The Report Committee for Kristen Alyssa Michaelson certifies that this is the approved version of the following report:

A Multiplicative Multi-State Constraint Kalman Filter

SUPERVISING COMMITTEE:

Renato Zanetti, Supervisor

Maruthi Akella

A Multiplicative Multi-State Constraint Kalman Filter

by

Kristen Alyssa Michaelson

REPORT

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN ENGINEERING

THE UNIVERSITY OF TEXAS AT AUSTIN

December 2020

To my parents

Acknowledgments

I would like to thank Dr. Zanetti for his enthusiastic support throughout my time in graduate school. This has been such a wonderful opportunity, and I've learned so much. I look forward to continuing our work together.

I would also like to thank my graduate school recommenders: Dr. Breuer, Dr. Henann, and Steve Gensheimer. Coming to graduate school has truly been the adventure of a lifetime, and I'm so grateful for all your help along the way. I share my success in completing my master's degree with you.

Thank you, Corey and Bryan, for helping me through coursework and quals. Any success I've had here is due in no small part to you, and I only hope I've been able to return the favor.

Finally, to my parents: Jeff Michaelson, Karen Michaelson, and Karen Meek. And to so many others in the village that raised me; in particular the Still family and the Lysik family, who always treated me like one of their own. None of this would have been possible without you. Thank you.

A Multiplicative Multi-State Constraint Kalman Filter

Kristen Alyssa Michaelson, M.S.E.
The University of Texas at Austin, 2020

Supervisor: Renato Zanetti

A multi-state constraint Kalman filter (MSCKF) is implemented with a multiplicative quaternion update. The filter is tested on data from simulated inertial measurement unit (IMU) and camera measurements. In the simulation, a vehicle views feature points in the environment as it travels along a circular path. The MSCKF is demonstrated to be consistent using Monte Carlo analysis.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The Multi-State Constraint Kalman Filter (MSCKF) is an autonomous navigation algorithm that fuses inertial measurement unit (IMU) measurements with information gathered from images of the environment [23], [24]. It is a member of a broad class of *vision-aided inertial navigation systems* (VINS). VINS algorithms seek to estimate the *state* of a robot in its environment. Typical states of interest include the position, velocity, and attitude of the robot, as well as the locations of *feature points* in the environment.

As the robot moves through the environment, the filter *propagates* the state estimates forward in time using the inertial measurements. The IMU contains accelerometers and gyroscopes, which provide basic information about changes in velocity and attitude. Unfortunately, even small inaccuracies in these measurements can cause the state estimate to drift significantly from the true state in a short time. Thus, the filter must rely on external measurements to correct the state estimate. In the case of VINS, a camera onboard the robot views feature points in the environment. By tracking these features through multiple images, the filter gains information about the state and *updates* the state estimate accordingly.

Camera measurements must undergo image processing before they are available for use by the filter. A raw camera measurement contains thousands of pixels, each of little use on its own. First, a *feature extraction* algorithm selects groups of pixels— often edges or corners of objects in the image— that are especially identifiable and have a good chance of being matched in other images. Then these features are compared to features in past images. Any features that have appeared before are associated with a tag that is common between images, and any new features are given new, unique tags.

Feature identification, association and tracking is a fascinating topic, and it has surely been the subject of many reports before this one. For our purposes, this brief introduction gives an appreciation for the time it takes an onboard computer to produce a usable image "measurement;" a series of 2D coordinate pairs in the image plane with associated feature tags. The filter propagates the states forward in time using the abundant IMU measurements until a camera measurement becomes available. Then the update is performed.

Perhaps the most straightforward implementation of a VINS estimator is EKF-SLAM [4], a *simultaneous localization and mapping* algorithm that uses a simple Extended Kalman Filter update [1]. In EKF-SLAM, the state vector includes the robot states (e.g. position, velocity, and attitude) and all the positions of features observed in the environment (the map). This algorithm quickly becomes impractical as more and more features are added to the map.

The MSCKF improves on EKF-SLAM by foregoing the map in favor of copies of estimated camera *poses*, or position-quaternion pairs. A new pose

is stored every time an image is recorded. Separately, features are tracked through consecutive images. When a feature has been observed multiple times, its 3D position can be estimated using the corresponding poses. These poses form the *constraints* that give the filter its name. This 3D feature position estimate is calculated using a nonlinear least squares solver. Next, the feature position estimate is used to calculate the measurement Jacobian for an EKF update based on that feature. Both the robot states and the pose list are updated. After the update is performed, all the data associated with the feature is discarded.

The main contribution of the MSCKF is that its computational complexity is *linear* in the number of features. This allows for lightweight, real-time state estimation in a number of practical applications.

1.1 Applications and comparison studies

The authors of [24] tested the algorithm on a dataset recorded by a camera-IMU pair mounted to a vehicle during a 9-minute drive around a neighborhood in Minneapolis, MN. Front-facing cameras on cars present a particular challenge for image-based motion estimators, since the vehicle always travels along the axis of the camera boresight. The only significant addition to the MSCKF implemented in the experiment was feature outlier rejection. The MSCKF assumes features are static in the environment and the robot moves relative to them; if a feature also moves in the environment, it should not be used in the MSCKF update.

While images were only recorded at 3 Hz due to storage limitations in the experimental equipment, the data was later processed at 14 Hz on a mobile-grade processor. For comparison, the IMU data was recorded at a rate of 100 Hz. After processing, the authors were able to achieve a final position estimate roughly 10 m from the true final position of the vehicle after a 3.2 km drive. The attitude 3σ bounds were on the order of 0.1° around both axes parallel to the ground (roll and pitch). For the axis perpendicular to the ground (yaw), the error was on the order of 1°.

The MSCKF has also been proposed for spacecraft applications [8], [25]. Visual navigation is a popular choice for *entry, descent, and landing*, where a spacecraft approaches a body in space and must quickly localize itself relative to that body while enduring challenging dynamic stresses. In one test, an MSCKF ran onboard a sounding rocket using a combination of pre-computed map landmarks and new image features observed one the fly during the test [25]. The same implementation was tested in simulation using images from the moon, Mars, and Europa. The MSCKF was also recently flight tested as part of NASA's Safe and Precise Landing Integrated Capabilites Evolution (SPLICE) program [8]. In this test, which also used landmarks from satellite imagery as features, GPS data was recorded to assess the performance of the filter after the test.

Because of its low computational complexity, the MSCKF is ideal for systems with constraints on payload weight and size. These include mobile and micro aerial vehicle (MAV) applications. In [19], the MSCKF is adapted

for a camera with a rolling shutter. Most consumer-grade cameras have rolling shutters, which can cause significant distortion in an image taken while the camera is in motion. The authors of [30] derived an MSCKF measurement model for stereo vision in an MAV application. Similarly, in [9], an MSCKF is applied to a system with an RGB-D camera, which measures distance to image features in addition to their locations in the image plane. The algorithm was deployed on a quadrotor.

The MSCKF was first published in 2007. A 2018 benchmark study compared it to several more modern *visual-inertial odometry* (VIO) algorithms [5]. These included OKVIS[1] [18], ROVIO [2], VINS-Mono [26], SVO+MSF [7], [22], and SVO+GTSAM [7], [6], many of which are also EKF-based. The study only considered visual-inertial algorithms. Vision-only methods were not included. The algorithms were tested on the publicly-available EuRoC datasets, which include camera, IMU, and ground truth data recorded during a series of indoor MAV flights [3]. The MSCKF was commended for its low resource usage and consistent behavior across platforms, but other algorithms were noted to "achieve higher overall accuracy with a manageable increase in resource requirements" [5].

[1]The authors of [18] offer some particularly insightful discussion about the tradeoffs between filtering- and batch-based strategies.

1.2 Observability and MSCKF 2.0

One important topic in the field of autonomous navigation is observability. A system is said to be *observable* if any initial state can be uniquely determined given a series of measurements [27]. Observability has important implications for filter *consistency*. A consistent filter has zero-mean estimation error and its predicted covariance matches the true error covariance [1]. Since it is a statistical property, consistency can be evaluated using Monte Carlo analysis.

