Filtering Techniques for Pose Estimation with Applications to Unmanned Air Vehicles

Bryce Benson Ready
Brigham Young University - Provo

Filtering Techniques for Pose Estimation With Applications to Unmanned Air Vehicles

Bryce Benson Ready

A dissertation submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Electrical and Computer Engineering

Brigham Young University

November 2012

ABSTRACT

Filtering Techniques for Pose Estimation With Applications to Unmanned Air Vehicles

Bryce Benson Ready
Department of Electrical and Computer Engineering
Doctor of Philosophy

This work presents two novel methods of estimating the state of a dynamic system in a Kalman Filtering framework. The first is an application specific method for use with systems performing Visual Odometry in a mostly planar scene. Because a Visual Odometry method inherently provides relative information about the pose of a platform, we use this system as part of the time update in a Kalman Filtering framework, and develop a novel way to propagate the uncertainty of the pose through this time update method. Our initial results show that this method is able to reduce localization error significantly with respect to pure INS time update, limiting drift in our test system to around 30 meters for tens of seconds.

The second key contribution of this work is the Manifold EKF, a generalized version of the Extended Kalman Filter which is explicitly designed to estimate manifold-valued states. This filter works for a large number of commonly useful manifolds, and may have applications to other manifolds as well. In our tests, the Manifold EKF demonstrated significant advantages in terms of consistency when compared to other filtering methods. We feel that these promising initial results merit further study of the Manifold EKF, related filters, and their properties.

Keywords: Riemannian manifold, Kalman filter, unmanned air vehicles, homography, image registration, direct image registration, visual odometry

ACKNOWLEDGMENTS

This dissertation is dedicated to my wife Camilla, whose support and encouragement made the writing of it possible.

I would like to thank Dr. Clark Taylor for his advice and support throughout my graduate work. I could not have asked for a better mentor or a better friend.

I would like to thank my committee members for their comments and support, particularly Dr. Jeff Humpherys for introducing me to his framework for understanding the Kalman Filter.

I would also like to thank Dr. Travis Oliphant, for introducing me to a number of beautiful ideas which have transformed the way I think, technically and otherwise.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Unmanned air vehicles have become increasingly popular in recent years, both commercially and in academic and military applications. These robotic platforms are inexpensive and lightweight, and have become increasingly capable in terms of sensor payloads and autonomy.

If any robotic platform is to interact with its environment, it must first have an accurate internal model of some sort that describes that environment and its place within it. Hence, estimation of a robotic platform's state has been the topic of a great deal of research for many years. Recent increases in the capability and ubiquity of robotics platforms like UAVs has brought some of the challenges of the state estimation problem to the forefront. In particular, most systems of interest have nonlinear system equations which must be solved "online", ideally in realtime, rather than using offline or batch-processing techniques. Many robotics applications, then, require a solution to the nonlinear filtering problem.

The nonlinear filtering problem has the goal of estimating the state of a dynamic system whose dynamic and measurement equations are nonlinear functions. It is one of the "hard problems" in engineering[1], one in which engineering skill, not rote copying of some optimal form, will be necessary for many years to come. This fact is an outgrowth of the inherent complexity of nonlinear functions: the probability density function which accurately describes out knowledge of the state of a system given some set of constraints can be arbitrarily complex. There are many ways to simplify the problem, all of them introducing some degree of sub-optimality.

There are many ways to perform nonlinear state estimation that are applicable to robotic vehicles, but the Kalman family of filters, from the provably optimal (for linear systems) Kalman Filter itself, to the wide range of nonlinear Kalman-style filters, has certainly found the broadest scope of application. The Extended Kalman Filter is without question the "workhorse" estimator of industry, and is the starting point for much academic research. While it can and has been improved upon, the EKF is, as we shall discuss in Chapter 6, based on a very important and flexible method of solving inverse problems. This, perhaps, explains why it works so well and is so tolerant of approximations.

This work makes two major contributions. Each of these are novel methods of performing state estimation with Kalman-style filters.

The first contribution of this work is discussed in Part I, Chapters 3 through 4. It is related to the use of an optical camera, an increasingly common and useful sensor found in robotic applications. There are two main ways in which EO cameras are used for pose estimation: they can perform some sort of place recognition, providing absolute estimates of portions of the pose, or they can be used to perform what is called Visual Odometry (VO), the process of estimating relative pose displacements from a series of images, thus yielding constraints on vehicle motion that can assist the estimation process. We describe a Visual Odometry system which is based on direct image registration. Since visual odometry inherently yields relative pose information, we use the VO system as the "time update" portion of the extended Kalman filtering framework. The fact that the visual odometry process is not differentiable presents a difficulty with this method, as it makes it difficult to propagate covariance estimates through this process. We present a novel way of performing this covariance propagation that takes advantage of the structure of the VO system, and show promising preliminary results of its use on a real small UAV platform.

The second, and much more general, contribution of this work is discussed in Part II, Chapters 6 through 10. This contribution is a new Kalman-family filter which we term the Manifold EKF. It is essentially a re-derivation of the Extended Kalman Filter, which extends its applicability to a broad and useful class of Riemannian Manifolds. We begin by highlighting a derivation of the EKF due to [2, 3], which provides a useful way to view the EKF and gives insights into its performance. The Manifold EKF is derived using this same framework, but with error terms defined in terms of operations that are generalized to manifolds. We present several examples of the implementation of these manifold operations for key manifolds of interest, give an analysis of the consistency of this new filter, and provide simulated results for a sample application.

Part I

Kalman Filtering with Visual Odometry

Chapter 2

Motivation and Related Work

Many of the advantages associated with fixed-wing MAVs stem from their ability to operate autonomously at two levels. On a basic level, MAVs must be able to autonomously perform low-level flight tasks such as taking off, flying straight and level, climbing, descending, banking, etc. On a higher level, MAVs must be able to combine these maneuvers to fly to specific locations, follow specific trajectories, and otherwise *navigate* autonomously. To perform these tasks, MAVs must be able to estimate their own pose, which consists of location (t_x, t_y, t_z) and attitude, which is commonly expressed using Euler angles for yaw(ψ), pitch(θ), and roll(ϕ). We can divide these six pose parameters into two sets of three parameters. The first group, which we will refer to as *aviation parameters* (pitch, roll, and altitude), are required to perform basic autonomous maneuvers, while the second, which we term *navigation parameters* (x, y, yaw), are additionally required for autonomous navigation. Current MAV systems [4, 5, 6, 7] carry a simple Inertial Navigation System (INS) consisting of accelerometers, rate gyros, and pressure sensors. These sensors can provide only relative information about navigation parameters[1]; thus INS based estimates of navigation parameters will drift without bound over time. Because MAVs use low-cost, lightweight MEMS-based inertial sensors, errors in navigation parameter estimates from the INS alone typically increase extremely rapidly, often becoming unacceptably large within a few seconds. For this reason, current MAV systems rely on the Global Positioning System (GPS) to provide estimates of the navigation parameters. While GPS does provide bounded-error estimates of geo-location (x,y) and heading, it makes the operation of the MAV dependent on external infrastructure–the network of orbiting GPS satellites. Signals from these satellites can be blocked, both by environmental obstacles (eg. urban terrain) and by deliberate or unintentional jamming [8]. Much effort has therefore been directed at finding ways to reduce the dependence of MAV platforms on GPS.

Vision-based pose estimation techniques are a promising way to estimate pose in GPS-denied environments, and thus reduce dependence on GPS. Vision sensors are typically already available on MAV platforms, and provide a rich source of information about the environment. There are two main methods of performing vision-based pose estimation reported in the literature. Visual Simultaneous Localization And Mapping, or Visual SLAM (e.g. [9, 10, 11, 12, 13]) is perhaps the most elegant and complete method. SLAM algorithms in general estimate both robot state and the location of landmarks in the environment simultaneously. If perfected, a solution to the Visual SLAM problem would allow a robot to function in a truly autonomous manner, using vision and other sensors to navigate an unfamiliar environment as a human being can, without relying on fiducial markings, GPS signals, or other external infrastructure. However, there are still a number of problems with Visual SLAM which make its practical application challenging.

[1]As we shall see, INS sensors can provide absolute measurements of aviation parameters. Exploitation of this fact is a critical component of this paper.

SLAM algorithms in general have non-constant computation time as more and more landmarks are observed, and managing and reducing this computational load is still a focus of ongoing research. Current Visual SLAM systems can use either a video camera alone (e.g. [13]) or a video camera with low-quality MEMS-based inertial sensors (e.g. [12]), and can provide impressive navigational accuracy and stability. Unfortunately, they rely on the assumption that the environment is bounded and relatively small, so that an excessive and growing collection of landmarks does not slow down the processor. This assumption is not valid for MAV platforms, which must navigate in extremely large unexplored environments. Furthermore, in navigation applications, the landmark locations are typically not of interest, meaning that much of this computational burden, while providing greater accuracy, does not contribute directly to the desired result.

In this work, our proposed scenario is a system in which GPS is typically available, but may drop out for an intermediate length of time (several tens of seconds). Our goal, then, is not to produce estimates that do not accumulate error, but estimates that accumulate error slowly enough that they are still accurate within this time window. Visual Odometry (VO) methods are a viable means of performing vision-based pose estimation under this scenario. Named in analogy to wheeled-robot odometry, VO methods use computer vision algorithms to estimate the relative orientation between image frames. With any VO method, then, the key goal is to somehow decrease the amount of error introduced at each step, thereby slowing the growth of error sufficiently that pose estimates are valid within a desired time window.

We employ two strategies to improve the error characteristics of VO. First, we utilize a novel VO system based upon prior work by Dellaert *et al.* [14, 15]. Our VO system uses a direct image registration algorithm, estimating a single parametric transformation mapping pixels in an image to ground locations. This is in contrast to more standard feature-based image registration methods, which track a series of feature points and use their motion to infer the relative pose between frames. Direct registration is generally able to produce more accurate results than such feature-based methods; this increased registration accuracy helps to slow the accumulation of pose error.

The second method we use to slow VO error growth is to fuse VO measurements with INS data in an EKF framework. In the literature, a technique known as vision-assisted inertial navigation [16, 17, 18, 19, 20, 21, references therein] is the usual method of doing this. Current techniques use the INS to provide relative measurements of the pose parameters, and these relative measurements are integrated in the EKF time update step to provide absolute pose. The relative pose measurements provided by VO are used in the EKF measurement update step to correct drift in this INS pose estimate. SLAM methods which incorporate inertial sensors similarly use inertial measurements in the time update (e.g. [9, 12]). In this work, we fuse INS and vision in a different way: rather than performing vision-assisted inertial navigation, we propose to perform inertially-aided visual odometry. The distinction is somewhat subtle, but nevertheless significant. One key contribution of this work is that we interpret INS measurements in a different way than the standard vision-aided inertial navigation literature, allowing us to make use of a low-quality MEMS-based INS without inducing INS integration error. This is in contrast to most vision-assisted inertial navigation literature, where the assumption is usually made that a relatively high fidelity INS system is available, or else that another sensor can slow or stop the drift induced by a low quality INS. Another interpretation is possible, however: the three-axis accelerometers in the INS measure the acceleration of the aircraft in each dimension, which for a fixed wing aircraft, will primarily measure the direction of the gravity vector. This information allows us to directly compute the pitch and roll of the aircraft [7]. INS data has been interpreted in this way in the computer vision community to estimate the pose of a camera [22, 23] as well as to perform such tasks as scene reconstruction and camera calibration [24, 25, 26]. Combining these pitch and roll estimates with the altitude estimate obtained from the INS pressure sensors, we have a complete estimate of the aviation pose parameters of the aircraft. Many current MAV systems interpret INS data in this way [4, 5, 6]: however, to our knowledge ours is the first work to apply this information to MAVs in a VO setting. Because we interpret the INS data as providing absolute measurements of the aviation pose parameters of the aircraft, we use INS data in the EKF measurement update, rather than as a time update. Since VO measures the relative pose between frames, we use VO measurements in the time-update step. Thus, VO is the main means of estimating aircraft pose, and its accuracy is improved by incorporating measurements of the aviation parameters from the INS. Use of VO as the time-update step in an EKF framework requires that we be able to propagate both the MAV pose and its covariance matrix. The standard EKF method for doing this (linearizing the time update function) will not work with our proposed VO system, as it is neither differentiable nor available in closed form. The second major contribution of this paper is thus a novel means of estimating uncertainty associated with our VO estimates.

In the remainder of Part I, we first give a general overview of our pose estimation system (Section 3), and then describe our VO method and the underlying direct image registration method (Section 3.1), giving further background on existing VO methods. We then develop our proposed method of uncertainty propagation through the VO system (Section 3.2). Finally, we will present MAV flight results obtained using our method (Section 4).

Chapter 3

Visual Odometry Based Pose Estimator

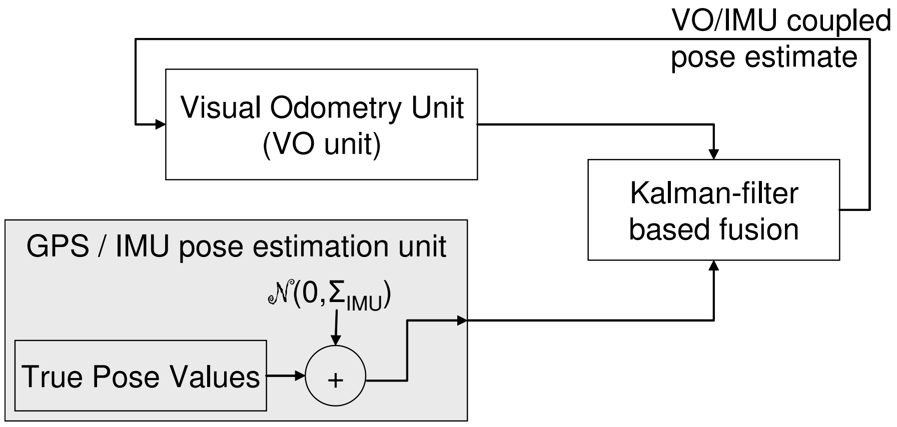

VO/IMU coupled pose estimate
Visual Odometry Unit (VO unit)
Kalman-filter based fusion
GPS / IMU pose estimation unit
$\mathcal{N}(0,\Sigma_{IMU})$
True Pose Values
+

Figure 3.0.1: Layout of our GPS/INS/VO pose estimation system

The operation of our pose estimation framework is summarized in Figure 3.0.1. The goal of this system is to estimate the pose of the aircraft, which we represent with a 6-vector χ:

$$\chi = \begin{bmatrix} t_x & t_y & t_z & \psi & \theta & \phi \end{bmatrix}^T, \qquad (3.0.1)$$

composed of a 3-D location state (t_x, t_y, t_z) and three Euler angles(ψ, θ, ϕ) representing attitude (yaw[1], pitch, and roll). Video frames Y_n and Y_{n-1} from an MAV camera are fed into the VO system, along with current pose estimates. The VO system then produces an estimated pose χ_n^V for each new video frame Y_n, using the image data Y_{n-1} and estimated pose $\hat{\chi}_{n-1}$ of the previous frame.

Independently, information from other on-board sensors are fed into the GPS/INS pose estimation system, which separately estimates a pose χ_n^G of the aircraft at the time each frame was taken. If GPS is unavailable, this estimate includes only the aviation parameters, which can be obtained from the INS alone. In our system, altitude is computed directly from the autopilot pressure sensors, while pitch and roll are computed by combining accelerometer

[1]Yaw is typically defined as the compass direction in which the nose of the aircraft is pointing, while heading is the direction in which the aircraft is moving. If the aircraft is flying with a crab angle (i.e. the nose is not pointing exactly in the direction of flight) due to wind conditions, these two quantities will not be identical. Pose estimates obtained using VO provide yaw information, while heading estimates can be provided directly by GPS. Heading can also be estimated by using the difference in location estimates.

and rate gyro measurements in a complimentary filter. Pitch/roll states are propagated forward in time using rate measurements from the gyroscopes. Accelerometer measurements are used to give bounded error estimates of pitch and roll, computed according to the following formula:

$$\begin{aligned} \begin{bmatrix} A_x & A_y & A_z \end{bmatrix} &\equiv \text{accelerometer readings}, && (3.0.2) \\ \phi_{acc} &= \tan^{-1}\left(\frac{A_y}{-A_z}\right), && (3.0.3) \\ \theta_{acc} &= \tan^{-1}\left(\frac{A_x}{-A_y \sin(\phi) - A_z \cos(\phi)}\right). && (3.0.4) \end{aligned}$$

The weighted sum of the accelerometer based measurements and predicted states become the new pitch/roll estimates. If the aircraft is turning, the accelerometers will measure not only the lift force opposing gravity, but also extra acceleration from the d'Alembert force due to centripetal acceleration. To help account for this fact, the relative weight of the accelerometer-based measurements is reduced as the turn rate of the MAV increases, causing the system to rely more heavily on the propagated values. In our experience, this method produces sufficiently accurate estimates to enable MAV navigation (see [7]).

Because these INS based pose measurements have bounded error as we have discussed, we model this estimate as the true pose of the aircraft corrupted with zero-mean Gaussian noise (ν):

$$\text{GPS unavailable: } \chi_n^G = \begin{bmatrix} t_z & \theta & \phi \end{bmatrix}^T + \nu. \quad (3.0.5)$$

We desire to fuse these partial measurements of aircraft pose with the pose information from the VO system using an EKF framework. The standard EKF framework estimates the state of a system given knowledge of the system dynamics and measurements that are functions of the state:

$$\begin{aligned} \chi_n &= \mathbf{f}(\chi_{n-1}, u_n) + \eta, && (3.0.6) \\ y_n &= \mathbf{h}(\chi_n) + \nu && (3.0.7) \end{aligned}$$

where η and ν are zero-mean, Gaussian random vectors with covariance matrices Q and R respectively. At each time step, we estimate the new state of the system and its covariance from the previous state:

$$\begin{aligned} \hat{\chi}_n^- &= \mathbf{f}(\hat{\chi}_{n-1}, u_n), && (3.0.8) \\ \hat{P}_n^- &= F\hat{P}_{n-1}F^T + Q && (3.0.9) \end{aligned}$$

where the matrix F is the Jacobian of the system dynamics function:

$$F = \left.\frac{\partial \mathbf{f}}{\partial \chi}\right|_{\chi=\hat{\chi}_{n-1}, u=u_n}. \quad (3.0.10)$$

When a measurement becomes available, we incorporate the information it provides about the state, performing what is known as a measurement update step:

$$\begin{aligned}\hat{\chi}_n &= \hat{\chi}_n^- + K\left(y_n - \mathbf{h}\left(\hat{\chi}_n^-\right)\right), &(3.0.11)\\ \hat{P}_n &= (I - KH)\,\hat{P}_n^- &(3.0.12)\end{aligned}$$

where H is the Jacobian of the measurement function (analogous to F):

$$H = \left.\frac{\partial \mathbf{h}}{\partial \chi}\right|_{\chi=\hat{\chi}^-}. \qquad (3.0.13)$$

As we have discussed, we use the aviation parameters available from the INS as our measurement function. This means that our $\mathbf{h}$ function is in fact just a linear operator, and we can find H directly:

$$\begin{aligned} y = \begin{bmatrix}\theta\\ \phi\\ t_z\end{bmatrix} &= \mathbf{h}(\chi)\\ &= \underbrace{\begin{bmatrix}0&0&0&0&1&0\\0&0&0&0&0&1\\0&0&1&0&0&0\end{bmatrix}}_{H}\chi. \end{aligned} \qquad (3.0.14)$$

By contrast, the time update in our EKF will come from VO, meaning that our $\mathbf{f}$ function represents the VO system as follows:

$$\begin{aligned}\chi_n^- &= \mathbf{f}(\chi_{n-1})\\ &= \text{vo\_system}\left(\chi_{n-1}, Y_{n-1}, Y_n\right). \end{aligned} \qquad (3.0.15)$$

In the next section, we will describe the operation of our VO pose estimation system, which is represented by the function $\mathbf{f}(\cdot)$. We will then proceed in section 3.2 to approximate F, enabling us to implement equation (3.0.9).

3.1 Visual Odometry System

Several VO frameworks are delineated in the literature. Most commonly used existing methods function by detecting and tracking feature points between frames in a video sequence, and using the motion of these points to estimate the relative pose between frames. This is done by using feature point motion to estimate either the essential matrix [27, 28, 29, 30, 31] or a homography [32, 33] relating pairs or sets of frames, and then decomposing these matrices [34, 35] to find the relative pose.

All VO methods share a common implementation challenge that must be addressed to allow absolute pose to be estimated. This problem is that of determining the scale factor of the estimated relative pose. Because a video camera is a bearing-only sensor and provides no depth information, it is impossible to distinguish whether a pair of frames are widely separated and observing large, distant objects or closely spaced and observing small, nearby objects. If care is not taken to ensure that relative pose estimates are expressed in the same scale then gross errors in absolute pose estimates can be accumulated very rapidly. This problem is typically addressed in the literature by triangulating the 3-D location of feature points common between two frame pairs.

In this work, we propose a novel VO strategy, based upon prior work by Dellaert *et al.* [14, 15]. Rather than infer the inter-frame relative pose by using the motion of extracted feature locations, we directly compute the absolute pose of the second frame from the absolute pose of the first frame by means of an iterative image registration approach. This approach works by projecting the first observed image onto the terrain and rendering a view of this projected data from the currently estimated pose of the second frame. The estimated pose of the second frame is iteratively adjusted by means of image Jacobians to make the second frame and the re-projected first frame match as closely as possible. Thus, this method directly estimates a single parametric transform using the captured image as a whole, rather than the estimated motion of selected feature points. This fact typically allows direct image registration methods to provide greater accuracy in image registration. Furthermore, since the absolute pose of each frame is estimated, the scale factor problem is handled implicitly.

Direct image registration such as we are performing depends upon three main assumptions: (1) that the scene being imaged is planar, (2) that all image motion is due to camera motion, meaning that motion due to independently moving objects is negligible, (3) there is sufficient texture in the imaged scene to allow the iterative descent registration algorithm to avoid converging to a local minimum.

While the planarity assumption is not appropriate for ground-based robots, it is generally a reasonable assumption in other applications. Unmanned Underwater Vehicle applications [36, 37, 38] commonly make this assumption. As MAVs are typically relatively distant from the terrain they observe, this assumption is often workable in many fixed-wing MAV situations. In addition, we have found that our direct registration algorithm is robust to small amounts of non-planarities (e.g. trees, small structures) in the images. This is a reasonable scenario for many applications; low flights in complicated (e.g. urban) terrain will violate this assumption, however.

The assumption that objects do not move independently is not problematic in most environments. At typical MAV altitudes, any moving objects will occupy only a tiny fraction of a captured image, and thus will have little effect on image registration. This assumption will fail in some cases (i.e. viewing a highway with heavy, fast-moving traffic) but will be a good assumption in many others.

Sufficient image texture is also usually a good assumption: most UAV flights are daytime flights, and most real flight environments contain significant visual texture. Certain flight environments could of course cause the direct image registration to not converge. In general, however, our practical experience suggests that direct registration is more robust than feature-based methods in low-texture video.

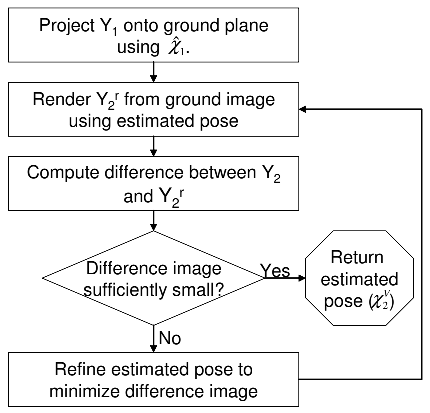

Project Y_1 onto ground plane using $\hat{\chi}_1$.
Render Y_2^r from ground image using estimated pose
Compute difference between Y_2 and Y_2^r
Difference image sufficiently small?
Yes
Return estimated pose (χ_2^V)
No
Refine estimated pose to minimize difference image

Figure 3.1.1: Our method for computing the pose of the MAV when Image 2 was captured assuming Image 1's pose is perfectly known.

Our VO algorithm for estimating the current pose of the MAV is illustrated in Figure 3.1.1. To estimate the pose of video frame Y_2 (or rather, the pose of the aircraft when this frame was captured), we assume that a previous frame Y_1 is available with associated pose information $\hat{\chi}_1$ for that frame. We also assume that we have a coarse estimate of the pose from which Y_2 was captured $\left(\chi_2^E\right)$. This coarse estimate could be obtained from the current GPS/INS estimate χ_2^G, the pose of the previous frame $\hat{\chi}_1$, or the result of a quick feature-based motion estimation algorithm. When registering sequential video frames (30 fps frame rate), simply using the pose of the previous frame as the initial pose estimate $\left(\text{i.e. let } \chi_2^E = \hat{\chi}_1\right)$ was found to produce the most rapid convergence, as the aircraft typically did not move far enough in one frame interval to make this a bad initial guess. The goal of our algorithm is to compute a more refined estimate χ_2^V of the MAV pose when frame Y_2 was captured.

The first step shown in Figure 3.1.1 is to project Y_1 onto a ground image. The projection process assumes that the terrain over which the MAV is flying is planar and horizontal and uses the estimated pose $\hat{\chi}_1$ with respect to this ground plane to produce an ortho-rectified image of the region of ground observed by Y_1. This projection is computed by perspective warping: i.e. the ground image is a perspectively warped version of the captured image. Rather than interpolating between pixel values, a gaussian point spread function is assumed to act on each pixel value, and the perspective projection of this point spread function determines the amount by which each pixel in Y_1 affects each ground image pixel. Further explanation and details of this warping process are given by Dellaert *et al.* [14, 15].

Once we have inferred the appearance of the ground plane using image Y_1, we desire to iteratively refine our initial pose estimate for Y_2. At the kth iteration, we produce a rendered image $Y_2^r(k)$ of this ground image using the current pose estimate $\chi_2^V(k)$ for image Y_2. This rendering process is simply the inverse of the projection process, and is performed using a projectively distorted gaussian point spread function to determine the amount by which each ground pixel affects a given pixel in $Y_2^r(k)$. The rendered image $Y_2^r(k)$ represents the visual information in Y_1 as it would appear in Y_2, assuming that the poses $\hat{\chi}_1$ and $\chi_2^V(k)$ used for projection and rendering were accurate. The difference or residual image $(Y_2^r(k) - Y_2)$ provides information about the error in the current pose estimate $\chi_2^V(k)$. To determine an update $\Delta\chi_2^V(k)$ to the current pose estimate, we use a variant of the popular Lucas-Kanade image

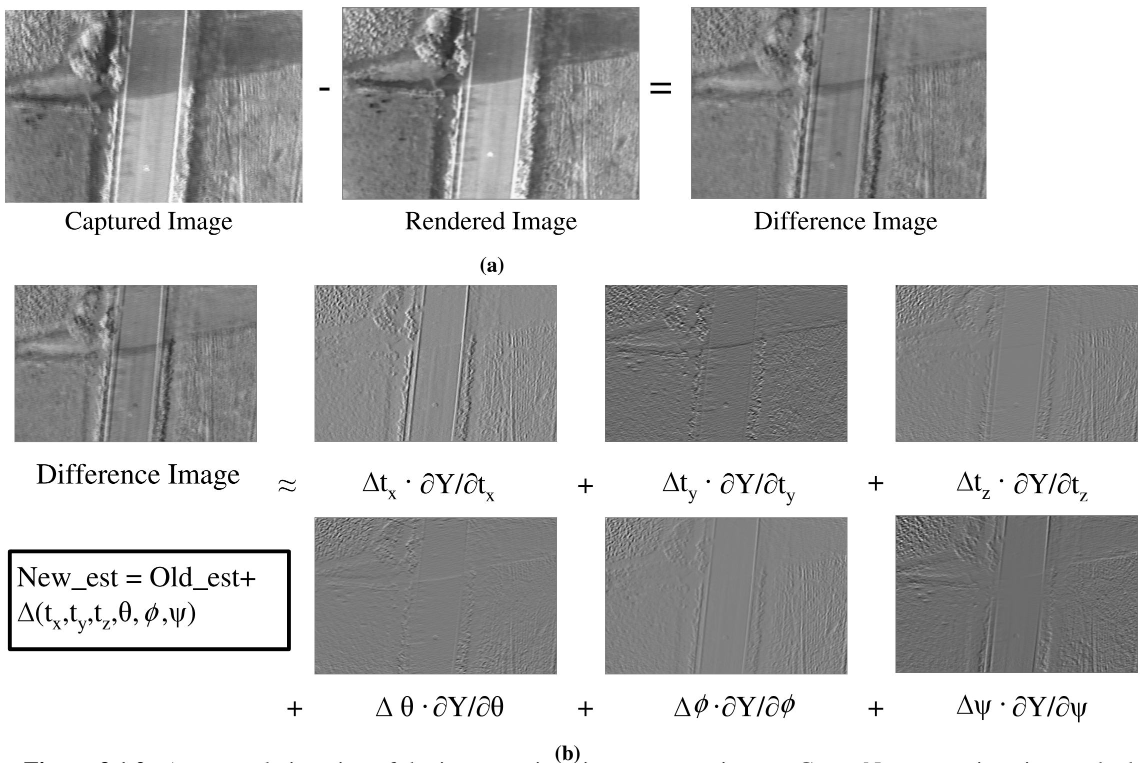


Figure 3.1.2: An example iteration of the image registration process using our Gauss-Newton registration method. In sub-figure (a), a difference image is created to evaluate how accurate the current pose estimate is. In sub figure (b), the new pose (New_est) is computed using the difference image and the Jacobian images.

registration method [39], based on Gauss-Newton gradient descent. We attempt to choose $\Delta\chi_2^V(k)$ to minimize the pixel for pixel squared magnitude of the residual image:

$$J = \sum_{p\in\mathbb{P}} \left(Y_{2,p} - Y_{2,p}^r\right)^2 \tag{3.1.1}$$

where $\mathbb{P}$ is the set of all pixels in image Y_2, $Y_{2,p}^r$ are the pixels in the rendered image and $Y_{2,p}$ are the pixels in Image 2. A Gauss-Newton iteration essentially consists of computing partial derivatives of the residual image with respect to all of the pose parameters. Each of these partial derivative images approximates the change in the residual image caused by a differential change in the associated pose parameter. The goal at each iteration is to express the residual image as a weighted sum of the different Jacobian images, after which the pose is changed according to these weights. A graphical example of a single iteration is shown in Figure 3.1.2.

As discussed in [14, 15], the partial derivatives or "Jacobian Images" of the residual can be approximated using the chain rule as follows:

$$\underbrace{\frac{\partial Y_2^r}{\partial \square}}_{\text{Jacobian Image}} = \underbrace{\frac{\partial Y_2^r}{\partial x}}_{\nabla_x}\frac{\partial x}{\partial \square} + \underbrace{\frac{\partial Y_2^r}{\partial y}}_{\nabla_y}\frac{\partial y}{\partial \square}. \tag{3.1.2}$$

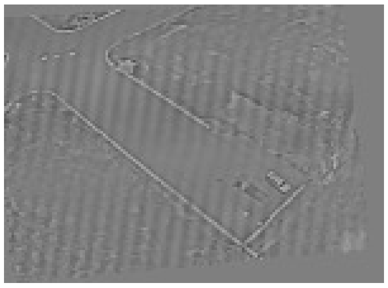
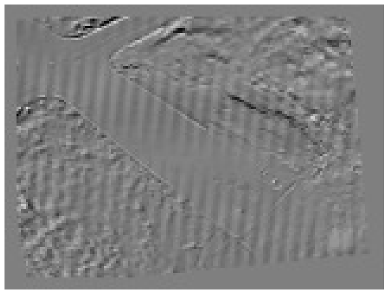



Figure 3.1.3: Residual image differences produced by our direct registration method (left image) and a standard feature-based registration method (right image). Feature-based registration was performed on a 640×480 video sequence, tracking Harris corners between frames using the OpenCV$^{\text{TM}}$toolbox, using RANSAC to estimate a homography matrix relating each frame pair, and warping the first frame to align it with the second. Direct registration was performed using the method indicated in this section on a $4\times$ downsampled version of the same video. Direct registration is able to consistently reduce the minimum mean squared pixel error compared to feature based registration.

Each of the terms in this equation represents an "image" or matrix of values, one for each pixel location. The symbol $\square$ represents one of the six pose parameters $[t_x, t_y, t_z, \psi, \theta, \phi]$, and the terms labeled ∇_x and ∇_y are gradients of the rendered image, i.e. partial derivatives of the luminance function in the vertical and horizontal image directions. The $\frac{\partial y}{\partial \square}$ and $\frac{\partial x}{\partial \square}$ terms represent the differential location change of the image of the preimage of each image point. That is, each pixel location (x, y) is imaging a particular world point P, and a differential change in any pose parameter $(\square)$ will cause a differential change in the (x, y) image coordinates of the projection of P. The $\frac{\partial y}{\partial \square}$ and $\frac{\partial x}{\partial \square}$ terms thus represent the way in which a feature observed at any point in the image will appear to move due to a differential change in the pose parameter $\square$. After multiple iterations like the one shown in Figure 3.1.2, the estimated change $(\Delta\chi_2^V(k))$ in the pose estimate will become very small. At this point, the current pose estimate $\chi_2^V(k)$ becomes the final χ_2^V returned from our VO algorithm.

Figure 3.1.3 demonstrates the potential improvement in registration given by direct registration methods versus more standard feature-based image registration. Since more accurate pixel registration implies more accurate relative pose estimates, improved accuracy can lead to slower growth in VO pose estimates, leading to more accurate overall pose estimates.

3.2 VO Covariance Estimation

As discussed in Section 3, to fuse INS and VO measurements in an EKF framework, we need to be able to find the matrix

$$F = \left.\frac{\partial \mathbf{f}}{\partial \chi}\right|_{\chi=\hat{\chi}_{n-1}} \tag{3.2.1}$$

that linearly approximates the state transition function. The state transition function $\mathbf{f}$ in our EKF is the VO system described in the previous section, which produces the current aircraft pose χ_2^V given the estimated previous pose $\hat{\chi}_1$. Unfortunately, this $\mathbf{f}$ function is not differentiable, and is expressed only as an iterative algorithm, not in mathematical closed form. In this section, we will make some simplifying assumptions that allow us to approximate the Jacobian F

of this algorithm. This will allow us to propagate covariance in pose $\chi_1$to covariance on pose χ_2^V, enabling the fusion of VO and INS measurements.

The final χ_2^V produced by our VO system is a function of both the pose $\hat{\chi}_1$ and the two images Y_1 and Y_2. The interplay of these two images in the iterative image registration algorithm leads to a non-differentiable **f**. Stated differently, changes in χ_2^V can be due either to changes in the quality of image registration between Y_1 and Y_2 or due to changes in the original pose $\hat{\chi}_1$: this fact makes it impossible to differentiate **f** directly. Even if we could parametrize and precisely describe "the quality of image registration" in a meaningful way, differentiating **f** with respect to this parametrization would still involve differentiation of an iterative procedure. To rectify this situation, we will first assume in Section 3.2.1 that the image registration process is able to perfectly register Y_1 with Y_2: this is the same as assuming that all our uncertainty about the final pose χ_2^V is due to propagated uncertainty in $\hat{\chi}_1$. We will find the desired matrix F using this assumption. We will then discuss error due to mis-registration in Section 3.2.2.

