathbf{R}}_{\ell+1}^T + \hat{\mathbf{R}}_{\ell}^T\right)\hat{\mathbf{T}}_g^{-1}\frac{\partial \mathbf{T}_s}{\partial \vec{\mathbf{T}}_s}\,{}^B\hat{\mathbf{a}}_{\ell+1} \tag{B.59}$$

$$\mathbf{\Gamma}_{p\vec{\mathbf{T}}_s} \simeq \frac{\Delta t}{2}\mathbf{\Gamma}_{v\vec{\mathbf{T}}_s} \tag{B.60}$$

$$\mathbf{\Gamma}_{q\vec{\mathbf{T}}_a} \simeq \frac{\Delta t}{2}\left(\hat{\mathbf{R}}_{\ell+1}^T + \hat{\mathbf{R}}_{\ell}^T\right)\hat{\mathbf{T}}_g^{-1}\hat{\mathbf{T}}_s\hat{\mathbf{T}}_a^{-1}\frac{\partial \mathbf{T}_a}{\partial \vec{\mathbf{T}}_a}\,{}^B\hat{\mathbf{a}}_{\ell+1} \tag{B.61}$$

$$\begin{aligned}\mathbf{\Gamma}_{v\vec{\mathbf{T}}_a} \simeq{}& \frac{\Delta t^2}{4}\lfloor{}^G\mathbf{a}_{\ell+1} - {}^G\mathbf{g}\times\rfloor\left(\hat{\mathbf{R}}_{\ell+1}^T + \hat{\mathbf{R}}_{\ell}^T\right)\hat{\mathbf{T}}_g^{-1}\hat{\mathbf{T}}_s\hat{\mathbf{T}}_a^{-1}\frac{\partial \mathbf{T}_a}{\partial \vec{\mathbf{T}}_a}\,{}^B\hat{\mathbf{a}}_{\ell+1} \\ &- \frac{\Delta t}{2}\left(\hat{\mathbf{R}}_{\ell+1}^T + \hat{\mathbf{R}}_{\ell}^T\right)\hat{\mathbf{T}}_a^{-1}\frac{\partial \mathbf{T}_a}{\partial \vec{\mathbf{T}}_a}\,{}^B\hat{\mathbf{a}}_{\ell+1}\end{aligned} \tag{B.62}$$

$$\mathbf{\Gamma}_{p\vec{\mathbf{T}}_a} \simeq \frac{\Delta t}{2}\mathbf{\Gamma}_{v\vec{\mathbf{T}}_a} \tag{B.63}$$

$$\tag{B.64}$$

Bibliography

[1] ARDrone Flyers. AR.Drone — ARDrone-Flyers.com, 2011. [`http://www.ardrone-flyers.com/`].

[2] H. Bay, T. Tuytelaars, and L.V. Gool. SURF: Speeded-up robust features. In *Proc. of the European Conference on Computer Vision (ECCV)*, 2008.

[3] G. Bradski. The opencv library. *Dr. Dobb's Journal of Software Tools*, 2000.

[4] Colm Buckley. Bezier curves for camera motion, Oct 1994.

[5] M. Calonder, V. Lepetit, M. Ozuysal, T. Trzcinski, C. Strecha, and P. Fua. Brief: Computing a local binary descriptor very fast. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(7):1281–1298, July 2012.

[6] J. Canny. A computational approach to edge detection. *Conference on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679 – 698, 1986.

[7] J. Civera, A. Davison, and J. Montiel. Inverse depth parametrization for monocular slam. *IEEE Transactions on Robotics*, 24(5):932 – 945, 2008.

[8] A. Davison. Real-time simultaneous localisation and mapping with a single camera. In *Proc. of the International Conference on Computer Vision (ICCV)*, 2003.

[9] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: Part I. *Robotics & Automation Magazine*, 13(2):99 – 110, 2006.

[10] J. Engel, J. Sturm, and D. Cremers. Semi-dense visual odometry for a monocular camera. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 1449–1456, Dec 2013.

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

[12] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, and A. Fitzgibbon. KinectFusion: real-time 3d reconstruction and interaction using a moving depth camera. In *Proc. of the 24th annual ACM symposium on User interface software and technology (UIST)*, 2011.

[13] G. Klein and D. Murray. Parallel tracking and mapping for small AR workspaces. In *Proc. of the International Symposium on Mixed and Augmented Reality (ISMAR)*, 2007.

[14] G. Klein and D. Murray. Parallel tracking and mapping on a camera phone. In *Mixed and Augmented Reality, 2009. ISMAR 2009. 8th IEEE International Symposium on*, pages 83–86, Oct 2009.

[15] J. Knight, A. Davison, and I. Reid. Towards constant time slam using postponement. In *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, volume 1, pages 405–413 vol.1, 2001.

[16] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. g2o: A general framework for graph optimization. In *Proc. of the International Conference on Robotics and Automation (ICRA)*, 2011.

[17] J.J. Leonard and H.F. Durrant-Whyte. Simultaneous map building and localization for an autonomous mobile robot. In *Intelligent Robots and Systems '91. 'Intelligence for Mechanical Systems, Proceedings IROS '91. IEEE/RSJ International Workshop on*, pages 1442–1447 vol.3, Nov 1991.