The unobservable directions of a system are not necessarily state values. For clarity, we begin with a discussion of the unobservable states of 2D SLAM. The 2D SLAM state vector includes the 2D position of the robot in the global frame, its angle (a single value), and the 2D positions of the features in the global frame. The robot measures 2D vectors between itself and feature points in a map. Using the nonlinear observability techniques from [13], the authors of [15] showed that the unobservable directions in 2D SLAM are the *global* translations and the *global* rotation. In other words, the robot can detect changes in position and angle *relative* to the map, but it does know where the map lies in the global frame.

The typical 3D visual-SLAM state vector includes the position, velocity, and attitude of the robot in the global frame, the positions of the features in the global frame, the biases of the accelerometers and gyroscopes in the IMU, and

the gravity vector.[2] The biases are modeled as random walks, and they have been shown to be observable from the camera measurements [17]. Explicitly estimating the IMU biases in turn improves the other state estimates, since the propagation of the inertial states depends on IMU data.

For VINS operating near the surface of the Earth, the unobservable directions are the three global translations and the global rotation about the gravity vector, or the yaw [14]. In other words, while the *relative* translations between the robot and the features are observable, the *global* locations of the robot and the features cannot be determined from camera measurements. This can be corrected with the addition of a GPS sensor or a laser tracking system; any device that records the position of the robot in the global frame. In a similar vein, the global yaw is not observable. Once again, the robot can only know its total change in yaw with respect to the static features; it cannot know the initial yaw of itself and the map. This is only true for vehicles operating in a consistent gravity field. In spacecraft applications, where the vehicle is in free fall, none of the global rotations are observable.

The other two rotations *are* observable, because the gravity vector is observable. For robots operating near the surface of the Earth, the gravity vector always points in the same global-frame direction. When the robot changes its roll or pitch, the change can be detected due to a change in the measured direction of the gravity vector. It is interesting to note that the

[2]While the MSCKF does not include feature positions in its state vector, it has the same observability properties as other VINS estimators [14].

four unobservable directions—the three global translations and the yaw—only hold for *general motion* in an environment with features that are points in 3D space. There exist *degenerate motions* which, combined with other types of features like lines and planes, can render VINS even less observable [31].

Because VINS is unobservable, extra care must be taken to ensure that the unobservable states in the estimator match the unobservable states in the real system. For EKF-based estimators, including the MSCKF, the linearization employed by the estimator causes the yaw to appear observable [15], [20], [21]. This occurs in 2D and 3D, and it results in an artificial reduction in the covariance of the yaw estimate. Three remedies have been proposed. The first, the *First Estimates Jacobian (FEJ)* EKF, uses the feature position estimate from the first observation of a feature to compute the filter Jacobians rather than later, updated estimates [15]. This idea was extended to 3D in [21] and, along with a few other improvements, deemed *MSCKF 2.0*. Another approach was introduced in [14], where the Jacobians are modified directly in order to enforce observability constraints. Recently, a third approach was suggested which takes advantage of the Lie algebra representation of three-dimensional robot poses [12].

Chapter 2

Background

2.1 Problem Formulation

A robot travels around a circular path, observing features in the environment. The robot is equipped with a camera and an IMU. The camera faces outward toward the features. The features are static. Figure 2.1 shows the trajectory and the feature points. Both the camera and the IMU are mounted at the body-frame origin, and the camera points along the body-frame z-axis. The gravity vector points downward along the global z-axis, which aligns with the body y-axis. Similar simulations were employed to test visual-inertial navigation algorithms in [12], [14], and [31].

As the robot moves along its path, data from the IMU and the camera are processed by an MSCKF. The filter maintains real-time estimates of the states and the covariance. It propagates these estimates using the IMU data. It also tracks features through successive images. Figure 2.2 shows an example simulated camera measurement. When a feature leaves the field of view, the filter updates the state and covariance estimates based on the information contained in the measurements of that feature.

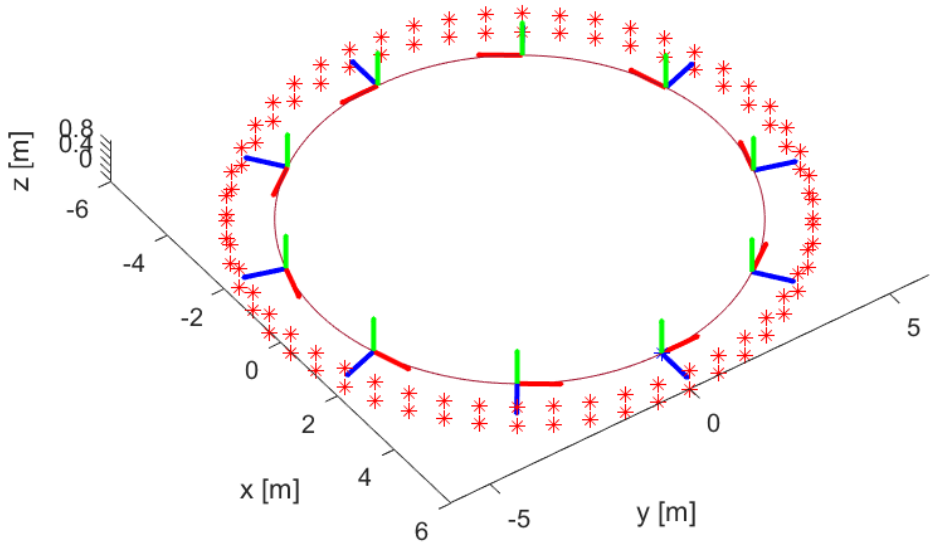

Figure 2.1: Simulated trajectory. The robot is represented by the three orthogonal body-frame axes; the x-axis (red), the y-axis (green), and the z-axis (blue). The camera points along the z-axis. The trajectory has a radius of 5 m, and feature points ($*$) are placed on the inner surface of a cylinder with a radius of 6 m.

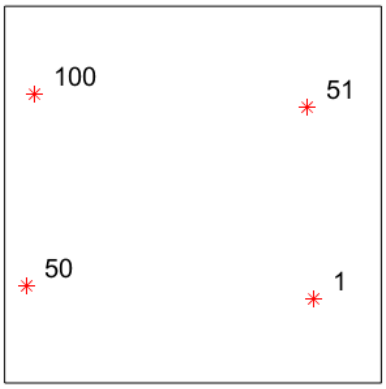

Figure 2.2: Sample image. Feature points move through the field of view as the camera passes them. In a real-world system, the feature points would be extracted from raw camera measurements, labeled, and matched with features in previous images using a data association algorithm. In the simulation, the camera knows which features it sees in each image.

2.2 State dynamics

The MSCKF state vector consists of the robot state and a list of poses. The IMU state is

$$\mathbf{X_{IMU}} = \begin{bmatrix} \mathbf{x}^T & \mathbf{v}^T & {}^B_I\mathbf{q}^T & \mathbf{b_a}^T & \mathbf{b_g}^T \end{bmatrix}^T \tag{2.1}$$

where $\mathbf{x}$ and $\mathbf{v}$ are the position and velocity of the robot in the inertial frame, ${}^B_I\mathbf{q}$ is the unit quaternion that encodes the passive rotation from the inertial frame to the body frame, and $\mathbf{b_a}$ and $\mathbf{b_g}$ are the 3×1 accelerometer and gyroscope biases. In this implementation, the IMU frame is taken as the body frame. The IMU is mounted at the body frame origin, and the IMU x-, y-, and z-axes correspond to the body frame x-, y-, and z-axes.