3.2.1 Propagating Errors In Image 1 Pose to Image 2

To find the component of uncertainty on χ_2^V (due to propagation of uncertainty from $\hat{\chi}_1$), we need to be able to characterize the function $\mathbf{f}(\cdot)$ such that $\chi_2^V = \mathbf{f}(\hat{\chi}_1)$. We seek to linearize this function so that we can perform a standard linear covariance update $P_2^V = F\hat{P}_1F^T$. To begin, we note that once the iterative image registration process has converged, $\hat{\chi}_1$ and χ_2^V can each be used to compute the homography matrices H_{G1} and H_{G2}, which map pixel locations in images Y_1 and Y_2 respectively to locations on the ground image. These homographies were used respectively in the projection and rendering processes of the VO system (see Section 3.1), and are thus available from the VO process. H_{G1} and H_{G2} individually have uncertainty associated with them, because each of their components is a function of $\hat{\chi}_1$ and χ_2^V, which are imperfectly known and thus have associated covariances. Because homography matrices can be composed by matrix multiplication and are invertible [40, 35], we can combine these two homography matrices into a single homography that maps pixel locations in Y_1 to pixel locations in Y_2 as:

$$H_{12} = H_{G2}H_{G1}^{-1} = H_{G2}H_{1G}. \tag{3.2.2}$$

If, as we have assumed, the registration between Y_1 and Y_2 is perfectly accurate, then this homography H_{12} is perfectly known. Thus, although both H_{G2} and H_{1G} (the inverse of H_{G1}) have associated uncertainty, their product H_{12} does not. This observation is a direct consequence of the fact that VO methods fundamentally measure relative poses: while χ_1 and χ_2 may both be incorrect, the accuracy of the relationship between them is constrained only by the accuracy of image registration.

The relationship in equation 3.2.2 forms the basis of our desired $\mathbf{f}(\cdot)$. We first post-multiply by H_{G1}:

$$H_{G2} = H_{12}H_{G1}. \tag{3.2.3}$$

Our insight about H_{G1},H_{G2}, and H_{12} allows us to write:

$$H_{G2}\left(\chi_2\right) = H_{12}H_{G1}\left(\chi_1\right). \tag{3.2.4}$$

To isolate χ_2 as a function of χ_1 from this last equation, it would be desirable if we could invert the function $H_{G2}\left(\chi_2\right)$; we would like to be able to deduce the pose of a camera given a homography mapping its image points to the ground. We will refer to this inverse function as ξ instead of H_{G2}^{-1}, to emphasize the fact that the inverse we would like is not the matrix inverse of the matrix H_{G2}, but rather a function mapping a homography matrix to a pose. If we could find such a function, we would have our desired formula for $\mathbf{f}$:

$$\begin{aligned} \xi\left(H_{G2}\left(\chi_2^V\right)\right) &= \xi\left(H_{12}H_{G1}\left(\hat{\chi}_1\right)\right), \qquad (3.2.5)\\ \chi_2^V &= \xi\left(H_{12}H_{G1}\left(\hat{\chi}_1\right)\right),\\ &= \mathbf{f}(\hat{\chi}_1). \qquad (3.2.6)\end{aligned}$$

We could then compute the desired derivative F by the chain rule:

$$\frac{\partial \mathbf{f}}{\partial \hat{\chi}_1} = \frac{\partial \chi_2^V}{\partial \hat{\chi}_1} = \frac{\partial \xi}{\partial H_{G2}}\frac{\partial H_{G2}}{\partial H_{G1}}\frac{\partial H_{G1}}{\partial \hat{\chi}_1}. \tag{3.2.7}$$

The function $H_{G1}\left(\hat{\chi}_1\right)$ can already be computed in closed form using standard computer vision techniques, and its derivative, the term $\frac{\partial H_{G1}}{\partial \hat{\chi}_1}$ in equation 3.2.7 is a 9×6 matrix of partial derivatives that can be computed from this closed-form expression in a straightforward manner. The term $\frac{\partial H_{G2}}{\partial H_{G1}}$ is a 9×9 matrix of partial derivatives: these partials are also straightforward to compute, as they are simply elements of H_{12} (see equation 3.2.3).

The only remaining obstacle is the first term in equation 3.2.7, determining the derivative of the ξ function. As discussed in [35], there are well documented methods of decomposing a homography matrix to determine a relative pose; these methods, however, in general produce four solutions among which we must choose based on the cheirality constraint. This fact makes the ξ function non-differentiable. Instead of attempting to differentiate ξ directly, we notice that we can approximate changes in the homography matrix H_{G2} due to changes in χ_2^V using the partial derivative matrix $\frac{\partial H_{G2}}{\partial \chi_2}$, just as we do for H_{G1}. The inverse of this linear mapping, if it existed, would give changes in χ_2^V due to changes in H_{G2} as desired. The inverse does not exist, as the 9×6 matrix $\frac{\partial H_{G2}}{\partial \chi_2^V}$ represents an over-determined system. We can, however, find the pseudo-inverse of the matrix $\frac{\partial H_{G2}}{\partial \chi_2}$, which still maps changes in H_{G2} to changes in χ_2, minimizing error in the elements of H_{G2} . We use this pseudo-inverse to approximate the derivative $\frac{\partial \xi}{\partial H_{G2}}$.

Combining these three terms, we compute the 6×6 matrix of partial derivatives $\frac{\partial \chi_2^V}{\partial \hat{\chi}_1}$ as:

$$\frac{\partial \chi_2^V}{\partial \hat{\chi}_1} = \left(Q^T Q\right)^{-1} Q^T \frac{\partial H_{G2}}{\partial H_{G1}}\frac{\partial H_{G1}}{\partial \hat{\chi}_1}, \tag{3.2.8}$$

where

$$Q = \frac{\partial H_{G2}}{\partial \chi_2^V}. \tag{3.2.9}$$

We now have the desired partial derivative matrix of our VO method, which we can use to approximate covariance propagation.

3.2.2 Determining Covariance With Imperfect Registration

Naturally, the process of determining H_{12} (i.e. registration) is not, as we assumed in the previous subsection, without error. This registration error will cause the difference image to have residual structure (which is not due to parallax) after the registration process completes. We address this source of error by computing the ratio of the residual cost function J of equation 3.1.1 with an empirically determined cost value, and boosting the diagonal values of the covariance matrix P_2^V proportionally. Doing this is similar to the "Q-boosting" technique common in EKF practice: we simply increase our estimated uncertainty on the VO pose such that the overall filter yields desirable results. In practice, this extra boost in the diagonal elements of P_2^V was not found to be necessary to yield good results, and was not used in generating result data.

Chapter 4

Results and Analysis

To evaluate our pose estimation framework, we estimate MAV pose without using GPS information, and show that fusion of VO and INS information allows MAV location to be estimated with accuracy similar to that of GPS for a reasonable period of time.

All results presented in this work are collected using an inexpensive, hand-launchable MAV platform, shown in Figure 4.0.1. The aircraft is a flying wing design, with a 6-foot wingspan, constructed of EPP foam. The on-board Kestrel$^{\text{TM}}$autopilot and associated Virtual Cockpit$^{\text{TM}}$software platform allow the aircraft to autonomously aviate and navigate. The Kestrel$^{\text{TM}}$autopilot comprises a small microcontroller and a collection of sensors that includes three-axis accelerometers, rate gyroscopes, and differential pressure sensors. Pitch, roll, and altitude are estimated on-board from these sensor readings, while 2-D location and heading are estimated using a small on-board GPS receiver [4, 7]. This pose estimate and other telemetry data is transmitted to a ground station at a rate of about 4 Hz. A separate camera/transmitter system collects video footage during flight and transmits this video to the ground station. This video stream is synchronized on the ground station with the stream of pose estimates from autopilot telemetry. VO and sensor fusion are performed off-line using this data.

4.1 Pose Estimation During GPS dropout

A key goal of this work is to explore the use of VO for MAV localization during GPS dropout. To evaluate the accuracy of our fused VO/INS pose estimates, we compare the estimated (t_x, t_y, t_z) location of the aircraft with baseline location measurements produced by the current MAV autopilot pose estimation method, which uses a GPS receiver to measure t_x and t_y, and a differential pressure sensor to measure t_z. We will hereafter refer to these baseline pose estimates as `gpsins` pose estimates. These baseline location estimates are here compared with location estimates from:

1. Our unaided VO system (referred to as `voonly`) shown in Figure 4.1.1
2. Our fusion system, incorporating only VO and INS measurements (referred to as `voins`) shown in Figure 4.1.2.

In each of these figures, we display the path of the MAV as estimated by `gpsins` and by one of the vision-aided fusion schemes, giving both a horizontal and vertical view (sub-figures (a) and (b), respectively), and plot the time-varying distance between these two paths (sub-figure (c)). Clearly the errors in `voonly` location are both much larger than those of `voins`, and increase dramatically over the course of the flight ($\sim$70 seconds). This demonstrates that fusion of VO and INS data can significantly reduce the drift in location estimates that is inherently part of VO

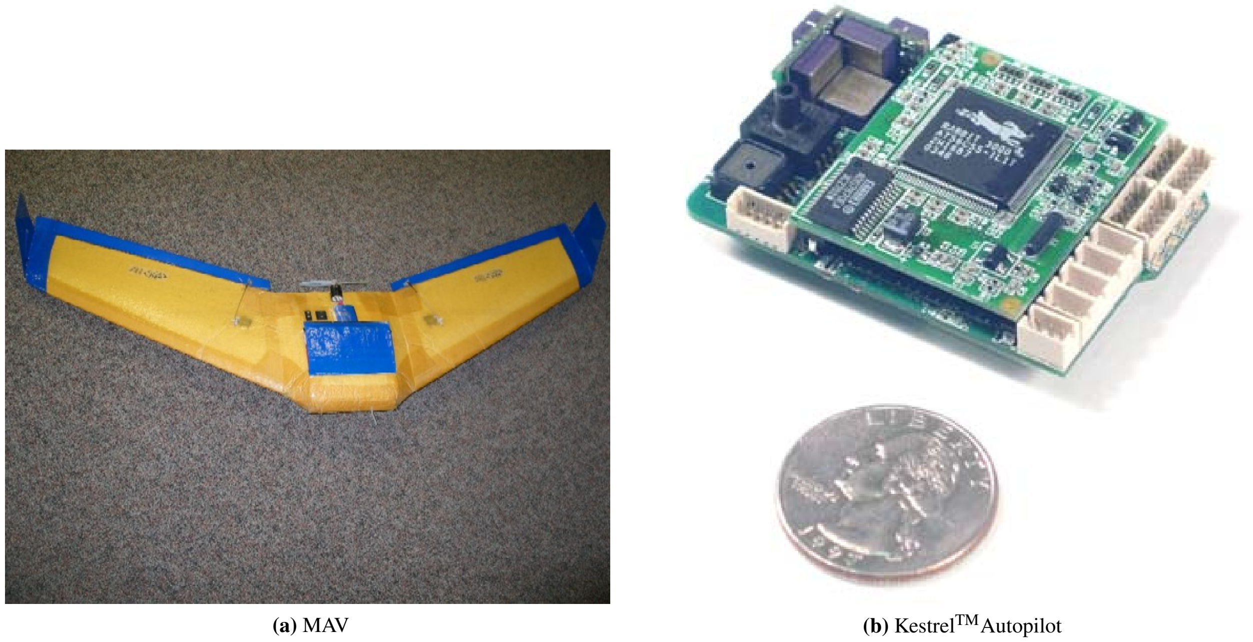
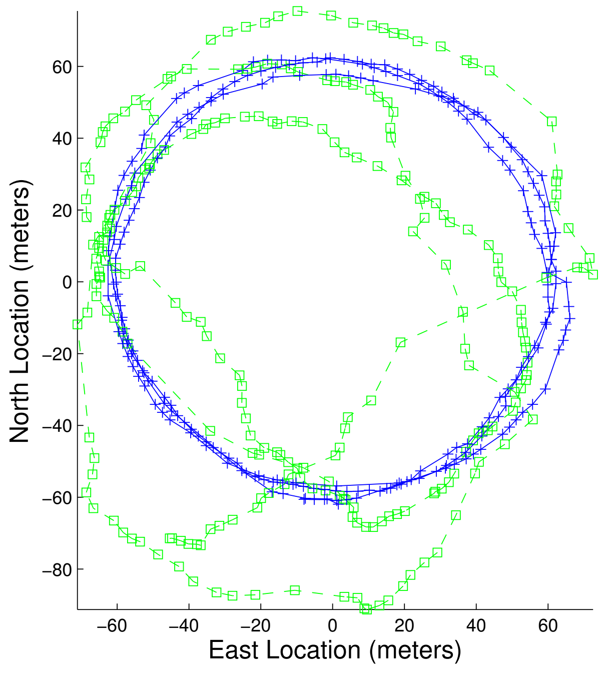

(a) MAV **(b)** Kestrel^{TM}Autopilot

Figure 4.0.1: The MAV platform used in this work. The MAV is a flying wing aircraft constructed of EPP foam with a 6 foot wingspan, controlled by a Kestrel^{TM}autopilot.

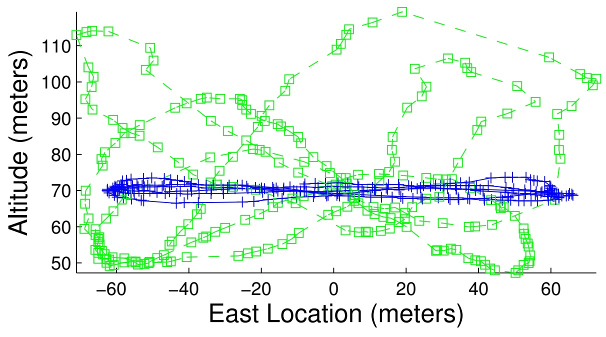
systems, as well as dramatically reducing ($\sim$40%) the worst-case location error. It is also meaningful to realize that the cyclic pattern of errors present in Figures 4.1.1(c) and 4.1.2(c) is likely a consequence of inaccuracies in camera mounting, wind estimation, and temporal data association. It is perhaps remarkable that position can be estimated to within <40m of GPS estimates in the presence of these inaccuracies. As the size, cost, and field use constraints on MAV platforms necessarily make them prone to these errors, the ability to function in their presence is an important benefit for MAV platforms. It should be noted that this error is comparable to the error obtained by some SLAM-based visual/inertial navigation solutions, such as that of Kim and Sukkarieh[9] and Bryson and Sukkarieh[10] (whose flight path is similar to ours). Several other SLAM-based solutions give much better accuracy in simulation (e.g. [41]) or in limited environments with relatively rapid loop closure (e.g. [13]).

4.2 Computational Complexity

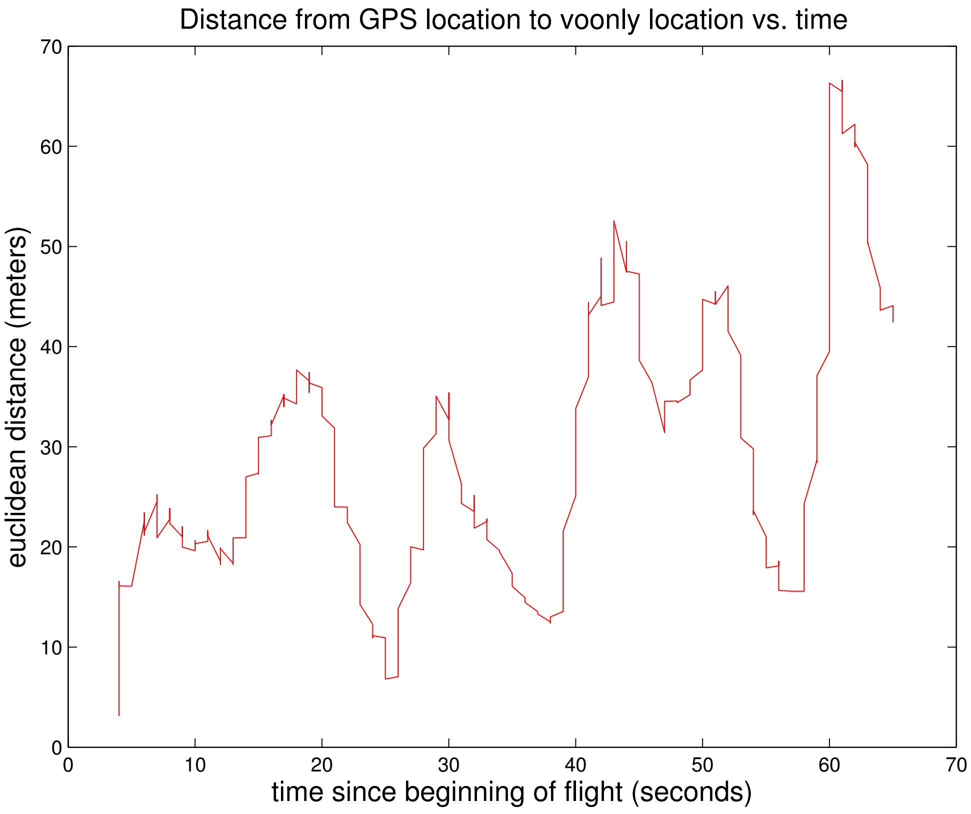
The results given here are presented as a proof of concept. The navigational results shown were obtained using real flight data, but processing was performed offline, in a framework that was not optimized for speed. A number of further optimizations are possible, which we feel would allow our estimation framework to run at a frame rate of about 5Hz or better on computing hardware compatible with MAV weight and power constraints ($\sim$1.5 GHz standard laptop-style CPU). These optimizations could include the following:



(a)

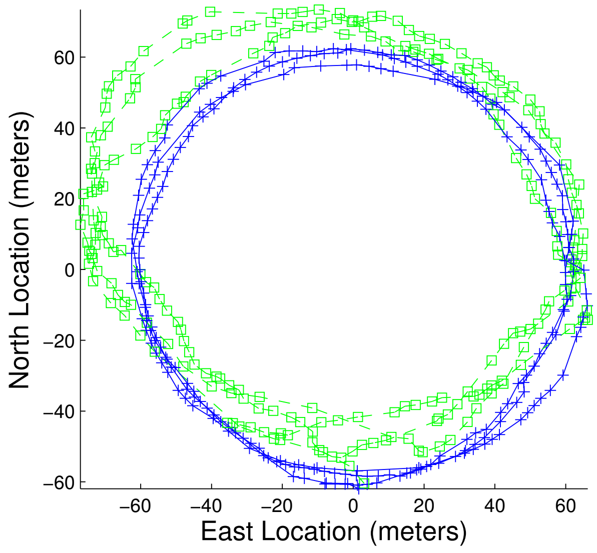


(b)

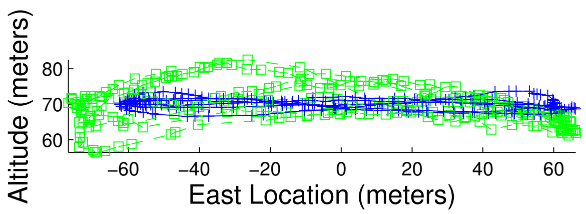


(c)

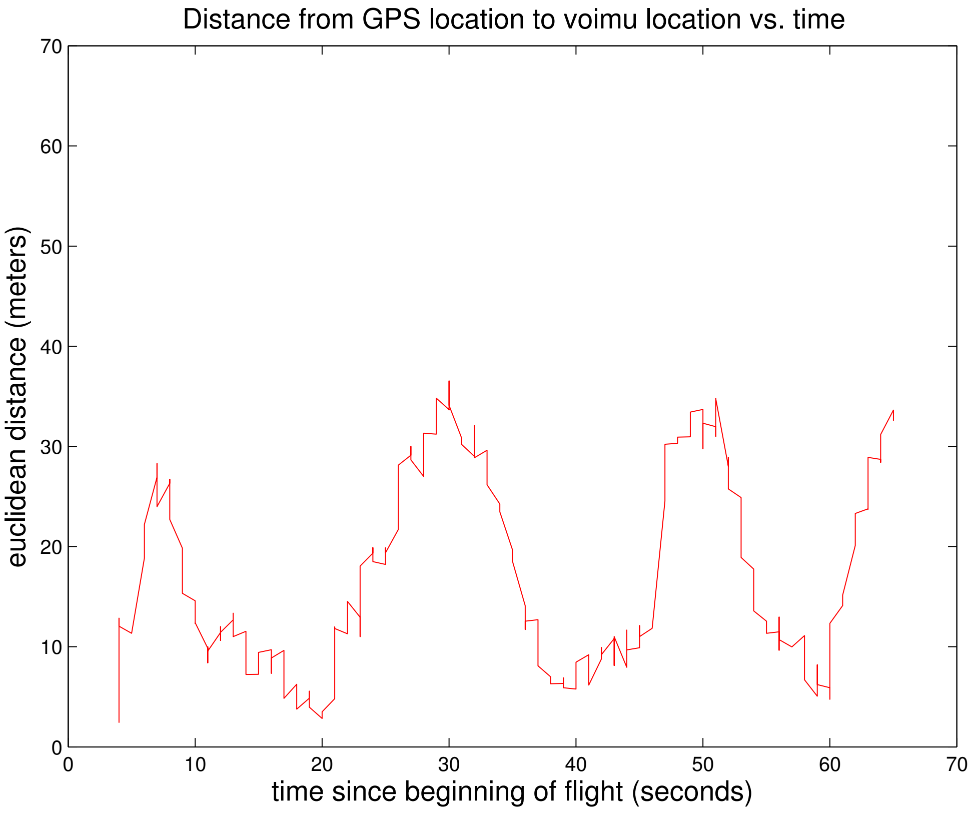
Figure 4.1.1: (a,b) `gpsins` location estimates (blue +) and `voonly` location estimates (green □) in a circular flight path. (c) Euclidean distance between `gpsins` and `voonly` location estimates at each point in time. Note rapid error growth.



(a)



(b)



(c)

Figure 4.1.2: (a,b) `gpsins` location estimates (blue $+$) and `voins` location estimates (green $\Box$) in a circular flight path. (c) Euclidean distance between `gpsins` and `voins` location estimates at each point in time. Notice the slow growth of error as compared to Figure 4.1.1(c).

- The simulation environment is currently coded in MATLAB, and no MEX functions are utilized. A functioning system would need a C/C++ implementation, which in itself would bring performance significantly closer to realtime.

- We currently perform forward additive image registration: that is, the second image is repeatedly warped (requiring expensive recomputation of image Jacobians) until it matches the first image. Baker *et al.* [39] present several ways of improving on this, notably the inverse compositional method, which needs to compute Jacobians only once and applies the inverse of the computed warping to the first image. This could also significantly reduce computational load.

- Not all pixels need actually be compared during the direct registration process: only a sampling of pixels would be needed. This could significantly reduce computation costs.

Memory requirements associated with this method are both fixed and reasonable, as only the previous video frame and the vehicle state estimates need to be stored.

Part II

Kalman Filtering on Riemannian Manifolds

Chapter 5

Motivation and Related Work

As we have discussed in Chapter 1, the Extended Kalman Filter (EKF) has been widely used in nonlinear filtering applications for over half a century. While theoretically suboptimal in general, the EKF is relatively easy to implement, and very tolerant of the approximations inherent in its use. Nevertheless, the EKF often has difficulties with highly nonlinear systems. Depending on the details of the system equations, the EKF can sometimes produce poor estimation results, and is further prone to becoming either inconsistent, producing incorrect results with high confidence, or else uninformative, offering an estimate with such high uncertainty as to make it effectively meaningless. Inconsistency is a particularly severe problem when the state and measurement models of the system in question are highly nonlinear. The nonlinear nature of the system will cause differences from predicted values which are completely predictable from available information, but differ from the approximate predictions given by the linearized models of the EKF. These differences are incorrectly assumed to represent valid information about the state, and cause the EKF to become overconfident in its estimates, which can eventually lead to the filter simply ignoring all incoming measurements and giving wildly inaccurate estimates, an effect known as filter divergence.

The literature documents many variants of the EKF designed to improve its performance for nonlinear systems (e.g. [42, 43, 44]). Some of these works deal with nonlinear systems in general. There are also large bodies of work on specific problems. For example, the estimation of the attitude of a rigid body has been the subject of a large amount of work spanning many decades, and a vast array of filtering strategies have been developed for this problem (see Section 5.1.2).

While there can be many sources of nonlinear behavior in system models, one very common source is that often the state being estimated is not a linear vector space, as the original Kalman Filter derivation assumes. Instead, these points often lie on Riemannian Manifolds, more general "curved" spaces. For example:

- The attitude of a rigid body (such as an aircraft or spacecraft) is a member of the set of all right-handed orthogonal rotations, called the *3-dimentional Special Orthogonal Group,* or SO^3. This is perhaps the most encountered and most studied manifold, not least because of its ubiquity across a wide field of applications.

- The pose (position and attitude) of a rigid body belongs to *3-dimensional Special Euclidean Group, or* SE^3, which is simply the combination of SO^3 and the familiar 3-dimensional linear vector space $\mathbb{R}^3$.

- Optical cameras and other bearing-only sensors inherently measure angles to a feature. This can be considered to belong to either the two-dimensional projective space P^2 (the set of all lines through the origin) or else to the surface of a two-dimensional sphere, denoted S^2.

- A commonly considered problem in computer vision is that of a pair of cameras imaging a planar surface. The pixel coordinates of a point in either camera's image will be related to the pixel coordinates of that point in the other camera's image by an invertible linear mapping termed a Homography matrix. Homography matrices are 3×3 invertible matrices, having in fact only eight degrees of freedom, and form an eight-dimensional manifold.

Manifolds lack some of the structure which linear vector spaces possess, making the filtering problem more difficult. In particular, two points in a manifold space cannot be meaningfully "added" together: i.e. vector addition and subtraction are not defined on manifolds in general. Since these operations provide the most common means of measuring distance on a vector space, the distance between two points on a manifold must be developed separately. Going a step further, such basic statistical operations as finding the mean and covariance of a set of points rely on the notion of integration and hence of summation. These facts present one with difficulties in applying the EKF to manifold-valued problems.

In general, the difficulties associated with estimating manifold-valued quantities have been overcome in one of two ways. Some works represent manifold quantities by embedding them in a higher dimensional space: for example, using a unit-length quaternion[45] to represent a rotation in SO^3. This method nominally renders the manifold as a vector space, and allows standard estimators like the EKF to be mostly re-used, but introduces the complication that the state must be constrained to lie on the manifold surface: not every 4-vector is a valid quaternion.

Other methods enforce the manifold constraint implicitly, either by using a minimal parameterization of some sort which guarantees conformance to the manifold surface, or by modifying the Kalman filter equations in a way that take the underlying structure of the manifold into account. These methods can perform unconstrained estimation, with the conceptual overhead of needing to account for manifold structure. SO^3 is clearly the manifold most studied in works of this type. Key papers of both types are discussed in more detail in Section 5.1, below.

In this work, we propose a new EKF-family filter, the Manifold Extended Kalman Filter (Manifold EKF). This filter parameterizes the space so as to guarantee adherence to the manifold constraint surface, and thus falls into the second category. Rather than being limited, however, to a particular manifold such as SO^3, the Manifold EKF is a general approach to filtering that works on a very broad class of manifolds. As we will demonstrate in Chapter 10, it has superior performance and consistency characteristics compared to the standard EKF, due to the fact that the manifold nature of the state and measurement spaces are accounted for in the derivation of the filter itself, rather than added on later. This filter can be viewed as a general extension of the EKF to manifold valued problems: it utilizes notions of "distance", "mean", and "covariance" that are generalized to a large and useful class of manifolds.

Our proposed method builds on the work of Pennec *et al.* [46, 47, 48], who have developed generalized versions of distance metrics and statistical measures for manifolds, as well as the recent work of Humpherys *et al.* [2, 3], who present a particularly useful derivation of the Extended Kalman Filter as a set of sub-optimal modifications to a least-squares estimation problem.

5.1 Related Work

The available literature on estimation and filtering is quite vast, and even a thorough summary is well beyond the scope of this work. We will instead discuss several key publications that are related to the Manifold EKF. We will first discuss the vast field of attitude estimation, much of which (historically at least) comes from the spacecraft community. We will then discuss general methods for nonlinear filtering, which can be applied to manifold estimation problems, as well as methods which address the constrained and re-parameterized approaches to the Kalman Filter for manifolds.

5.1.1 Nonlinear Filtering

Several general nonlinear filtering methods can be used to help solve the problems of estimating attitude. These include approaches using the Unscented Kalman Filter, Particle Filters, and filters that numerically propagate a state pdf based on the Fokker-Planck equation. Descriptions of the entire field of nonlinear filtering are well beyond the scope of this work, and useful surveys are given by Crassidis et al. [49, 50] as well as Daum [1].

A number of extensions to the EKF are also very useful in terms of manifold value estimation. More directly in the EKF family of filters is Psiaki's Backward Smoothing EKF [51], which maintains a sliding window of past states rather than a single state estimate. As we shall see in Chapter 6 the EKF, which estimates only the current state of a system, can be viewed as a suboptimal simplification of a full least squares estimator which estimates the entire state history. The Backward Smoothing EKF can be thought of as "bridging the gap" between these two extremes.

5.1.2 Attitude Estimation

SO^3 is without a doubt the most studied manifold in terms of estimation. This is understandable, as the attitude of a rigid body is a basic component of many physical systems. A survey of attitude estimation methods is given in [49, 50], and a helpful survey of the various parameterizations of SO^3 used is given in [45].

Many early works began with the least squares cost function defined by Wahba [52], which was motivated by the need for estimating the attitude of spacecraft. Wahba's problem seeks to find a rotation matrix between the spacecraft body frame and another frame, given pairs of vectors whose values are known or measured in each frame. It minimizes the squared difference between body-frame vectors and their counterparts rotated into the body frame. Early work on this problem notably produced the QUEST estimator [53, 54, 55], which solves for the rotation matrix in the form of a unit quaternion, by finding the eigenvalue of a sum of outer product matrices. This can also be done using the SVD [56], which gives covariance information about the resulting solution. Later work extends the QUEST algorithm into filtering frameworks [57, 58, 59]. Psiaki's extended QUEST framework, in particular, is a least-squares optimization based recursive filtering framework which is quite general in that it can estimate other states in addition to quaternion values.

While all of these methods provide a framework to deal with the estimation of values on SO^3, and possibly even other vector-space values, they do not provide a framework for estimating values on other manifolds. If other manifolds, such as S^2 or P^2 are involved in a system's equations, there are a number of more general filtering methods which provide a way to deal with quantities on any manifold.

5.1.3 Constraint-based Kalman Estimation of Manifold Quantities

One general way of using a Kalman-style filter to estimate manifold quantities is to view the manifold as a smooth surface embedded in a higher-dimensional space. The Whitney theorem[60, 61] guarantees that this is always possible for any manifold, and that the embedding space will need to have a dimension of at most $2N+1$, where N is the dimension of the manifold itself. The space S^2, for instance, is inherently a two-dimensional space, but is readily visualized as the surface of the unit sphere in $\mathbb{R}^3$. A filter using this method may thus use a three-vector to represent a point in S^2. Standard Extended Kalman Filtering equations can be used in this case, with the additional complication that there is nothing to guarantee that the state estimate will remain on the surface of the sphere. In fact, the mean of several unit-length vectors will always have a norm that is less than or equal to unity. Therefore, some extra modifications to the EKF framework are necessary to ensure that the manifold surface constraint is obeyed.

Historically, two main methods were used to enforce manifold constraints, often referred to as the projection and pseudo-measurement approaches.

The projection approach performs the standard (Extended) Kalman Filter method, but adds a "reprojection" step after each measurement update, which forces the state estimate back onto the manifold. For example, a filter whose state space is S^2 might estimate the state as a 3-vector in the standard way, but then normalize the vector after each step. This approach was first described in the EKF literature in [62], and several other papers deal with this topic
.

The pseudo-measurement approach works by adding an extra "measurement" to the system, with zero measurement uncertainty, in order to enforce the manifold constraint. Thus, our example system working on S^2 might "measure" the difference between the length of the vector and unity, with the result being always forced to zero, with zero uncertainty. This "measurement" is treated in the EKF framework like any other measurement, and tends to keep the estimate on the manifold surface. This method is as old as the Kalman Filter itself, with Kalman's original paper giving an example using a measurement with zero covariance [63]. Several later works expanded this method for use in nonlinear systems, and to inequality constraints.

More recently, Juler and LaViola[42] published an important paper that arguably describes the state of the art in terms of constrained Kalman Filtering. They discuss two types of constraints: one which guarantees that the distribution is constrained to the manifold, and one which guarantees that the mean of the distribution is constrained to the manifold. For a manifold such as S^2, enforcing the former constraint will usually cause the latter constraint to be violated. They prove that for general nonlinear problems, the pseudo-measurement approach does not enforce either of these constraints, and therefore use a projection-based method. They implement their technique using the Unscented Kalman Filter [64], adding a step after the measurement update which first projects each individual sample point onto the constraint surface, computes the mean, and then projects that mean onto the constraint surface, appropriately increasing the covariance in the process.

5.1.4 Parameterization-based Kalman Estimation of Manifold Quantities

Another way to allow the Kalman Filter to operate on manifolds is to choose a minimal parameterization of the manifold, such that every possible state is guaranteed to be a manifold point. Perhaps the most common example is to use yaw, pitch, and roll angles to parameterize SO^3, rather than the quaternions often used in constraint-based filters. Another common example is the use of two spherical coordinate angles rather than unit-length 3-vectors to parameterize points in S^2.

There have been many works that have applied this principle in specific applications, such as vision-based motion estimation [65], medical imaging [44], and platform attitude estimation [43]. We will discuss some of these methods which relate particularly closely to our method. Our proposed method is a general extension of the EKF to a particular class of manifolds, and is thus applicable to a broader class of problems than these works address individually. More specific differences between these method and our proposed Manifold EKF will be discussed on a case-by-case basis.

In many ways, the special-case filter described by [44] uses a similar approach to ours. This work uses some of the statistical tools described in [46, 47, 48] to construct a filter whose state space is a particular application-specific manifold. Rather than defining and minimizing manifold distance, however, it chooses a particular base point and approximates measurement errors as the difference between two vectors based at that point. As will be discussed in Chapter 7, this amounts to using distance on a particular chart in place of distance on the manifold, which will mean that the filter will return a different answer if a different base point is chosen. Put differently, the distance which the EKF is attempting to minimize will change in ways that do not match physical intuition, which will introduce error into this technique.

The Multiplicative Extended Kalman Filter of [43] deals with the manifold SO^3, meaning it gives a way to estimate the attitude of a rigid body. It represents attitude by using both a unit quaternion and a three-dimensional representation such as the rotation vector or (modified) Gibbs vector[66]. This three-dimensional, minimal representation is not able to uniquely represent every value on the manifold, but it is unconstrained in that any collection of three numbers does correspond to a rotation (i.e. it is onto, though not one-to-one). Thus, this minimal representation and an associated covariance matrix represent the information in the filter, while the quaternion shifts this information to be centered at the proper point. In this way, the rotation antipodal to the base point, which is not uniquely representable, is always kept very far away (in fact approximately antipodal to) the filter's region of interest. If the measurement space is also SO^3 (as when quaternion measurements are available from a star tracker on a satellite), then the MEKF finds the tangent vector to the predicted measurement and uses this as the innovation vector in a standard Kalman Filter update.

In some ways, our work could be viewed as generalizing the MEKF to manifolds other than SO^3. Specifically, the MEKF properly uses the logarithmic mapping on SO^3 to determine an error vector, meaning its distance metric results will match physical intuition in that it will be invariant under rotational shifts. Aside from applicability to only a single manifold, the MEKF differs from our approach in that it always uses only a single iteration to estimate the updated state, whereas our filter applies multiple iterations. As will be seen in Chapter 10, the number of iterations has a significant effect on the performance of our filter.

5.2 Outline

Chapter 6 is devoted to providing a derivation of the Extended Kalman Filter as a sub-optimal modification of a least-squares estimation problem, using the derivation strategy of [2, 3]. We will then proceed in Chapter 7 to briefly sketch the manifold development of Pennec *et al*. With these two frameworks in place, we will devote Chapter 8 to the derivation of the key equations of the Manifold EKF. In Chapter 9 we describe the implementation of the basic set of manifold operations on several important manifolds. Finally, we examine the advantages and performance of the Manifold EKF in Chapter 10.