[18] M. Li, B.H. Kim, and A. I. Mourikis. Real-time motion estimation on a cellphone using inertial sensing and a rolling-shutter camera. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 4697–4704, Karlsruhe, Germany, May 2013.

[19] M. Li and A. I. Mourikis. Improving the accuracy of ekf-based visual-inertial odometry. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 828–835, Minneapolis, MN, May 2012.

[20] M. Li and A. I. Mourikis. Vision-aided inertial navigation for resource-constrained systems. In *Proceedings of the IEEE/RSJ International Conference on Robotics and Intelligent Systems (IROS)*, pages 1056–1063, Vilamoura, Portugal, November 2012.

[21] M. Li and A. I. Mourikis. 3-d motion estimation and online temporal calibration for camera-imu systems. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 5689–5696, Karlsruhe, Germany, May 2013.

[22] D. Lowe. Object recognition from local scale-invariant features. In *Proc. of the International Conference on Computer Vision (ICCV)*, 1999.

[23] A. I. Mourikis M. Li. Consistency of ekf-based visual-inertial odometry. Technical report, Dept. of Electrical Engineering, University of California, Riverside, 2012. http://www.ee.ucr.edu/ mli/ICRA2012REPORT.pdf.

[24] X. Zheng M. Li, H. Yu and A. I. Mourikis. High-fidelity sensor modeling and self-calibration in vision-aided inertial navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Hong Kong, May 2014.

[25] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *Proc. of the AAAI National Conference on Artificial Intelligence*, 2002.

[26] A. I. Mourikis and S. I. Roumeliotis. A multi-state constraint kalman filter for vision-aided inertial navigation. Technical report, Dept. of Computer Science and Engineering, University of Minnesota, 2006. www.cs.umn.edu/∼mourikis/tech_reports/TR_MSCKF.pdf.

[27] A.I. Mourikis and S.I. Roumeliotis. A multi-state constraint kalman filter for vision-aided inertial navigation. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 3565–3572, April 2007.

[28] R.A. Newcombe, S. Lovegrove, and A.J. Davison. DTAM: Dense tracking and mapping in real-time. In *Proc. of the International Conference on Computer Vision (ICCV)*, 2011.

[29] G. Nützi, S. Weiss, D. Scaramuzza, and R. Siegwart. Fusion of IMU and vision for absolute scale estimation in monocular SLAM. *Journal of Intelligent Robotic Systems*, 61(1–4):287 – 299, 2010.

[30] P. Pinies, T. Lupton, S. Sukkarieh, and J.D. Tardos. Inertial aiding of inverse depth slam using a monocular camera. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 2797–2802, April 2007.

[31] M. Pizzoli, C. Forster, and D. Scaramuzza. Remode: Probabilistic, monocular dense reconstruction in real time. In *IEEE International Conference on Robotics and Automation (ICRA)*, Hong Kong, May 2014.

[32] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully B. Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. ROS: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[33] E. Rosten and T. Drummond. Machine learning for high-speed corner detection. In *Proc. of the European Conference on Computer Vision (ECCV)*, 2006.

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

[35] J. Shi and C. Tomasi. Good features to track. In *Proc. of the Conference on Computer Vision and Pattern Recognition (CVPR)*, 1994.

[36] H. Strasdat, A. Davison, J. Montiel, and K. Konolige. Double window optimisation for constant time visual SLAM. In *Proc. of the International Conference on Computer Vision (ICCV)*, 2011.

[37] H. Strasdat, J. Montiel, , and A. Davison. Scale-drift aware large scale monocular SLAM. In *Proc. of Robotics: Science and Sytems*, 2010.

[38] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents series)*. MIT Press, 2005.

[39] N. Trawny and S. I. Roumeliotis. Indirect kalman filter for 6d pose estimation. Technical Report Tech. Rep. 2005-002, University of Minnesota, Dept. of Comp. Sci. & Eng., Jan. 2005.

[40] Tinne Tuytelaars and Krystian Mikolajczyk. K.: Local invariant feature detectors: A survey. *FnT Comp. Graphics and Vision*, pages 177–280, 2008.

[41] Stephan Weiss, Markus W. Achtelik, Simon Lynen, Michael C. Achtelik, Laurent Kneip, Margarita Chli, and Roland Siegwart. Monocular vision for long-term micro aerial vehicle state estimation: A compendium. *Journal of Field Robotics*, 30(5):803–831, 2013.

[42] Wikipedia. Gauss-newton algorithm, 2014. [http://en.wikipedia.org/wiki/Gauss-Newton_algorithm].

[43] Wikipedia. Google driverless car — Wikipedia, the free encyclopedia, 2014. [http://en.wikipedia.org/wiki/Google_driverless_car].

[44] Wikipedia. Roomba — Wikipedia, the free encyclopedia, 2014. [http://en.wikipedia.org/wiki/Roomba].

[45] Wikipedia. Spirit (rover) — Wikipedia, the free encyclopedia, 2014. [http://en.wikipedia.org/wiki/Spirit_rover].

[46] Youtube. MSCKF results, 2014. [http://youtu.be/Xrq56SZONvo].