The full state vector is

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}^T & \mathbf{v}^T & {}^{B}_{I}\mathbf{q}^T & \mathbf{b_a}^T & \mathbf{b_g}^T & {}^{C_1}_{I}\mathbf{q}^T & {}^{C_1}\mathbf{x} & \dots & {}^{C_N}_{I}\mathbf{q}^T & {}^{C_N}\mathbf{x} \end{bmatrix}^T \tag{2.2}$$

where the vector $\begin{bmatrix} {}^{C_1}_{I}\mathbf{q}^T & {}^{C_1}\mathbf{x} & \dots & {}^{C_N}_{I}\mathbf{q}^T & {}^{C_N}\mathbf{x} \end{bmatrix}^T$ stores a copy of the robot's *pose* every time an image is recorded. A pose is a position-quaternion pair. For ease of later processing, the filter stores camera poses instead of body poses. The transformation from the body frame to the camera frame is assumed to be known. If the camera frame coincides with the body frame, then the filter simply copies pairs $\begin{bmatrix} {}^{B}_{I}\mathbf{q}^T & \mathbf{x}^T \end{bmatrix}^T$ from the IMU state. Once the state vector has attained a threshold of N poses, old poses are discarded.

The poses are static. They do not change over time. The dynamics of the IMU states are

$$\dot{\mathbf{x}}(t) = \mathbf{v}(t) \tag{2.3}$$

$$\dot{\mathbf{v}}(t) = \mathbf{a}(t) \tag{2.4}$$

$$\dot{\mathbf{q}}(t) = \frac{1}{2}\omega(t) \otimes \mathbf{q}(t) \tag{2.5}$$

$$\dot{\mathbf{b}}_{\mathbf{a}}(t) = \mathbf{n}_{\mathbf{wa}}(t) \tag{2.6}$$

$$\dot{\mathbf{b}}_{\mathbf{g}}(t) = \mathbf{n}_{\mathbf{wg}}(t) \tag{2.7}$$

where $\mathbf{a}(t)$ and $\omega(t)$ are the true inertial-frame acceleration and the true angular rate, respectively. The accelerometer and gyroscope biases are modeled as random walks, and their time evolution is given by 3×1 zero-mean white Gaussian noises $\mathbf{n_{wa}}(t)$ and $\mathbf{n_{wg}}(t)$. The covariance values of the distributions from which $\mathbf{n_{wa}}(t)$ and $\mathbf{n_{wg}}(t)$ are drawn depend on the specific IMU in use.

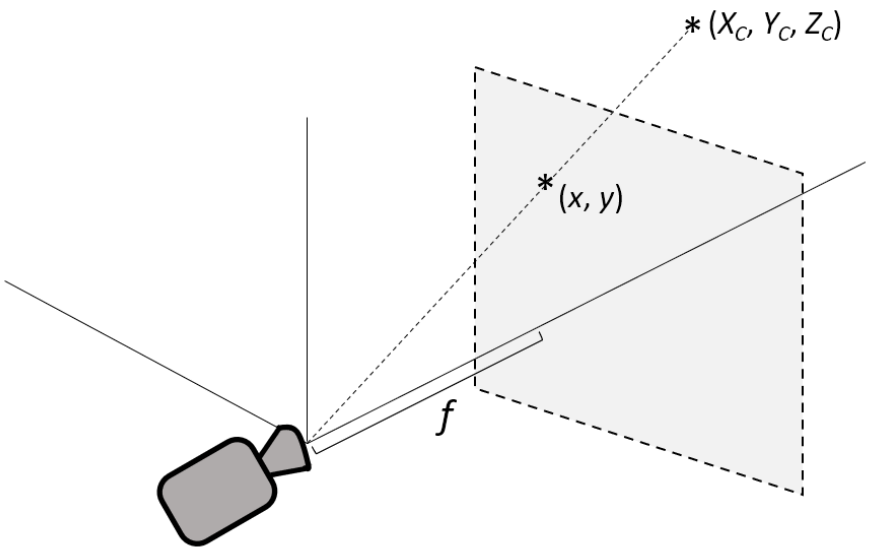

Figure 2.3: Pinhole camera geometry

2.3 Camera measurements

When a camera views a feature in the environment, the feature is projected into a 2D image plane. The camera's *focal length* is the distance from the camera to the image plane. The camera measurement (i.e. the 2D location of the feature in the image plane) is

$$h(X_C, Y_C, Z_C) = \begin{bmatrix} x & y \end{bmatrix}^T = \begin{bmatrix} fX_C/Z_C & fY_C/Z_C \end{bmatrix}^T \tag{2.8}$$

where (X_C, Y_C, Z_C) is the location of the feature in the camera frame [11]. Figure 2.3 shows this simple *pinhole camera* model.

A single image does not contain enough information to determine the 3D location of the feature point in the global frame. At least two images must be used [11]. If a feature point appears in many images with known

camera poses, its 3D location can be estimated using a nonlinear least squares solver [24].

Chapter 3

Simulation

Inertial and camera measurement data were collected during a simulated experiment where a robot executed the trajectory in Fig. 2.1. The data was fused by an MSCKF. While it is difficult to replicate real-world conditions in a simulation, errors were added to the simulated measurements to account for noises in the IMU and camera measurements and biases in the IMU measurements.

3.1 Sensor characteristics

The simulated IMU measurements are based on the Sensonor STIM300, a small device "well-suited for stabilization, guidance and navigation applications in Industrial, Aerospace and Defense markets" [28]. A STIM300 has been acquired for experimental use in our laboratory. Table 3.1 shows the power spectral density characteristics of the IMU.

The true state values $\mathbf{x}(t)$, $\mathbf{v}(t)$, and $\mathbf{q}(t)$ are calculated by numerically integrating 2.3-2.7 over time using the true values $\mathbf{a}(t)$ and $\mathbf{w}(t)$. In this implementation, the true values $\mathbf{b_a}(t)$ and $\mathbf{b_g}(t)$ are constant.

	Symbol	Units in Datasheet [28]	Units in simulation
Accelerometer noise	Q_a	$0.07\ m/s/\sqrt{hr}$	$1.4 \times 10^{-6}\ m^2/s^2/s$
Accelerometer bias	Q_{wa}	$0.05\ mg$	$4.9 \times 10^{-4}\ m/s^2$
Gyro noise	Q_g	$0.15\ deg/\sqrt{hr}$	$1.9 \times 10^{-9}\ rad^2/s$
Gyro bias	Q_{wg}	$0.3\ deg/hr$	$1.5 \times 10^{-6}\ rad/s$

Table 3.1: IMU power spectral density values

The IMU measurement at time t is

$$\mathbf{a_m}(t) = T_I^B(\mathbf{a}(t) - \mathbf{g}) + \mathbf{n_a}(t) + \mathbf{b_a}(t) \tag{3.1}$$

$$\omega_\mathbf{m}(t) = \omega(t) + \mathbf{n_g}(t) + \mathbf{b_g}(t) \tag{3.2}$$

where

$$\mathbf{n_a}(t) \sim \mathcal{N}(\mathbf{0}_{3\times1}, (Q_a/\Delta t)\mathbf{I}_{3\times3}) \tag{3.3}$$

$$\mathbf{n_g}(t) \sim \mathcal{N}(\mathbf{0}_{3\times1}, (Q_g/\Delta t)\mathbf{I}_{3\times3}) \tag{3.4}$$

and $\mathbf{g}$ is the gravity vector, which points downward along the inertial-frame z-axis. The time step Δt is the rate at which the IMU reports data. In this implementation, $\Delta t = 0.01$ s. First, the gravity vector is subtracted from the true inertial-frame acceleration. The gravity vector is *subtracted* from the true acceleration $\mathbf{a}(t)$ since $-\mathbf{g}$ is the robot's *reaction* to gravity, which keeps it at a constant height. Then, that difference is expressed in the body frame using direction cosine matrix T_I^B. T_I^B is computed from the true quaternion $\mathbf{q}(t)$. Next the measurement is corrupted by a random noise $\mathbf{n_g}(t)$ and bias $\mathbf{b_a}(t)$. A similar procedure is used to compute the gyroscope measurement $\omega_\mathbf{m}(t)$.

The camera measurement is based on the pinhole camera model in Fig. 2.3. For simplicity, the focal length of the camera is $f = 1$. The camera

measurement is

$$\mathbf{c_m} = \begin{bmatrix} X_C/Z_C & Y_C/Z_C \end{bmatrix}^T + \mathbf{n}_{cam} \tag{3.5}$$

where $\mathbf{n}_{cam}$ is a 2×1 vector of zero-mean Gaussian-distributed random noise with standard deviation $\sigma_{cam} = 0.01$. The camera has a 90° field of view in both dimensions, which produces a square image (see Fig. 2.2).