Chapter 6

The EKF As A Suboptimal Least Squares Estimator

6.1 The Extended Kalman Filter

This chapter presents a derivation of the EKF which is due to [2, 3]. This derivation is useful in many ways, not the least of which being that it highlights the relationship between the EKF and the full nonlinear least squares problem. This derivation is presented here for two reasons. First, it provides a novel way to understand the well-known sub-optimality of the Extended Kalman Filter, which gives some insight into the EKF's performance and how it can be improved. Second, this derivation framework can be easily generalized for manifolds, and thus an analogous method will be used to derive the Manifold EKF in Chapter 8.

6.1.1 System and Notation

The (discrete-time) EKF can be thought of as an estimator of the state history of a **dynamic system**. Such a system is described at any particular time index k by an N-dimensional **state** vector x_k[1]. This state evolves according to the **system dynamic function** $f_k(x_{k-1})$, which computes the value of the k-th state from the $(k-1)$th state:

$$x_k = f_k(x_{k-1}) + v_k. \tag{6.1.1}$$

Note that there can in general be a different system dynamic function at each time step k. v_k is a zero-mean random variable with covariance

$$Q_k = E\left[v_k v_k^T\right]. \tag{6.1.2}$$

Often, it is assumed that v_k is a Gaussian process (that is, that being zero mean with covariance Q_k completely specifies the statistics of v_k). This this is not actually an assumption made by the EKF, which requires only that v_k have zero mean and be uncorrelated with the state and other noise sources.

The complete state history of such a system up to time index k will be denoted $\mathbf{X}_k$, where

$$\mathbf{X}_k = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_k \end{bmatrix} \tag{6.1.3}$$

[1]We are here assuming a discrete-time system.

is a vector of length $(k+1)\cdot N$. We will also find it convenient to define the aggregate system dynamic function $\mathbf{f}_k(\mathbf{X}_k)$, which produces $\mathbf{X}_k$ from $\mathbf{X}_{k-1}$:

$$\begin{aligned}\mathbf{X}_k &= \mathbf{f}_k(\mathbf{X}_{k-1}) \\ &\doteq \begin{bmatrix} x_0 \\ f_1(x_0) \\ f_2(x_1) \\ \vdots \\ f_k(x_{k-1}) \end{bmatrix} + \begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ \vdots \\ v_k \end{bmatrix}\end{aligned} \tag{6.1.4}$$

as well as the aggregate $(k+1)\cdot N \times (k+1)\cdot N$ block-diagonal matrix $\mathbf{Q}_k$:

$$\mathbf{Q}_k = \begin{bmatrix} Q_0 & & & \mathbf{0} \\ & Q_1 & & \\ & & \ddots & \\ \mathbf{0} & & & Q_k \end{bmatrix}. \tag{6.1.5}$$

The state of the system is in general not directly observable. Rather, information about the value of x_k must be inferred by observing y_k, an M-dimensional vector of measured values. The complete measurement history is denoted $\mathbf{Y}_k$, where

$$\mathbf{Y}_k \doteq \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_k \end{bmatrix} \tag{6.1.6}$$

is a vector of length $k\cdot M$. Note that, for simplicity, we will assume that there is one measurement for every entry in the state history except for x_0, though relaxing this assumption is straightforward. These measurements can be predicted from the state history by means of the **system measurement function**, h_k:

$$y_k = h_k(x_k) + w_k. \tag{6.1.7}$$

Like v_k, w_k is zero-mean noise with covariance $R_k = E\left[w_k w_k^T\right]$, which is uncorrelated with v_k. We here make the common Markov assumption that y_k depends only on x_k and not on previous states $(x_{k-1}, x_{k-2}, \cdots)$.

As with the dynamic function, we will find it convenient to define $\mathbf{h}_k(\mathbf{X}_k)$, which produces $\mathbf{Y}_k$ given $\mathbf{X}_k$:

$$\mathbf{Y}_k = \mathbf{h}_k(\mathbf{X}_k) \doteq \begin{bmatrix} h_1(x_1) \\ h_2(x_2) \\ \vdots \\ h_k(x_k) \end{bmatrix} + \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_k \end{bmatrix} \tag{6.1.8}$$

as well as the $k \cdot M \times k \cdot M$ block matrix $\mathbf{R}_k$:

$$\mathbf{R}_k = \begin{bmatrix} R_1 & & & \mathbf{0} \\ & R_2 & & \\ & & \ddots & \\ \mathbf{0} & & & R_k \end{bmatrix}. \tag{6.1.9}$$

The EKF derivation will involve the computation of the Jacobians of the dynamic and measurement functions, respectively. We thus define Jacobians of f_k and h_k as

$$F_k \doteq \frac{\partial f_k(x_{k-1})}{\partial x_{k-1}}, \tag{6.1.10}$$

$$H_k \doteq \frac{\partial h_k(x_k)}{\partial x_k}. \tag{6.1.11}$$

6.1.2 Cost Function

We will begin by setting up a least squares problem to estimate the state history $\mathbf{X}_k$ of the system. We seek to find $\mathbf{X}_k$ to minimize the following cost function:

$$\begin{aligned} & 2S(\mathbf{X}_k) \\ = \; & \left(\mathbf{X}_0 - \hat{\mathbf{X}}_0\right)^T \mathbf{Q}_0^{-1} \left(\mathbf{X}_0 - \hat{\mathbf{X}}_0\right) + (\mathbf{X}_k - \mathbf{f}_k(\mathbf{X}_{k-1}))^T \mathbf{Q}_k^{-1} (\mathbf{X}_k - \mathbf{f}_k(\mathbf{X}_{k-1})) \\ & + (\mathbf{Y}_k - \mathbf{h}_k(\mathbf{X}_k))^T \mathbf{R}_k^{-1} (\mathbf{Y}_k - \mathbf{h}_k(\mathbf{X}_k)) \end{aligned} \tag{6.1.12}$$

$$\begin{aligned} = \; & \underbrace{(x_0 - \hat{x}_0)^T Q_0^{-1} (x_0 - \hat{x}_0)}_{A} + \sum_{k=1}^{K} \underbrace{(x_k - f_k(x_{k-1}))^T Q_k^{-1} (x_k - f_k(x_{k-1}))}_{B} \\ & + \underbrace{(y_k - h_k(x_k))^T R_k^{-1} (y_k - h_k(x_k))}_{C}. \end{aligned} \tag{6.1.13}$$

The terms in Equation 6.1.13 describe the distance between values in X_k and Y_k and available estimates thereof:

1. the Mahalanobis distance (term A) between the initial state x_0 and its *a-priori* estimate $\hat{x}_0$
2. the Mahalanobis distance (term B) between each state x_k and $f_k(x_{k-1})$, its value as predicted by the system dynamic model

3. the Mahalanobis distance (term C) between each past measurement y_k and $h_j(x_k)$, its value as predicted by the measurement model

In some sense the minimization of this cost function is the "correct" problem to solve: since the prior estimate $(\hat{x}_0, Q_0)$, the dynamic model $(f_k(\cdot), Q_k)$, the measurement model $(h_k(\cdot), R_k)$ and the measured values y_k are the only information available for the system, minimizing the squared error between actual and predicted values is in many cases a useful problem to solve.

Since each successive new value of k will only add extra terms to this cost function, we can re-write it in a recursive fashion:

$$\begin{aligned} S(\mathbf{X}_k) &= S(\mathbf{X}_{k-1}) \\ &\quad + \frac{1}{2}(x_k - f_k(x_{k-1}))^T Q_k^{-1}(x_k - f_k(x_{k-1})) \\ &\quad + \frac{1}{2}(y_k - h_k(x_k))^T R_k^{-1}(y_k - h_k(x_k)). \end{aligned} \tag{6.1.14}$$

In order to determine a new estimate of $\mathbf{X}_k$, the central idea is to begin with an initial estimate and use Newton's method to find a critical point of the gradient $\mathscr{J}_k$ of the cost function S_k. This will involve computation of the Hessian $\mathscr{H}_k$. Of course, the standard caveat of iterative nonlinear estimation applies: the non-quadratic nature of S_k (due to the nonlinearity of the functions f_k and h_k) means that we are not guaranteed to converge to the global optimum of S_k.

6.1.3 Jacobian and Hessian of Cost Function

We can easily derive an expression for the gradient. We will express this in two block rows, the first containing derivatives with respect to prior states and the second the derivative with respect to the most recent state only:

$$\mathscr{J}_k = \begin{bmatrix} \frac{\partial S_k(\mathbf{X}_k)}{\partial \mathbf{X}_{k-1}} \\ \frac{\partial S_k(\mathbf{X}_k)}{\partial x_k} \end{bmatrix} = \begin{bmatrix} \left(\frac{\partial S(\mathbf{X}_{k-1})}{\partial \mathbf{X}_{k-1}} + \begin{bmatrix} \mathbf{0}_{(k-2)\cdot N} \\ -F_k^T Q_k^{-1}(x_k - f_k(x_{k-1})) \end{bmatrix} \right) \\ \left(Q_k^{-1}(x_k - f_k(x_{k-1})) - H_k^T R_k^{-1}(y_k - h_k(x_k)) \right) \end{bmatrix} \tag{6.1.15}$$

and the Hessian:

$$\begin{aligned}
\frac{\partial \mathscr{J}_k}{\partial X_k} &= \begin{bmatrix} \frac{\partial \mathscr{J}_k}{\partial \mathbf{X}_{k-1}} & \frac{\partial \mathscr{J}_k}{\partial x_k} \end{bmatrix} \qquad (6.1.16)\\
&= \underbrace{\begin{bmatrix} \left(\frac{\partial S(\mathbf{X}_{k-1})}{\partial \mathbf{X}_{k-1} \partial \mathbf{X}_{k-1}} + \begin{bmatrix} \mathbf{0}_{(k-2)\cdot N,(k-2)\cdot N} & \mathbf{0}_{(k-2)\cdot N} \\ \mathbf{0}^T_{(k-2)\cdot N} & F_k^T Q_k^{-1} F_k \end{bmatrix} \right) & \begin{bmatrix} \mathbf{0}_{(k-2)\cdot N} \\ -F_k^T Q_k^{-1} \end{bmatrix} \\ \begin{bmatrix} \mathbf{0}_{(k-2)\cdot N} & -Q_k^{-1} F_k \end{bmatrix} & \left(Q_k^{-1} + H_k^T R_k^{-1} H_k \right) \end{bmatrix}}_{\text{first order terms}} \\
&+ \underbrace{\begin{bmatrix} \begin{bmatrix} \mathbf{0}_{(k-2)\cdot N,(k-2)\cdot N} & \mathbf{0}_{(k-2)\cdot N} \\ \mathbf{0}^T_{(k-2)\cdot N} & -\nabla^2 f_k Q_k^{-1} \left(x_k - f_k\left(x_{k-1}\right) \right) \end{bmatrix} & \mathbf{0}_{(k-1)\cdot N} \\ \mathbf{0}^T_{(k-1)\cdot N} & -\nabla^2 h_k R_k^{-1} \left(y_k - h_k\left(x_k\right) \right) \end{bmatrix}}_{\text{second order terms}}.
\end{aligned}$$

An iterative least squares solution can then be found by choosing an initial estimate of $\mathbf{X}_k$, evaluating $\frac{\partial \mathscr{J}_k}{\partial X_k}$ and $\mathscr{J}_k$ at that estimate, and forming the next estimate using Newton's method:

$$\mathbf{X}_k^{i+1} = \mathbf{X}_k^i - \left(\frac{\partial \mathscr{J}_k}{\partial X_k} \right)^{-1} \mathscr{J}_k\left(\mathbf{X}_k^i\right). \qquad (6.1.17)$$

6.1.4 EKF Approximations

Thus far, the method we describe is simply a least squares method taking the dynamics and measurements into account, sometimes referred to as bundle adjustment[67]. The EKF equations can be arrived at from this problem by making the following simplifications:

6.1.4.1 Gauss-Newton Iteration

We use a Gauss-Newton iteration, instead of the full Newton's method. That is, we ignore the second derivative terms in the Hessian, using instead the approximate Hessian $\mathscr{H}_k$:

$$\mathscr{H}_k = \begin{bmatrix} \left(\frac{\partial S(\mathbf{X}_{k-1})}{\partial \mathbf{X}_{k-1} \partial \mathbf{X}_{k-1}} + \begin{bmatrix} \mathbf{0}_{N(k-2),N(k-2)} & \mathbf{0}_{N(k-2)} \\ \mathbf{0}^T_{N(k-2)} & F_k^T Q_k^{-1} F_k \end{bmatrix} \right) & \begin{bmatrix} \mathbf{0}_{N(k-2)} \\ -F_k^T Q_k^{-1} \end{bmatrix} \\ \begin{bmatrix} \mathbf{0}_{N(k-2)} & -Q_k^{-1} F_k \end{bmatrix} & \left(Q_k^{-1} + H_k^T R_k^{-1} H_k \right) \end{bmatrix} \qquad (6.1.18)$$

where $\mathscr{H}_{k-1}\left(\hat{\mathbf{X}}_{k-1}\right)$ represents only the first order terms of the second derivative

$$\left. \frac{\partial S\left(\mathbf{X}_{k-1}\right)}{\partial \mathbf{X}_{k-1} \partial \mathbf{X}_{k-1}} \right|_{\mathbf{X}_{k-1} = \hat{\mathbf{X}}_{k-1}}. \qquad (6.1.19)$$

The use of a Gauss-Newton approximation to the Hessian matrix rather than the full Hessian will not in general affect the validity of the solution. In fact, there is an entire class of iterative minimization algorithms that use various approximations to the Hessian matrix: the specific approximation will affect the performance of the solver, but will in general not cause the algorithm to converge to an incorrect result. Of course, any nonlinear solver algorithm could converge to a local minimum of the cost function, and this is true even if Newton's method were used directly.

6.1.4.2 Prior Initial Value

For the initial value of $\mathbf{X}_k$, we choose its prior estimate, which consists of the prior estimate of all past states $\hat{\mathbf{X}}_{k-1}$, along with the value of x_k predicted using the system dynamic model:

$$\hat{\mathbf{X}}_k = \begin{bmatrix} \hat{\mathbf{X}}_{k-1} \\ \hat{x}_k \end{bmatrix} = \begin{bmatrix} \hat{\mathbf{X}}_{k-1} \\ f_k(\hat{x}_{k-1}) \end{bmatrix} \tag{6.1.20}$$

evaluating the gradient and Hessian at this point greatly simplifies their form. This is the case because we are using the prior estimate $\hat{\mathbf{X}}_{k-1}$, *which is assumed to be the minimizer* of the prior cost function S_{k-1}. Letting

$$\hat{H}_k \doteq \left.\frac{\partial h_k(x_k)}{\partial x_k}\right|_{x_k=\hat{x}_k} \tag{6.1.21}$$

and

$$\hat{F}_k \doteq \left.\frac{\partial f_k(x_{k-1})}{\partial x_{k-1}}\right|_{x_{k-1}=\hat{x}_{k-1}}, \tag{6.1.22}$$

we can find the simplified value of the Jacobian:

$$\begin{aligned} \mathscr{J}_k(\hat{\mathbf{X}}_k) &= \begin{bmatrix} \left(\underbrace{\frac{\partial S(\mathbf{X}_{k-1})}{\partial \mathbf{X}_{k-1}}}_{\mathbf{0}} + \begin{bmatrix} \mathbf{0}_{N(k-2)} \\ -\hat{F}_k^T Q_k^{-1} \underbrace{(f_k(\hat{x}_{k-1}) - f_k(\hat{x}_{k-1}))}_{\mathbf{0}} \end{bmatrix} \right) \\ \left(Q_k^{-1} \left(\underbrace{f_k(\hat{x}_{k-1}) - f_k(\hat{x}_{k-1})}_{\mathbf{0}} \right) - \hat{H}_k^T R_k^{-1} (y_k - h_k(f_k(\hat{x}_{k-1}))) \right) \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{0}_{N(k-1)} \\ -\hat{H}_k^T R_k^{-1} (y_k - h(f_k(\hat{x}_{k-1}))) \end{bmatrix}. \end{aligned} \tag{6.1.23}$$

We similarly find a simplified Hessian:

$$\mathscr{H}_k(\hat{\mathbf{X}}_k) = \begin{bmatrix} \left(\mathscr{H}_{k-1}(\hat{\mathbf{X}}_{k-1}) + \triangle_1\right) & \begin{bmatrix} \mathbf{0}_{N(k-2)} \\ -\hat{F}_k^T Q_k^{-1} \end{bmatrix} \\ \begin{bmatrix} \mathbf{0}_{N(k-2)} & -Q_k^{-1}\hat{F}_k \end{bmatrix} & \left(Q_k^{-1} + \hat{H}_k^T R_k^{-1} \hat{H}_k\right) \end{bmatrix} \tag{6.1.24}$$

where

$$\triangle_1 = \begin{bmatrix} \mathbf{0}_{N(k-2),N(k-2)} & \mathbf{0}_{N(k-2)} \\ \mathbf{0}_{N(k-2)}^T & F_k^T Q_k^{-1} F_k \end{bmatrix}. \tag{6.1.25}$$

Note that the $\mathbf{0}_{(k-1)N}$ vector in $\mathscr{J}_k\left(\hat{\mathbf{X}}_k\right)$ means that only the rightmost block column of $\left(\mathscr{H}_k\left(\hat{\mathbf{X}}_k\right)\right)^{-1}$ will contribute to the updated estimate.

6.1.4.3 Estimate Only Current State

We do not update any of the past system states $\mathbf{X}_{k-1}$, but only the most recent system state, x_k. This further reduces the portion of the inverse Hessian we are required to compute for the first iteration, from the rightmost block column to the lower-right sub-block, which is of size $N \times N$. This block of the inverse Hessian is denoted $\left(\mathscr{H}_k\left(\hat{\mathbf{X}}_k\right)\right)_{2,2}^{-1}$, and represents a first order approximation of the covariance of the state x_k. We therefore will denote it P_k, as in standard EKF equations.

We can compute this block using Lemma 1 from Appendix A, which is allows us to explicitly compute any given term of the inverse of a block matrix, as

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} = \begin{bmatrix} A^{-1}+A^{-1}B\left(D-CA^{-1}B\right)^{-1}CA^{-1} & -A^{-1}B\left(D-CA^{-1}B\right)^{-1} \\ -\left(D-CA^{-1}B\right)^{-1}CA^{-1} & \left(D-CA^{-1}B\right)^{-1} \end{bmatrix} \tag{6.1.26}$$

$$= \begin{bmatrix} \left(A-BD^{-1}C\right)^{-1} & -\left(A-BD^{-1}C\right)^{-1}BD^{-1} \\ -D^{-1}C\left(A-BD^{-1}C\right)^{-1} & -D^{-1}C\left(A-BD^{-1}C\right)^{-1} \end{bmatrix}. \tag{6.1.27}$$

If we identify terms of $\mathscr{H}$ as

$$A = \left(\mathscr{H}_{k-1}\left(\hat{\mathbf{X}}_{k-1}\right) + \begin{bmatrix} \mathbf{0}_{N(k-2),N(k-2)} & \mathbf{0}_{N(k-2)} \\ \mathbf{0}_{N(k-2)}^T & F_k^T Q_k^{-1} F_k \end{bmatrix}\right), \tag{6.1.28}$$

$$B = \begin{bmatrix} \mathbf{0}_{N(k-2)} \\ -\hat{F}_k^T Q_k^{-1} \end{bmatrix}, \tag{6.1.29}$$

$$C = \begin{bmatrix} \mathbf{0}_{N(k-2)} & -Q_k^{-1}\hat{F}_k \end{bmatrix}, \tag{6.1.30}$$

and

$$D = \left(Q_k^{-1} + \hat{H}_k^T R_k^{-1} \hat{H}_k\right), \tag{6.1.31}$$

we can apply Lemma 1 to yield

$$\left(\mathscr{H}_k\left(\hat{\mathbf{X}}_k\right)\right)_{2,2}^{-1} = P_k = \left(D-CA^{-1}B\right)^{-1} \tag{6.1.32}$$

and

$$P_k = \left(\begin{array}{c} \hat{H}_k^T R_k^{-1} \hat{H}_k + Q_k^{-1} - \\ \left[\begin{array}{cc} \mathbf{0}_{N(k-2)}^T & -Q_k^{-1}\hat{F}_k \end{array} \right] \left(\mathscr{H}_{k-1}\left(\hat{\mathbf{X}}_{k-1}\right) + \triangle_1 \right)^{-1} \left[\begin{array}{c} \mathbf{0}_{(k-2)N} \\ -\hat{F}_k^T Q_k^{-1} \end{array} \right] \end{array} \right)^{-1} \tag{6.1.33}$$

where

$$\triangle_1 = \left[\begin{array}{cc} \mathbf{0}_{(k-2)N,(k-2)N} & \mathbf{0}_{(k-2)N} \\ \mathbf{0}_{(k-2)N}^T & \hat{F}_k^T Q_k^{-1} \hat{F}_k \end{array} \right].$$

Equation 6.1.33 can be further simplified by applying Lemma 2 from Appendix A, which gives

$$\left(D - CA^{-1}B\right)^{-1} = D^{-1} + D^{-1}C\left(A - BD^{-1}C\right)^{-1} BD^{-1}. \tag{6.1.34}$$

By identifying terms in Equation 6.1.33

$$P_k = \left(\begin{array}{c} \hat{H}_k^T R_k^{-1} \hat{H}_k + \underbrace{Q_k^{-1}}_{D} - \\ \underbrace{\left[\begin{array}{cc} \mathbf{0}_{N(k-2)}^T & -Q_k^{-1}\hat{F}_k \end{array} \right]}_{C} \left(\underbrace{\mathscr{H}_{k-1}\left(\hat{\mathbf{X}}_{k-1}\right) + \triangle_1}_{A} \right)^{-1} \underbrace{\left[\begin{array}{c} \mathbf{0}_{N(k-2)} \\ -\hat{F}_k^T Q_k^{-1} \end{array} \right]}_{B} \end{array} \right)^{-1}, \tag{6.1.35}$$

we can compute the intermediate products in Equation 6.1.34, as

$$D^{-1}C = \left[\begin{array}{cc} \mathbf{0}_{N(k-2)}^T & -\hat{F}_k \end{array} \right], \tag{6.1.36}$$

$$BD^{-1} = \left[\begin{array}{c} \mathbf{0}_{N(k-2)} \\ -\hat{F}_k^T \end{array} \right], \tag{6.1.37}$$

and

$$BD^{-1}C = \left[\begin{array}{c} \mathbf{0}_{N(k-2)} \\ -\hat{F}_k^T Q_k^{-1} \end{array} \right] \left[\begin{array}{cc} \mathbf{0}_{N(k-2)}^T & -\hat{F}_k \end{array} \right] = \left[\begin{array}{cc} \mathbf{0}_{(k-2)N,(k-2)N} & \mathbf{0}_{(k-2)N} \\ \mathbf{0}_{(k-2)N} & \hat{F}_k^T Q_k^{-1} \hat{F}_k \end{array} \right]. \tag{6.1.38}$$

Plugging these term identifications back into Equation 6.1.34 allows us to compute to compute P_k:

$$P_k = \left(\hat{H}_k^T R_k^{-1} \hat{H}_k + \left(Q_k + \hat{F}_k \left(\mathscr{H}_{k-1}\left(\hat{\mathbf{X}}_{k-1}\right)\right)_{2,2}^{-1} F_k^T \right)^{-1} \right)^{-1} \tag{6.1.39}$$

$$= \left(\hat{H}_k^T R_k^{-1} \hat{H}_k + \left(Q_k + \hat{F}_k P_{k-1} F_k^T \right)^{-1} \right)^{-1}. \tag{6.1.40}$$

For the EKF, P_k is a first order approximation to the covariance of the final estimate of x_k. Plugging this value and the initial estimates (6.1.20) into the Newton Update equation (6.1.17) gives us a form of the EKF equations:

$$\begin{aligned} P_k &= \left(\hat{H}_k^T R_k^{-1} \hat{H}_k + \left(Q_k + \hat{F}_k P_{k-1} \hat{F}_k^T\right)^{-1}\right)^{-1}, \quad (6.1.41) \\ \hat{x}_k^+ &= \hat{x}_k^- - P_k \hat{H}_k^T R_k^{-1} \left(y_k - h\left(f_k\left(\hat{x}_{k-1}\right)\right)\right) \\ &= f_k\left(\hat{x}_{k-1}\right) - P_k \hat{H}_k^T R_k^{-1} \left(y_k - h\left(f_k\left(\hat{x}_{k-1}\right)\right)\right). \quad (6.1.42) \end{aligned}$$

6.1.4.4 Single Iterative Update

The EKF stops after performing the single Gauss-Newton update step just described, rather than iterating the full Newton's method until convergence. This simplifies the algorithm, as it means that only the $(2,2)$ sub-block of the inverse approximate Hessian need ever be computed, whereas updating the current estimate of x_k and iterating would necessitate computation of the entire last block row of $\mathscr{H}^{-1}$.

6.2 Analysis of EKF Assumptions

It is interesting to note that in the linear case, the cost function S_k in equation 6.1.13 is truly quadratic in $\mathbf{X}_k$, which means that Newton's method is guaranteed to converge in a single step, regardless of the initial estimate. If that is the case, then the above simplifications are based on correct assumptions, and result in the original Kalman Filter which has been shown to be the optimal estimator for linear system models.

In the general nonlinear case optimality guarantees are quickly lost. In general, we can think of the non-optimality of the EKF in terms of three major sources of error:

1. Even if we correctly find a critical point of the cost function S_k, there is no guarantee that we will find the global optimum. This is in general the case with all nonlinear estimators.

2. By assuming the prior estimate was the minimizer of the prior cost function and updating only the most recent state (second and third assumptions), the EKF sacrifices the information in past state and measured values. Instead, all information about the state history comes from the estimate $\hat{x}_{k-1}$, and the prior inverse block Hessian (covariance) P_{k-1}. In the general nonlinear case, this amounts to the optimization of a different cost function than that given by S_k. Another way of understanding this problem is to realize that during any particular update, discrepancies between actual and predicted values are assumed to be purely due to errors in the estimate of the current state x_k, whereas they may actually be due partly to errors propagated from prior state estimates.

3. The EKF does not find the optimum point according to this cost function, but instead takes only a single Gauss-Newton optimization step on this cost function.

The EKF is quite robust to the effects of these simplifications in many practical applications. The key problem with the EKF is that its estimate tends to become overconfident, leading it to trust incorrect state estimates too highly (due to the covariance matrix becoming too small). This is typically countered by artificially increasing the values of Q_k, which serves to help prevent P_k from becoming too small and leading to filter divergence, but which also may cause the filter covariance to be so large as to make the filter uninformative. The performance of a filter can thus be cast as the extent to which it can strike the proper balance between being inconsistent and being uninformative [68].

Since there is in general no way to guarantee that a nonlinear estimator will converge to the global optimum, the first source of error is difficult to eliminate. The second source of error, updating only the most recent state, is inherent in the nature of the filtering problem. If sufficient computational resources are available, it is likely that a full optimization of the least squares cost function will yield a better answer than any filter. However, in many situations this is simply not possible due to resource constraints, which is why filtering methods are resorted to. The Backward-Smoothing EKF of [51] gives a scalable way to solve this problem partially. By allowing the number of prior states maintained to be fixed at a number other than 1, the BSEKF makes it possible in a given application to explore the tradeoff between resource usage and filter performance.

The last of the problems listed above is perhaps the easiest to address, at least partially. The Iterated Extended Kalman Filter attempts to do so by using the Kalman Filter update equation iteratively, in effect re-computing the inverse Hessian at each new estimate in hopes of obtaining a more accurate estimate. In the framework we have described, this amounts to using a value of x_k that is not equal to $f(x_{k-1})$, while still assuming that $x_k = f(x_{k-1})$ for purposes of computing the Jacobian. The IEKF measurement update has been shown to outperform the EKF measurement update in certain cases. In particular, if a single measurement completely observes the state variables in which the measurement function is highly nonlinear, the final linearization used by the IEKF will be close to the linearization at the correct state, which will cause the IEKF estimate to be better in general[68].

Chapter 7

Manifolds

As the mathematics of manifolds may be unfamiliar to many readers who are familiar with Kalman filtering, the purpose of this section is to give a relatively informal and intuitive explanation of this material. While some derivations are sketched in moderate detail for the sake of gaining insight, this section is not intended to provide complete proofs or mathematical rigor. Readers desiring a rigorous treatment should to consult [47] and [60], from which this material is drawn.

The Kalman Filter formulation assumes that the state to be estimated is a member of an inner product space. This assumption is evident in the formulation of the cost function used in the EKF Derivation in Chapter 6: the covariance weighted squared distance between some value x and its estimate $\hat{x}$ was always computed as a quadratic form

$$(x-\hat{x})^T Q_x^{-1} (x-\hat{x}) . \tag{7.0.1}$$

This distance is effectively a particular inner product (specified by the value of Q_x) of the vector $(x-\hat{x})$ with itself. This vector is the vector based at $\hat{x}$ which points to x, and is computed by vector subtraction of the coordinates of those two points.

Inner product spaces have a number of important properties, in particular the notion of vector subtraction, that cannot be meaningfully defined on many spaces of interest. Thus, the purpose of this chapter is to develop the definition of a **Riemannian manifold**, which is a generalization of the notion of an inner product space. As we shall see, on Riemannian manifolds we can define operations called the exponential and logarithmic mappings which can be seen as generalized replacements for vector addition and subtraction.

This chapter will proceed as follows. After giving some background information on inner product spaces in Section 7.1, we will motivate and define the concept of a manifold in Section 7.2. The key motivation for manifolds will be the fact that we cannot always find a way to uniquely assign coordinates to points on curved spaces. Once manifolds are defined, we will of necessity discuss the definition of distance between points on a manifold (Section 7.3), defining along the way the important notion of the tangent space to a manifold point. The tangent space to a manifold point can be thought of as the set of all velocity vectors a curve passing through that point could have. These velocity vectors give us a way to define speed, which in turn gives us a way to define distance (by integration over a time interval). Once distance is defined, we can introduce the exponential and logarithmic mappings which will allow us to replace vector addition and subtraction respectively. These mappings are defined in terms of motion along the geodesics, or minimum-length curves, of the manifold.

Once we have a way to define distance, we will discuss (Section 7.4) the importance of ensuring that the distance between manifold points remains invariant under a certain class of transformations. If some operations do not change the distance between points in a space associated with some real system, our distance metric must reflect this fact in order to be useful in the process of modeling and estimating the state of that system. To this end, we will introduce the notion of a Lie Group of operations under which distance should be invariant. This set of operations will give us a way to "move between" manifold points and thus relate tangent vectors at different points to one another. It will also lead (Section 7.5) to a convenient representation for points on a manifold called the principal chart, and to a way to actually implement the exponential and logarithmic mappings.

With the foregoing discussions in place, we note (Section 7.6) that all of the concepts developed for manifolds are applicable to vector spaces, and yield familiar operations in those spaces: the exponential mapping on a vector space is vector addition, the logarithmic mapping is vector subtraction, etc... Finally (Section 7.7), we briefly discuss the representation of statistical and probabilistic concepts on manifolds.

7.1 Definition of Inner Product Spaces

A **scalar field** can be thought of as the generalization of the set of real numbers. Two real numbers can be added and/or multiplied, and the operations of addition and multiplication follow the rules of algebra (multiplication distributes over addition, etc...). Generalizing this, then, if we can define any collection of mathematical objects, and define "add" and "multiply" operations on those objects that follow the same rules of algebra, that set of objects is termed a scalar field, and each element of the set is called a **scalar**.

A finite dimensional **vector space** is a generalization of the familiar three-dimensional Euclidean space. A vector space is a collection of vectors, each of which is a fixed-length collection of scalars[1]. The number of scalars making up each vector is the **dimension** of the vector space. A vector space is more than just a set of n-tuples of scalars, however; a vector space must define the operations of scalar multiplication and vector addition. The former allows a scalar and a vector to be multiplied to produce another vector, and the latter allows two vectors to be added (or subtracted) to produce another vector.

In addition to scalar multiplication and vector addition, most commonly used vector spaces also define the notion of an **inner product**, and such a space is more specifically termed **inner product space**. The inner product is an operation that takes two vectors and produces a single scalar. The inner product is often used to define the geometrically important notions of length, distance, and angle. In the most familiar three-dimensional euclidean space:

- The length of a vector is the square root of the inner product of the vector with itself: $||a|| = \sqrt{\langle a,a \rangle}$.
- The distance between points a and b is the length of the vector $\overrightarrow{ab} = b - a$.
- The angle between two vectors can be found from the inner product: $\theta = \arccos\left(\frac{\langle a,b \rangle}{\sqrt{\langle a,a \rangle \langle b,b \rangle}}\right)$.

[1]Vectors have a fixed number of scalar components for finite dimensional vector spaces. There are also infinite dimensional vector spaces, where each vector is has infinitely many components. In this work the term vector space will always refer to finite dimensional vector spaces.

These definitions of length, distance, and angle work in any inner product space, regardless of dimension.

Perhaps the most intuitive inner product space is $\mathbb{R}^3$, whose elements are ordered triples of real numbers. This space is commonly identified with the three dimensional euclidean space E^3, the collection of all geometric points in the three dimensional space that human beings experience. This identification consists of associating or "labeling" each point in E^3 with a point in $\mathbb{R}^3$, commonly referred to as the assignment of coordinates to points. We define a **chart** as a one-to-one mapping that identifies points in some space with points in $\mathbb{R}^n$. More generally, any vector space of dimension n can be assigned coordinates by identifying each point in the space with a single point in $\mathbb{R}^n$. Assigning coordinates to each vector in a vector space allows us to treat the space as $\mathbb{R}^n$, which means that we can perform geometric operations in that space by performing algebraic operations.[2] A vector space can in fact be defined by this process of assigning coordinates: if we can define a one-to-one and onto correspondence between any space and $\mathbb{R}^n$, we have effectively proved that that space is a finite dimensional inner product space.

7.2 Definition of a Manifold

Unfortunately, not all finite dimensional spaces of interest can be corresponded to $\mathbb{R}^n$ in this way. For example, the unit sphere, S^2, is the collection of all points in $\mathbb{R}^3$ whose distance from the origin is 1. While it is typically pictured as embedded in a three-dimensional space as just described, the sphere is inherently two dimensional. In attempting to assign two dimensional coordinates to points on the sphere, we run into the famous map-maker's problem: there is no way to form a $1:1$ correspondence between S^2 and $\mathbb{R}^2$, because there will always be at least one point on the sphere that can have many possible coordinate values. The "north pole" and "south pole" are the classic examples of such singularity points if latitude and longitude angles are used to correspond S^2 and $\mathbb{R}^2$.

This problem of not being able to unambiguously assign coordinates to every point on such spaces leads to the definition of a **differentiable manifold**, or simply a **manifold**, which generalizes the definition of a vector space. To assign coordinates in a vector space, we must be able to find a chart that is one-to-one and onto: a unique assignment of coordinates to every point in the space. For manifolds, this constraint is relaxed in a specific way.

The definition of a manifold allows for many charts, each corresponding between $\mathbb{R}^n$ and only a portion of the points in the space. For the space to be a manifold, these charts must satisfy the following conditions:

- The charts must collectively cover the entire manifold: there may be no points that are not assigned coordinates in $\mathbb{R}^n$ by some chart.
- Wherever two charts ϕ and ψ overlap (where two different charts identify the same manifold point with points in $\mathbb{R}^n$), there must be a differentiable and invertible function $s_{\phi\psi}$ taking the coordinates of a point in one chart to the coordinates of that point in the other chart.
- The range of each chart (the portion of $\mathbb{R}^n$ onto which it maps manifold points) must be an open subset of $\mathbb{R}^n$.