3.2 Propagation

After each IMU measurement is received, the filter propagates the state estimate $\hat{\mathbf{X}}(t)$ and the covariance estimate $\mathbf{P}(t)$ forward in time by one step Δt. The time domain is divided into discrete steps $t_{k+1} = t_k + \Delta t$. The poses are static and do not change in time. The IMU states are propagated using time-linearizations of 2.3-2.7.

$$\bar{\mathbf{x}}(t_{k+1}) = \hat{\mathbf{x}}(t_k) + \hat{\mathbf{v}}(t_k)\Delta t + \frac{1}{2}\hat{\mathbf{a}}(t_k)\Delta t^2 \tag{3.6}$$

$$\bar{\mathbf{v}}(t_{k+1}) = \hat{\mathbf{v}}(t_k) + \hat{\mathbf{a}}(t_k)\Delta t \tag{3.7}$$

$$\bar{\mathbf{q}}(t_{k+1}) = \begin{bmatrix} \cos(\frac{1}{2}||\boldsymbol{\Delta}\hat{\theta}(t_k)||) \\ \frac{\boldsymbol{\Delta}\hat{\theta}(t_k)}{||\boldsymbol{\Delta}\hat{\theta}(t_k)||}\sin(\frac{1}{2}||\boldsymbol{\Delta}\hat{\theta}(t_k)||) \end{bmatrix} \otimes \hat{\mathbf{q}}(t_k) \tag{3.8}$$

$$\bar{\mathbf{b}}_\mathbf{a}(t_{k+1}) = \hat{\mathbf{b}}_\mathbf{a}(t_k) \tag{3.9}$$

$$\bar{\mathbf{b}}_\mathbf{g}(t_{k+1}) = \hat{\mathbf{b}}_\mathbf{g}(t_k) \tag{3.10}$$

The vector $\boldsymbol{\Delta}\hat{\theta}(t_k)$ is a vector of angles and $\hat{\mathbf{a}}(t_k)$ is a vector of acceleration values calculated from the IMU measurement.

$$\hat{\mathbf{a}}(t_k) = \hat{T}_B^I(\mathbf{a_m}(t_k) - \hat{\mathbf{b}}_\mathbf{a}(t_k)) + \mathbf{g} \tag{3.11}$$

$$\boldsymbol{\Delta}\hat{\theta} = (\omega_{\mathbf{m}}(t_k) - \hat{\mathbf{b}}_{\mathbf{g}}(t_k))\Delta t \tag{3.12}$$

The direction cosine matrix $\hat{T}_B^I$ is the transpose of $\hat{T}_I^B$, which is calculated from the estimated quaternion $\hat{\mathbf{q}}(t_k)$.

The covariance matrix $\mathbf{P}(t)$ is calculated using the linearized error dynamics. The time evolution of the estimation error is

$$\dot{\tilde{\mathbf{X}}}_{\mathbf{IMU}} = \mathbf{F}\tilde{\mathbf{X}}_{\mathbf{IMU}} + \mathbf{B}\mathbf{n}_{\mathbf{IMU}} \tag{3.13}$$

where $\tilde{\mathbf{X}}_{\mathbf{IMU}}$ is the difference between the estimated IMU state $\hat{\mathbf{X}}_{\mathbf{IMU}}$ and the true IMU state $\mathbf{X}_{\mathbf{IMU}}$ [24]. The vector $\mathbf{n}_{\mathbf{IMU}}$ is $\begin{bmatrix}\mathbf{n_a}^T & \mathbf{n_g}^T & \mathbf{n_{wa}}^T & \mathbf{n_{wg}}^T\end{bmatrix}^T$, a vector of all the IMU noises. The matrices $\mathbf{F}$ and $\mathbf{B}$ are

$$\mathbf{F} = \begin{bmatrix} \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & -\hat{T}_B^I[(\mathbf{a_m} - \hat{\mathbf{b}}_{\mathbf{a}})\times] & -\hat{T}_B^I & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & -[(\omega_{\mathbf{m}} - \hat{\mathbf{b}}_{\mathbf{g}})\times] & \mathbf{0}_{3\times3} & -\mathbf{I}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \end{bmatrix} \tag{3.14}$$

$$\mathbf{B} = \begin{bmatrix} \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ -\hat{T}_B^I & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & -\mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} \end{bmatrix} \tag{3.15}$$

The state transition matrix from time t_k to time t_{k+1} is

$$\boldsymbol{\Phi}(t_{k+1}, t_k) = e^{\mathbf{F}\Delta t} \tag{3.16}$$

and the discrete-time $\mathbf{B}$ matrix is

$$\mathbf{B}_d = \begin{bmatrix} -\frac{1}{2}\hat{T}_B^I\Delta t & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ -\hat{T}_B^I & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & -\mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} \end{bmatrix} \tag{3.17}$$

The covariance matrix can be divided into four regions. At the top left, $\mathbf{P}_{II}(t_k)$ is the covariance of the IMU states. At the bottom right, $\mathbf{P}_{CC}(t_k)$ is the covariance of the poses. The off-diagonal terms $\mathbf{P}_{IC}(t_k)$ are the correlations between the IMU states and the poses.

The propagated covariance matrix is

$$\bar{\mathbf{P}}(t_{k+1}) = \begin{bmatrix} \mathbf{\Phi}(t_{k+1}, t_k)\mathbf{P}(t_k)\mathbf{\Phi}(t_{k+1}, t_k)^T + \mathbf{B}_d Q \mathbf{B}_d^T & \mathbf{\Phi}(t_{k+1}, t_k)\mathbf{P}_{IC}(t_k) \\ \mathbf{P}_{IC}(t_k)^T \mathbf{\Phi}(t_{k+1}, t_k)^T & \mathbf{P}_{CC}(t_k) \end{bmatrix} \tag{3.18}$$

where $\mathbf{Q}$ is a diagonal matrix of the IMU noise power spectral densities:

$$\mathbf{Q} = \begin{bmatrix} Q_a \Delta t \mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & Q_g \Delta t \mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \end{bmatrix} \tag{3.19}$$

where zeroes replace the Q_{wa} and Q_{wg} terms since the biases are static. The IMU covariance increases in the propagation step due to the IMU noises. The camera pose covariance stays the same. After the propagation, the covariance is adjusted to enforce symmetry.

$$\bar{\mathbf{P}}(t_{k+1}) \leftarrow \frac{1}{2}(\bar{\mathbf{P}}(t_{k+1}) + \bar{\mathbf{P}}(t_{k+1})^T) \tag{3.20}$$

3.3 Measurement Update

The filter propagates the IMU states forward in time until a camera measurement is available. If a feature point has left the field of view, the filter performs a Kalman update [16]. It also stores a copy of the current pose and augments the covariance matrix accordingly.

3.3.1 Prediction of feature point location

In the MSCKF, feature points are tracked through series of images. Each time a feature appears in an image, it is associated with a 2×1 measurement $\mathbf{c_m}(t_k)$ and a camera pose $\begin{bmatrix} {}^{C}_{I}\hat{\mathbf{q}}(t_k)^T & \hat{\mathbf{x}}_C(t_k)^T \end{bmatrix}^T$. When a feature leaves the camera's field of view, the filter performs a measurement update.[1] The first task in the measurement update is to estimate the inertial-frame location of the feature $\mathbf{p_f}$ using the measurement-pose pairs.