[2]The assigning of coordinates, allowing the use of algebra to solve geometric problems, is due to Descartes and underlies the field of algebraic geometry.

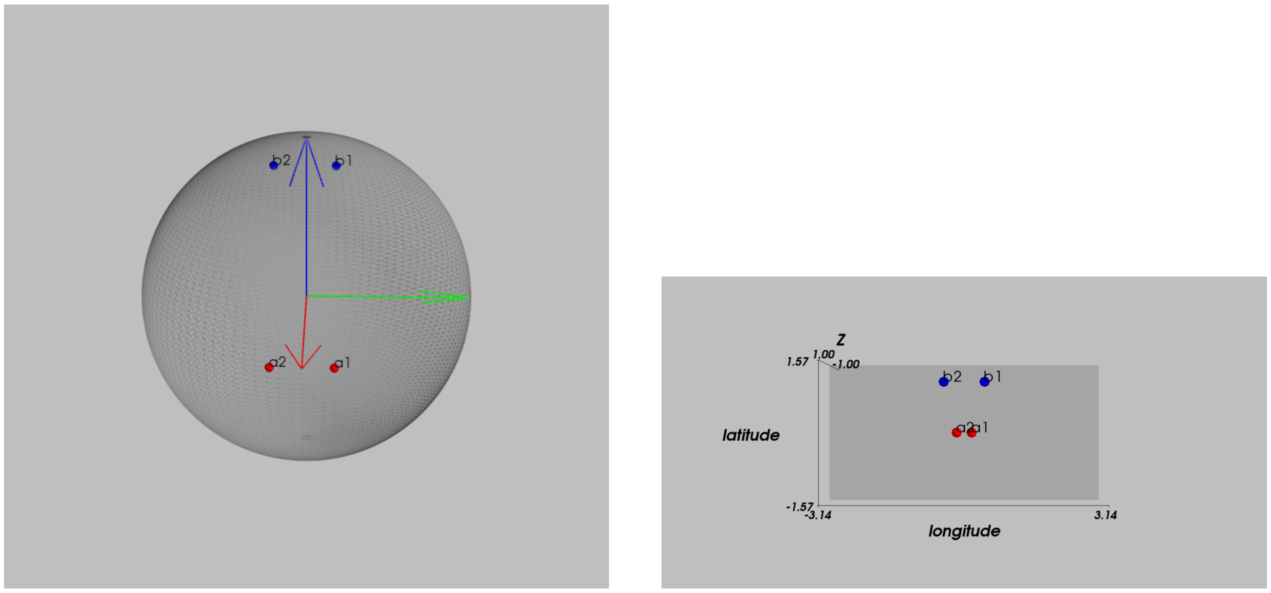


Figure 7.3.1: Manifold distance is not chart distance: the standard distance metric based on vector subtraction and vector norm is not in general invariant under many desired transformations on a manifold. The distance along the manifold surface between points a_1 and a_2 is equal to the distance between b_1 and b_2: this result is physically meaningful in many common problems. This is the case because $b_1 = f(a_1)$ and $b_2 = f(a_2)$, where f is a 3D rotation about the sphere center: "along the surface" distance on S^2 is invariant to such rotations. However, if we use vector subtraction of the *coordinates* of those points (i.e. $\text{dist}^2(b_1, b_2) = \|b_1 - b_2\|^2$) in the commonly used latitude, longitude coordinate system for S^2, the distance on the chart is changed considerably by the operation f.

7.3 Distance on Manifolds

In an inner product space, as discussed above, the definition of an inner product automatically gives us a way to define the distance between two points: $\text{dist}(a,b) = \sqrt{\langle (b-a),(b-a)\rangle}$. In order to be useful any definition of distance must conform to the physical properties of the system being modeled. These properties can be understood as a series of invariants, operations under which distance will not change. For example, the distance between two points on the surface of a sphere should not be changed by operations which rotate the points about the sphere center, effectively rotating the sphere beneath the points.

If we can find a chart for a space such that the standard chart distance does not change under all of the invariant transformations we care about, then we can simply use chart coordinates to represent manifold points in all our operations, much like we use tuples of real numbers in $\mathbb{R}^3$ to represent points in three-dimensional euclidean space. Finding a chart that has this property is often easy for linear vector spaces, but not for manifolds in general. Geometrically speaking, it is fairly easy to see why this is so. An illustrative example is given in Figure 7.3.1. The manifold S^2 is inherently "curved", and assigning 2-d coordinates to it amounts to an attempt to "flatten" the curved space. This has two effects: it requires "tearing" of the sphere at some point (causing the chart to not be one-to-one over the whole manifold), and it produces nonlinear distortions in the mapping process. These distortions cause chart distance to be different after a transformation, even when actual manifold distance is not.

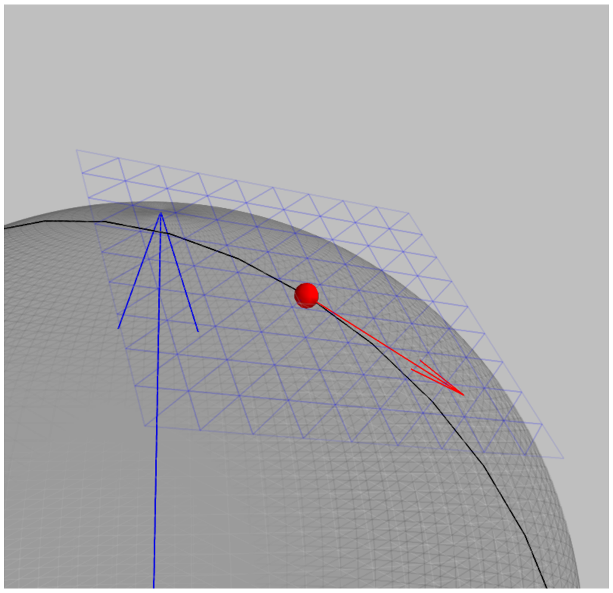


Figure 7.3.2: An illustration of the tangent space to a manifold point. Any curve $\gamma(t)$ (black line) on the manifold passing through a point can be differentiated with respect to time, and the resulting vector $\frac{d\gamma(t)}{dt}$ (red) resides in the tangent space $T_x\mathscr{M}$ (blue grid).

Because of practical constraints such as this, we cannot simply use chart distance as a proxy for manifold distance on general manifolds.

Fortunately, the notion of vector addition, and thus the standard definition of distance, can be generalized to manifolds. To do this, we will define the **tangent space** to each point on the manifold, which is intuitively the set of vectors which "point from" that base point to every other point on the manifold, just as the difference between two points a and b in a vector space specifies a vector $\bar{ab}$ that "translates" from a to b.

The actual mapping which tells us the point b to which a particular tangent vector v based at a points is termed the **exponential mapping**. Its inverse function, termed the **logarithmic mapping**, determines the vector v given a and b. The former function is the manifold generalization of vector addition, while the latter is the generalization of vector subtraction.

7.3.1 Tangent Space to a Manifold Point

Every point a on a manifold $\mathscr{M}$ has an associated tangent space, denoted $T_a\mathscr{M}$. The tangent space can be thought of as representing the set of all possible velocities a manifold-valued curve $\gamma(t)$ passing through a may possess.

To see that this is the case, we can think of ourselves moving in time along the path $\gamma(t)$ on a manifold $\mathscr{M}$, as shown (using S^2 for ease of visualization) in Figure 7.3.2. Note that this function γ returns an abstract manifold point, *not* the coordinates of this point in any particular chart. Suppose that at some time t_0, we are located at the manifold point γ_0, so that $\gamma_0 = \gamma(t_0)$. We can choose a chart that covers the point γ_0 and other points that are close to γ_0 (in the temporal sense). We can then compose the abstract function γ with the chart's mapping ϕ to produce a new function

$\gamma_\phi(t)$. This function takes a scalar (time) to a vector in $\mathbb{R}^n$ (the coordinates of the abstract point $\gamma(t)$ under the chart ϕ). The function γ_ϕ is now a standard vector-valued scalar function, and each of its components can thus be differentiated using ordinary calculus. This means that the velocity vector of the curve γ_ϕ can be computed at the point t_0. The key insight is that this velocity vector expresses in some sense a true, abstract property of the function γ, regardless of the chart we choose. This is the case because we can convert the coordinates of the vector $\frac{\partial \gamma_\phi}{\partial t}$, which depend on chart ϕ, to the coordinates of the corresponding vector under some other chart ψ (which of course must also cover the temporal neighborhood of interest around γ_0), using the invertible, differentiable mapping between overlapping charts $m_{\phi\psi}$. Recall that this mapping must by definition exist in order for the space to be a manifold: the use of $m_{\phi\psi}$ here highlights the key reason why this is the case. Since knowing a curve's velocity at a point under a certain choice of chart implies knowing its velocity at that point under *any* choice of chart, the velocity is not an arbitrary value: the choice of chart is merely a choice of the representation of that velocity.

It is important to keep in mind that the tangent spaces at any two points on a manifold are two distinct spaces. If we wish to compare vectors in different spaces, we must define some way of identifying vectors in these two spaces. Put differently, we need a way to map vectors in some tangent space $T_x\mathscr{M}$ to another tangent space $T_y\mathscr{M}$. One way to do this is to use a differentiable function which takes the coordinates of some point x (expressed in some chart) to the coordinates of another point y (also expressed in some chart):

$$f(x) = y. \tag{7.3.1}$$

This function, then, would take any curve γ_ϕ passing through x to a corresponding curve η_ψ passing through y:

$$\gamma_\phi(0) = x, \tag{7.3.2}$$
$$f(\gamma_\phi(0)) = \eta_\psi(0) = y. \tag{7.3.3}$$

If this is the case, the Jacobian of the function f will be a linear operator mapping vectors in $T_x\mathscr{M}$ to corresponding vectors in $T_y\mathscr{M}$. To see that this is so, we need only find the time derivative of the new curve $\eta_\phi(t)$ evaluated at $t = 0$, applying the chain rule:

$$\left.\frac{d\eta_\psi(t)}{dt}\right|_{t=0} = \left.\frac{df(\gamma_\phi(t))}{dt}\right|_{t=0} = \left.\frac{\partial f(\gamma)}{\partial \gamma}\right|_{\gamma=x} \cdot \left.\frac{d\gamma_\phi(t)}{dt}\right|_{t=0}. \tag{7.3.4}$$

If we recall that the tangent space $T_x\mathscr{M}$ was defined as the collection of all possible velocity vectors which a manifold-valued curve could possess at point x, we see that $\left.\frac{d\gamma_\phi(t)}{dt}\right|_{t=0}$ is in fact such a vector: it is the velocity of the curve γ_ϕ as it passes through the point x (at time 0). Likewise, $\left.\frac{d\eta_\psi(t)}{dt}\right|_{t=0}$ is the corresponding vector in $T_y\mathscr{M}$, and we can see from Equation 7.3.4 that they are linearly related by the Jacobian $\left.\frac{\partial f(\gamma)}{\partial \gamma}\right|_{\gamma=x}$ of f. Thus we can see that the Jacobian of any mapping taking one manifold point to another provides a linear mapping between the tangent spaces to those two points.

7.3.2 Riemannian Manifolds and Manifold Distance

All of the tangent spaces to points on some n-dimensional manifold $\mathscr{M}$ are equivalent to $\mathbb{R}^n$. If we define an inner product operation on each tangent space, it becomes an inner product space. This allows us to talk about the length of vectors in that space, or equivalently the instantaneous speed of any curve passing through that point. A **Riemannian manifold** is a manifold which "smoothly" defines an inner product to every tangent space.

To be rigorous about the meaning of "smoothly" we must briefly introduce the concept of a vector field. A **vector field** is a mapping that assigns a vector to each point in a vector space. More specifically, given the coordinates of a point a in some vector space, the vector field gives the coordinates of a vector based at a. A vector field is **differentiable** if the vector coordinates can be differentiated w.r.t. the vector space coordinates. We can define a vector field on a manifold as well, by using the same argument we used to define the tangent space: the choice of a chart lets us use the standard definition of a vector field, and where charts overlap, the overlap functions allow us to convert vector components between them.

If a manifold $\mathscr{M}$ has an inner product $\langle \cdot, \cdot \rangle_x$ defined on each of its tangent spaces $T_x\mathscr{M}$, then we can compute the inner product between any two differentiable vector fields V_1 and V_2 at every point on the manifold as $\langle V_1(x), V_2(x) \rangle_x$. This result is a scalar function taking the coordinates x of a manifold point (in some chart) to a scalar value. If this function is differentiable with respect to x at every point x on $\mathscr{M}$ (no matter what the chart used) then the manifold is by definition a Riemannian manifold.

Intuitively, a Riemannian manifold is thus the generalization of an inner product space: it allows us to define manifold distance. We have seen that we can compute the velocity vector to a manifold-valued curve in any chart of our choice. For a Riemannian manifold we can use the Riemannian metric to compute the length of any such vector. This means that we can compute the instantaneous speed of a curve at a manifold point. By integrating this speed over time, we can find the length of the curve over any time interval. This fact can be used to define distance: the distance between point a and point b is defined to be the length of the shortest curve that begins at a and ends at b. The shortest curve between two points is termed a **geodesic**.

Practically speaking, the inner product in the tangent space $T_x\mathscr{M}$ at any point x on an N-dimensional Riemannian manifold $\mathscr{M}$ is specified by an $N \times N$ Hermitian symmetric matrix. In practice, there are a number of ways of assigning these matrices to points in $\mathscr{M}$, all of which result in a Riemannian metric. The choice of which of these Riemannian metrics is to be used must be made before any computation can be performed. As we have said previously, we will typically require that certain geometric intuition holds, meaning that the distance between points should be invariant under certain transformations. We will see shortly that this provides us with a meaningful way to choose a particular Riemannian metric.

7.3.3 Geodesics and the Exponential and Logarithmic Mappings

In order to find out what the geodesic curves are for a particular manifold, we can conceptually use the calculus of variations to set up a second order differential equation, the solutions to which are the geodesics (details are given in [47]). It turns out due to the theory of second order differential equations that there is a one-to-one correspondence between tangent vectors at a point and geodesic curves passing through that point (keeping in mind that every geodesic curve has a particular speed). Intuitively, this correspondence gives us a way to get from the tangent space $T_a\mathscr{M}$ back onto the manifold $\mathscr{M}$ itself: it allows the definition of the **exponential mapping** $\exp_a(v)$, which produces the point reached by traveling along the geodesic corresponding to v for unit time.

The mapping from $T_a\mathscr{M}$ to $\mathscr{M}$ provided by $\exp_a(\cdot)$ is one-to-one for the entire manifold except a set of points known as the **cut locus** of a, which is the set of points where multiple geodesics leaving a meet. For example, on the manifold S^2, the cut locus of any point will be the antipodal point, since all geodesic curves leaving a are great circles, and will intersect again at the antipodal point to a. The vectors in $T_a\mathscr{M}$ corresponding to the cut locus points form a set called the **tangential cut locus**. The exponential map is invertible everywhere except on the cut locus. In this region, we can define the inverse of the exponential mapping which is termed the **logarithmic mapping**, denoted $\log_a(b)$. This mapping takes the endpoint of a geodesic and returns the corresponding tangent vector at a.

The logarithmic mapping at a specified point a maps manifold points near a to $T_a\mathscr{M}$, which is a vector space and thus equivalent to $\mathbb{R}^n$. This means that the logarithmic mapping can be considered the mapping function of a chart, called the **exponential chart** based at a. This chart uses tangent vectors in $T_a\mathscr{M}$ to represent points on $\mathscr{M}$. If we choose one point on $\mathscr{M}$ to be the origin, the exponential chart based there is termed the **principal chart**.

Note that the length of the vector $v = \log_a(b)$ is by definition exactly the geodesic distance between a and b. This is the case because the exponential mapping was defined using the geodesics, the same curves used to define distance. This fact allows us to use the exponential and logarithmic mappings as generalized version of vector addition and vector subtraction, respectively, since $\log_a(\cdot)$ allows us to find the vector "pointing from" a to b whose length is the distance between them.

7.4 Respecting Distance Invariants on Manifolds

As we have discussed, it is important to have a way of measuring distance on the manifold that is invariant under certain classes of transformations, so that our distance metrics accurately model physical intuition. We can do this by choosing the Riemannian metric such that it is invariant to the group of transformations we are concerned with. Following [47], we presume that the transformations under which distance should be invariant themselves belong to a special type of differential manifold called a **Lie Group**.

A Lie Group $\mathscr{G}$ is a differentiable manifold which has defined a *differentiable* composition operator, here denoted $\circ$, under which the elements of the manifold form a **group**. Among other things, this means that individual elements $g_1, g_2 \in \mathscr{G}$ of a Lie Group can be "multiplied" (composed) together to yield another element of the Lie Group: $g_1 \circ g_2 \in \mathscr{G}$. The differentiability of this operator can be defined in terms of any chart of choice, analogously with the differentiability of a vector field on a manifold, as above. Because it is differentiable, the mapping $f(g) = g_1 \circ g$ defines an invertible linear mapping between the tangent spaces $T_g\mathscr{G}$ and $T_{g_1 \circ g}\mathscr{G}$.

In practice, the geometric transformations under which distance is physically invariant are frequently members of Lie Groups. For example, in many physical situations, distance is invariant to three-dimensional rotation and linear translation: these operations correspond to the manifolds SO^3 and $\mathbb{R}^3$ respectively, both of which are also Lie Groups.

Thus, for every manifold $\mathscr{M}$ on which we wish to work, we must also specify the Lie Group $\mathscr{G}$ of transformations under which distance should be invariant. We assume that there is an action operator, denoted $\star$, by which group operations can "act" on a manifold point to produce a manifold point. It is this group action under which we would like distance to be invariant. We will require that this action distributes over composition between group members:

$$\begin{aligned}(g_1 \circ g_2) \star f &= g_1 \star (g_2 \star f) \\ &= g_1 \star g_2 \star f. \qquad (7.4.1)\end{aligned}$$

In some cases, the manifold itself is its own Lie Group. This is the case, for example, with both $\mathbb{R}^3$ and SO^3. In $\mathbb{R}^3$, distance should physically be invariant to linear translation, but since every linear translation can be identified with a vector in $\mathbb{R}^3$, we have that $\mathscr{M} = \mathscr{G}$. Similarly, the angular distance between two orientations in SO^3 should be invariant to rotations, but every rotation is a member of SO^3.

Since we have a group of operations under which distance is invariant, we will use this set of operations to allow us to translate between manifold points. The Jacobian of these operations, then, give us a way to translate vectors between tangent spaces. In practice this means that we can shift calculations on some tangent space $T_x\mathscr{M}$ to any other tangent space, perform them there, and shift the results back without worrying that the shifting process affects the outcome. To this end, we will also require that the manifold is **homogeneous** under the group, meaning that every manifold point $x \in \mathscr{M}$ can be obtained by acting on the origin with some member $f_x \in \mathscr{G}$ of the group. This requirement allows us to associate with every manifold point a set $\mathscr{F}_x$ of group operations which take the origin to that point:

$$\mathscr{F}_x = \{ f \in \mathscr{G} | f \star o = x \}. \qquad (7.4.2)$$

This association allows us to choose (from this set) a single **placement function** $f_x \in \mathscr{F}_x$ for every manifold point x. The inverse f_x^{-1} of this placement function allows us to move from any point x on the manifold back to the origin, and the inverse of the Jacobian of f_x, denoted $J(f_x)^{-1}$, allows us to translate vectors in any tangent space $T_x\mathscr{M}$ into corresponding tangent vectors on $T_O\mathscr{M}$, at the origin.

Because of the group structure of $\mathscr{G}$, the set of all potential placement functions is a sub-group of the manifold $\mathscr{G}$ (a subset which is also a group in its own right). $\mathscr{G}$ will always contain some operations which take the origin to itself, and these operations will form a sub-group of $\mathscr{G}$ which we call the **isotropy group**, written $\mathscr{I}$. If we have some placement function f_x which takes the origin to the point x, we can find another function that does the same by simply composing this function with any function $I \in \mathscr{I}$. Note that if the manifold $\mathscr{M}$ and the Lie group $\mathscr{G}$ are the same space (as is the case with SO^3), then the isotropy group has only a single member (the identity), and there is but a single unique choice of placement function for each point, which is the point itself.

Given a choice of placement function for every point on a homogeneous manifold, we are now ready to choose a Riemannian metric which is invariant under the action of points in $\mathscr{G}$. Our strategy is to choose the matrix Q_O which defines the Riemannian metric at the origin, allowing us to define the length of any vector in the principal chart, and thus the distance of any point from the origin. This length is computed as:

$$\operatorname{len}(v_O) = \sqrt{\langle v_O, v_O \rangle_O} = \sqrt{v_O^T Q_O v_O}. \tag{7.4.3}$$

Since the placement function f_x takes the origin to the point x, it's Jacobian evaluated at the origin:

$$J(f_x) = \left. \frac{\partial f_x \star e}{\partial e} \right|_{e=O} \tag{7.4.4}$$

is a linear mapping from $T_O\mathscr{M}$ to $T_x\mathscr{M}$. With this mapping, we can define the inner product on any tangent space $T_x\mathscr{M}$ by first mapping vectors back to $T_0\mathscr{M}$ and then applying the inner product in $T_O\mathscr{M}$:

$$\begin{aligned} \langle v_x, w_x \rangle_x &\doteq \left\langle J^{-1}(f_x)\, v_x, J^{-1}(f_x)\, w_x \right\rangle_O \\ &= v_x^T \underbrace{J^{-T}(f_x)\, Q_O J^{-1}(f_x)}_{G_x} w_x \\ &= v_x^T Q_x w_x. \end{aligned} \tag{7.4.5}$$

.

By defining the Riemannian inner product in this way, we can in many cases guarantee that the inner product between two vectors is invariant to the action of any group element g. To see when this is possible, we can examine the inner product between two vectors under the action of some $g \in \mathscr{G}$:

$$\begin{aligned} \left\langle v_{g\star x}, w_{g\star x} \right\rangle_{g\star x} &\doteq \left\langle J^{-1}(f_{g\star x})\, v_{g\star x}, J^{-1}(f_{g\star x})\, w_{g\star x} \right\rangle_O \\ &= \left\langle J^{-1}(f_{g\star x}) \left. \frac{\partial g\star\Box}{\partial \Box} \right|_{\Box=x} v_x, J^{-1}(f_{g\star x}) \left. \frac{\partial g\star\Box}{\partial \Box} \right|_{\Box=x} w_x \right\rangle_O \end{aligned} \tag{7.4.6}$$

recall that by definition,

$$J(f_{g\star x}) = \left. \frac{\partial f_{g\star x} \star e}{\partial e} \right|_{e=O}. \tag{7.4.7}$$

Furthermore, since $f_{g\star x}$ takes the origin to the point $g \star x$, it will necessarily be equal to $g \circ f_x \circ I$, for the proper choice of $I \in \mathscr{I}$, the isotropy group. This is the case because the operation $g \circ f_x$, which is a member of $\mathscr{G}$, takes the origin to the point $g \star x$. Hence, as discussed in it must differ from $f_{g\star x}$ only by composition with a member of $\mathscr{I}$ (which may simply be the identity mapping), since the set $\mathscr{F}_x$ defined above contains all functions in $\mathscr{G}$ taking the origin to the point x, and its members differ only by composition with members of $\mathscr{I}$. This, then, means that we can write:

$$\begin{aligned}
J\left(f_{g\star x}\right) &= \left.\frac{\partial\left(g\circ f_x\circ I\right)\star e}{\partial e}\right|_{e=O} \\
&= \left.\frac{\partial g\star\left(f_x\star\left(I\star e\right)\right)}{\partial e}\right|_{e=O} \\
&= \left.\frac{\partial g\star\square}{\partial\square}\right|_{\square=x}\cdot\left.\frac{\partial f_x\star\triangle}{\partial\triangle}\right|_{\triangle=O}\cdot\left.\frac{\partial\left(I\star e\right)}{\partial e}\right|_{e=O} \\
&= \left.\frac{\partial g\star\square}{\partial\square}\right|_{\square=x}\cdot J\left(f_x\right)\cdot J\left(I\right), && (7.4.8) \\
J^{-1}\left(f_{g\star x}\right) &= J^{-1}\left(I\right)\cdot J^{-1}\left(f_x\right)\cdot\left.\frac{\partial g\star\square}{\partial\square}\right|_{\square=x}^{-1} && (7.4.9)
\end{aligned}$$

substituting 7.4.9 back into 7.4.6, and simplifying, we obtain:

$$\begin{aligned}
\left\langle v_{g\star x},w_{g\star x}\right\rangle_{g\star x} &= \left\langle J^{-1}\left(I\right)J^{-1}\left(f_x\right)\cdot v_x,J^{-1}\left(I\right)J^{-1}\left(f_x\right)w_x\right\rangle_O \\
&= v_x^TJ^{-T}\left(f_x\right)J^{-T}\left(I\right)Q_OJ^{-1}\left(I\right)J^{-1}\left(f_x\right)w_x. && (7.4.10)
\end{aligned}$$

In order for this to be equal to Equation 7.4.5, a sufficient condition is that:

$$J^{-T}\left(I\right)Q_OJ^{-1}\left(I\right) = Q_0 \qquad (7.4.11)$$

for any choice of $I\in\mathscr{I}$.

We will need to take care to verify that Equation 7.4.11 hold on any manifolds we work on, for whatever distance metric Q_O we choose. Fortunately, as we have discussed, the isotropy group $\mathscr{I}$ always contains only the identity mapping if the manifold $\mathscr{M}$ is its own Lie Group. This means that for Lie Groups like SO^3 and SE^3, Equation 7.4.11 is always satisfied, regardless of the value of Q_O. On other manifolds, we may need to carefully select a value of Q_O to satisfy this condition, and in some cases it may simply not be possible to choose a Riemannian metric that is invariant.

7.5 The Principal Chart of a Manifold

The exponential and logarithmic mappings take tangent space vectors to abstract manifold points and back again. In order to actually express and use these functions, they must be made to express manifold points using some particular chart. There are multiple ways to do this, but a particularly parsimonious way is to use the manifold's principal chart. If we do this, the coordinates of the tangent space vector from O to some point x *are also* the chart coordinates of that point x on the manifold. This means that both the exponential and the logarithmic mappings are identity if the base point is the origin. This convenient mapping simplifies a number of operations.

In order to numerically express the exponential and logarithmic mappings based at some other point x, we can proceed much as we did for the Riemannian metric. Between any point x and any other point y, there is a geodesic curve whose velocity vector at x is the vector $\log_x(y)$. Mapping this curve through any group operations will not alter this vector, as we showed in the last section. Therefore, we will map the curve through the inverse of the placement function of x, f_x^{-1}, compute the logarithm at the origin, and then map the resulting vector back to $T_x\mathscr{M}$, using $J(f_x)$:

$$\begin{aligned}
\log_x(y) &= J(f_x)\log_{f_x^{-1}\star x}\left(f_x^{-1}\star y\right) \\
&= J(f_x)\log_O\left(f_x^{-1}\star y\right) && (7.5.1) \\
&= J(f_x)\left(f_x^{-1}\star y\right) && (7.5.2)
\end{aligned}$$

where Equation 7.5.2 follows from 7.5.1 since $\log_O$ is the identity mapping.

We can safely perform these operations precisely because the distance metric was chosen such that group action operations like $f_x^{-1}\star\Box$ preserve distance, and therefore preserve geodesics, which are defined as minimum distance curves. To make this more rigorous, consider that there is exactly one geodesic curve γ which joins the point x with the point y in unit time: that is, $\gamma(0)=x$ and $\gamma(1)=y$. The image of this curve under f_x^{-1} is some other curve η, and we know by definition of f_x^{-1} that $\eta(0)=O$ and $\eta(1)=f_x^{-1}\star y$. The logarithmic function $\log_O\left(f_x^{-1}\star y\right)$ will by definition find the time derivative of the unique geodesic curve joining $\eta(0)$ and $\eta(1)$, which is a tangent vector in $T_O\mathscr{M}$. Because f_x^{-1} is a member of $\mathscr{G}$, and the image of a geodesic curve under any member of $\mathscr{G}$ is also a geodesic, the curve η is a geodesic, and therefore must be precisely the geodesic joining the points $\eta(0)$ and $\eta(1)$. The tangent vector $\log_O\left(f_x^{-1}\star y\right)$, then, is precisely the time derivative of the geodesic curve $\eta(t)$ at $t=0$. Therefore, multiplication by $J(f_x)=\left.\frac{\partial f_x\star e}{\partial e}\right|_{e=O}$ will necessarily convert this time derivative into the time derivative of the original curve $\gamma(t)$ at $t=0$:

$$\begin{aligned}
\left.\frac{\partial f_x\star e}{\partial e}\right|_{e=O}\left.\frac{d\eta(t)}{dt}\right|_{t=0} &= \left.\frac{\partial f_x\star e}{\partial e}\right|_{e=O}\left.\frac{d f_x^{-1}\star\gamma(t)}{dt}\right|_{t=0} \\
&= \left.\frac{\partial f_x\star e}{\partial e}\right|_{e=O}\left.\frac{\partial f_x^{-1}\star\Box}{\partial\Box}\right|_{\Box=x}\left.\frac{d\gamma(t)}{dt}\right|_{t=0} \\
&= J(f_x)J\left(f_x^{-1}\right)\left.\frac{d\gamma(t)}{dt}\right|_{t=0} \\
&= J(f_x)J(f_x)^{-1}\left.\frac{d\gamma(t)}{dt}\right|_{t=0} \\
&= \left.\frac{d\gamma(t)}{dt}\right|_{t=0}. && (7.5.3)
\end{aligned}$$

Similarly for the exponential, we can map a vector v_x back to the origin using $J(f_x)^{-1}$, apply the exponential mapping (identity) and then transform the result using f_x:

$$\begin{aligned}
\exp_x(v_x) &= f_x\star\exp_{f_x^{-1}\star x}\left(J(f_x)^{-1}v_x\right) \\
&= f_x\star\exp_O\left(J(f_x)^{-1}v_x\right) && (7.5.4) \\
&= f_x\star\left(J(f_x)^{-1}v_x\right). && (7.5.5)
\end{aligned}$$

In terms of implementation, Equations 7.5.2 and 7.5.5 work because the group action operator $\star$ is parameterized such that its input and output are both expressed in the principal chart. This means that we must find the way to express this operation in this fashion for a manifold/group pair in order to use this technique. For example, as we shall see in Chapter 9, the principal chart representation of SO^3 is the rotation vector parameterization (unit-length axis of rotation scaled by angle of rotation). We will therefore need a way to compose two such rotation vectors directly. Furthermore, our implementation must be differentiable, so that quantities like $J(f_x)$ can be computed.

Of course, not every point on a general manifold can be represented by a single chart. It will be important, therefore, to examine what the cut locus of the principal chart is. For many manifolds of interest, including SO^3 and S^2, the cut locus is a single point, the antipodal point to the origin. We will in general simply use principal chart coordinates as though they were a global covering of the entire manifold, since the cut locus will in general be a single point, and thus have null measure. See Chapter 11 in Section III for further discussion.

7.6 Vector Spaces Are Also Manifolds

Note that all vector spaces are also manifolds, and all of the developments here apply to vector spaces as well. By way of a common example, suppose that the manifold $\mathscr{M}$ is in fact $\mathbb{R}^n$. In this case, the chart mapping ϕ is simply the identity mapping, the exponential mapping is vector addition, and the logarithmic mapping is vector subtraction. Since the exponential mapping has no cut locus, the logarithmic mapping is everywhere defined. This means that the tangent space at a is in fact just a copy of the space $\mathbb{R}^n$ with its origin at a, and every point in $\mathscr{M}$ corresponds to a point in $T_a\mathscr{M}$ trivially.

7.7 Probability and Mahalanobis Distance on Manifolds

A Riemannian metric on a manifold can be used to define a measure (volume metric) $d\mathscr{M}$ which can be used to integrate real-valued functions of manifold coordinates:

$$\int f(x)\,\mathrm{d}\mathscr{M}. \tag{7.7.1}$$

The availability of a measure, in turn, allows us to define probability density functions on manifolds.

Defining the moments of a pdf on a manifold is somewhat more complicated. Mean is typically defined:

$$\bar{x} = \int x\mathrm{d}\mathscr{M}, \tag{7.7.2}$$

but this definition requires the use of vector addition. Fortunately, there are several alternative ways to define mean that coincide with this definition in the vector case. Pennec [47] uses the Frechet or Karcher mean, defined as the point $\bar{x}$ about which covariance is minimized:

$$\arg\min\left\{(\log_{\bar{x}}(x))(\log_{\bar{x}}(x))^T\right\}. \tag{7.7.3}$$

Note that the covariance is simply the mean outer product of vectors in $T_{\bar{x}}\mathscr{M}$, which is a vector space: that is, we have no difficulties using the standard definition of covariance (adapted to use $\log_{\bar{x}}(\Box)$ instead of $(\Box - \bar{x})$), since it naturally lives on a tangent space. A simple recursive algorithm for the computation of mean value has been developed, and is given in [46, 47].

A definition of Gaussian pdf that follows from this definition of $\mathrm{d}\mathscr{M}$ is given in [46]. In our application, we care less about the actual definition of the Gaussian pdf than we do about Mahalanobis distance, since the EKF derivation does not assume that noise is Gaussian in nature, but only seeks to minimize a Mahalanobis distance metric. The Mahalanobis distance is also easy to define, and is again simply the standard definition of Mahalanobis distance, with vector subtraction replaced by the logarithmic map:

$$\mathrm{dist}_{Maha} = \left(\log_{\bar{x}}(x)\right) Q^{-1} \left(\log_{\bar{x}}(x)\right). \tag{7.7.4}$$

It is important to note that Mahalanobis distance is sometimes not invariant to group operations, even when other invariant distance metrics can be found for a manifold. Note that in the definition given above, the log function is defined using an invariant distance metric, and that Mahalanobis distance is defined in terms of this log function.

Chapter 8

Kalman Filtering on Manifolds

Having laid the foundation in Chapter 7 of the theoretical concepts of manifolds, we will now proceed to derive the equations of the Manifold EKF. This derivation will follow the same process as the standard EKF derivation in Chapter 6: we will first frame a cost function $S_k(\mathbf{X}_k)$ to be minimized, which will in general involve error vectors on both the state and measurement manifolds. These error vectors, as we have discussed in Chapter 7, will be computed by using the logarithmic mapping of their corresponding manifolds, rather than by vector subtraction in the coordinates of a particular chart.

In implementing the Manifold EKF, as with any other filtering method, we will often encounter situations where we wish to predict the current state from a previous state absent any new measurements. This corresponds to the "time update" step employed by the typical EKF. There will also be the "measurement update" case, wherein a new measurement collected at the current time is available, and what we term the "joint update" case which performs a simultaneous time and measurement update. Each of these cases will need to be derived from a separate cost function: we therefore will present cost functions for the time-only and joint update cases and present results for each. The measurement-only case can be implemented by simply performing a joint update with the amount of time lapse set to zero: i.e. by setting $x_k = x_{k-1}$, and letting $f_k(x_{k-1})$ be simply the identity function.

Given the cost function, we will seek to find the value of only the most recent state x_k which minimizes this cost function by using the Gauss-Newton iterative minimization technique:

$$\begin{aligned}\Delta x_k &= -\left\{\frac{\partial^2 S_k(\mathbf{X}_k)}{\partial \mathbf{X}_k \partial \mathbf{X}_k}\right\}^{-1} \frac{\partial S_k(\mathbf{X}_k)}{\partial \mathbf{X}_k} \\ &= -\left\{\frac{\partial^2 S_k(\mathbf{X}_k)}{\partial \mathbf{X}_k \partial \mathbf{X}_k}\right\}^{-1} \mathscr{J}(\mathbf{X}_k), \end{aligned} \tag{8.0.1}$$

we can then apply the resulting delta vector to the current estimate of x_k to obtain a new one, x_k^+. This application will be performed using the state manifold's exponential mapping, rather than vector addition on a particular chart:

$$x_k^+ = \exp_{x_k}(\Delta x_k). \tag{8.0.2}$$

As with the EKF, we will use $\mathscr{H}_k(\mathbf{X}_k)$, the Gauss-Newton approximation of the Hessian, rather than the actual Hessian matrix. We will therefore proceed to compute the the Jacobian $\mathscr{J}(\mathbf{X}_k)$ of S, as well as $\mathscr{H}(\mathbf{X}_k)$ and its inverse, which we will denote $\mathscr{P}_k(\mathbf{X}_k)$. Unlike the EKF, however, we will not require that only a single iteration of the Gauss-Newton iterative procedure may be performed, but will allow the procedure to run for any given number of iterations or until some convergence criteria are met. This means that we must compute the last row of the inverse Hessian matrix $\mathscr{P}_k(\mathbf{X}_k)$, rather than simply the last block column of that row. At the end of this chapter, Algorithms 8.1 and 8.2 provide a complete summary of the Manifold EKF filter equations.

8.1 Joint Update Case

8.1.1 Cost Function

To form a cost function analogous to equation 6.1.13 in Chapter 6, we desire to find the sum of the Mahalanobis error between (1) the current state and its value as predicted by the system dynamic function and (2) the actual measured value and the measured value predicted by the system measurement function. We can form such a cost function S_k, using the logarithmic mapping to find error vectors:

$$\begin{aligned} S_k(\mathbf{X}_k) \quad = \quad & \frac{1}{2}\sum_{i=1}^{k}\left(\log_{x_i}(f_i(x_{i-1}))\right)^T Q_i^{-1}\left(\log_{x_i}(f_i(x_{i-1}))\right) \\ & +\frac{1}{2}\sum_{i=1}^{k}\left(\log_{z_i}(h_i(x_i))\right)^T R_i^{-1}\left(\log_{z_i}(h_i(x_i))\right). \end{aligned} \tag{8.1.1}$$

Re-writing this cost function recursively, we have:

$$\begin{aligned} S_k(\mathbf{X}_k) \quad = \quad & \frac{1}{2}S_{k-1}(\mathbf{X}_{k-1}) \\ & +\frac{1}{2}\left(\log_{x_k}(f_k(x_{k-1}))\right)^T Q_k^{-1}\left(\log_{x_k}(f_k(x_{k-1}))\right) \\ & +\frac{1}{2}\left(\log_{z_k}(h_k(x_k))\right)^T R_k^{-1}\left(\log_{z_k}(h_k(x_k))\right). \end{aligned} \tag{8.1.2}$$

Other than the replacement of vector subtraction with it's generalization the logarithmic mapping, this cost function is identical to that used by the EKF. This cost function applies to the general case where both dynamic motion (resulting in a newly added state x_k) and a measurement must be considered.

8.1.2 Computing the Jacobian $\mathscr{J}(X_k)$

The Jacobian vector $\mathscr{J}(\mathbf{X}_k)$ is a vector of the same length as $\mathbf{X}_k$ which represents the partial derivatives of the cost function $S_k(\mathbf{X}_k)$ with respect to each element of $\mathbf{X}_k$.

We will first compute the Jacobian of several pieces of the cost function, which will prove to be slightly different than the corresponding portions of the EKF derivation.

The expression $\log_{x_k}(f_k(x_{k-1}))$ represents the state space error vector. It is the vector in the tangent space at x_k which points at $f_k(x_{k-1})$, the propagated version of the prior state. We will need the Jacobian of this expression with respect to both x_k and x_{k-1}. Both of these Jacobians will be $N \times N$ matrices:

$$\begin{aligned} \phi_k &\doteq \frac{\partial \log_{x_k}\left(f_k\left(x_{k-1}\right)\right)}{\partial x_{k-1}} \\ &= \frac{\partial \log_{x_k}(f)}{\partial f} \cdot \frac{\partial f_k\left(x_{k-1}\right)}{\partial x_{k-1}}, \end{aligned} \tag{8.1.3}$$

$$F_k \doteq \frac{\partial \log_{x_k}\left(f_k\left(x_{k-1}\right)\right)}{\partial x_k}. \tag{8.1.4}$$

Note that ϕ_k and F_k allow us to map changes in x_{k-1} and x_k, respectively, into changes in the state space error vector. We will also find it convenient to define a larger $N \times kN$ matrix containing ϕ_k :

$$\Phi_k \doteq \left[\begin{array}{cc} \mathbf{0}_{N\times(k-1)N} & \phi_k \end{array}\right]. \tag{8.1.5}$$

Similarly, $\log_{z_k}\left(h_k\left(x_k\right)\right)$ is the measurement space error vector, the vector in the tangent space $T_{z_k}\mathscr{X}$ which points at $h_k\left(x_k\right)$, the estimated measurement value. We will need the Jacobian of this expression with respect to x_k. This Jacobian will be an $M \times N$ matrix:

$$\begin{aligned} H_k &\doteq \frac{\partial \log_{z_k}\left(h_k\left(x_k\right)\right)}{\partial x_k} \\ &= \frac{\partial \log_{z_k}(h)}{\partial h} \cdot \frac{\partial h_k\left(x_k\right)}{\partial x_k}. \end{aligned} \tag{8.1.6}$$

Recall that the cost function is given by:

$$\begin{aligned} S_k\left(\mathbf{X}_k\right) = &\frac{1}{2} S_{k-1}\left(\mathbf{X}_{k-1}\right) \\ &+\frac{1}{2}\left(\log_{x_k}\left(f_k\left(x_{k-1}\right)\right)\right)^T Q_k^{-1}\left(\log_{x_k}\left(f_k\left(x_{k-1}\right)\right)\right) \\ &+\frac{1}{2}\left(\log_{z_k}\left(h_k\left(x_k\right)\right)\right)^T R_k^{-1}\left(\log_{z_k}\left(h_k\left(x_k\right)\right)\right). \end{aligned} \tag{8.1.7}$$

As with the EKF derivation in Section 6, we will break up the Jacobian into derivatives w.r.t. X_{k-1} and x_k:

$$\begin{aligned} \mathscr{J}_k = \frac{\partial S_k\left(\mathbf{X}_k\right)}{\partial \mathbf{X}_k} &= \left[\begin{array}{c} \frac{\partial S_k(\mathbf{X}_k)}{\partial \mathbf{X}_{k-1}} \\ \frac{\partial S_k(\mathbf{X}_k)}{\partial x_k} \end{array}\right] \\ &= \left[\begin{array}{c} \frac{1}{2}\frac{\partial S_{k-1}(\mathbf{X}_{k-1})}{\partial \mathbf{X}_{k-1}} + \Phi_k^T Q_k^{-1} \log_{x_k}\left(f_k\left(x_{k-1}\right)\right) \\ F_k^T Q_k^{-1}\left(\log_{x_k}\left(f_k\left(x_{k-1}\right)\right)\right) + H_k^T R_k^{-1}\left(\log_{z_k}\left(h_k\left(x_k\right)\right)\right) \end{array}\right]. \end{aligned} \tag{8.1.8}$$

8.1.3 Computing the Approximate Hessian $\mathscr{H}_k(\mathbf{X}_k)$

The approximate Hessian of $S(\mathbf{X}_k)$ is a matrix of shape $(k+1)N \times (k+1)N$ which gives all the partial second derivatives of S w.r.t. each element of $\mathbf{X}_k$. We define the symbol $\mathscr{H}_k(\mathbf{X}_k)$ to be the Gauss-Newton approximation to this Hessian, which simply neglects all second order terms.

Recalling that the Jacobian of S_k is given by

$$\begin{aligned}\mathscr{J}_k = \frac{\partial S_k(\mathbf{X}_k)}{\partial \mathbf{X}_k} &= \begin{bmatrix} \frac{\partial S_k(\mathbf{X}_k)}{\partial \mathbf{X}_{k-1}} \\ \frac{\partial S_k(\mathbf{X}_k)}{\partial x_k} \end{bmatrix} \\ &= \begin{bmatrix} \mathscr{J}_{k-1}(\mathbf{X}_{k-1}) + \Phi_k^T Q_k^{-1} \log_{x_k}(f_k(x_{k-1})) \\ F_k^T Q_k^{-1}\left(\log_{x_k}(f_k(x_{k-1}))\right) + H_k^T R_k^{-1}\left(\log_{z_k}(h_k(x_k))\right) \end{bmatrix},\end{aligned} \tag{8.1.9}$$

we can compute the derivative of $\mathscr{J}_k$ (neglecting second order terms) as:

$$\begin{aligned}\frac{\partial \mathscr{J}_k(\mathbf{X}_k)}{\partial \mathbf{X}_k} &= \begin{bmatrix} \frac{\partial^2 S_k(\mathbf{X}_k)}{\partial \mathbf{X}_{k-1}\partial \mathbf{X}_{k-1}} & \frac{\partial^2 S_k(\mathbf{X}_k)}{\partial \mathbf{X}_{k-1}\partial x_k} \\ \frac{\partial^2 S_k(\mathbf{X}_k)}{\partial x_k \partial \mathbf{X}_{k-1}} & \frac{\partial^2 S_k(\mathbf{X}_k)}{\partial x_k \partial x_k} \end{bmatrix} \\ &= \begin{bmatrix} \frac{1}{2}\frac{\partial^2 S_{k-1}(\mathbf{X}_{k-1})}{\partial \mathbf{X}_{k-1}\partial \mathbf{X}_{k-1}} + \Phi_k^T Q_k^{-1}\Phi_k & \Phi_k^T Q_k^{-1} F_k \\ F_k^T Q_k^{-1}\Phi_k & F_k^T Q_k^{-1} F_k + H_k^T R_k^{-1} H_k \end{bmatrix}\end{aligned} \tag{8.1.10}$$

which is simply the approximated Hessian:

$$\mathscr{H}_k = \begin{bmatrix} \mathscr{H}_{k-1}(X_{k-1}) + \Phi_k^T Q_k^{-1}\Phi_k & \Phi_k^T Q_k^{-1} F_k \\ F_k^T Q_k^{-1}\Phi_k & F_k^T Q_k^{-1} F_k + H_k^T R_k^{-1} H_k \end{bmatrix}. \tag{8.1.11}$$

8.1.4 Computing the Inverse Hessian $\mathscr{P}_k(X_k)$

As shown in equation 8.0.1, computing Δx_k requires the inverse Hessian, not the Hessian itself. Since we are interested only in updating the most recent state x_k and not the entire past history X_k, we need only compute the last row of the block inverse Hessian.

The inverse Hessian matrix is a first-order approximation of the covariance matrix of the error of a least-squares estimator. We will therefore call the inverse of the approximate Hessian (the covariance of X_k) $\mathscr{P}_k$:

$$\mathscr{P}_k \doteq \mathscr{H}_k^{-1}. \tag{8.1.12}$$

The lower right sub-block of this matrix is (to first order) the covariance of x_k, and will be denoted P_k:

$$P_k \doteq \left\{\mathscr{H}_k^{-1}\right\}_{(2,2)}. \tag{8.1.13}$$

Because the EKF performs only a single iterative step, it need only compute P_k. The Manifold EKF, by contrast, will iterate until convergence, meaning that we will be interested in computing both $\left\{\mathscr{H}_k^{-1}\right\}_{(2,1)}$ and P_k, which together constitute the last block row of $\mathscr{P}_k$. We will first proceed to compute the former, as it will turn out that the latter is simply a sub-term.

8.1.4.1 Computing $\{\mathscr{H}_k^{-1}\}_{(2,1)}$

Recalling the GN-approximate Hessian for the full case, and identifying terms as in Lemma 1:

$$\mathscr{H}_k = \begin{bmatrix} \underbrace{\mathscr{H}_{k-1}(X_{k-1}) + \Phi_k^T Q_k^{-1} \Phi_k}_{A} & \underbrace{\Phi_k^T Q_k^{-1} F_k}_{B} \\ \underbrace{F_k^T Q_k^{-1} \Phi_k}_{C} & \underbrace{F_k^T Q_k^{-1} F_k + H_k^T R_k^{-1} H_k}_{D} \end{bmatrix}, \tag{8.1.14}$$

we can easily compute the $(2,1)$ term of $\mathscr{H}_k^{-1}$ to be either

$$\begin{aligned} \{\mathscr{H}_k^{-1}\}_{(2,1)} &= -\left(F_k^T Q_k^{-1} F_k + H_k^T R_k^{-1} H_k\right)^{-1} F_k^T Q_k^{-1} \Phi_k \cdot \\ &\quad \begin{pmatrix} \mathscr{H}_{k-1}(X_{k-1}) + \Phi_k^T Q_k^{-1} \Phi_k \\ -\Phi_k^T Q_k^{-1} F_k \left(F_k^T Q_k^{-1} F_k + H_k^T R_k^{-1} H_k\right)^{-1} F_k^T Q_k^{-1} \Phi_k \end{pmatrix}^{-1} \end{aligned} \tag{8.1.15}$$

or

$$\begin{aligned} \{\mathscr{H}_k^{-1}\}_{(2,1)} &= -\begin{pmatrix} F_k^T Q_k^{-1} F_k + H_k^T R_k^{-1} H_k \\ -F_k^T Q_k^{-1} \Phi_k \underbrace{\left(\mathscr{H}_{k-1}(\mathbf{X}_{k-1}) + \Phi_k^T Q_k^{-1} \Phi_k\right)^{-1}} \Phi_k^T Q_k^{-1} F_k \end{pmatrix}^{-1} \cdot \\ &\quad F_k^T Q_k^{-1} \Phi_k \underbrace{\left(\mathscr{H}_{k-1}(X_{k-1}) + \Phi_k^T Q_k^{-1} \Phi_k\right)^{-1}} \end{aligned} \tag{8.1.16}$$

depending on which portion of Lemma 1 we choose to use. We will prefer the latter form, equation 8.1.16.

A number of simplifications can be made to equation 8.1.16. The indicated last factor can be altered using Lemma A.2, and recalling the definition of $\mathscr{P}_k = \mathscr{H}_k^{-1}$ to give:

$$\begin{aligned} \left(\mathscr{H}_{k-1}(X_{k-1}) + \Phi_k^T Q_k^{-1} \Phi_k\right)^{-1} &= \mathscr{P}_{k-1} + \mathscr{P}_{k-1} \Phi_k^T \left(-Q_k - \Phi_k \mathscr{P}_{k-1} \Phi_k^T\right)^{-1} \Phi_k \mathscr{P}_{k-1} \\ &= \mathscr{P}_{k-1} - \mathscr{P}_{k-1} \Phi_k^T \left(Q_k + \Phi_k \mathscr{P}_{k-1} \Phi_k^T\right)^{-1} \Phi_k \mathscr{P}_{k-1} \\ &= \mathscr{P}_{k-1} - \mathscr{P}_{k-1} \Phi_k^T \left(Q_k + \Phi_k \mathscr{P}_{k-1} \Phi_k^T\right)^{-1} \Phi_k \mathscr{P}_{k-1}. \end{aligned} \tag{8.1.17}$$

Furthermore, recalling that Φ_k is a $N \times kN$ matrix which is zero except for the last length N block column, and that $\mathscr{P}_{k-1}$ is $kN \times kN$ we find that

$$\Phi_k \mathscr{P}_{k-1} \Phi_k^T = \phi_k \{\mathscr{P}_{k-1}\}_{(2,2)} \phi_k^T = \phi_k P_{k-1} \phi_k^T. \tag{8.1.18}$$

Substituting 8.1.18 into equation 8.1.17 gives

$$\left(\mathscr{H}_{k-1}(X_{k-1}) + \Phi_k^T Q_k^{-1} \Phi_k\right)^{-1} = \mathscr{P}_{k-1} - \mathscr{P}_{k-1} \Phi_k^T \left(Q_k + \phi_k P_{k-1} \phi_k^T\right)^{-1} \Phi_k \mathscr{P}_{k-1}. \tag{8.1.19}$$

The term $\phi_k P_{k-1} \phi_k^T$ is seen frequently, and in fact has particularly intuitive geometric meaning. Recalling that ϕ_k is the Jacobian of the state error vector $\log_{x_k}(f_k(x_{k-1}))$ w.r.t. x_{k-1}, we can see that this term is the first order approximation of the covariance on this error vector induced by uncertainty in x_{k-1}. We will denote this term by

$$P'_{k-1} = \phi_k P_{k-1} \phi_k^T. \tag{8.1.20}$$

Equation 8.1.19, in turn, can be substituted back into 8.1.16, which can then be again simplified with Equation 8.1.18 to obtain

$$\begin{aligned} \{\mathscr{H}_k^{-1}\}_{(2,1)} &= -\left(\begin{array}{c} F_k^T Q_k^{-1} F_k + H_k^T R_k^{-1} H_k \\ \underbrace{-F_k^T Q_k^{-1} \left\{ I - P'_{k-1} \left(Q_k + \phi_k P_{k-1} \phi_k^T \right)^{-1} \right\} P'_{k-1}} Q_k^{-1} F_k \end{array} \right)^{-1} \\ &\quad \cdot F_k^T Q_k^{-1} \left\{ \Phi_k \mathscr{P}_{k-1} - P'_{k-1} \left(Q_k + P'_{k-1} \right)^{-1} \Phi_k \mathscr{P}_{k-1} \right\}. \end{aligned} \tag{8.1.21}$$

We will also find it convenient to define an identifier for the indicated term in Equation 8.1.21

$$M_k = F_k^T Q_k^{-1} \left\{ I - P'_{k-1} \left(Q_k + P'_{k-1} \right)^{-1} \right\} P'_{k-1}, \tag{8.1.22}$$

giving us:

$$\begin{aligned} \{\mathscr{H}_k^{-1}\}_{(2,1)} &= -\left(F_k^T Q_k^{-1} F_k + H_k^T R_k^{-1} H_k - M_k Q_k^{-1} F_k \right)^{-1} \\ &\quad \cdot F_k^T Q_k^{-1} \left\{ I - P'_{k-1} \left(Q_k + P'_{k-1} \right)^{-1} \right\} \Phi_k \mathscr{P}_{k-1} && (8.1.23) \\ &= -\left(F_k^T Q_k^{-1} F_k + H_k^T R_k^{-1} H_k - M_k Q_k^{-1} F_k \right)^{-1} \\ &\quad \cdot F_k^T Q_k^{-1} \left(I - P'_{k-1} \left(Q_k + P'_{k-1} \right)^{-1} \right) \phi_k \{\mathscr{P}_{k-1}\}_{(2,1)}. && (8.1.24) \end{aligned}$$

8.1.4.2 Computing P_k

As discussed, P_k is simply a sub-term in the computation of $\{\mathscr{H}_k\}_{(2,1)}$. This can be seen by examining the $(2,2)$ sub-block of an inverse matrix from Lemma 1, and comparing the form to that of the $(2,1)$ sub-block, which we have already computed:

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} = \begin{bmatrix} A^{-1} + A^{-1}B\left(D - CA^{-1}B\right)^{-1} CA^{-1} & -A^{-1}B\left(D - CA^{-1}B\right)^{-1} \\ -\underbrace{\left(D - CA^{-1}B\right)^{-1}} CA^{-1} & \underbrace{\left(D - CA^{-1}B\right)^{-1}} \end{bmatrix}. \tag{8.1.25}$$

Note that the $(2,1)$ sub-block contains the $(2,2)$ sub-block as a factor. Finding the corresponding term in Equation 8.1.24, we have that:

$$P_k = \left(F_k^T Q_k^{-1} F_k + H_k^T R_k^{-1} H_k - M_k Q_k^{-1} F_k \right)^{-1}. \tag{8.1.26}$$

8.1.5 Total Filter Update Equations

Now that we have computed the relevant terms, we can return to Equations 8.0.1 and 8.0.2. These equations will in general need to be iteratively applied.

We can iteratively compute new estimates for x_k by using Gauss-Newton update steps, recalling that the update computed at each step must be applied using the manifold's exponential mapping:

$$\mathbf{X}_k^+ = \exp_{X_k^-}\left(-\mathscr{H}_k^{-1}\mathscr{J}_k\right). \tag{8.1.27}$$

As discussed, we will only update x_k, giving

$$x_k^+ = \exp_{x_k^-}\left(-\left\{\mathscr{H}_k^{-1}\right\}_{(2,:)}\mathscr{J}_k\right). \tag{8.1.28}$$

We can now substitute the values of $\mathscr{J}_k$ and $\mathscr{H}_k^{-1}$ into Equation 8.0.1 and 8.0.2 to yield

$$x_k^+ = \exp_{x_k^-}\left(\begin{array}{c} -\left[\begin{array}{cc}\left\{\mathscr{H}_k^{-1}\left(X_k^-\right)\right\}_{(2,1)} & P_k\left(X_k^-\right)\end{array}\right] \\ \cdot\left[\begin{array}{c}\mathscr{J}_{k-1}\left(\mathbf{X}_{k-1}\right)+\Phi_k^T Q_k^{-1}\log_{x_k}\left(f_k\left(x_{k-1}\right)\right) \\ F_k^T Q_k^{-1}\left(\log_{x_k}\left(f_k\left(x_{k-1}\right)\right)\right)+H_k^T R_k^{-1}\left(\log_{z_k}\left(h_k\left(x_k\right)\right)\right)\end{array}\right]\end{array}\right). \tag{8.1.29}$$

As with the EKF derivation above, we assume that S_{k-1} was actually minimized by $\hat{\mathbf{X}}_{k-1}$, meaning that $\mathscr{J}_{k-1}\left(\hat{\mathbf{X}}_{k-1}\right)$ is zero. This gives us:

$$x_k^+ \quad = \quad \exp_{x_k^-}\left(\delta_{x_k}\right), \tag{8.1.30}$$

where

$$\begin{aligned}\delta_{x_k} \quad = \quad & \underbrace{-\left\{\mathscr{H}_k^{-1}\left(\mathbf{X}_k^-\right)\right\}_{(2,1)}\Phi_k^T Q_k^{-1}\log_{x_k}\left(f_k\left(x_{k-1}\right)\right)} \\ & -P_k\left(X_k^-\right)\left(F_k^T Q_k^{-1}\left(\log_{x_k}\left(f_k\left(x_{k-1}\right)\right)\right)+H_k^T R_k^{-1}\left(\log_{z_k}\left(h_k\left(x_k\right)\right)\right)\right).\end{aligned} \tag{8.1.31}$$

The first term (indicated) in the expression for δ_{x_k} can be simplified by recalling Equations 8.2.4 and 8.1.26:

$$\begin{aligned}
&\underbrace{\left\{\mathscr{H}_k^{-1}\left(\mathbf{X}_k^-\right)\right\}_{(2,1)}\Phi_k^T Q_k^{-1}\log_{x_k}\left(f_k\left(x_{k-1}\right)\right)}\\
= \quad & -P_k F_k^T Q_k^{-1}\left(I-P_{k-1}^{'}\left(Q_k+P_{k-1}^{'}\right)^{-1}\right)\phi_k\left\{\mathscr{P}_{k-1}\right\}_{(2,1)}\Phi_k^T Q_k^{-1}\log_{x_k}\left(f_k\left(x_{k-1}\right)\right)\\
= \quad & -P_k F_k^T Q_k^{-1}\left(I-P_{k-1}^{'}\left(Q_k+P_{k-1}^{'}\right)^{-1}\right)\underbrace{\phi_k P_{k-1}\phi_k^T}_{P_{k-1}^{'}} Q_k^{-1}\log_{x_k}\left(f_k\left(x_{k-1}\right)\right)\\
= \quad & -P_k \underbrace{F_k^T Q_k^{-1}\left(I-P_{k-1}^{'}\left(Q_k+P_{k-1}^{'}\right)^{-1}\right)P_{k-1}^{'}}_{M_k} Q_k^{-1}\log_{x_k}\left(f_k\left(x_{k-1}\right)\right)\\
= \quad & -P_k M_k Q_k^{-1}\log_{x_k}\left(f_k\left(x_{k-1}\right)\right). \qquad (8.1.32)
\end{aligned}$$

Note our identifications of factors equivalent to $P_{k-1}^{'}$ and M_k as defined in equations 8.1.20 and 8.1.22 respectively.

A summary of the equations for the joint update is given in Algorithm 8.1.

8.2 Time-only Update

It is sometimes the case that we wish to propagate our state estimate forward in time, without having new measurements to incorporate. When this is the case, we can use a cost function which simply omits the last term in Equation 8.1.2:

$$\begin{aligned}
S_k\left(\mathbf{X}_k\right) \quad = \quad & \frac{1}{2}S_{k-1}\left(\mathbf{X}_{k-1}\right)\\
& +\frac{1}{2}\left(\log_{x_k}\left(f_k\left(x_{k-1}\right)\right)\right)^T Q_k^{-1}\left(\log_{x_k}\left(f_k\left(x_{k-1}\right)\right)\right). \qquad (8.2.1)
\end{aligned}$$

This corresponds to performing a time update only: we add a new state x_k, but do not apply any measurements based on that state.

The Jacobian of this function is simply:

$$\mathscr{J}_k=\frac{\partial S_k\left(\mathbf{X}_k\right)}{\partial \mathbf{X}_k}=\begin{bmatrix}\frac{\partial S_k(\mathbf{X}_k)}{\partial \mathbf{X}_{k-1}}\\ \frac{\partial S_k(\mathbf{X}_k)}{\partial x_k}\end{bmatrix} = \begin{bmatrix}\frac{1}{2}\frac{\partial S_{k-1}(\mathbf{X}_{k-1})}{\partial \mathbf{X}_{k-1}}+\Phi_k^T Q_k^{-1}\log_{x_k}\left(f_k\left(x_{k-1}\right)\right)\\ F_k^T Q_k^{-1}\left(\log_{x_k}\left(f_k\left(x_{k-1}\right)\right)\right)\end{bmatrix}. \qquad (8.2.2)$$

Unlike the joint case, the optimal value of x_k, which will cause $\mathscr{J}_k$ to be zero, is readily apparent: simply setting $x_k = f_k\left(x_{k-1}\right)$ will accomplish this. This means that we need not iterate to perform a time update, but can simply compute the new value. However, as it is still useful to compute the covariance of the predicted estimate, we will proceed to compute the approximate Hessian and its inverse.

Algorithm 8.1 Joint Time and Measurement update step for the Manifold Extended Kalman Filter

For a system with state x_k, dynamic equations given by:

$$x_k = f_k(x_{k-1})$$

and measurement equations given by:

$$y_k = h_k(x_k).$$

We can perform a combined time and measurement update by repeatedly applying:

$$x_k^+ = \exp_{x_k^-}\left(\begin{array}{c} P_k M_k Q_k^{-1}\left(\log_{x_k}(f_k(x_{k-1}))\right) \\ -P_k F_k^T Q_k^{-1}\left(\log_{x_k}(f_k(x_{k-1}))\right) \\ -P_k H_k^T R_k^{-1}\left(\log_{z_k}(h_k(x_k))\right) \end{array}\right)$$

until convergence, where

$$\begin{aligned} P_k &= \left(F_k^T Q_k^{-1} F_k + H_k^T R_k^{-1} H_k - M_k Q_k^{-1} F_k\right)^{-1}, \\ M_k &= F_k^T Q_k^{-1}\left(I - P'_{k-1}\left(Q_k + P'_{k-1}\right)^{-1}\right) P'_{k-1}, \\ P'_{k-1} &= \phi_k P_{k-1} \phi_k^T \end{aligned}$$

and

$$\begin{aligned} F_k &\doteq \frac{\partial \log_{x_k}(f_k(x_{k-1}))}{\partial x_k}, \\ \phi_k &\doteq \frac{\partial \log_{x_k}(f_k(x_{k-1}))}{\partial x_{k-1}} = \frac{\partial \log_{x_k}(f)}{\partial f} \cdot \frac{\partial f_k(x_{k-1})}{\partial x_{k-1}}, \\ H_k &\doteq \frac{\partial \log_{z_k}(h_k(x_k))}{\partial x_k} = \frac{\partial \log_{z_k}(h)}{\partial h} \cdot \frac{\partial h_k(x_k)}{\partial x_k}. \end{aligned}$$

8.2.1 Computing the Approximate Hessian $\mathscr{H}_k(\mathbf{X}_k)$

Similarly to the joint case, we can simply compute the approximate Hessian to be

$$\mathscr{H}_{k|k-1}(X_k) = \begin{bmatrix} \mathscr{H}_{k-1}(X_{k-1}) + \Phi_k^T Q_k^{-1} \Phi_k & \Phi_k^T Q_k^{-1} F_k \\ F_k^T Q_k^{-1} \Phi_k & F_k^T Q_k^{-1} F_k \end{bmatrix}. \tag{8.2.3}$$

8.2.2 Computing $\left\{\mathscr{H}_k^{-1}\right\}_{(2,1)}$

Since the only difference between the time-only Hessian of Equation 8.2.3 and the joint Hessian of Equation 8.1.11 is the removal of the $H_k^T R_k^{-1} H_k$ term in the $(2,2,)$ sub-block, it is easy to show that $\left\{\mathscr{H}_k^{-1}\right\}_{(2,1)}$ is given by:

$$\begin{aligned} \left\{\mathscr{H}_k^{-1}\right\}_{(2,1)} &= -\left(F_k^T Q_k^{-1} F_k - M_k Q_k^{-1} F_k\right)^{-1} \\ &\quad \cdot F_k^T Q_k^{-1}\left(I - P'_{k-1}\left(Q_k + P'_{k-1}\right)^{-1}\right) \phi_k \left\{\mathscr{P}_{k-1}\right\}_{(2,1)} \end{aligned} \tag{8.2.4}$$

Algorithm 8.2 Time-only update step for the Manifold Extended Kalman Filter

For a system with state x_k, dynamic equations given by:

$$x_k = f_k(x_{k-1})$$

and measurement equations given by:

$$y_k = h_k(x_k).$$

We can perform a time update by setting :

$$x_k = f_k(x_{k-1}).$$

The covariance estimate of x_k is given by P_k:

$$P_k = \left(F_k^T Q_k^{-1} F_k - M_k Q_k^{-1} F_k\right)^{-1}$$

where

$$\begin{aligned} M_k &= F_k^T Q_k^{-1}\left(I - P'_{k-1}\left(Q_k + P'_{k-1}\right)^{-1}\right)P'_{k-1}, \\ P'_{k-1} &= \phi_k P_{k-1} \phi_k^T \end{aligned}$$

and

$$\begin{aligned} F_k &\doteq \frac{\partial \log_{x_k}(f_k(x_{k-1}))}{\partial x_k}, \\ \phi_k &\doteq \frac{\partial \log_{x_k}(f_k(x_{k-1}))}{\partial x_{k-1}} = \frac{\partial \log_{x_k}(f)}{\partial f} \cdot \frac{\partial f_k(x_{k-1})}{\partial x_{k-1}}, \\ H_k &\doteq \frac{\partial \log_{z_k}(h_k(x_k))}{\partial x_k} = \frac{\partial \log_{z_k}(h)}{\partial h} \cdot \frac{\partial h_k(x_k)}{\partial x_k}. \end{aligned}$$

which is simply Equation 8.1.24 with said term removed.

8.2.3 Computing P_k

As with the joint update, we observe that P_k is a factor in $\left\{\mathscr{H}_k^{-1}\right\}_{(2,1)}$: using the appropriate portion of Equation 8.2.4, we have:

$$P_k = \left(F_k^T Q_k^{-1} F_k - M_k Q_k^{-1} F_k\right)^{-1}. \tag{8.2.5}$$

A summary of the equations for the joint update is given in Algorithm 8.2.

8.3 Measurement-only Update

The Kalman Filter implementation allows the time and measurement updates to be fully separated: a time update followed by a measurement update produces the same result as if a joint update had been performed. For the EKF, this is no longer the case. We will therefore elect to always perform joint updates. In some cases, it will be necessary to store the value of the prior state x_{k-1} in addition to the time-propagated state $x_{k|k-1}$. In this way, a measurement update performed after a time update will simple discard $x_{k|k-1}$, and perform a joint update.

8.4 Discussion

The above Manifold EKF equations follow generally from the same derivation process used for the Extended Kalman Filter in Chapter 6. As discussed in that Chapter, the EKF makes what can be considered three important approximations:

1. The EKF updates only the most recent state x_k, not any prior states. This is equivalent to assuming that newly available measurements give information only about the noise in the estimate of x_k when in fact they also give information about the noise in prior measurements.

2. The EKF assumes that the estimate of the last state x_{k-1} obtained by previous steps is the minimizer of the cost function.

3. The EKF does not iterate until convergence.

The last of these approximations is not (or at least need not be) made by the Manifold EKF, and results in significant improvement in its performance (see Chapter 10).

Chapter 9

Manifold EKF Implementation Examples

Chapter 8 provided the key derivation of the Manifold EKF equations. Actual implementation of the Manifold EKF, however, requires numerical implementations of the basic manifold operations described in Chapter 7. In this chapter, we provide examples of the process of implementing these specific operations for several manifolds of interest. These will include the commonly used manifolds SO^3 (rotations) and S^2 (directions/bearings). We also include a hybrid manifold which we term Fb^3, which can be used to specify the state of a moving robotic platform equipped with an inertial measurement unit. This manifold is the product of:

- position ($\mathbb{R}^3$)
- attitude (SO^3)
- velocity ($\mathbb{R}^3$)
- Accelerometer biases ($\mathbb{R}^3$)
- Gyro biases ($\mathbb{R}^3$)

For each of these manifolds, we will:

1. Determine the parameterization of the manifold which corresponds to the principal chart
2. For manifolds which are also Lie groups, derive functions which operate on and return principal chart coordinates:
   (a) the composition operator on the group, and
   (b) the inversion operator on the group.
3. For all manifolds, derive functions which operate on and return principal chart coordinates:
   (a) the action ($\star$) of the group on the manifold,
   (b) the placement function f_x which returns the group operation which takes the origin to a point, and
   (c) the rescaling operator on the manifold.
4. Determine which Riemannian Metrics are invariant under the group action

These basic operations, once known, allow the logarithmic and exponential mappings to be computed as shown in Equations 7.5.2 and 7.5.5 respectively, and enable implementation of the Manifold EKF.

Before proceeding with the examples, we note in this process we will find ourselves computing the derivative of many scalar functions. Many of these functions have numerical instabilities when evaluated near zero, and therefore we give a Taylor series approximation that can be used instead of the function. Except where otherwise noted, this Taylor series expansion is about 0. The term $\mathscr{O}\left(\square^n\right)$ indicates a sum of discarded terms containing only powers of $\square$ raised to the power of n or greater.