The feature location estimate $\hat{\mathbf{p}}_\mathbf{f}$ is the solution to the nonlinear least-squares optimization problem

$$\hat{\mathbf{p}}_\mathbf{f} = \min_{\mathbf{p_f}} \sum_{i=1}^{N} (\mathbf{c_m}^i - h({}^{C_i}_{I}\hat{\mathbf{q}}, \hat{\mathbf{x}}_{C_i}, \mathbf{p_f}))^T (\mathbf{c_m}^i - h({}^{C_i}_{I}\hat{\mathbf{q}}, \hat{\mathbf{x}}_{C_i}, \mathbf{p_f})) \tag{3.21}$$

In 3.21, there are N measurements $\mathbf{c_m}^i$ of the feature $\mathbf{p_f}$. The location of $\mathbf{p_f}$ in camera frame C_i is

$${}^{C_i}\mathbf{p_f} = \begin{bmatrix} X_{C_i} \\ Y_{C_i} \\ Z_{C_i} \end{bmatrix} = T_I^{C_i}(\mathbf{p_f} - \hat{\mathbf{x}}_{C_i}) \tag{3.22}$$

and the function h is

$$h({}^{C_i}_{I}\hat{\mathbf{q}}, \hat{\mathbf{x}}_{C_i}, \mathbf{p_f}) = \begin{bmatrix} X_{C_i}/Z_{C_i} \\ Y_{C_i}/Z_{C_i} \end{bmatrix} \tag{3.23}$$

This problem can be solved using a nonlinear least squares solver. A detailed explanation of the method implemented for this report is given in Ref. [24].

[1] Ref. [24] includes an additional condition that triggers an update. When the maximum number of allowable poses in the state vector is reached, a few poses are removed together. These poses are evenly-spaced over time, and all features common to them are processed at the time they are removed. In this implementation, only a single pose—the oldest pose—is removed after each camera measurement.

3.3.2 Kalman update

The Kalman filter adjusts the states by predicting the measurement with the propagated state values. If the predicted measurement is close to the measurement $\mathbf{c_m}$, then the propagated state values are close to the truth. The update is minor. On the other hand, if the predicted measurement is very different from $\mathbf{c_m}$, the state values may change significantly during the update.

When a feature point that has been tracked through multiple frames leaves the field of view, its location in 3D space is estimated using 3.21. Then the estimate $\hat{\mathbf{p}}_\mathbf{f}$ is used with the list of poses to calculate a series of predicted measurements $h({}^{C_i}_{I}\hat{\mathbf{q}}, \hat{\mathbf{x}}_{C_i}, \hat{\mathbf{p}}_\mathbf{f})$. The difference between the measurements and the predicted measurements is the measurement residual.

$$\mathbf{r} = \begin{bmatrix} \mathbf{c_m}^1 \\ \mathbf{c_m}^2 \\ \vdots \\ \mathbf{c_m}^N \end{bmatrix} - \begin{bmatrix} h({}^{C_1}_{I}\hat{\mathbf{q}}, \hat{\mathbf{x}}_{C_1}, \hat{\mathbf{p}}_\mathbf{f}) \\ h({}^{C_2}_{I}\hat{\mathbf{q}}, \hat{\mathbf{x}}_{C_2}, \hat{\mathbf{p}}_\mathbf{f}) \\ \vdots \\ h({}^{C_N}_{I}\hat{\mathbf{q}}, \hat{\mathbf{x}}_{C_N}, \hat{\mathbf{p}}_\mathbf{f}) \end{bmatrix} \tag{3.24}$$

In 3.24, all the measurements of the feature point are stacked together. This way, only one update is performed per feature.

The measurement $h({}^{C_i}_{I}\hat{\mathbf{q}}, \hat{\mathbf{x}}_{C_i}, \hat{\mathbf{p}}_\mathbf{f})$ is related to the state values and the estimated feature position. Therefore, we must calculate two measurement Jacobians for the Kalman update [23], [24]:

$$\mathbf{H_X} = \begin{bmatrix} \mathbf{0}_{2\times15} & \mathbf{J_1}[{}^{C_1}\hat{\mathbf{p}}_\mathbf{f}\times] & -\mathbf{J_1}\hat{T}_I^{C_1} & \mathbf{0}_{2\times3} & \mathbf{0}_{2\times3} & \cdots & \mathbf{0}_{2\times3} & \mathbf{0}_{2\times3} \\ \mathbf{0}_{2\times15} & \mathbf{0}_{2\times3} & \mathbf{0}_{2\times3} & \mathbf{J_2}[{}^{C_2}\hat{\mathbf{p}}_\mathbf{f}\times] & -\mathbf{J_2}\hat{T}_I^{C_2} & \cdots & \mathbf{0}_{2\times3} & \mathbf{0}_{2\times3} \\ \vdots & & & & & \ddots & & \vdots \\ \mathbf{0}_{2\times15} & \mathbf{0}_{2\times3} & \mathbf{0}_{2\times3} & \mathbf{0}_{2\times3} & \mathbf{0}_{2\times3} & \cdots & \mathbf{J_N}[{}^{C_N}\hat{\mathbf{p}}_\mathbf{f}\times] & -\mathbf{J_N}\hat{T}_I^{C_N} \end{bmatrix} \tag{3.25}$$

$$\mathbf{H_f} = \begin{bmatrix} \mathbf{J_1}\hat{T}_I^{C_1} \\ \mathbf{J_2}\hat{T}_I^{C_2} \\ \vdots \\ \mathbf{J_N}\hat{T}_I^{C_N} \end{bmatrix} \tag{3.26}$$

where

$$\mathbf{J_i} = \nabla_{C_i \hat{\mathbf{p}}_\mathbf{f}} \mathbf{c_m}^i = \frac{1}{\hat{Z}_{C_i}} \begin{bmatrix} 1 & 0 & -\hat{X}_{C_i}/\hat{Z}_{C_i} \\ 0 & 1 & -\hat{Y}_{C_i}/\hat{Z}_{C_i} \end{bmatrix} \tag{3.27}$$

Note that in 3.25, the first fifteen columns of $\mathbf{H_X}$ are all zeroes. These correspond to the IMU states, which are not related to the camera measurements currently being processed. By definition of the event that triggers the update, the feature point $\mathbf{p_f}$ does not appear in the current camera frame. Therefore, it is only related to past poses.

The measurement residual is a function of the state error, the error in the feature location estimate, and the measurement noise [23]:

$$\mathbf{r} \simeq \mathbf{H_X}\tilde{\mathbf{X}} + \mathbf{H_f}\tilde{\mathbf{p}}_\mathbf{f} + \mathbf{n} \tag{3.28}$$

where $\mathbf{n}$ is a vector of length $2N$ with covariance matrix $\mathbf{R} = \sigma_{cam}^2 \mathbf{I}_{2N \times 2N}$. In 3.28, the feature error $\mathbf{H_f}\tilde{\mathbf{p}}_\mathbf{f}$ is correlated to the state error $\mathbf{H_X}\tilde{\mathbf{X}}$, since state values are used to estimate the feature position.

In order to rid ourselves of $\mathbf{H_f}$, we employ the QR factorization $\mathbf{H_f} = QR$. The matrix Q is a $2N \times 2N$ unitary matrix, and R is a $2N \times 3$ upper-triangular matrix [10]. Since R is tall *and* upper-triangular, it is mostly zeroes. Therefore, if we apply Q^T to the terms in 3.28, we can form a residual that is

only dependent on the state error:

$$Q^T\mathbf{r} \simeq Q^T\mathbf{H_X}\tilde{\mathbf{X}} + Q^T\mathbf{H_f}\tilde{\mathbf{p}}_\mathbf{f} + Q^T\mathbf{n} \tag{3.29}$$

$$= Q^T\mathbf{H_X}\tilde{\mathbf{X}} + Q^TQR\tilde{\mathbf{p}}_\mathbf{f} + Q^T\mathbf{n} \tag{3.30}$$

$$= Q^T\mathbf{H_X}\tilde{\mathbf{X}} + R\tilde{\mathbf{p}}_\mathbf{f} + Q^T\mathbf{n} \tag{3.31}$$

If we choose $\mathbf{r}_0$ as the last $2N-3$ rows of $Q^T\mathbf{r}$ and $\mathbf{H_0}$ as the last $2N-3$ rows of $Q^T\mathbf{H_X}$, then:

$$\mathbf{r_0} = \mathbf{H_0}\tilde{\mathbf{X}} + \mathbf{n_0} \tag{3.32}$$

where the covariance matrix of $\mathbf{n_0}$ is $\mathbf{R_0} = \sigma^2_{cam}\mathbf{I}_{(2N-3)\times(2N-3)}$ [24].

The Kalman gain is:

$$\mathbf{K} = \bar{\mathbf{P}}\mathbf{H_0^T}(\mathbf{H_0}\bar{\mathbf{P}}\mathbf{H_0^T} + \mathbf{R_0})^{-1} \tag{3.33}$$

and the state correction is:

$$\mathbf{\Delta X} = \mathbf{K}\mathbf{r_0} \tag{3.34}$$

which includes corrections to the pose list $\begin{bmatrix} {}^{C_1}_{I}\mathbf{q}^T & {}^{C_1}\mathbf{x} & \dots & {}^{C_N}_{I}\mathbf{q}^T & {}^{C_N}\mathbf{x} \end{bmatrix}^T$.

The propagated IMU states 3.6-3.10 are updated as:

$$\hat{\mathbf{x}}(t_{k+1}) = \bar{\mathbf{x}}(t_{k+1}) + \mathbf{\Delta x} \tag{3.35}$$

$$\hat{\mathbf{v}}(t_{k+1}) = \bar{\mathbf{v}}(t_{k+1}) + \mathbf{\Delta v} \tag{3.36}$$

$$\hat{\mathbf{q}}(t_{k+1}) = \begin{bmatrix} \cos(\frac{1}{2}||\mathbf{\Delta}\theta||) \\ \frac{\mathbf{\Delta}\theta}{||\mathbf{\Delta}\theta||}\sin(\frac{1}{2}||\mathbf{\Delta}\theta||) \end{bmatrix} \otimes \bar{\mathbf{q}}(t_{k+1}) \tag{3.37}$$

$$\hat{\mathbf{b}}_\mathbf{a}(t_{k+1}) = \bar{\mathbf{b}}_\mathbf{a}(t_{k+1}) + \mathbf{\Delta b_a} \tag{3.38}$$

$$\hat{\mathbf{b}}_\mathbf{g}(t_{k+1}) = \bar{\mathbf{b}}_\mathbf{g}(t_{k+1}) + \mathbf{\Delta b_g} \tag{3.39}$$

where a multiplicative update is used for the quaternion in 3.37. Here, $\mathbf{\Delta}\theta$ comes from the vector $\mathbf{\Delta X}$, not an IMU measurement as in 3.8. The poses are updated using:

$$ {}^{C_i}_{I}\mathbf{q}(t_{k+1}) = \begin{bmatrix} \cos(\frac{1}{2}||\mathbf{\Delta}\theta||) \\ \frac{\mathbf{\Delta}\theta}{||\mathbf{\Delta}\theta||}\sin(\frac{1}{2}||\mathbf{\Delta}\theta||) \end{bmatrix} \otimes {}^{C_i}_{I}\mathbf{q}(t_k) \tag{3.40} $$

$$ \mathbf{x}_{C_i}(t_{k+1}) = \mathbf{x}_{C_i}(t_k) + \mathbf{\Delta x}_{C_i} \tag{3.41} $$

The covariance is updated using the Joseph form [1]:

$$ \mathbf{P}(t_{k+1}) = (\mathbf{I} - \mathbf{KH_0})\bar{\mathbf{P}}(t_k)(\mathbf{I} - \mathbf{KH_0})^T + \mathbf{KR_0K}^T \tag{3.42} $$

and symmetry is enforced using

$$ \mathbf{P}(t_{k+1}) \leftarrow \frac{1}{2}(\mathbf{P}(t_{k+1}) + \mathbf{P}(t_{k+1})^T) \tag{3.43} $$

If more than one feature leaves the field of view at once, multiple updates may be performed in the same step by stacking the vectors $\mathbf{H_0}$ and $\mathbf{r_0}$ corresponding to each feature [24].[2] After the update, the new state and covariance values are propagated forward in time until the next camera measurement is available.

3.3.3 Storing a new camera pose

Every time an image is taken, a copy of the current camera pose is appended to the end of the state vector. First, the camera pose is calculated

[2]If *many* features leave the field of view at the same time, the stack of $\mathbf{H_0}$ matrices may become very large. An improvement in computational complexity can be achieved using a *second* QR decomposition. The procedure is outlined in Ref. [24].

from the state estimate using known information about the transformation between the camera frame and the body frame.

$$ {}^{C}_{I}\mathbf{q} = {}^{C}_{B}\mathbf{q} \otimes \hat{\mathbf{q}}(t_{k+1}) \tag{3.44} $$

$$ \mathbf{x}_C = \hat{\mathbf{x}}(t_{k+1}) + \hat{T}^{I}_{B}{}^{B}\mathbf{x}_C \tag{3.45} $$

In 3.44-3.45, ${}^{C}_{B}\mathbf{q}$ is the known rotation from the body frame to the camera frame, and ${}^{B}\mathbf{x}_C$ is the position of the camera in the body frame.

Next, the covariance matrix must be augmented with the covariance of the new pose $\begin{bmatrix} {}^{C}_{I}\mathbf{q} & \mathbf{x}_C \end{bmatrix}$. The augmented covariance matrix is [23]:

$$ \mathbf{P}(t_{k+1}) \leftarrow \begin{bmatrix} \mathbf{I}_{6M+15} \\ \mathbf{J} \end{bmatrix} \mathbf{P}(t_{k+1}) \begin{bmatrix} \mathbf{I}_{6M+15} \\ \mathbf{J} \end{bmatrix}^{T} \tag{3.46} $$

where there were M poses in the state vector before the addition of the new one, and

$$ \mathbf{J} = \begin{bmatrix} \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \hat{T}^{I}_{B} & \mathbf{0}_{3\times6} & \mathbf{0}_{3\times6M} \\ \mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} & [\hat{T}^{I}_{B}{}^{B}\mathbf{x}_C\times] & \mathbf{0}_{3\times6} & \mathbf{0}_{3\times6M} \end{bmatrix} \tag{3.47} $$

Because $\mathbf{J}$ is not full rank, a diagonal matrix with very small values may be added to the augmented covariance in 3.46 to enforce positive-definiteness.

Finally, if the number of poses in the state vector is too large, the oldest pose is removed. The corresponding rows and columns of $\mathbf{P}(t_{k+1})$ are also removed. In this implementation, the filter maintains a sliding window of 20 poses.

Chapter 4

Results

The filter is initialized with state values close to the truth. Then, it relies on camera and IMU data to estimate the trajectory. Fig. 4.1 shows a sample estimated trajectory. The path is smooth between updates, when the filter is propagating the states using the high-frequency IMU data. Then, when feature leaves the field of view, the Kalman update places the robot closer to the true path. The IMU data is received at a rate of 100 Hz, and the camera data is received at 5 Hz.

Figs. 4.2-4.6 show the estimation error and predicted standard deviation of all the states over time. The estimation error is the difference between the true states and the estimated states. For all the states except attitude, the true value is simply subtracted from the estimated value. The attitude error $\alpha = \begin{bmatrix}\phi & \theta & \psi\end{bmatrix}^T$ is calculated using

$$\boldsymbol{\Delta q} = {}^{B}_{I}\mathbf{q}_{true} \otimes {}^{B}_{I}\hat{\mathbf{q}}^{-1} \tag{4.1}$$

$$\alpha = 2\boldsymbol{\Delta q}_v / \boldsymbol{\Delta q}_s \tag{4.2}$$

where $\boldsymbol{\Delta q}_v$ is the vector part of the quaternion $\boldsymbol{\Delta q}$, and $\boldsymbol{\Delta q}_s$ is the scalar part.