9.1 SO^3 Under SO^3

SO^3 is the space of all orthogonal matrices with positive determinant. This means that SO^3 is a surface in $\mathbb{R}^9$ (the general set of $3x3$ matrices) which is defined by the following constraints:

$$\begin{aligned} R^T R &= I, && (9.1.1) \\ |R| &= +1. && (9.1.2) \end{aligned}$$

9.1.1 Principal Chart Representation

In order to determine the principal chart representation of SO^3, we can think of a manifold valued curve $R(t)$, and differentiate Equation 9.1.1 for this curve. The goal is to find the form of $\dot{R}$, the time derivative of a rotation, which is of course a vector in TSO^3. In particular, we wish to find $T_O SO^3$, the tangent space at the origin, as this space is used as the principal chart. The constraint is thus:

$$R(t) R^T(t) = I. \qquad (9.1.3)$$

Taking the derivative, we find:

$$\begin{aligned} \frac{d}{dt} R(t) R^T(t) &= 0, \\ \dot{R}(t) R^T(t) + R(t) \dot{R}^T(t) &= 0, \\ \dot{R}(t) R^T(t) &= -R(t) \dot{R}^T(t), \\ \dot{R}(t) R^T(t) &= -\left(\dot{R}(t) R^T(t)\right)^T. && (9.1.4) \end{aligned}$$

Equation 9.1.4 shows that the matrix $\dot{R}(t) R^T(t)$ is a skew-symmetric matrix, i.e. transposing and negation are the same. Since every skew-symmetric matrix has only three independent values, we can write this as a vector r under the so-called "hat operator"[35, chap. 2][45]:

$$\begin{aligned} \dot{R}(t) R^T(t) &= \hat{r}(t), && (9.1.5) \\ \dot{R}(t) &= \hat{r}(t) R(t) && (9.1.6) \end{aligned}$$

where we define

$$\begin{bmatrix} r_1 \\ r_2 \\ r_3 \end{bmatrix}^{\wedge} = \begin{bmatrix} 0 & r_3 & -r_2 \\ -r_3 & 0 & r_1 \\ r_2 & -r_1 & 0 \end{bmatrix}. \tag{9.1.7}$$

Thus, every derivative of a function on SO^3 is the product of a skew-symmetric matrix and a rotation matrix. Since we are trying to discover the principal chart, we want the tangent space at the origin, meaning we set $R(t) = I$ for $t = t_0$. This, then, gives us

$$\dot{R}(t) = \hat{r}(t), \tag{9.1.8}$$

meaning that derivatives at the origin are simply skew symmetric matrices. In turn, this means that that principal chart vectors can be identified with a 3-vector r. Geometrically, the vector r is termed a rotation vector, which is the axis of rotation scaled by the angle of rotation[45].

9.1.2 Group Operations

SO^3 is a Lie group, as well as a manifold, and therefore we will define the composition of two rotations and the inversion of a rotation.

9.1.2.1 Composition

The composition of two rotations R_b and R_a are determined in the standard way by matrix multiplication:

$$R_b \circ R_a = R_b R_a. \tag{9.1.9}$$

We need to be able to perform this composition on the principal chart representation of SO^3, meaning that we need a way to compose two rotation vectors directly. We will do this by first converting the rotation vectors to quaternions, composing, and converting back to rotation vectors[69, Sec. 7.3.4]:

$$r_b \circ r_a = r(q(r_b) \otimes q(r_a)). \tag{9.1.10}$$

The functions $r()$ and $q()$ represent quaternion-to-rotation-vector and rotation-vector-to-quaternion transformations respectively, while $\otimes$ represents quaternion composition[45]. The Jacobians of the composition function, then, are given by:

$$\frac{\partial (r_b \circ r_a)}{\partial r_b} = \frac{\partial r(q)}{\partial q} \cdot \frac{\partial (q_b \otimes q_a)}{\partial q_b} \cdot \frac{\partial q(r_b)}{\partial r_b} \tag{9.1.11}$$

and

$$\frac{\partial\left(r_b \circ r_a\right)}{\partial r_a} = \frac{\partial r\left(q\right)}{\partial q} \cdot \frac{\partial\left(q_b \otimes q_a\right)}{\partial q_a} \cdot \frac{\partial q\left(r_a\right)}{\partial r_a}. \tag{9.1.12}$$

For the conversion of quaternions (assumed to be unit-length) to rotation vectors, we have[47, Sec. 7.3.4]:

$$r\left(q\right) = 2\mathrm{sign}\left(q_4\right) \cdot \frac{\arcsin\left(\|v\|\right)}{\|v\|} v, \tag{9.1.13}$$

$$\frac{\partial r\left(q\right)}{\partial q} = \left[\begin{array}{cc} \left(\tau I_{3x3} + \xi v v^T\right) & -2v \end{array}\right] \tag{9.1.14}$$

where

$$v = \left[\begin{array}{c} q_1 \\ q_2 \\ q_3 \end{array}\right], \tag{9.1.15}$$

$$\begin{aligned} \tau &= 2\mathrm{sign}\left(q_4\right) \frac{\arcsin\left(\|v\|\right)}{\|v\|} && (9.1.16) \\ &= 2\mathrm{sign}\left(q_4\right) \left(1 + \frac{1}{6}\|v\|^2 + \mathcal{O}\left(\|v\|^4\right)\right) && (9.1.17) \end{aligned}$$

and

$$\begin{aligned} \xi &= 2\mathrm{sign}\left(q_4\right) \frac{\|v\| \sqrt{1 - \|v\|^2} - \arcsin\left(\|v\|\right)}{\|v\|^3} && (9.1.18) \\ &= -2\mathrm{sign}\left(q_4\right) \left(\frac{2}{3} + \frac{1}{5}\|v\|^2 + \mathcal{O}\left(\|v\|^4\right)\right). && (9.1.19) \end{aligned}$$

For the conversion of rotation vectors to unit quaternions, we have[47, Sec. 7.3.4]:

$$q\left(r\right) = \pm \left[\begin{array}{c} \kappa r \\ \cos\left(\frac{\|r\|}{2}\right) \end{array}\right] \tag{9.1.20}$$

and

$$\frac{\partial q\left(r\right)}{\partial r} = \pm \left[\begin{array}{c} \kappa I_{3x3} - \lambda r r^T \\ -\frac{1}{2}\kappa r^T \end{array}\right] \tag{9.1.21}$$

where

$$\kappa = \frac{\sin\left(\frac{\|r\|}{2}\right)}{\|r\|} \tag{9.1.22}$$

$$= \frac{1}{2} - \frac{1}{48}\|r\|^2 + \mathscr{O}\left(\|r\|^4\right) \tag{9.1.23}$$

and

$$\lambda = \frac{\sin\left(\frac{\|r\|}{2}\right)}{\|r\|^3} - \frac{\cos\left(\frac{\|r\|}{2}\right)}{2\|r\|^2} \tag{9.1.24}$$

$$= \frac{1}{24}\left(1 - \frac{\|r\|^2}{40}\right) + \mathscr{O}\left(\|r\|^4\right). \tag{9.1.25}$$

Note that there is always a sign ambiguity, as q and $-q$ represent the same rotation[45].

Finally, the quaternion composition operation is given by [45, eq. 172-174]

$$q_b \otimes q_a = \begin{bmatrix} q_{b4} & q_{b3} & -q_{b2} & q_{b1} \\ -q_{b3} & q_{b4} & q_{b1} & q_{b2} \\ q_{b2} & -q_{b1} & q_{b4} & q_{b3} \\ -q_{b1} & -q_{b2} & -q_{b3} & q_{b4} \end{bmatrix} q_a$$

$$= \begin{bmatrix} q_{a4} & -q_{a3} & q_{a2} & q_{a1} \\ q_{a3} & q_{a4} & -q_{a1} & q_{a2} \\ -q_{a2} & q_{a1} & q_{a4} & q_{a3} \\ -q_{a1} & -q_{a2} & -q_{a3} & q_{a4} \end{bmatrix} q_b, \tag{9.1.26}$$

with Jacobians given by

$$\frac{\partial\left(q_b \otimes q_a\right)}{\partial q_b} = \begin{bmatrix} q_{a4} & -q_{a3} & q_{a2} & q_{a1} \\ q_{a3} & q_{a4} & -q_{a1} & q_{a2} \\ -q_{a2} & q_{a1} & q_{a4} & q_{a3} \\ -q_{a1} & -q_{a2} & -q_{a3} & q_{a4} \end{bmatrix} \tag{9.1.27}$$

and

$$\frac{\partial\left(q_b \otimes q_a\right)}{\partial q_a} = \begin{bmatrix} q_{b4} & q_{b3} & -q_{b2} & q_{b1} \\ -q_{b3} & q_{b4} & q_{b1} & q_{b2} \\ q_{b2} & -q_{b1} & q_{b4} & q_{b3} \\ -q_{b1} & -q_{b2} & -q_{b3} & q_{b4} \end{bmatrix}. \tag{9.1.28}$$

9.1.2.2 Inversion

The inverse of a rotation matrix R is found by taking the transpose, R^T. This corresponds to negating the first three elements of the corresponding quaternion. As can be seen from Equation 9.1.20, negation of the first three elements of a quaternion is the same as negating the corresponding rotation vector r. Thus, the inversion operation is given by:

$$r^{-1} = -r \tag{9.1.29}$$

with a Jacobian given by:

$$\frac{\partial r^{-1}}{\partial r} = -I_{3x3}. \tag{9.1.30}$$

9.1.3 Manifold Operations

9.1.3.1 Placement Function

Since SO^3 is both a manifold and the chosen Lie Group of invariant operations on itself, the isotropy group $\mathscr{I}$ of operations which take the origin to itself is made up only of a single element, the identity. Thus, there is only a single choice of placement function for each manifold point, which is the point itself.

9.1.3.2 Group Action

Because SO^3 is its own Lie group, the action of SO^3 on itself is simply defined to be the same as the composition operation.

9.1.3.3 Action on $\mathbb{R}^3$

When they are thought of as operations, elements of SO^3 are perhaps most commonly used to rotate vectors in $\mathbb{R}^3$. This operation is not strictly necessary for dealing with SO^3 under SO^3, but will be useful in our other example manifolds.

The most straightforward way to compute the action of some rotation r on a vector x is to use the Rodriguez formula to convert r back to a direction cosine matrix R, and then perform the matrix multiplication[69, Sec. 7.3.3.3]:

$$\begin{aligned} r \star x &= R(r)x \\ &= \left(I + \alpha(\theta)\hat{r} + \beta(\theta)\hat{r}^2\right)x \end{aligned} \tag{9.1.31}$$

where

$$\theta = \|r\|, \tag{9.1.32}$$

$$\begin{aligned}\alpha &= \frac{\sin(\theta)}{\theta} \\ &\approx 1-\frac{\theta^2}{6}+\mathscr{O}\left(\theta^4\right) \qquad (9.1.33)\end{aligned}$$

and

$$\begin{aligned}\beta &= \frac{1-\cos(\theta)}{\theta^2} \\ &= \frac{1}{2}-\frac{\theta^2}{24}+\mathscr{O}\left(\theta^4\right). \qquad (9.1.34)\end{aligned}$$

The derivative of this operation with respect to x is simply R:

$$\frac{\partial(r\star x)}{\partial x}=R(r)=\left(I+\alpha\hat{r}+\beta\hat{r}^2\right), \qquad (9.1.35)$$

while the derivative with respect to r is given by[69, Sec. 7.3.3.3]:

$$\frac{\partial(r\star x)}{\partial x} = \hat{x}\left(\gamma(\theta)rr^T+\beta\hat{r}+\alpha I_{3x3}\right)-\hat{r}\hat{x}\left(\delta(\theta)rr^T+2\beta I_{3x3}\right) \qquad (9.1.36)$$

where

$$\begin{aligned}\gamma(\theta) &= \frac{\dot{\alpha}}{\theta}=\frac{(\cos(\theta)-\alpha)}{\theta^2} \qquad (9.1.37)\\ &= \frac{1}{3}-\frac{\theta^2}{30}+\mathscr{O}\left(\theta^4\right) \qquad (9.1.38)\end{aligned}$$

and

$$\begin{aligned}\delta(\theta) &= \frac{\dot{\beta}}{\theta}=\frac{\alpha-2\beta}{\theta^2} \qquad (9.1.39)\\ &= -\frac{1}{12}+\frac{\theta^2}{180}+\mathscr{O}\left(\theta^4\right). \qquad (9.1.40)\end{aligned}$$

9.1.3.4 Rescaling

Geometrically, the length of the rotation vector r is the angle by which to rotate about the unit-length axis of rotation given by $\frac{r}{\|r\|}$. There is thus an angle wrapping ambiguity, with a rotation vector $r=(n\pi+\theta)v$ for some unit-length v representing either $r=v\theta$ for even n or $r=-v\theta$ for odd n. As discussed in Chapter 7, the exponential mapping is only invertible for points within the tangential cut locus, which for SO^3 is a circle of radius π. Therefore, for some applications, it will be important to implement a “rescaling” operation, which maps an r value with a length greater than π to the corresponding vector with length $<\pi$.

In order to compute a rescaled vector, we must first determine the number of copies of 2π to subtract from the length of the vector such that its absolute length is minimized:

$$k_0 = \arg\min_{k\in\mathbb{N}} |\|r\| - 2\pi k| = \left\lfloor \frac{\|r\|}{2\pi} + \frac{1}{2} \right\rfloor . \tag{9.1.41}$$

With this value, we can compute[69, sec. 7.3.3.2] the rescaled vector r_r:

$$r_r = \left(1 - \frac{2\pi k_0}{\|r\|}\right) r \tag{9.1.42}$$

and its Jacobian:

$$\frac{\partial r_r}{\partial r} = I_{3x3} + \frac{2\pi k_0}{\|r\|^3} \hat{r}^2 . \tag{9.1.43}$$

9.1.4 Invariant Metrics

As discussed in Section 7.4, we need to verify that Equation 7.4.11 holds for whatever metric we choose for the manifold. Reproducing that equation here for convenience, we must verify that:

$$J^{-T}(I)\, Q_O J^{-1}(I) \quad = \quad Q_0 \tag{9.1.44}$$

for any choice of $I \in \mathscr{I}$, a member of the isotropy group. For SO^3, this is trivially verified for any choice of Q_O, since $\mathscr{I}$ consists only of the identity rotation, whose Jacobian is the identity matrix.

9.2 S^2 Under SO^3

The manifold S^2 is a very important space, as it can be used to represent 2-d measurements of direction or bearing. EO/IR cameras are bearing-only sensors, and thus S^2 is a natural manifold to use for camera-based measurements.

The space S^2 is a two-dimensional manifold, most readily visualizable as the unit-sphere embedded in standard 3-space. Distance on the unit-sphere is physically invariant to rotations of the sphere, meaning that we will choose SO^3 as the Lie Group of operations used to translate between points.

9.2.1 Principal Chart Representation

We must first determine what the nature of the tangent spaces to S^2 are. Geometrically, it is obvious that these tangent spaces are 2-d tangent planes to the sphere. To show this more rigorously, we can begin by viewing S^2 as a smooth surface constraint on coordinates in $\mathbb{R}^3$: S^2 is the set of all points $p \in \mathbb{R}^3$ such that $p^T p = 1$. If we consider some curve $\rho(t)$, where the function $\rho(\cdot)$ takes a time value $t \in \mathbb{R}$ to a unit length vector, we can compute the derivative of the constraint at any point:

$$\begin{aligned} \rho^T(t)\rho(t) &= 1, && (9.2.1)\\ \frac{d}{dt}\rho^T(t)\rho(t) &= \frac{d}{dt}1, && (9.2.2)\\ 2\rho^T(t)\frac{d\rho(t)}{dt} &= 0, && (9.2.3)\\ 2\rho^T(t)\dot{\rho}(t) &= 0. && (9.2.4)\end{aligned}$$

From the last equation, it is easy to see that at any point t, any valid value of the vector $\dot{\rho}$ must be orthogonal to the unit-length 3-vector $\rho(t)$, meaning they must lie on the 2-d plane whose normal vector is $\rho(t)$. Since $\dot{\rho}(t)$ is by definition the velocity vector to a manifold-valued curve, it is necessarily a member of the tangent space $T_{\rho(t)}S^2$. We arbitrarily choose the origin of S^2 to be the point $\begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^T$, meaning that the tangent space to the origin in the global parameterization is the set of all vectors of the form $\begin{bmatrix} 0 & a & b \end{bmatrix}^T$ for any $a, b \in \mathbb{R}$. This 2-d tangent space is $T_O S^2$, and is isomorphic to $\mathbb{R}^2$.

9.2.2 Global to Principal Chart Coordinate Conversions

Knowing the structure of the tangent space, we can choose a minimal 2-d coordinate representation for this space. This representation can be fully specified by finding a function which takes a unit-length 3-vector to principal chart coordinates, and vice versa.

9.2.2.1 Global to Principal Chart Coordinates

If we consider a global coordinate g:

$$g = \begin{bmatrix} g_x & g_y & g_z \end{bmatrix}^T. \qquad (9.2.5)$$

Simply choosing the y and z coordinates of this vector gives us a 2-vector

$$v = \begin{bmatrix} g_y & g_z \end{bmatrix}^T. \qquad (9.2.6)$$

The vector v could be used as the principal chart representation were it not for the fact that its length is not correct. As we have seen, the length of the principal chart representation of a point must be the distance between the origin and that point, which for S^2 means it must be the angle θ between the vector g and the origin vector $O = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$. This is (part of) what allows the logarithmic mapping to be used in lieu of vector subtraction to determine distance.

In fact, the length of the vector v is $\sqrt{g_y^2 + g_z^2}$, which geometrically is easily seen to be equal to $\sin(\theta)$. Therefore, we must scale the vector v by $\frac{\theta}{\sin(\theta)}$ in order to find our principle chart coordinates. This operation is trivial in most cases, but can be numerically unstable if θ is very close to either O or the antipodal point $-O = \begin{bmatrix} -1 & 0 & 0 \end{bmatrix}$. In those cases, we can compute a Taylor series expansion of the function $\frac{\theta}{\sin(\theta)}$ about the points $\theta = 0$

and $\theta = \pi$. The former expansion is

$$\frac{\theta}{\sin(\theta)} \approx 1 + \frac{1}{6}\theta^2 + \frac{7}{360}\theta^4 + \mathscr{O}\left(\theta^5\right), \tag{9.2.7}$$

while the latter is

$$\begin{aligned}\frac{\pi}{\sin(\pi)} &\approx -1 - \frac{\pi}{(\theta-\pi)} - \frac{\pi}{6}(\theta-\pi) - \frac{1}{6}(\theta-\pi)^2 \\ &\quad + \frac{7\pi}{360}(\theta-\pi)^3 - \frac{7}{360}(\theta-\pi)^4 + \mathscr{O}\left((\theta-\pi)^5\right).\end{aligned} \tag{9.2.8}$$

Note that this value will become arbitrarily large as θ approaches π. Since the point $-O$ is in fact the cut locus of S^2, the point where multiple geodesics meet and thus the point which has multiple coordinate values in the principal chart, this is a convenient place for this numerical instability to be located: $-O$ is already not uniquely representable.

The Jacobian of the principal chart coordinate vector p w.r.t. the global representation g is thus:

$$\frac{\partial p}{\partial g} = \frac{\partial}{\partial g}\frac{\theta}{\sin(\theta)}v \tag{9.2.9}$$

$$= \frac{\theta}{\sin(\theta)}\frac{\partial v}{\partial g} + \left(\frac{(\sin(\theta) - \theta\cos(\theta))}{\sin^2\theta}\right)v\frac{\partial\theta}{\partial g}. \tag{9.2.10}$$

It is helpful to note that

$$\theta = \operatorname{arctan2}(\|v\|, g_x), \tag{9.2.11}$$

$$= \operatorname{arctan2}\left(\sqrt{g_y^2 + g_z^2}, g_x\right) \tag{9.2.12}$$

and to further note that

$$\|v\| = \sqrt{g_y^2 + g_z^2} = \sin(\theta) \tag{9.2.13}$$

and

$$g_x = \cos(\theta). \tag{9.2.14}$$

We can thus compute the derivative of θ w.r.t. g:

$$\begin{aligned}\frac{\partial\theta}{\partial g} &= \frac{\partial \operatorname{arctan2}(y,x)}{\partial y}\cdot\frac{\partial\sqrt{g_y^2+g_z^2}}{\partial g} + \frac{\partial \operatorname{arctan2}(y,x)}{\partial x}\cdot\frac{\partial g_x}{\partial g} \\ &= \frac{g_x}{\|g\|^2}\cdot\left[\begin{array}{ccc} 0 & \frac{g_y}{\sqrt{g_y^2+g_z^2}} & \frac{g_z}{\sqrt{g_y^2+g_z^2}}\end{array}\right] + \frac{-\sqrt{g_y^2+g_z^2}}{\|g\|^2}\cdot\left[\begin{array}{ccc} 1 & 0 & 0\end{array}\right] \\ &= \frac{g_x}{\|v\|}\cdot\left[\begin{array}{cc} 0 & v^T\end{array}\right] - \|v\|\cdot\left[\begin{array}{ccc} 1 & 0 & 0\end{array}\right].\end{aligned} \tag{9.2.15}$$

Substituting this into Equation 9.2.10, we can find:

$$\begin{aligned}
\frac{\partial p}{\partial g} &= \frac{\theta}{\sin(\theta)}\frac{\partial v}{\partial g} + \left(\frac{(\sin(\theta) - \theta\cos(\theta))}{\sin^2\theta}\right) v \frac{\partial \theta}{\partial g} \\
&= \frac{\theta}{\sin(\theta)}\frac{\partial v}{\partial g} + \left(\frac{(\sin(\theta) - \theta\cos(\theta))}{\sin^2\theta}\right) v \left(\frac{g_x}{\|v\|}\cdot\begin{bmatrix} 0 & v^T \end{bmatrix} - \|v\|\cdot\begin{bmatrix} 1 & 0 & 0 \end{bmatrix}\right). \qquad (9.2.16)
\end{aligned}$$

Using equations 9.2.13 and 9.2.14, we have:

$$\begin{aligned}
\frac{\partial p}{\partial g} &= \frac{\theta}{\|v\|}\begin{bmatrix} \mathbf{0}_{2x1} & I_{2x2} \end{bmatrix} + \left(\frac{(\|v\| - \theta g_x)}{\|v\|^2}\right)\left(\frac{g_x}{\|v\|}\cdot\begin{bmatrix} \mathbf{0}_{2x1} & vv^T \end{bmatrix} - \frac{\|v\|^2}{\|v\|}\cdot\begin{bmatrix} v & \mathbf{0}_{2x2} \end{bmatrix}\right) \\
&= \frac{\theta}{\|v\|}\begin{bmatrix} \mathbf{0}_{2x1} & I_{2x2} \end{bmatrix} + \left(\frac{(\|v\| - \theta g_x)}{\|v\|^3}\right)\left(g_x\cdot\begin{bmatrix} \mathbf{0}_{2x1} & vv^T \end{bmatrix} - \|v\|^2\cdot\begin{bmatrix} v & \mathbf{0}_{2x2} \end{bmatrix}\right) \\
&= \frac{\theta}{\|v\|}\begin{bmatrix} \mathbf{0}_{2x1} & I_{2x2} \end{bmatrix} \\
&\quad + \left(\frac{g_x\|v\|}{\|v\|^3}\cdot\begin{bmatrix} \mathbf{0}_{2x1} & vv^T \end{bmatrix} - \frac{\theta g_x^2}{\|v\|^3}\begin{bmatrix} \mathbf{0}_{2x1} & vv^T \end{bmatrix} - \begin{bmatrix} v & \mathbf{0}_{2x2} \end{bmatrix} + \frac{\theta g_x}{\|v\|^3}\|v\|^2\cdot\begin{bmatrix} v & \mathbf{0}_{2x2} \end{bmatrix}\right) \\
&= \frac{\theta}{\|v\|}\begin{bmatrix} \mathbf{0}_{2x1} & I_{2x2} \end{bmatrix} + \left(\frac{g_x(\|v\| - \theta g_x)}{\|v\|^3}\right)\begin{bmatrix} \mathbf{0}_{2x1} & vv^T \end{bmatrix} + \left(\frac{\theta g_x}{\|v\|} - 1\right)\begin{bmatrix} v & \mathbf{0}_{2x2} \end{bmatrix} \\
&= \frac{\theta}{\|v\|}\begin{bmatrix} \mathbf{0}_{2x1} & I_{2x2} \end{bmatrix} + \left(\frac{g_x}{\|v\|^2} - g_x^2\frac{\theta}{\|v\|^3}\right)\begin{bmatrix} \mathbf{0}_{2x1} & vv^T \end{bmatrix} + \left(g_x\frac{\theta}{\|v\|} - 1\right)\begin{bmatrix} v & \mathbf{0}_{2x2} \end{bmatrix}. \\
&= \frac{\theta}{\sin(\theta)}\begin{bmatrix} \mathbf{0}_{2x1} & I_{2x2} \end{bmatrix} \\
&\quad + \left(\frac{\cos(\theta)}{\sin^2(\theta)} - \cos^2(\theta)\frac{\theta}{\sin^3(\theta)}\right)\begin{bmatrix} \mathbf{0}_{2x1} & vv^T \end{bmatrix} \\
&\quad + \left(\frac{\theta\cos(\theta)}{\sin(\theta)} - 1\right)\begin{bmatrix} v & \mathbf{0}_{2x2} \end{bmatrix}. \qquad (9.2.17)
\end{aligned}$$

Note that this derivative doesn't exist at the origin, when $g = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$, or $\theta = 0$. This will cause difficulties in evaluation of the Jacobian of the group action operation discussed in Section 9.2.3.2. The Jacobian at this point is seldom needed, but we have found the following approximation, when used for this special case, to be functional in practice:

$$\frac{\partial p}{\partial g} = \left(1 + \frac{1}{6}\|v\|^2 + 0.075\|v\|^4\right) I - \left(\frac{2}{3} + 0.2\|v\|^2\right) vv^T - \begin{bmatrix} v & 0_{2\times 2} \end{bmatrix}. \qquad (9.2.18)$$

9.2.2.2 Principal Chart to Global Coordinate Conversion

Given a principle chart vector, will see in Section 9.2.3.1 below how to find the rotation which takes the origin to that point. We can thus find the global representation of a point by simply finding its placement function f_p, and then rotating the 3-vector $\begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$ by that rotation:

$$\begin{aligned} p_g &= f_p \star O \\ &= f_p \star \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}. \end{aligned} \tag{9.2.19}$$

The derivative of this operation can be found by the chain rule:

$$\frac{\partial p_g}{\partial p} = \frac{\partial p_g}{\partial f_p} \cdot \frac{\partial f_p}{\partial p}. \tag{9.2.20}$$

The former derivative factor is the derivative of the action of SO^3 on $\mathbb{R}^3$, and is given in Section 9.1.3.3. The latter derivative is the Jacobian of the placement function of S^2, and will be given by Equation 9.2.23 below.

9.2.3 Manifold Operations

Since S^2 is not a Lie Group [60], we need only examine its manifold operations: we have already discussed the composition and inversion functions on SO^3 above.

9.2.3.1 Placement Function

We will choose SO^3 as the group $\mathscr{G}$ for this manifold, since intuition dictates that any rotation of the sphere should not change physical distance on the sphere. The placement function f_p corresponding to a sphere point p is thus the rotation that takes the origin vector $O = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$ to the global representation of the point p. As shown in Figure 9.2.1, we can find a rotation which takes the origin to any sphere point by simply finding the cross product

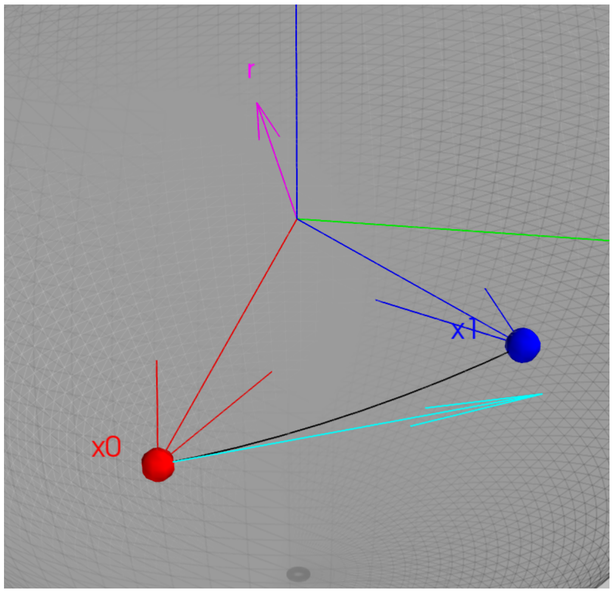


Figure 9.2.1: Figure illustrating the choice of principal chart coordinates on S^2. Point x_0, the origin, has global coordinates $\begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$. The principal chart vector pointing to any other point x_1 has coordinates $\begin{bmatrix} a & b \end{bmatrix}$, such that a vector $\begin{bmatrix} 0 & a & b \end{bmatrix}$ (cyan) "points in the direction of" x_1. The vector r is the rotation which takes the vector $\begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$ to the global coordinates of the point x_1.

of the origin with a vector of the form $\begin{bmatrix} 0 & a & b \end{bmatrix}$, yielding another vector perpendicular to $\begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$:

$$\begin{aligned} f_x &= \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \times \begin{bmatrix} 0 \\ x \end{bmatrix} \\ &= \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \times \begin{bmatrix} 0 \\ x_1 \\ x_2 \end{bmatrix} \\ &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ x_1 \\ x_2 \end{bmatrix} \\ &= \begin{bmatrix} 0 \\ x_2 \\ -x_1 \end{bmatrix}, \end{aligned} \tag{9.2.21}$$

$$pc = \frac{\arctan\left(\frac{\sqrt{g_y^2+g_z^2}}{g_x}\right)}{\sqrt{g_y^2+g_z^2}} \begin{bmatrix} g_y \\ g_z \end{bmatrix}. \tag{9.2.22}$$

The Jacobian of the placement function is straightforward to compute, and is given by:

$$\frac{\partial}{\partial x} f_x = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ -1 & 0 \end{bmatrix}. \qquad (9.2.23)$$

9.2.3.2 Group Action

Physically, the action of a member of SO^3 on a point in S^2 is straightforward: if a point on the surface of the sphere is identified with a unit-length 3-vector, then the action simply corresponds to the rotation of this vector about the point $\begin{bmatrix} 0 & 0 & 0 \end{bmatrix}$. As discussed in Chapter 7, we will require a 2-d principal chart representation of points on S^2, and will need to parameterize the group action operator so that its inputs and outputs are parameterized in terms of principal chart coordinates. In order to express this action in terms of values expressed in the principal chart, we can proceed by converting a principal chart value to its global representation (unit length 3-vector), rotate it, and then find the principal chart coordinates corresponding to the rotated vector

$$r \star p = \mathrm{g2pc}\left(r \star \mathrm{pc2g}\left(p\right)\right). \qquad (9.2.24)$$

The Jacobians of the group action, then, are easily found by the chain rule:

$$\frac{\partial r \star p}{\partial p} = \left.\frac{\partial \mathrm{g2pc}\left(\square\right)}{\partial \square}\right|_{\square = r\star \mathrm{pc2g}(p)} \cdot \left.\frac{\partial \left(r \star \triangle\right)}{\partial \triangle}\right|_{\triangle = \mathrm{pc2g}(p)} \cdot \frac{\partial \mathrm{pc2g}\left(p\right)}{\partial p}, \qquad (9.2.25)$$

$$\frac{\partial r \star p}{\partial r} = \left.\frac{\partial \mathrm{g2pc}\left(\square\right)}{\partial \square}\right|_{\square = r\star \mathrm{pc2g}(p)} \cdot \frac{\partial \left(r \star \mathrm{pc2g}\left(p\right)\right)}{\partial r}. \qquad (9.2.26)$$

The only difficulty with these equations lies with the first factor: as we have seen, $\mathrm{g2pc}\left(\square\right)$ is not differentiable when $\square = O$. This derivative will be evaluated at this point when the rotation r operating on a point p is in fact the inverse of the placement function of that point, i.e. when $r = f_p^{-1}$. This means that computing the logarithm of a point about itself will require this computation. Fortunately, there is a straightforward way around this problem: The inverse of the Jacobian of the placement function is equal to the Jacobian of the inverse placement function:

$$J^{-1}\left(f_p\right) \doteq \left.\frac{\partial \left(f_p \star \square\right)}{\partial \square}\right|_{\square = O}^{-1} = \left.\frac{\partial \left(f_p^{-1} \star \triangle\right)}{\partial \triangle}\right|_{\triangle = p} = J\left(f_p^{-1}\right). \qquad (9.2.27)$$

The Jacobian of the placement function $J\left(f_p\right)$ of any point p will involve evaluation of the derivative $\left.\frac{\partial \mathrm{g2pc}(\square)}{\partial \square}\right|_{\square = f_p \star O = p}$. Provided that the point p is not itself the origin, this places the evaluation point $\square$ away from the singularity which exists there. Thus only the derivatives of computations close to $\log_O\left(O\right)$ will require use of the approximation discussed in Section 9.2.2.1.

9.2.3.3 Rescale Operation

Analogously to the rescale operation on SO^3, we can re-normalize an S^2 vector x by computing the number of angular wraps:

$$k_0 = \left\lfloor \frac{\|x\|}{2\pi} + \frac{1}{2} \right\rfloor, \tag{9.2.28}$$

and then computing the rescaled vector as:

$$x_r = \left(1 - \frac{2\pi k_0}{\|x\|}\right) x. \tag{9.2.29}$$

The Jacobian of this operation is given by:

$$\frac{\partial x_r}{\partial x} = \left(1 - \frac{2\pi k_0}{\|x\|}\right) I_{2x2} + \frac{2\pi k_0}{\|x\|^3} x x^T. \tag{9.2.30}$$

9.2.4 Invariant Metrics

Unlike SO^3, S^2 is not a Lie group. Hence, it has a nontrivial isotropy group $\mathscr{I}$ of rotations which take the origin to itself. In fact, the group $\mathscr{I}$ is the set of all rotations with a rotation vector of the form $\begin{bmatrix} a & 0 & 0 \end{bmatrix}$ for any $a \in \mathbb{R}$. The Jacobians $J(I)$ of these group operations w.r.t. the S^2 point being acted on are rotations about the x axis, of the form

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix}. \tag{9.2.31}$$

Thus, according to Equation 7.4.11

$$J^{-T}(I) Q_O J^{-1}(I) = Q_0 \tag{9.2.32}$$

which in this case will be

$$R_x(\theta) Q_O R_x^T(\theta) = Q_O. \tag{9.2.33}$$

This equation is satisfied if $Q_O = aI_{2x2}$ for any real number a; that is, if Q_O is isotropic. This is sufficient for many applications.

9.3 Fb^3: State Space for a Flying Vehicle in 3-space

We here define the space Fb^3 as the set of all possible states of a flying vehicle.

A flying vehicle must keep track of its own position, attitude, and velocity, and typically will also estimate the current bias value added to its onboard rate gyro and accelerometer sensors. We define Fb^3 to have the following states:

- **position:** a 3-vector in some "fixed" or "world" coordinate frame

- **attitude:** a member of SO^3. When combined with position, this can be viewed as a member of the Lie Group SE^3[70, 71].
- **velocity**: 3-vector expressed in the body reference frame[1]
- **gyroscope bias**: additive bias to the rate gyro measurements, a member of $\mathbb{R}^3$.
- **accelerometer bias**: additive bias to the accelerometer measurements, a member of $\mathbb{R}^3$

9.3.1 Principal Chart Representation

Since it is in many ways simply a composition of multiple subspaces, all but one of which is simply $\mathbb{R}^3$, determining the principal chart of Fb^3 is quite straightforward: it is simply the concatenation of the principal chart representations of its constituent manifolds. Thus, the principle chart representation of Fb^3 is given by:

$$x = \begin{bmatrix} p^T & r^T & v^T & b_a^T & b_g^T \end{bmatrix}^T \tag{9.3.1}$$

where:

- p is the global position of the vehicle
- r is expressed in the principal chart of SO^3, and describes the vehicle's attitude
- v is the vehicle's velocity vector, expressed in the body frame
- b_a is the additive bias on the accelerometer measurements
- b_g is the additive bias on the rate gyroscope measurements

Another equivalent way to regard Fb^3 is to replace p and r with a frame representation f which represents a member of SE^3[70, 71].

9.3.2 Group Operations

Fb^3 is composed entirely of manifolds which are Lie Groups, and is a Lie Group itself. We will therefore define the operations of composition and inversion on Fb^3.

9.3.2.1 Composition

To motivate the composition of Fb^3 values, we will first examine the composition of SE^3 values, in a motivating example which will be reused for the remainder of this section. Let us define a frame $f_1 = \begin{bmatrix} t_1^T & r_1^T \end{bmatrix}^T$. This frame is a member of SE^3: it is an ordered trihedron of vectors with a common base point. It thus has a pose (consisting of both a position and an orientation) and can be used to express the pose of a rigid body. We presume

[1]Of course, velocity need not be estimated or expressed in the body frame. The choice of what frame velocity is expressed in can have effects on the definition of some of the basic manifold operations, in particular the action of Fb^3 on itself.

the presence of a similar reference or world frame f_g. The vector t_1 is a vector pointing from the origin of f_g to the origin of f_1, and its coordinates give the coordinates of f_1 expressed in frame f_g. The rotation vector r_1 specifies the orientation of f_1, and is defined such that its action on a 3-vector expressed in f_g produces a vector expressed in f_1:

$$v_{f_1} = r_1 \star v_{f_g}. \tag{9.3.2}$$

Now, consider that we have another frame f_2 which is defined relative to f_1. t_2 is then the position vector pointing from the origin of f_1 to the origin of f_2, expressed in f_1's coordinates. Similarly, r_2 takes free vectors in f_1's coordinates to the corresponding free vectors in f_2's coordinates. We define the composition operator[2] on SE^3 such that the composition of f_2 with f_1 yields the expression of the frame f_2 relative to the global frame f_g:

$$\begin{aligned} f_t = f_2 \circ f_1 &\doteq \begin{bmatrix} t_t^T & r_t^T \end{bmatrix}^T && (9.3.3)\\ &= \begin{bmatrix} \left(t_1 + r_1^{-1} \star t_2\right)^T & \left(r_2 \circ r_1\right)^T \end{bmatrix}^T. && (9.3.4)\end{aligned}$$

Intuitively, we have rotated the vector t_2, which was expressed in f_1, such that it is now expressed in f_g. We can then add this result to t_1, to find the location (relative to f_g) of frame f_2's origin. Similarly, the composition of r_2 and r_1 gives the total rotation from frame f_g coordinates to frame f_2 coordinates.

The Jacobian of the composition operator on SE^3 is straightforward to compute. We have that:

$$\begin{aligned} \frac{\partial f_t}{\partial f_2} &= \begin{bmatrix} \frac{\partial t_t}{\partial t_2} & \frac{\partial t_t}{\partial r_2} \\ \frac{\partial r_t}{\partial t_2} & \frac{\partial r_t}{\partial r_2} \end{bmatrix} \\ &= \begin{bmatrix} \frac{\partial\left(r_1^{-1} \star t_2\right)}{\partial t_2} & \mathbf{0}_{3x3} \\ \mathbf{0}_{3x3} & \frac{\partial\left(r_2 \circ r_1\right)}{\partial r_2} \end{bmatrix}. && (9.3.5)\end{aligned}$$

Both of the nonzero blocks of this derivative matrices have been derived already in Section 9.1: the upper and lower blocks can be found using Equations 7.4.9 and 9.1.11 respectively. Similarly, the Jacobian with respect to f_1 is given by:

$$\begin{aligned} \frac{\partial f_t}{\partial f_2} &= \begin{bmatrix} \frac{\partial t_t}{\partial t_1} & \frac{\partial t_t}{\partial r_1} \\ \frac{\partial r_t}{\partial t_1} & \frac{\partial r_t}{\partial r_1} \end{bmatrix} \\ &= \begin{bmatrix} I_{3x3} & \frac{\partial\left(r_1^{-1} \star t_2\right)}{\partial r_1} \\ \mathbf{0}_{3x3} & \frac{\partial\left(r_2 \circ r_1\right)}{\partial r_1} \end{bmatrix}. && (9.3.6)\end{aligned}$$

[2]This is certainly not the only way to define the composition operator on SE^3, nor is it the way employed by [47]. This method, however, has the advantage that it implies that the action of a frame f defined relative to some global frame f_g on a 3-vector expressed in f_g yields the same geometric vector expressed in frame f.

We can extend this definition of the composition of two frames in SE^3 to the composition of two Fb^3 values in a straightforward manner:

$$x_t = x_2 \circ x_1 \doteq \left[\begin{array}{cccc} (f_2 \circ f_1)^T & (v_2+v_1)^T & (b_{a2}+b_{a1})^T & (b_{g2}+b_{g1})^T \end{array}\right] \tag{9.3.7}$$

$$= \left[\begin{array}{ccccc} \left(t_1 + r_1^{-1} \star t_2\right)^T & (r_2 \circ r_1)^T & (v_2+v_1)^T & (b_{a2}+b_{a1})^T & (b_{g2}+b_{g1})^T \end{array}\right]. \tag{9.3.8}$$

The Jacobians of this operation with respect to both x_2 and x_1 is straightforward to find, given previous developments, and are given by

$$\frac{\partial x_t}{\partial x_2} = \left[\begin{array}{cccc} \frac{\partial(f_2 \circ f_1)}{\partial f_2} & \mathbf{0}_{6x3} & \mathbf{0}_{6x3} & \mathbf{0}_{6x3} \\ \mathbf{0}_{3x6} & I_{3x3} & \mathbf{0}_{3x3} & \mathbf{0}_{3x3} \\ \mathbf{0}_{3x6} & \mathbf{0}_{3x3} & I_{3x3} & \mathbf{0}_{3x3} \\ \mathbf{0}_{3x6} & \mathbf{0}_{3x3} & \mathbf{0}_{3x3} & I_{3x3} \end{array}\right] \tag{9.3.9}$$

and

$$\frac{\partial x_t}{\partial x_1} = \left[\begin{array}{cccc} \frac{\partial(f_2 \circ f_1)}{\partial f_1} & \mathbf{0}_{6x3} & \mathbf{0}_{6x3} & \mathbf{0}_{6x3} \\ \mathbf{0}_{3x6} & I_{3x3} & \mathbf{0}_{3x3} & \mathbf{0}_{3x3} \\ \mathbf{0}_{3x6} & \mathbf{0}_{3x3} & I_{3x3} & \mathbf{0}_{3x3} \\ \mathbf{0}_{3x6} & \mathbf{0}_{3x3} & \mathbf{0}_{3x3} & I_{3x3} \end{array}\right], \tag{9.3.10}$$

respectively.

9.3.2.2 Inversion

As with composition, the key to defining an inversion rule for Fb^3 is to examine the corresponding rule on SE^3. The defining characteristic of the inversion function is that the composition of any frame f with its inverse f^{-1} should yield the origin:

$$f \circ f^{-1} = O = \left[\begin{array}{cc} \mathbf{0}_{3x1} & \mathbf{0}_{3x1} \end{array}\right]^T. \tag{9.3.11}$$

From this constraint, we can readily derive the inverse of a frame:

$$f^{-1} = \left[\begin{array}{cc} (r \star -t)^T & (-r)^T \end{array}\right]^T. \tag{9.3.12}$$

Intuitively, the inverse expresses the negative of t in the f frame, and uses the inverse of the frame's rotation.

The Jacobian of the inverse is given by:

$$\begin{aligned}\frac{\partial f^{-1}}{\partial f} &= \begin{bmatrix} \frac{\partial(r\star -t)}{\partial t} & \frac{\partial(r\star -t)}{\partial r} \\ \mathbf{0}_{3x3} & \frac{\partial(-r)}{\partial r} \end{bmatrix} \\ &= \begin{bmatrix} \left(-\left.\frac{\partial(r\star\tau)}{\partial \tau}\right|_{\tau=-t}\right) & \left(\left.\frac{\partial(r\star\tau)}{\partial r}\right|_{\tau=-t}\right) \\ \mathbf{0}_{3x3} & -I_{3x3} \end{bmatrix}. \end{aligned} \tag{9.3.13}$$

9.3.3 Manifold Operations

9.3.3.1 Placement Function

Since Fb^3 is a Lie Group, its isotropy group $\mathscr{I}$ is a single identity element, meaning that its placement function under itself is simply identity.

9.3.3.2 Group Action

Because Fb^3 is its own group, the group action is the same as the composition operator we have already described:

$$x_2 \star x_1 \doteq x_2 \circ x_1. \tag{9.3.14}$$

9.3.3.3 Action of Fb^3 on $\mathbb{R}^3$

In order to define the action of Fb^3 on any vector $v \in \mathbb{R}^3$, we will first define the action of SE^3 on $\mathbb{R}^3$. Since we have opted to define the translational portion t of a frame $f = \begin{bmatrix} t & r \end{bmatrix}$ to be expressed relative to a global frame, this action is given by finding the vector from t to the vector's endpoint, and then rotating the result:

$$f \star v \doteq r \star (v - t). \tag{9.3.15}$$

This operation has an extremely intuitive interpretation: the action of a frame on a vector gives an expression of the same geometric vector expressed relative to that frame.

The Jacobian of this operation is easily found with respect to f:

$$\begin{aligned}\frac{\partial (f\star v)}{\partial f} &= \begin{bmatrix} \frac{\partial(f\star v)}{\partial t} & \frac{\partial(f\star v)}{\partial r} \end{bmatrix} \\ &= \begin{bmatrix} -\left.\frac{\partial(r\star y)}{\partial y}\right|_{y=v-t} & \left.\frac{\partial(r\star y)}{\partial r}\right|_{y=v-t} \end{bmatrix}, \end{aligned} \tag{9.3.16}$$

and with respect to v:

$$\frac{\partial (f\star v)}{\partial v} = \left.\frac{\partial (r\star y)}{\partial y}\right|_{y=v-t}. \tag{9.3.17}$$

The action of an Fb^3 vector x on a vector $v \in \mathbb{R}^3$ is most easily defined as being simply the action of its frame on that vector:

$$x \star v \doteq f_x \star v \tag{9.3.18}$$

where

$$x \doteq \begin{bmatrix} f^T & v^T & b_a^T & b_g^T \end{bmatrix}^T. \tag{9.3.19}$$

9.3.3.4 Rescaling

A member $x = \begin{bmatrix} t^T & r^T & v^T & b_a^T & b_g^T \end{bmatrix}^T$ of the manifold Fb^3 has only a single element, r, for which increasing the length by multiples of some value identifies the same point, the rescale operation of Fb^3 is a straightforward extension of that of SO^3:

$$\text{rescale}(x) = \begin{bmatrix} t^T & \left(\text{rescale}(r)^T\right) & v^T & b_a^T & b_g^T \end{bmatrix}^T \tag{9.3.20}$$

where rescale(r) is given by Equation 9.1.42. The Jacobian is given by:

$$\frac{\partial \text{rescale}(x)}{\partial x} = \begin{bmatrix} I_{3x3} & \mathbf{0}_{3x3} & \mathbf{0}_{3x3} & \mathbf{0}_{3x3} & \mathbf{0}_{3x3} \\ \mathbf{0}_{3x3} & \frac{\partial \text{rescale}(r)}{\partial r} & \mathbf{0}_{3x3} & \mathbf{0}_{3x3} & \mathbf{0}_{3x3} \\ \mathbf{0}_{3x3} & \mathbf{0}_{3x3} & I_{3x3} & \mathbf{0}_{3x3} & \mathbf{0}_{3x3} \\ \mathbf{0}_{3x3} & \mathbf{0}_{3x3} & \mathbf{0}_{3x3} & I_{3x3} & \mathbf{0}_{3x3} \\ \mathbf{0}_{3x3} & \mathbf{0}_{3x3} & \mathbf{0}_{3x3} & \mathbf{0}_{3x3} & I_{3x3} \end{bmatrix} \tag{9.3.21}$$

where the $(2,2)$ term of this block matrix is given by Equation 9.1.43.

9.3.4 Invariant Metrics

As with SO^3, the Lie group nature of Fb^3 ensures that any choice of inner product matrix in $T_O Fb^3$ will result in an invariant distance metric under the group action.

Chapter 10

Manifold EKF Performance

As discussed in Chapter 6, if we assume that the state and measurements of a system are members of vector spaces, the suboptimality of the EKF can be thought of as deriving from three major sources:

1. The EKF attempts to solve a nonlinear optimization problem, meaning it may in general produce a local rather than the global optimum.
2. The EKF allows updates to only the most recent system state, rather than the entire state history.
3. At each update step, the EKF performs only a single iteration of what optimally needs to be a multi-iteration estimation process.

The first source of error in inherent in the nature of nonlinear estimation, and cannot in general be removed by any practical estimator. If we assume that due to resource constraints only a single prior state may be stored and updated, then the second source of error is likewise not removable.

If the state and/or measurements of a system are members of more general Riemannian manifolds, the EKF has the additional problem of not inherently recognizing this portion of the system structure. Since the Manifoild EKF solves this problem and is furthermore not limited to a single iteration at each update step, it in some sense represents the best that can be done in terms of a filtering solution for manifolds. Further improvements over the Manifold EKF will likely need to resort to methods which store and update multiple prior states.

Even if the Manifold EKF were optimal according to some cost function, the actual performance of the filter in practice is a separate and highly relevant question. In this chapter, we present results demonstrating the performance of the Manifold EKF. In Section 10.1 we present a series of simulated results which demonstrate that the Manifold EKF is typically more consistent than other filter types in the case where state and measurement covariances are highly non-isotropic. We then proceed in Section 10.2 to show simulated results for a rapidly and unpredictably moving camera, comparing the Manifold EKF to a regular EKF using the same principal chart representation, demonstrating the benefit of the Manifold EKF equations.

10.1 Consistency of Measurement Update

In order to assess the performance of the Manifold EKF, we simulate and compare the performance of each of four filters in a simple experiment. This simple experiment consists of estimating the true platform body frame bearing to a target of interest, given a (noisy) measurement of the target bearing in the frame of a rigidly attached camera and perfect knowledge of the rotation between body and camera axes. A noisy prior estimate of this bearing is also available. The state and measurement spaces of this system are thus both S^2, and we here simulate only a measurement update. Multiple Monte Carlo trials of but a single measurement update are performed, in order to assess the relative performance of the measurement update of each of the four filters.

The four filters that are compared in this experiment use different parameterizations of the state space, but all use the principal chart on S^2 for the measurement space. These four filters are:

- **simpleEKF**: A straightforward EKF implementation, using a minimal spherical parameterization of the state space. That is, the simpleEKF filter uses a particular set of chart coordinates as its state space representation
- **simpleUKF**: An unscented[72, 73, 74, 42] version of the simpleEKF filter, using the same state-space representation.
- **constrainedUKF**: the constrained unscented filter introduced in [42], which was discussed in Section 5.1.3. This filter uses a 3-d vector as the state space, and constrains this vector to have unit norm.
- **manifoldEKF**: the Manifold EKF herein derived, which uses the principal chart of S^2 as the state space.

10.1.1 Metrics

Consistency and **conservativeness** are important attributes of estimators. If an estimator is consistent, its covariance estimate P is greater than or equal to the true covariance of its estimates $\hat{x}$ about the true value x, [75] satisfying:

$$E\left[(x-\hat{x})(x-\hat{x})^T\right] \quad \leq \quad P. \tag{10.1.1}$$

Intuitively, for a consistent estimator, the estimate of its own uncertainty of the state remains close to the true value, and information from each new incoming measurement will be appropriately incorporated. When a filter becomes inconsistent, it has a much higher probability of ignoring information in incoming measurements, because its artificially small covariance estimate falsely indicates that the propagated state estimate from the prior time step is more reliable than it really is (overconfidence). This can lead to **filter divergence**, wherein the state covariance becomes so small at some point that the filter effectively ignores all incoming measurements thereafter, causing it to drift far from the true value. Typically, a filter cannot recover from such divergence normally, and must be re-initialized.

Any estimate can be made consistent by, for example, setting $P = aI$, where a is a larger than the maximum eigenvalue of the true covariance. Such an estimate, however, may not be very useful, as an unnecessarily large covariance estimate effectively indicates that the estimate contains little information about the actual value. Thus, a filter may be consistent but overly conservative. Ideally, a filter would give covariance estimates which were just large enough for the filter to be consistent, but no larger.

One way to measure consistency in simulation is to compute a scaled version of the normalized estimation error squared (NEES)[76, 77]. This amounts to the mean of the squared Mahalanobis error between the true state and the state estimate:

$$\begin{aligned}\text{err}_{\text{Maha}} &= (\hat{x}-x)^T \hat{P}^{-1} (\hat{x}-x), && (10.1.2)\\ \text{NEES} &= E\left[\text{err}_{\text{Maha}}\right] \\ &= E\left[(\hat{x}-x)^T \hat{P}^{-1} (\hat{x}-x)\right]. && (10.1.3)\end{aligned}$$

For a consistent filter with gaussian state errors, the normalized squared error vector $(\hat{x}-x)^T \hat{P}^{-1} (\hat{x}-x)$ will be a sample from a chi-squared distribution with N degrees of freedom, where N is the dimension of the state vector x. The mean of this squared value should thus be equal to N: we therefore divide the mean squared value by N to cause the nominal value to be equal to 1, giving

$$\begin{aligned}\text{SNEES} &= \frac{1}{N}\text{NEES} \\ &= \frac{1}{N}E\left[(\hat{x}-x)^T \hat{P}^{-1} (\hat{x}-x)\right]. && (10.1.4)\end{aligned}$$

This Scaled NEES (SNEES) is an indicator of filter consistency: if it is greater than 1, the the average squared distance of the true value from the mean estimate is greater that predicted by the covariance matrix $\hat{P}$, hence the filter is inconsistent. Conversely, a value less than one represents a conservative filter, and the closer to 1 this value is, the less conservative (and hence more desirable) the estimates of that filter are.

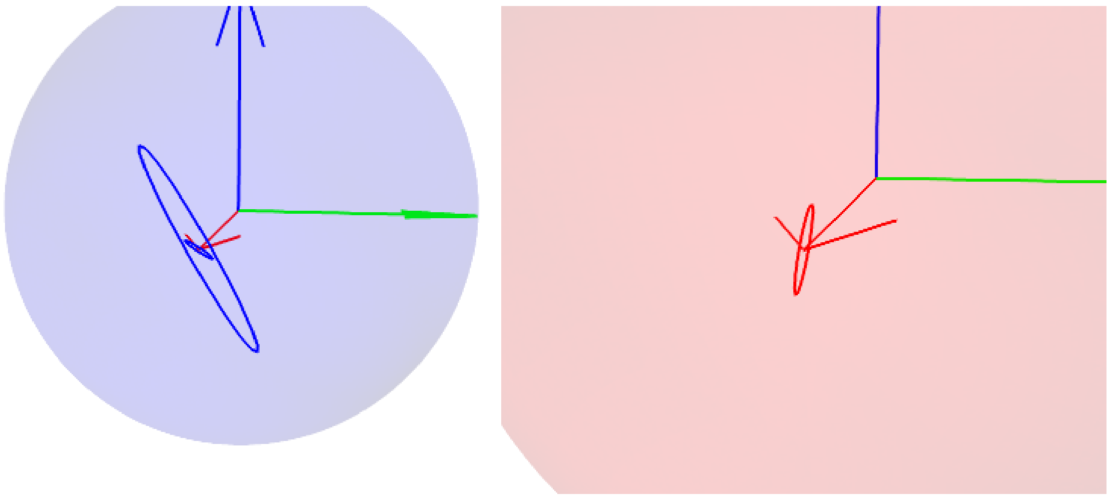
Because of the way the Kalman Filter operates, the measurement update never causes an increase in the covariance estimate. For this reason, there is a strong tendency for a filter, once it has an inconsistent estimate of the current state, to continue to produce increasingly inconsistent results, and hence diverge. Thus, the more often a filter produces inconsistent state estimates, the more often it is likely to diverge.

For our desired comparison between the four filters listed, we plot both the absolute *RMS* error between the true state and a given estimate, and the NEES value.

10.1.2 Results and Discussion

The Manifold EKF update equations require an iterative procedure in order to converge to the correct value. We experimented with the use of several different limits on the number of iterations the filter is allowed to perform. We found that the number of iterations has a strong effect on Manifold EKF performance, but allowing more than 3 iterations did not contribute significantly to performance. All Manifold EKF results presented in this section were obtained with 3 iterations in the measurement update.

It was found that both absolute error and SNEES varied significantly depending on the covariance of the prior state estimate $\left(\hat{P}\right)$ and that of the measurement (R). To examine this effect, values of both the state covariance $\hat{P}$ and the measurement covariance R were generated whose maximum eigenvalue was $10\times$ larger than their smaller eigenvalue, making the covariance matrices highly non-isotropic. Furthermore, covariances were generated with different orientations, covering the range $[0,\pi)$ in 5 degree increments. Thus, a set of 36 values of $\hat{P}$ were generated which were


(a)

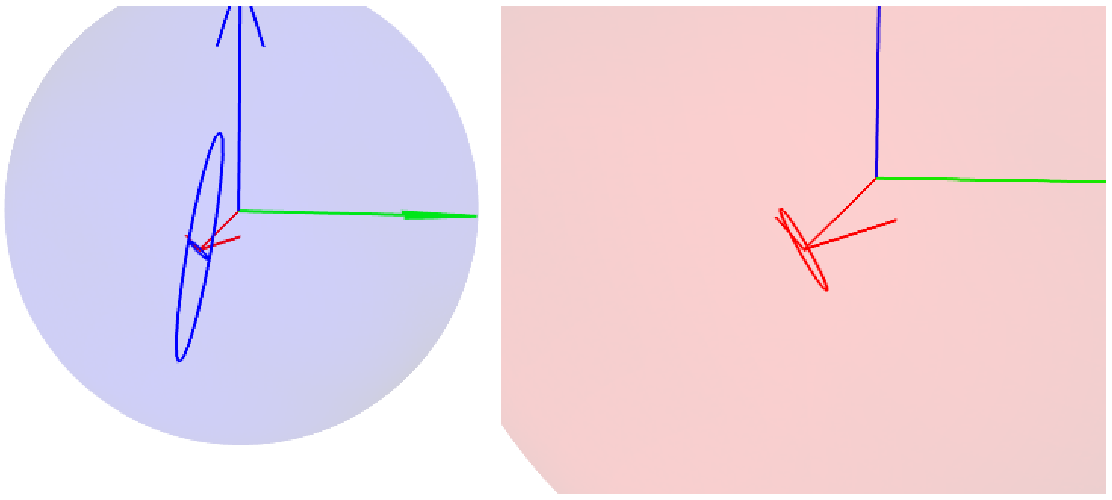

(b)

Figure 10.1.1: Examples of non-isotropic covariance matrices used in this experiment. Each sub-figure shows the covariance of the pre-update (larger ellipse) and post-update (smaller ellipse) state estimates (left image), and the covariance of the measurement (right image).

identical up to a rotation, and likewise for R, the measurement covariance. Some example $\hat{P}, R$ pairings are illustrated in Figure 10.1.1. Every possible combination of these candidates was tried, resulting in $36 \times 36 = 1296$ individual trials. For each such trial, 100 Monte Carlo sub-trials with different measurement noise values were performed, and the absolute and SNEES errors from the truth were recorded. The mean of these error metrics across the 100 Monte Carlo sub-trials generates an average error metric for each of the 1296 trials. These average values are plotted in Figure 10.1.2 for each of the four filters. Aggregate results are given in Table 10.1, which shows the percentage of cases in which each filter had an SNEES value ≤ 1. That table also shows an inconsistency value for each filter, which is the sum of the SNEES value over all inconsistent cases, divided by the number of such cases. This value gives some relative indication of how badly each of the four filters become inconsistent.

Table 10.1: Aggregate consistency data for each of the four filters used.

filter	% consistent cases	inconsistency
simpleEKF	22.3	7540.4
simpleUKF	54.8	245.6
constrainedUKF	44.4	11.0
manifoldEKF	65.2	1.2

There are several notable conclusions which can be drawn from these results:

1. The manifoldEKF is not always consistent: there are many rotations of the P and R matrices which cause the filter's SNEES to be greater than one. This is to be expected, given that the manifoldEKF, like the EKF itself, is a sub-optimal version of the complete least squares estimator.

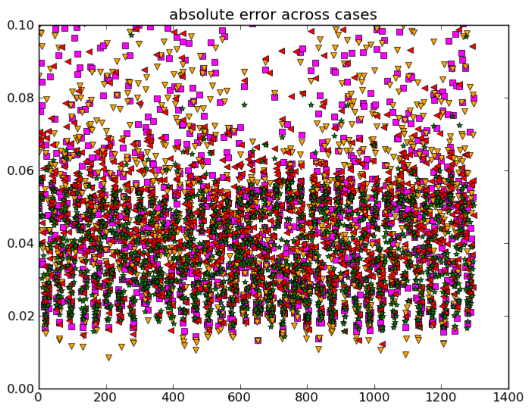
2. The manifoldEKF is generally significantly more accurate than any of the other filters used. There are many cases where other filters produce an estimate that is closer to the true value than the manifoldEKF's, but the manifoldEKF's RMS error is usually between 0.02 and 0.06 radians, while the accuracy of other methods can vary more widely depending on the rotation of the covariance matrices in a particular trial. (See Figure 10.1.2b).

3. The manifoldEKF is generally significantly more consistent than any of the other filters used. It's mean Mahalanobis error is below 1.0 in more cases, and is consistently much smaller when it is above 1.0 (see Table 10.1). Furthermore, the SNEES value is more tightly contained within a narrow band, while the consistency of the other filters varies wildly: they are sometimes both consistent and less conservative than the manifoldEKF, and sometimes wildly inconsistent.

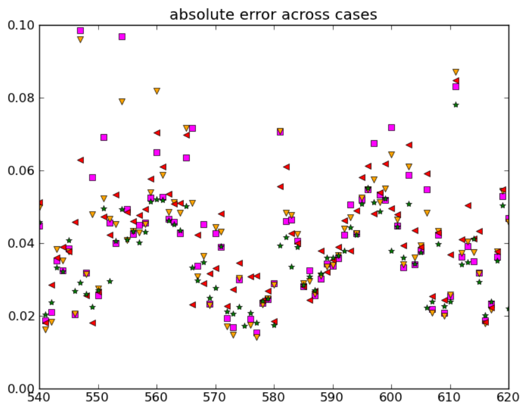
10.2 Camera-In-Hand Performance Comparison

The results presented in Section 10.1 deal only with the performance of the measurement update of the Manifold EKF under very specific circumstances. In order to give a broader feel for the Manifold EKF's performance, we will in this section present simulated results in a more complete problem setting.

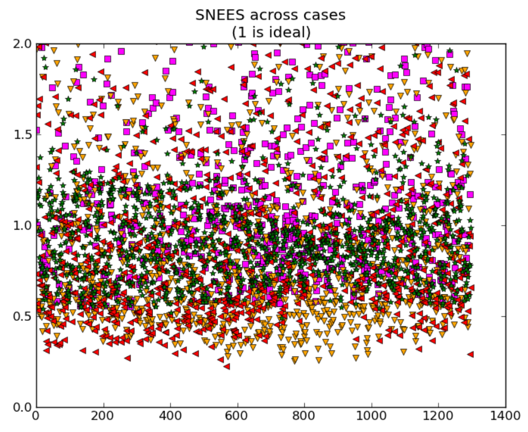
Specifically we deal with a variation on the so-called "camera-in-hand" problem: a platform consisting of an optical camera and an IMU is allowed to move through space with little knowledge of its motion dynamics. We use a simple constant-velocity model for the system dynamics, presuming that the world-frame velocity of the platform is altered over time by the addition of white random acceleration.



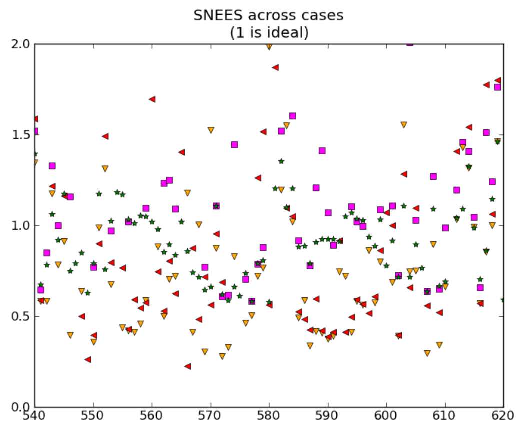
(a) full



(b) zoomed



(c) full



(d) zoomed

Figure 10.1.2: Average distance over 100 trials between truth and final filter estimate, for four filters: simpleEKF (magenta squares), simpleUKF (orange triangles, facing down), constrainedUKF (red triangles, facing left), and manifoldEKF (green stars). These results are for non-isotropic initial state and measurement covariances, where the larger eigenvector is $10\times$ the smaller in all cases. Each point on the x-axis represents a different rotation of the state and measurement covariances, in five degree increments.

Sub-figures 10.1.2a and 10.1.2b give absolute error in radians, while sub-figures 10.1.2c and 10.1.2d give SNEES. Sub-figures 10.1.2b and 10.1.2d are identical to 10.1.2a and 10.1.2c respectively, except that they are zoomed in the x direction, in order to more clearly show typical values.

SNEES is defined such that a perfectly consistent filter would always give a value of 1.0 on these plots, while a conservative filter would have values <1, and any value greater than 1 indicates filter inconsistency. Note that, while the Manifold EKF is not always consistent, it is consistent in significantly more cases than all the other filters, and tends to have lower SNEES values even when it is inconsistent.

10.2.1 Simulation Methodology

10.2.1.1 True System Dynamics

The simulated true system generates acceleration and angular velocity values by integrating gaussian white noise with a specified covariance. We used a standard deviation of $1\,\frac{m}{s^3}$ for acceleration noise (jerk) and $1\,\frac{rad}{s^2}$ for angular velocity noise (angular acceleration). In addition, the true system has a certain probability of using 100 times this noise level at any particular point. The probability of these random "course changes" was set to 0.5 for acceleration and 0.01 for angular velocity. This setup allows for sharp changes in the course of the system.

The principal chart representation used for the state, as discussed, is a minimal representation and will thus have a cut locus. This can be problematic for the SO^3 portion of the state, as a 180° rotation about the z axis is a cut locus point. The system's attitude is thus prevented from deviating from the origin by any rotation with magnitude greater than $\frac{\pi}{2}$: if this limit is ever about to be exceeded, the angular velocity is reversed in sign and multiplied by 0.5, allowing the system to "bounce off" this attitude boundary. This simple setup keeps the system attitude away from the singularity point of the principal chart representation. See Section 11.2.2 for a discussion of how future work could eliminate the need for such precautions.

The acceleration and angular velocity measurements generated by the true system are corrupted by biases, which biases are produced by integration of white noise (random walk). The driving random walk noise used is small, with a standard deviation of $1 \times 10^{-3}\,\frac{m}{s^2}$ for acceleration bias and $\frac{\pi}{180} \times 10^{-3}\,\frac{rad}{s}$ for angular velocity bias.

10.2.1.2 Estimator System Dynamics

The state of this system is a member of the space Fb^3 discussed in Chapter 9. We thus estimate the position p, velocity v, and attitude r, in addition to accelerometer bias b_a and gyro bias b_g. The estimator performs a simple integration of IMU inputs, with the dynamic equations given by:

$$\begin{bmatrix} p(t+\Delta t) \\ r(t+\Delta t) \\ v(t+\Delta t) \\ b_a(t+\Delta t) \\ b_g(t+\Delta t) \end{bmatrix} = \begin{bmatrix} p(t) + \Delta t \left((r(t))^{-1} \star v(t) \right) \\ \exp_{r(t)} \left(\Delta t \left(g(t) - b_g(t) \right) \right) \\ v(t) + \Delta t \left(a(t) - b_a(t) + r(t) \star G \right) \\ b_a(t) \\ b_g(t) \end{bmatrix} \tag{10.2.1}$$

where $g(t)$ and $a(t)$ represent the biased gyro and accelerometer measurements respectively, which are treated as inputs, and G is the world-frame gravity vector, which is taken to be $\begin{bmatrix} 0 & 0 & -9.80665 \end{bmatrix}$. Note also the use of the action $(\star)$ of the inverse attitude vector $(r(t))^{-1}$ on $v(t)$, which has the effect of mapping $v(t)$ from the body frame back to the inertial frame, and the exponential mapping based at $r(t)$ $\left(\exp_{r(t)}(\cdot)\right)$, which properly updates the attitude with the estimated bias.

10.2.1.3 System Measurements

A number of point targets, whose positions are a-priori known exactly by the system, are scattered throughout the environment, and the camera on the platform makes a noisy measurement of the bearing to each visible target. The measurements provided by the system are thus points in S^2. The equations of this measurement process are given by:

$$\begin{aligned} x_{bt} &= r_c \star r(t) \star (T - p(t)), && (10.2.2) \\ m(t) &= \text{g2pc}\left(\frac{x_{bt}}{\|x_{bt}\|}\right) && (10.2.3) \end{aligned}$$

where x_{bt} is the camera-frame vector from the body origin to the target, T is the target location in inertial coordinates, r_c is a fixed rotation from the body frame to the camera frame, and g2pc is the conversion between a unit-length vector and the principal chart representation of S^2, given in Section 9.2.2.1. The body-to-camera rotation r_c is chosen such that the camera looks along the body-frame $-z$ axis. The camera is chosen to have a circular field of view of 160°.

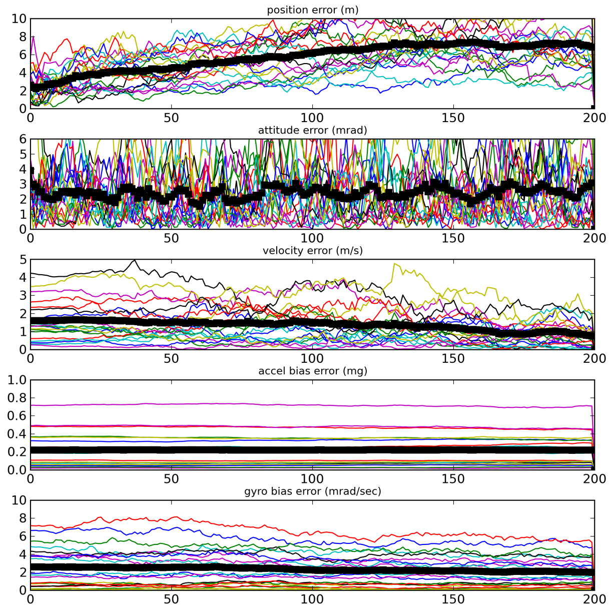
In actual application, recognizing which target corresponds to a given bearing measurement, the so-called data association problem, would be a significant portion of the effort required in the implementation of such a system. As the data association and filtering problems are in many (though not all) ways separable from one another, and as these results are intended to demonstrate the potential of the Manifold EKF, we make the common assumption that the data association problem is perfectly solved, giving the simulated system perfect knowledge of the position of the target associated with each bearing measurement.

10.2.2 Results and Discussion

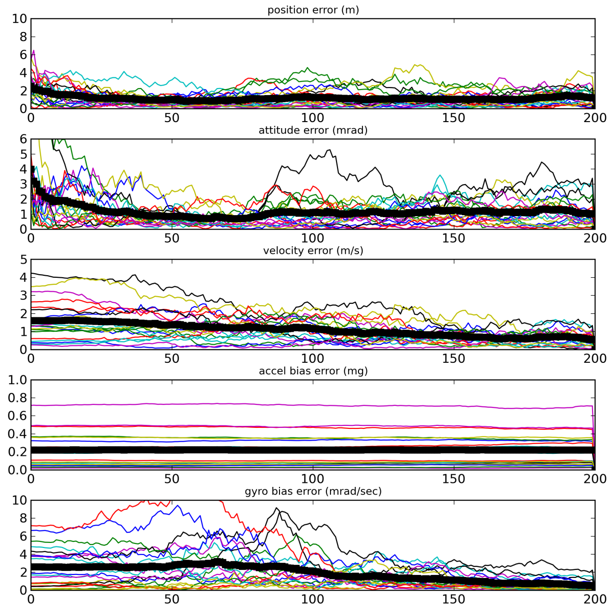
We simulate the system by performing 20 Monte Carlo trials. Each trial consists of allowing the platform to move within a virtual box 30 meters on a side, with targets placed randomly on each of the "walls" of the box. The platform is allowed to move for two seconds. These run parameters were chosen such that the platform would always be within the box, and would thus always be able to see some landmarks. This simulation is performed using the Manifold EKF, as well as a regular EKF which uses the same parameterization. Thus, any performance differences between the two filters will be due to the Manifold EKF equations, not simply to the principal chart representation used.