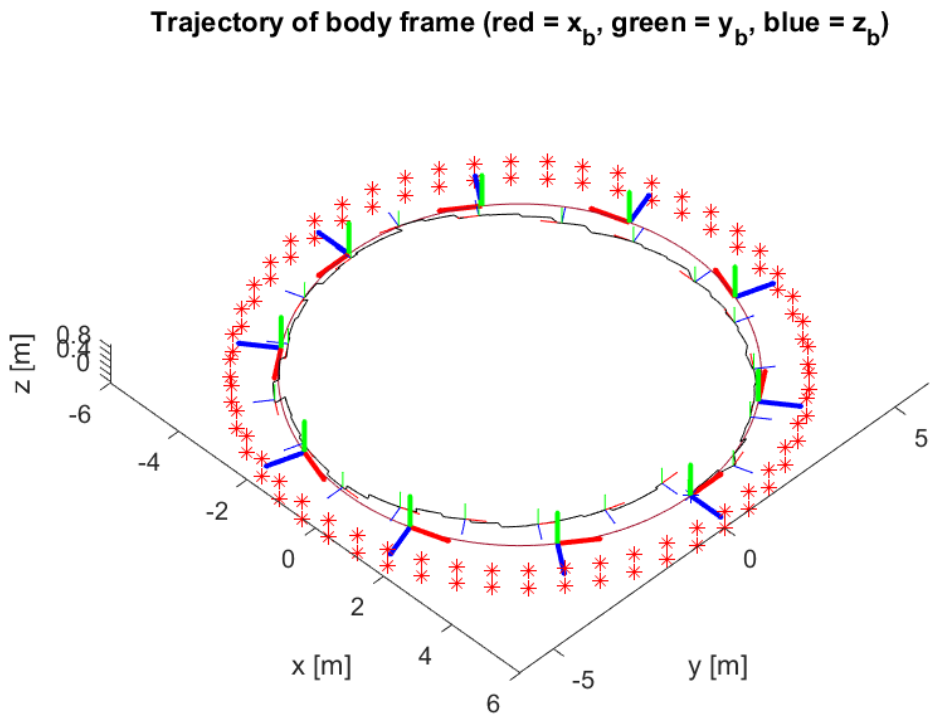

Figure 4.1: A sample estimated trajectory. The red line shows true trajectory. The true attitude is shown with large, bold axes. The estimated trajectory and attitude are shown with the black line and the smaller, lighter axes respectively.

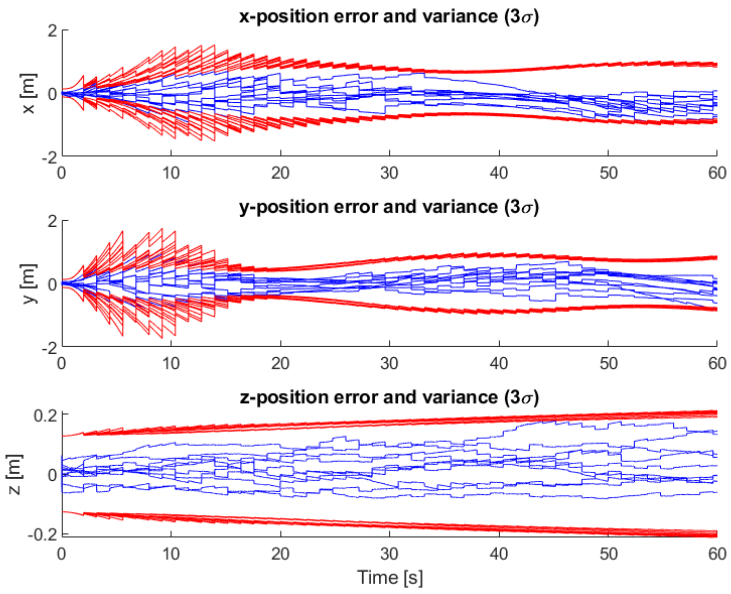

Figure 4.2: Position error Monte Carlo plot

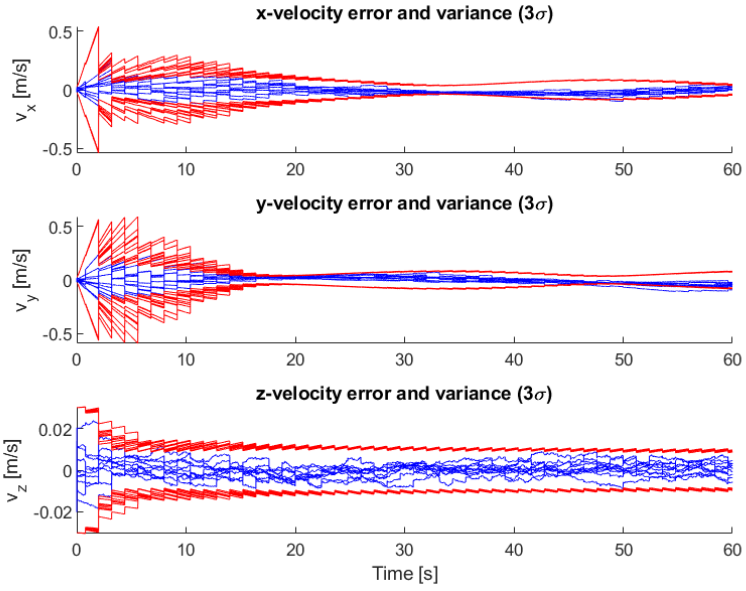

Figure 4.3: Velocity error Monte Carlo plot

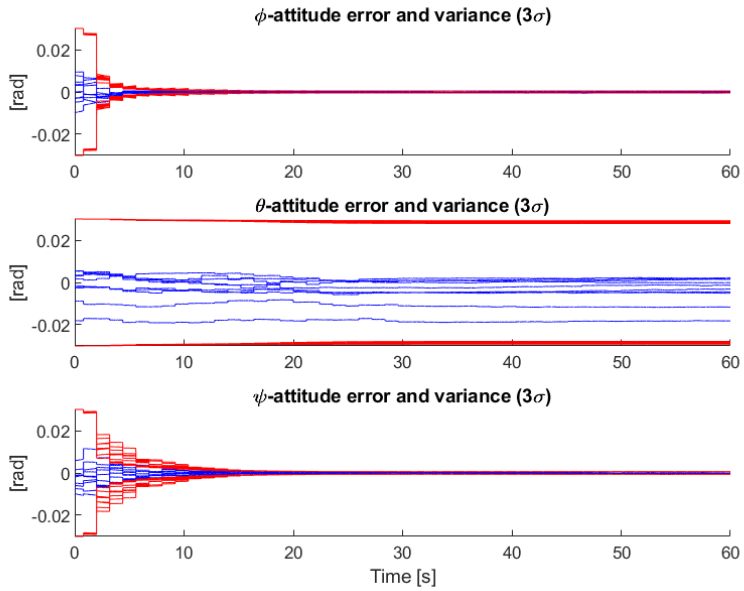

Figure 4.4: Attitude error Monte Carlo plot

In Fig. 4.4, the yaw errors (θ) remain relatively constant over the course of the run. The covariance bounds stay wide. This is a nice visual demonstration of the fact that global yaw is unobservable in camera-IMU systems. The yaw is initialized with a small error, and the filter does not learn any more information about it as the robot moves along the path.

Figs. 4.2-4.6 show that the filter is consistent, since the error values (blue) remain within three standard deviations (red).