In terms of results, we show the absolute error of the state estimate from truth for each of the twenty runs, broken down by component (position error, attitude error, etc...), as well as the mean value of these errors across all trials. We also show the scaled NEES of the state estimate for each of the trails, and the mean of this value across all trials. In all cases, the Manifold EKF was limited to 3 iterations during each measurement update.

Several experiments were attempted, with different values of measurement uncertainty. With a small amount of measurement noise, (standard deviation of 3 degrees) the results of the EKF and the Manifold EKF are comparable, with the Manifold EKF having slightly less absolute error, especially in attitude, and being slightly more conservative. Results for a measurement noise standard deviation of 20 degrees are given in Figures 10.2.1 and 10.2.2, for the Manifold EKF and regular EKF respectively. In all of these plots, the x axis represents a sample number, with samples

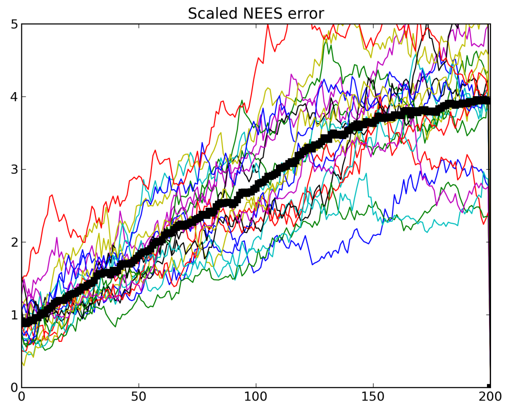


(a) EKF

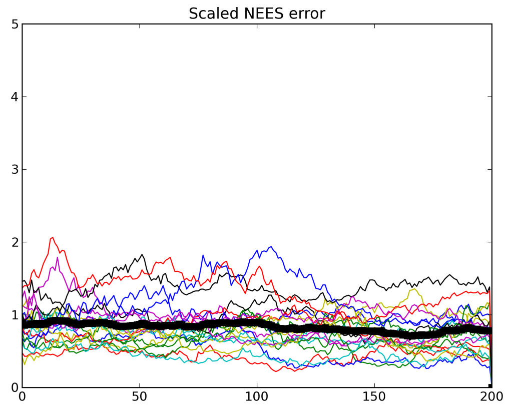


(b) Manifold EKF

Figure 10.2.1: Absolute error by state component, for the for the modified "camera-in-hand" problem..



(a) EKF



(b) Manifold EKF

Figure 10.2.2: Scaled NEES for the modified "camera-in-hand" problem.

being spaced by $\Delta_t = 0.01$ seconds. Individual Monte-Carlo trials are plotted as thin lines in various colors, and the heavy dark lines represent the mean of these individual trials.

From these results, it is clear that the Manifold EKF offers significant benefit in terms of accuracy and consistency that is due not to its parameterization of the space using the principal chart (since the EKF implementation shares that parameterization), but to the structure of its update procedures. This benefit is much more pronounced when there is high uncertainty in the measurements . Such results are to be expected, since the Manifold EKF performs multiple iterations during its measurement update, allowing the nonlinear measurement equations to be more correctly modeled by the filter.

Part III

Conclusion

Chapter 11

Conclusions and Future Work

11.1 Conclusions

This work presents two novel methods of estimating the state of a dynamic system. The first is an application specific method for use with systems performing Visual Odometry in a mostly planar scene. The second is the Manifold EKF, a generalized version of the Extended Kalman Filter which is explicitly designed to estimate manifold-valued states. We have derived the Manifold EKF as a simplification of an optimal least squares estimator, and have provided examples of the application of this filter to the commonly used manifolds SO^3 and S^2, as well as a space encompassing SO^3 which represents the state of a robotic vehicle. Our tests have demonstrated that the Manifold EKF is advantageous in terms of consistency and error performance under certain conditions when compared to other filtering methods. We feel that these promising results merit further study of the Manifold EKF, related filters, and their properties.

11.2 Future Work

We here address a number of possible next steps in the development of the ideas introduced in this work.

11.2.1 Application to a Broader Array of Manifolds

The manifolds to which the Manifold EKF is applied in this work are some of the most common and straightforward, with the chief ones being SO^3 and S^2, and combinations of these with $\mathbb{R}^n$. There are a number of other manifolds to which this technique could be applied. It would be interesting to thoroughly study the use of the Manifold EKF in tasks such as estimation of the essential matrix and/or homography matrices, both of which are manifold quantities[35, 40]. More complicated applications like bearing-only SLAM would also be interesting applications to study.

11.2.2 Removing Effect of Coordinate Singularity on Filter Performance

The Manifold EKF uses only principal chart coordinates to express the state. Since principal chart coordinates are minimal, there will always be a set of cut locus points, to which multiple coordinates could be assigned. For critically important manifolds like SO^3 and S^2, the cut locus is a set of null measure (a single point), and thus does not pose a problem in terms of integration over the manifold surface [69]. However, this singularity can still lead to filter instability. This work makes no attempt to remove this limitation. Numerically, these singularity points can often be avoided by suitably defining the geometry of the problem, perhaps even doing so dynamically in a manner similar to that used by the MEKF [78], where a separate "super-state" is kept which gives, in global coordinates, the location of a floating base point, and the actual state stores error from this point. It would be interesting to numerically analyze the effects of the cut locus, and determine the conditions under which it introduces numerical instability. Another possibility is to use a non-minimal representation of the manifold but still implement the exponential and logarithmic mappings (e.g. see [79]): it is certainly possible to use such a strategy for the Manifold EKF, though not using the principal chart means that the exponential mapping based at the origin will in general no longer be identity.

11.2.3 Optimal Cost Function of the Manifold EKF

The Manifold EKF is expressly not optimal in terms of the cost function used to derive it, as given in Equation 8.1.2, as it only allows updating of the most recent state x_k. Taking a cost function of the same form but with x_0 through x_{k-1} considered constants would also not produce the Manifold EKF. It would be interesting to find the cost function under which the Manifold EKF is optimal, as this could shed light on the reasons for its performance improvements. It would also be interesting to know how the performance of a filter which is optimal under some other cost function would compare with that of the Manifold EKF.

Bibliography

[1] F. Daum, "Nonlinear filters: beyond the Kalman filter," *IEEE Aerospace and Electronic Systems Magazine*, vol. 20, pp. 57–69, Aug. 2005. 1, 25

[2] J. Humpherys and J. West, "Kalman Filtering with Newton's Method," *IEEE Control Systems Magazine*, vol. 30, no. 6, pp. 101–106, 2010. 2, 24, 28, 29

[3] J. Humpherys, P. Redd, and J. West, "A Fresh Look at the Kalman Filter," *SIAM Review*, 2012. 2, 24, 28, 29

[4] R. Beard, D. Kingston, M. Quigley, D. Snyder, R. Christiansen, W. Johnson, T. McLain, and M. Goodrich, "Autonomous Vehicle Technologies for Small Fixed Wing UAVs," *AIAA Journal of Aerospace Computing, Information, and Communication*, vol. 2, Jan 2005. 4, 6, 17

[5] J. B. Saunders, B. Call, A. Curtis, R. W. Beard, and T. W. McLain, "Static and dynamic obstacle avoidance in miniature air vehicles," in *AIAA 5th Aviation, Technology, Integration, and Operations Conference*, Sep 2005. 4, 6

[6] D. B. Kingston and R. W. Beard, "Real-Time Attitude and Position Estimation for Small UAV's Using Low-Cost Sensors," in *AIAA Unmanned Unlimited Systems Conference and Workshop*, (Chicago, IL), Sept 2004. 4, 6

[7] R. S. Christiansen, "Design of an Autopilot for Small Unmanned Aerial Vehicles," Master's thesis, Brigham Young University, August 2004. 4, 6, 8, 17

[8] J. Volpe, "Vulnerability assessment of the transport infrastructure relying on the global positioning system," tech. rep., Office of the Assistant Secretary for Transportation Policy, U.S. Department of Transportation, Center, J. A. V. N. T. S., Aug 2001. 4

[9] J. Kim and S. Sukkarieh, "Airborne Simultaneous Localisation and Map Building," in *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on*, vol. 1, pp. 406–411 vol.1, 2003. 4, 5, 18

[10] M. Bryson and S. Sukkarieh, "Bearing-Only SLAM for an Airborne Vehicle," in *Australasian Conference on Robotics and Automation*, 2005. 4, 18

[11] J. Kim and S. Sukkarieh, "SLAM aided GPS/INS Navigation in GPS Denied and Unknown Environments," in *The 2004 International Symposium on GNSS/GPS*, 2004. 4

[12] J. Langelaan and S. Rock, "Passive gps-free navigation for small uavs," in *Proc. IEEE Aerospace Conference*, pp. 1–9, 2005. 4, 5

[13] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse, "Monoslam: Real-time single camera slam," vol. 29, pp. 1052–1067, June 2007. 4, 5, 18

[14] F. Dellaert, S. Thrun, and C. Thorpe, "Jacobian Images of Super-Resolved Texture Maps for Model-Based Motion Estimation and Tracking," in *1998 IEEE Workshop on Applications of Computer Vision*, (Princeton, NJ), pp. 2–7, IEEE Computer Society, October 1998. 5, 10, 11, 12

[15] F. Dellaert, C. Thorpe, and S. Thrun, "Super-Resolved Texture Tracking of Planar Surface Patches," in *1998 IEEE/RSJ International Conference on Intelligent Robotic Systems*, vol. 1, pp. 197–203, October 1998. 5, 10, 11, 12

[16] P. Corke, J. Lobo, and J. Dias, "An Introduction to Inertial and Visual Sensing," *The International Journal of Robotics Research*, vol. 26, no. 6, pp. 519–535, 2007. 5

[17] S. Roumeliotis, A. Johnson, and J. Montgomery, "Augmenting inertial navigation with image-based motion estimation," in *2002 IEEE International Conference on Robotics and Automation*, vol. 4, pp. 4326–4333, 2002. 5

[18] D. D. Diel, "Stochastic Constraints for Vision Aided Inertial Navigation," Master's thesis, MIT, Jan 2005. 5

[19] D. S. Bayard and P. B.Brugarolas, "An Estimation Algorithm for Vision-Based exploration of small bodies in space," in *2005 American Control Conference*, 2005. 5

[20] A. I. Mourikis and S. I. Roumeliotis, "A multi-state constraint kalman filter for vision-aided intertial navigation," in *2007 IEEE International Conference on Robotics and Automation*, 2007. 5

[21] L. Armesto, J. Tornero, and M. Vincze, "Fast Ego-motion Estimation with Multi-rate Fusion of Inertial and Vision," *The International Journal of Robotics Research*, vol. 26, no. 6, pp. 577–589, 2007. 5

[22] T. Viéville, E. Clergue, and P. Facao, "Computation of ego-motion and structure from visual and inertial sensors using the vertical cue," in *International Conference on Computer Vision*, pp. 591–598, 1993. 6

[23] J. Domke and Y. Aloimonos, "Integration of Visual and Inertial Information for Egomotion: A Stochastic Approach," in *2006 IEEE International Conference on Robotics and Automation*, pp. 2053–2059, 2006. 6

[24] J. Lobo and J. Dias, "Vision and inertial sensor cooperation using gravity as a vertical reference," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 12, pp. 1597–1608, 2003. 6

[25] J. Lobo and J. Dias, "Inertial sensed ego-motion for 3d vision," *Journal of Robotic Systems*, vol. 21, no. 1, pp. 3–12, 2004. 6

[26] J. Lobo and J. Dias, "Relative Pose Calibration Between Visual and Inertial Sensors," *The International Journal of Robotics Research*, vol. 26, no. 6, pp. 561–575, 2007. 6

[27] D. Nister, O. Naroditsky, and J. Bergen, "Visual Odometry," in *2004 IEEE Conference on Computer Vision and Pattern Recognition*, vol. 1, 2004. 9

[28] D. Nister, O. Naroditsky, and J. Bergen, "Visual odometry for ground vehicle applications," *Journal of Field Robotics*, vol. 23, pp. 3–20, Jan 2006. 9

[29] T. Kanade, O. Amidi, and Q. Ke, "Real-time and 3D Vision for Autonomous Small and Micro Air Vehicles," in *2004 IEEE Conference on Decision and Control*, vol. 2, pp. 1655–1662, 2004. 9

[30] J. Kehoe, R. Causey, A. Arvai, and R. Lind, "Partial Aircraft State Estimation from Optical Flow Using Non-Model-Based Optimization," in *2006 American Control Conference*, p. 6, 2006. 9

[31] J. Kehoe, A. Watkins, R. Causey, and R. Lind, "State Estimation using Optical Flow from Parallax-Weighted Feature Tracking," in *2006 AIAA Guidance, Navigation, and Control Conference*, 2006. 9

[32] K. Kaiser, N. Gans, and W. Dixon, "Position and orientation of an aerial vehicle through chained, vision-based pose reconstruction," in *2006 AIAA Guidance, Navigation, and Control Conference*, AIAA, Aug 2006. 9

[33] K. Kaiser, N. Gans, and W. Dixon, "Localization and Control of an Aerial Vehicle through Chained, Vision-Based Pose Reconstruction," in *2007 American Control Conference*, pp. 5934–5939, 2007. 9

[34] D. Nister, "An Efficient Solution To The Five-Point Relative Pose Problem," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 6, pp. 756–770, 2004. 9

[35] Y. Ma, S. Soatto, J. Kosecka, and S. S. Sastry, *An Invitation to 3-D Vision: From Images to Geometric Models*. SpringerVerlag, 2004. 9, 14, 15, 65, 95

[36] R. Eustice, H. Singh, J. Leonard, M. Walter, and R. Ballard, "Visually navigating the rms titanic with slam information filters," in *Robotics: Science and Systems*, (Cambridge, Mass), Jun 2005. 10

[37] K. Richmond and S. Rock, "A real-time visual mosaicking and navigation system," *Unmanned Untethered Submersible Technology*, 2005. 10

[38] S. Fleischer, *Bounded-Error Vision-Based Navigation Of Autonomous Underwater Vehicles*. PhD thesis, stanford university, 2000. 10

[39] S. Baker and I. Matthews, "Lucas-Kanade 20 years on: A Unifying Framework. Part 1: The Quantity Approximated, the Warp Update Rule, and the Gradient Descent Approximation," *International Journal of Computer Vision*, vol. 56, pp. 221–255, February 2004. 12, 21

[40] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2004. 14, 95

[41] J. Kim and S. Sukkarieh, "Robust Multi-loop Airborne SLAM in Unknown Wind Environments," in *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pp. 1536–1541, 2006. 18

[42] S. J. Julier and J. J. LaViola, Jr., "On Kalman Filtering With Nonlinear Equality Constraints," *IEEE Transactions on Signal Processing*, vol. 55, pp. 2774–2784, June 2007. 23, 26, 85

[43] F. L. Markley, "Attitude error representations for Kalman filtering," *Journal of Guidance Control and Dynamics*, vol. 26, no. 2, pp. 311–317, 2003. 23, 27

[44] A. Tyagi and J. W. Davis, "A recursive filter for linear systems on Riemannian manifolds," *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–8, June 2008. 23, 27

[45] M. D. Shuster, "A Survey of Attitude Representations," *The Journal of the Astronautical Sciences*, vol. 41, no. 4, pp. 439–517, 1993. 24, 25, 65, 66, 68

[46] X. Pennec, "Intrinsic Statistics on Riemannian Manifolds: Basic Tools for Geometric Measurements," *Journal of Mathematical Imaging and Vision*, vol. 25, no. 1, pp. 127–154, 2006. 24, 27, 52

[47] X. Pennec, *L'Incertitude dans les problemes de reconnaissance et de recalage: application en imagerie medicale et biologie moleculaire*. Phd, INRIA, Sophia-Antipolis, 1996. 24, 27, 39, 45, 46, 51, 52, 67, 80

[48] X. Pennec and N. Ayache, "Uniform distribution, distance and expectation problems for geometric features processing," *Journal of Mathematical Imaging and Vision*, vol. 9, no. 1, pp. 49–67, 1998. 24, 27

[49] J. L. Crassidis, F. L. Markley, and Y. Cheng, "A Survey of Nonlinear Attitude Estimation Methods," *Journal of Guidance, Control, and Dynamics*, vol. 30, no. 1, pp. 12–28, 2007. 25

[50] F. L. Markley, J. L. Crassidis, and Y. Cheng, "Nonlinear attitude filtering methods," in *AIAA Guidance, Navigation, and Control Conference*, Citeseer, 2005. 25

[51] M. Psiaki, "Backward-smoothing extended Kalman filter," *Journal of Guidance Control and Dynamics*, vol. 28, no. 5, p. 885, 2005. 25, 38

[52] G. Wahba, "A Least Squares Estimate of Spacecraft Attitude," *SIAM Review*, vol. 7, no. 3, p. 409, 1965. 25

[53] J. E. Keat, "Analysis of Least-Squares Attitude Determination Routine DOAOP," tech. rep., Computer Sciences Corporation, Feb. 1977. 25

[54] P. B. Davenport, "A Vector Approach to the Algebra of Rotations With Applications," Tech. Rep. August, National Aeronautics and Space Administration, 1968. 25

[55] M. D. Shuster and S. D. Oh, "Three-axis attitude determination from vector observations," *Journal of Guidance and Control*, vol. 4, pp. 70–77, 1981. 25

[56] F. L. Markley, "Attitude determination using vector observations and the singular value decomposition," *Journal of the Astronautical Sciences*, vol. 36, no. 3, p. 245258, 1988. 25

[57] M. D. Shuster, "A Simple Kalman Filter and Smoother for Spacecraft Attitude," *The Journal of the Astronautical Sciences*, vol. 37, no. 1, pp. 89–106, 1989. 25

[58] I. Y. Bar-Itzhack, "'REQUEST' - A recursive 'QUEST' algorithm for sequential attitude determination," *Journal of Guidance, Control, and Dynamics*, vol. 19, no. 5, pp. 1034–1038, 1996. 25

[59] M. L. Psiaki, "Extended quest attitude determination filtering," *Quest*, vol. 23, no. 2, pp. 206–214, 2000. 25

[60] T. Frankel, *The Geometry of Physics: An Introduction*. Cambridge University Press, 1997. 26, 39, 75

[61] X. Pennec, "Intrinsic Statistics on Riemannian Manifolds: Basic Tools for Geometric Measurements," *Journal of Mathematical Imaging and Vision*, vol. 25, pp. 127–154, July 2006. 26

[62] D. Simon and T. L. Chia, "Kalman Filtering with state equality constraints," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 38, no. 1, pp. 128–136, 2002. 26

[63] R. Kalman, "A New Approach to Linear Filtering and Prediction Problems," *Transactions of the ASME Journal of Basic Engineering*, pp. 35–45, 1960. 26

[64] S. J. Julier and J. Uhlmann, "A new extension of the Kalman filter to nonlinear systems," in *Int. Symp. Aerospace/Defense Sensing, Simul. and Controls*, vol. 3, (Oxford, UK), p. 26, University of Oxford, Citeseer, 1997. 26

[65] S. Soatto, R. Frezza, and P. Perona, "Recursive Motion Estimation on the Essential Manifold," *Lecture Notes in Computer Science*, vol. 801, pp. 60–72, 1994. 27

[66] M. D. Shuster and G. Natanson, "Quaternion computation from a geometric point of view," *Journal of the Astronautical Sciences*, vol. 41, no. 4, pp. 545–556, 1993. 27

[67] B. Triggs, P. McLauchlan, R. Hartley, and A. Fitzgibbon, "Bundle adjustment–a modern synthesis," *Vision Algorithms: Theory and Practice*, vol. 1883, pp. 298–372, 2000. 33

[68] T. Lefebvre, H. Bruyninckx, and J. De Schutter, "Kalman filters for non-linear systems: a comparison of performance," *International Journal of Control*, vol. 77, pp. 639–653, May 2004. 38

[69] X. Pennec, *L'Incertitude dans les problemes de reconnaissance et de recalage: application en imagerie medicale et biologie moleculaire*. Phd, INRIA, Sophia-Antipolis, 1996. 66, 69, 70, 71, 95

[70] Y. Ma, S. Soatto, J. Kosecka, and S. S. Sastry, *An Invitation to 3-D Vision*. Springer-Verlag, 2004. 79

[71] R. Hartley and A. Zisserman, *Multiple View Geometry in Compute Vision*. Cambridge, UK: Cambridge University Press, 2006. 79

[72] R. Van Der Merwe and E. Wan, "The square-root unscented Kalman filter for state and parameter-estimation," in *IEEE INTERNATIONAL CONFERENCE ON ACOUSTICS SPEECH AND SIGNAL PROCESSING*, vol. 6, pp. 3461–3464, Citeseer, 2001. 85

[73] E. Wan and R. Van Der Merwe, "The unscented Kalman filter," in *Kalman filtering and neural networks*, pp. 221–280, Wiley Publishing, 2001. 85

[74] S. J. Julier, Jeffrey, and K. Uhlmann, "Unscented Filtering and Nonlinear Estimation," *PROCEEDINGS OF THE IEEE*, vol. 92, pp. 401 – 422, 2004. 85

[75] T. Lefebvre, H. Bruyninckx, and J. De Schuller, "Comment on "A new method for the nonlinear transformation of means and covariances in filters and estimators" [and authors' reply]," *IEEE Transactions on Automatic Control*, vol. 47, no. 8, pp. 1406–1409, 2002. 85

[76] G. P. Huang, A. I. Mourikis, and S. I. Roumeliotis, "Analysis and improvement of the consistency of extended Kalman filter based SLAM," *2008 IEEE International Conference on Robotics and Automation*, pp. 473–479, May 2008. 85

[77] S. Huang and G. Dissanayake, "Convergence and Consistency Analysis for Extended Kalman Filter Based SLAM," *IEEE Transactions on Robotics*, vol. 23, no. 5, pp. 1036–1049, 2007. 85

[78] F. L. Markley, “Attitude Error Representations for Kalman Filtering,” *AIAA Journal of Guidance, Control, and Dynamics*, vol. 26, p. 2003, 2003. 96

[79] T. Lee, N. H. Mcclamroch, and M. Leok, “A Lie Group Variational Integrator for the Attitude Dynamics of a Rigid Body with Applications to the 3D Pendulum,” in *Proc. of the 2005 IEEE Conference on Control Applications*, (Toronto, Canada), pp. 962–967, 2005. 96

Index

Part IV

Appendices

Appendix A

Useful Matrix Lemmata

In this appendix, we state and then prove several matrix lemmata used in the text, especially in the context of the EKF and Manifold EKF derivations (Chapters 6 and 8 respectively).

We will provide proofs for the following lemmata:

Lemma 1. *Inverse of an Augmented Matrix*

$$\begin{aligned} M^{-1} &= \begin{bmatrix} A^{-1}+A^{-1}B\left(D-CA^{-1}B\right)^{-1}CA^{-1} & -A^{-1}B\left(D-CA^{-1}B\right)^{-1} \\ -\left(D-CA^{-1}B\right)^{-1}CA^{-1} & \left(D-CA^{-1}B\right)^{-1} \end{bmatrix} \\ &= \begin{bmatrix} \left(A-BD^{-1}C\right)^{-1} & -\left(A-BD^{-1}C\right)^{-1}BD^{-1} \\ -D^{-1}C\left(A-BD^{-1}C\right)^{-1} & D^{-1}+D^{-1}C\left(A-BD^{-1}C\right)^{-1}BD^{-1} \end{bmatrix} \end{aligned} \tag{A.0.1}$$

where M is a block matrix with sub-blocks labeled A,B,C, and D:

$$M = \begin{bmatrix} A & B \\ C & D \end{bmatrix}, \tag{A.0.2}$$

and where A and D are invertible matrices.

Lemma 2. *Sherman-Morrison-Woodbury Lemma, a.k.a. Matrix Inversion Lemma*

This Lemma allows us to add two matrices together and find the inverse of the sum from the inverses of the individual components. Specifically, we will see that

$$\left(D-CA^{-1}B\right)^{-1} = D^{-1}+D^{-1}C\left(A-BD^{-1}C\right)^{-1}BD^{-1}. \tag{A.0.3}$$

Lemma 3. Inverse Swapping Lemma

This Lemma allows us to "swap" a specific formula involving the inverse of the matrix D for one involving the inverse of the matrix A:

$$\left(D-CA^{-1}B\right)^{-1}CA^{-1} = D^{-1}C\left(A-BD^{-1}C\right)^{-1}. \tag{A.0.4}$$

A.1 Inversion of an "Augmented" Matrix

In many cases, we have an invertible matrix A, and we wish to find the inverse of a larger matrix of which A is the $(1,1)$ sub-block. That is, given a matrix

$$M = \begin{bmatrix} A & B \\ C & D \end{bmatrix}, \tag{A.1.1}$$

we would like to compute M^{-1}, given that A and D are both invertible[1].

[1]Note that both A and D need not both be invertible for M to be invertible, and one of the two formulas given in this section for M^{-1} will always work provided that M is in fact invertible. The only requirement for M to be invertible is that either $\left(D-CA^{-1}B\right)$ or $\left(A-BD^{-1}C\right)$ must be invertible (which necessarily implies that A and D are invertible respectively).

In order to determine this result, we will perform Gaussian elimination to determine a series of block-matrix multiplies which to both the matrix M and the block identity matrix:

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \quad : \quad \begin{bmatrix} I & 0 \\ 0 & I \end{bmatrix}. \tag{A.1.2}$$

Our goal will be to transform the left hand matrix into identity, which will cause the right-hand matrix to be M^{-1}.

First, we will eliminate the $(2,1)$ sub-block, summing scaled versions of the two rows to form a new second row:

$$\begin{bmatrix} I & 0 \\ -CA^{-1} & I \end{bmatrix}\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} A & B \\ 0 & -CA^{-1}B+D \end{bmatrix}. \tag{A.1.3}$$

We can then perform a right-multiplication to eliminate the $(1,2)$ sub-block, summing scaled versions of the two columns to form a new second column:

$$\begin{bmatrix} I & 0 \\ -CA^{-1} & I \end{bmatrix}\begin{bmatrix} A & B \\ C & D \end{bmatrix}\begin{bmatrix} I & -A^{-1}B \\ 0 & I \end{bmatrix} = \begin{bmatrix} A & 0 \\ 0 & D-CA^{-1}B \end{bmatrix}. \tag{A.1.4}$$

Bringing the two block-triangular matrix operations to the other side of the equation yields:

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} I & 0 \\ -CA^{-1} & I \end{bmatrix}^{-1}\begin{bmatrix} A & 0 \\ 0 & D-CA^{-1}B \end{bmatrix}\begin{bmatrix} I & -A^{-1}B \\ 0 & I \end{bmatrix}^{-1} \tag{A.1.5}$$

which is effectively a decomposition of M into a product of diagonal and triangular matrices. Note that the inverses of the triangular matrices are easily computed:

$$\begin{bmatrix} I & 0 \\ -Y & I \end{bmatrix}^{-1} = \begin{bmatrix} I & 0 \\ Y & I \end{bmatrix}, \tag{A.1.6}$$

because

$$\begin{aligned} \begin{bmatrix} I & 0 \\ -Y & I \end{bmatrix}\begin{bmatrix} I & 0 \\ Y & I \end{bmatrix} &= \begin{bmatrix} I & 0 \\ -Y & I \end{bmatrix}\begin{bmatrix} I & 0 \\ Y & I \end{bmatrix} \\ &= \begin{bmatrix} II+0Y & I0+0I \\ -YI+IY & -Y0+II \end{bmatrix} \\ &= \begin{bmatrix} I & 0 \\ 0 & I \end{bmatrix}, \end{aligned} \tag{A.1.7}$$

and similarly:

$$\begin{bmatrix} I & -Y \\ 0 & I \end{bmatrix}^{-1} = \begin{bmatrix} I & Y \\ 0 & I \end{bmatrix}, \tag{A.1.8}$$

because

$$\begin{aligned} \begin{bmatrix} I & -Y \\ 0 & I \end{bmatrix}\begin{bmatrix} I & Y \\ 0 & I \end{bmatrix} &= \begin{bmatrix} I & -Y \\ 0 & I \end{bmatrix}\begin{bmatrix} I & Y \\ 0 & I \end{bmatrix} \\ &= \begin{bmatrix} II-Y0 & IY-YI \\ 0I+I0 & 0Y-II \end{bmatrix} \\ &= \begin{bmatrix} I & 0 \\ 0 & I \end{bmatrix}, \end{aligned} \tag{A.1.9}$$

yielding

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} I & 0 \\ CA^{-1} & I \end{bmatrix} \begin{bmatrix} A & 0 \\ 0 & D-CA^{-1}B \end{bmatrix} \begin{bmatrix} I & A^{-1}B \\ 0 & I \end{bmatrix}. \tag{A.1.10}$$

Having this *LDU* decomposition of M allows us to easily find its inverse:

$$\begin{aligned}
M^{-1} &= \left(\begin{bmatrix} I & 0 \\ CA^{-1} & I \end{bmatrix} \begin{bmatrix} A & 0 \\ 0 & D-CA^{-1}B \end{bmatrix} \begin{bmatrix} I & A^{-1}B \\ 0 & I \end{bmatrix} \right)^{-1}, & \text{(A.1.11)} \\
\begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} &= \begin{bmatrix} I & A^{-1}B \\ 0 & I \end{bmatrix}^{-1} \begin{bmatrix} A^{-1} & 0 \\ 0 & \left(D-CA^{-1}B\right)^{-1} \end{bmatrix} \begin{bmatrix} I & 0 \\ CA^{-1} & I \end{bmatrix}^{-1} \\
&= \begin{bmatrix} I & -A^{-1}B \\ 0 & I \end{bmatrix} \begin{bmatrix} A^{-1} & 0 \\ 0 & \left(D-CA^{-1}B\right)^{-1} \end{bmatrix} \begin{bmatrix} I & 0 \\ -CA^{-1} & I \end{bmatrix} \\
&= \begin{bmatrix} A^{-1} & -A^{-1}B\left(D-CA^{-1}B\right)^{-1} \\ 0 & \left(D-CA^{-1}B\right)^{-1} \end{bmatrix} \begin{bmatrix} I & 0 \\ -CA^{-1} & I \end{bmatrix}, & \text{(A.1.12)} \\
\begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} &= \begin{bmatrix} A^{-1}+A^{-1}B\left(D-CA^{-1}B\right)^{-1}CA^{-1} & -A^{-1}B\left(D-CA^{-1}B\right)^{-1} \\ -\left(D-CA^{-1}B\right)^{-1}CA^{-1} & \left(D-CA^{-1}B\right)^{-1} \end{bmatrix}. & \text{(A.1.13)}
\end{aligned}$$

Equation A.1.13 is the first half of the lemma.

To see the second half, note that we arbitrarily chose to chancel the C term using row additions and the B term using column additions. If we choose the other alternative, we can similarly find:

$$\begin{aligned}
\begin{bmatrix} I & -BD^{-1} \\ 0 & I \end{bmatrix} \begin{bmatrix} A & B \\ C & D \end{bmatrix} &= \begin{bmatrix} A-BD^{-1}C & 0 \\ C & D \end{bmatrix}, \\
\begin{bmatrix} I & -BD^{-1} \\ 0 & I \end{bmatrix} \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} I & 0 \\ -D^{-1}C & I \end{bmatrix} &= \begin{bmatrix} A-BD^{-1}C & 0 \\ 0 & D \end{bmatrix}, & \text{(A.1.14)}
\end{aligned}$$

$$\begin{aligned}
\begin{bmatrix} A & B \\ C & D \end{bmatrix} &= \begin{bmatrix} I & -BD^{-1} \\ 0 & I \end{bmatrix}^{-1} \begin{bmatrix} A-BD^{-1}C & 0 \\ 0 & D \end{bmatrix} \begin{bmatrix} I & 0 \\ -D^{-1}C & I \end{bmatrix}^{-1}, \\
\begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} &= \begin{bmatrix} I & 0 \\ -D^{-1}C & I \end{bmatrix} \begin{bmatrix} \left(A-BD^{-1}C\right)^{-1} & 0 \\ 0 & D^{-1} \end{bmatrix} \begin{bmatrix} I & -BD^{-1} \\ 0 & I \end{bmatrix} \\
&= \begin{bmatrix} \left(A-BD^{-1}C\right)^{-1} & 0 \\ -D^{-1}C\left(A-BD^{-1}C\right)^{-1} & D^{-1} \end{bmatrix} \begin{bmatrix} I & -BD^{-1} \\ 0 & I \end{bmatrix}, \\
\begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} &= \begin{bmatrix} \left(A-BD^{-1}C\right)^{-1} & -\left(A-BD^{-1}C\right)^{-1}BD^{-1} \\ -D^{-1}C\left(A-BD^{-1}C\right)^{-1} & -D^{-1}C\left(A-BD^{-1}C\right)^{-1} \end{bmatrix}. & \text{(A.1.15)}
\end{aligned}$$

In summary, then, the inverse of a block matrix can be found in either of two ways. One involves inverses of A,

$$M^{-1} = \begin{bmatrix} A^{-1}+A^{-1}B\left(D-CA^{-1}B\right)^{-1}CA^{-1} & -A^{-1}B\left(D-CA^{-1}B\right)^{-1} \\ -\left(D-CA^{-1}B\right)^{-1}CA^{-1} & \left(D-CA^{-1}B\right)^{-1} \end{bmatrix}, \tag{A.1.16}$$

while the other involves the explicit inverse of D

$$M^{-1} = \begin{bmatrix} \left(A-BD^{-1}C\right)^{-1} & -\left(A-BD^{-1}C\right)^{-1}BD^{-1} \\ -D^{-1}C\left(A-BD^{-1}C\right)^{-1} & D^{-1}+D^{-1}C\left(A-BD^{-1}C\right)^{-1}BD^{-1} \end{bmatrix}. \tag{A.1.17}$$

This derivation was performed assuming that both A and D are invertible. It is easy to show, however, that if one of these matrices is not invertible, the form of M^{-1} which does not involve that inverse is still valid. Thus, Equation A.1.16 is still valid if D is rank deficient, provided A is not: if the matrix M is full rank, this equation will give the inverse. A similar statement is true of Equation A.1.17.

As we shall see, most of the other useful lemmata that we present in this section follow directly from equating terms in these two matrices.

A.2 Woodbury Matrix Identity

The Woodbury matrix identity allows us to compute the inverse of an additively updated invertible matrix with an additive update to its inverse:

$$\left(A - BD^{-1}C\right)^{-1} \quad = \quad A^{-1} + A^{-1}B\left(D - CA^{-1}B\right)^{-1}CA^{-1}. \tag{A.2.1}$$

The left-hand side of this equation is simply the $(1,1)$ sub-block in equation A.1.17, which the right-hand side is the $(1,1)$ sub-block in equation A.1.16. Note also that equating the $(2,2)$ sub-blocks of the matrices in those two equations and swapping A for D and B for C yields the same result.

A more common form of the same result can be obtained by letting $E = -D^{-1}$:

$$(A + BEC)^{-1} \quad = \quad A^{-1} - A^{-1}B\left(E^{-1} + CA^{-1}B\right)^{-1}CA^{-1}. \tag{A.2.2}$$

A.3 Inverse Swapping

As we have seen, the Woodbury identity comes from equating either of the diagonal terms in equations A.1.17 and A.1.16. Similarly, by equating the off-diagonal terms, we obtain:

$$\left(D - CA^{-1}B\right)^{-1}CA^{-1} = D^{-1}C\left(A - BD^{-1}C\right)^{-1}. \tag{A.3.1}$$

This lemma allows us to swap the computation of A^{-1} for that of D^{-1} and vice versa.