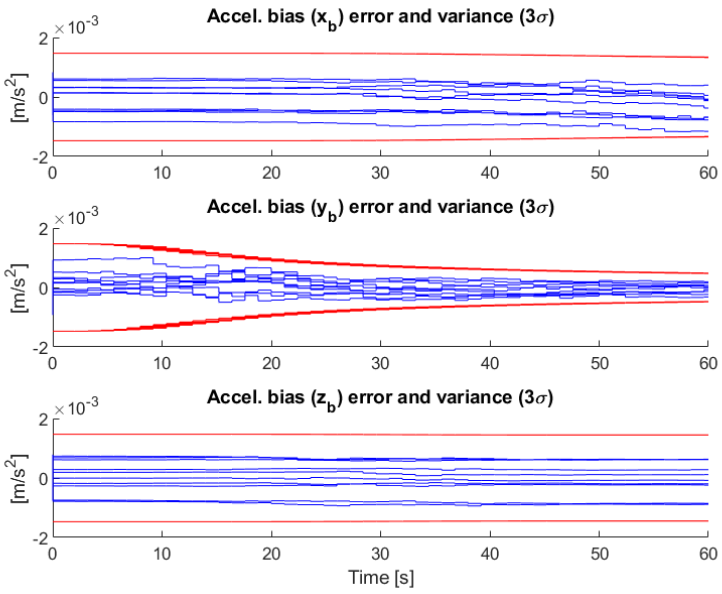

Figure 4.5: Accelerometer bias error Monte Carlo plot

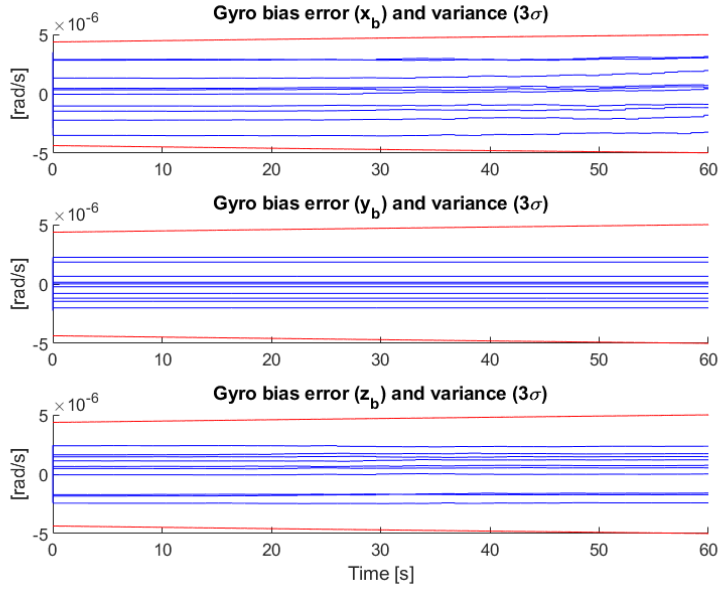

Figure 4.6: Gyro bias error Monte Carlo plot

Chapter 5

Conclusion

This report presents an implementation of the MSCKF. The performance of the filter is evaluated in simulation, and it is shown to be consistent using Monte Carlo analysis. Further, Fig. 4.4 shows that no information is gained about the global yaw as the robot moves along the path.

In this implementation, the attitude quaternions are adjusted using a *multiplicative* update. While we are not the first to implement a multiplicative quaternion update in the MSCKF (see, e.g. [5], [29]), we are curious about the observability implications of doing so. In future work, we hope to derive a mathematical foundation for the observability properties of the multiplicative MSCKF.

Bibliography

[1] Yaakov Bar-Shalom, X. Rong Li, and Thiagalingam Kirubarajan. *Estimation with Applications to Tracking and Navigation: Theory Algorithms and Software*. John Wiley and Sons, Inc, New York, 2001.

[2] Michael Bloesch, Sammy Omari, Marco Hutter, and Roland Siegwart. Robust visual inertial odometry using a direct ekf-based approach. In *International Conference on Intelligent Robots and Systems (IROS)*, 2015.

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

[4] P Cheeseman, R Smith, and M Self. A stochastic map for uncertain spatial relationships. In *4th International Symposium on Robotic Research*, pages 467–474. MIT Press Cambridge, 1987.

[5] Jeffrey Delmerico and Davide Scaramuzza. A benchmark comparison of monocular visual-inertial odometry algorithms for flying robots. In *IEEE Conference on Robotics and Automation (ICRA)*, May 2018.

[6] Christian Forster, Luca Carlone, Frank Dellaert, and Davide Scaramuzza. On-manifold preintegration for real-time visual–inertial odometry. *IEEE*

Transactions on Robotics, 33(1):1–21, 2016.

[7] Christian Forster, Zichao Zhang, Michael Gassner, Manuel Werlberger, and Davide Scaramuzza. Svo: Semidirect visual odometry for monocular and multicamera systems. *IEEE Transactions on Robotics*, 33(2), 2016.

[8] Matthew P. Fritz, Andrew S. Olguin, Kyle W. Smith, Ronney Lovelace, Ronald Sostaric, Samual Pedrotty, Jay Estes, Teming Tse, and Reuben Garcia. Operational constraint analysis of terrain relative navigation for landing applications. In *AIAA SciTech Forum*, January 2020.

[9] Marissa N. Galfond. Visual-inertial odometry with depth sensing using a multi-state constraint kalman filter. Master's thesis, MIT, June 2014.

[10] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, 2013.

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

[12] Sejong Heo and Chan Gook Park. Consistent ekf-based visual-inertial odometry on matrix lie group. *IEEE Sensors Journal*, 18, May 2018.

[13] R. Hermann and A. Krener. Nonlinear controllability and observability. *IEEE Transactions on Automatic Control*, 22:728–740, 1977.

[14] Joel A. Hesch, Dimitrios G. Kottas, Sean L. Bowman, and Stergios I. Roumeliotis. Camera-imu-based localization: Observability analysis and

consistency improvement. *The International Journal of Robotics Research*, 33(1):182–201, 2014.

[15] Guoquan P. Huang, Anastasios I. Mourikis, and Stergios I. Roumeliotis. Analysis an dimprobement of the consistency of extended kalman filter based slam. In *IEEE International Conference on Robotics and Automation*, May 2008.

[16] R. E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82:35–45, 1960.

[17] Jonathan Kelly and Gaurav S. Sukhatme. Visual-inertial sensor fusion: Localization, mapping, and sensor-to-sensor self-calibration. *International Journal of Robotics Research*, 1:56–79, 2011.

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

[19] Mingyang Li, Byung Hyung Kim, and Anastasios Mourikis. Real-time motion tracking on a cellphone using inertial sensing and a rolling-shutter camera. In *IEEE International Conference on Robotics and Automation (ICRA)*, May 2013.

[20] Mingyang Li and Anastasios Mourikis. Improving the accuracy of ekf-based visual-inertial odometry. In *IEEE International Conference on*

Robotics and Automation (ICRA), May 2012.

[21] Mingyang Li and Anastasios I. Mourikis. High-precision, consistent ekf-based visual-inertial odometry. *International Journal of Robotics Research*, 32(6):690–711, 2012.

[22] Simon Lynen, Markus W Achtelik, Stephan Weiss, Margarita Chli, and Roland Siegwart. A robust and modular multi-sensor fusion approach applied to mav navigation. In *2013 IEEE/RSJ international conference on intelligent robots and systems*, pages 3923–3929. IEEE, 2013.

[23] Anastasios I. Mourikis and Stergios Roumeliotis. A multi-state constraint kalman filter for vision-aided inertial navigation, September 2006.

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

[25] Anastasios I. Mourikis, Nikolas Trawny, Stergios I. Roumeliotis, Andrew E. Johnson, Adnan Ansar, and Larry Matthies. Vision-aided inertial navigation for entry, descent, and landing. *IEEE Transactions on Robotics*, 25(2):264–280, April 2009.

[26] Tong Qin, Peiliang Li, and Shaojie Shen. Vins-mono: A robust and versatile monocular visual-inertial state estimator. *IEEE Transactions on Robotics*, 34(4), 2018.

[27] Wilson J. Rugh. *Linear System Theory (2nd Ed.)*. Prentice-Hall, Inc., USA, 1996.

[28] Sensonor AS, Horten, Norway. *Ultra-High Performance Inertial Measurement Unit (IMU): STIM300*.

[29] K. Sun, K. Mohta, B. Pfrommer, M. Watterson, S. Liu, Y. Mulgaonkar, C. J. Taylor, and V. Kumar. Robust stereo visual inertial odometry for fast autonomous flight. *IEEE Robotics and Automation Letters*, 3(2):965–972, 2018.

[30] Ke Sun, Kartik Mohta, Bernd Pfrommer, Michael Watterson, Sikang Liu, Yash Mulgaonkar, Camillo J. Taylor, and Vijay Kumar. Robust stereo visual inertial odometry for fast autonomous flight. In *IEEE Robotics and Automation Letters*, volume 3 of *2*, April 2018.

[31] Yulin Yang and Guoquan Huang. Observability analysis of aided ins with heterogeneous features of points, lines, and planes. *IEEE Transactions on Robotics*, 35(6), December 2019.