MULTI-CAMERA VISUAL-INERTIAL ODOMETRY: ALGORITHM DESIGN AND PERFORMANCE EVALUATION

by

Jesse Bloecker

A thesis submitted to the Faculty of the University of Delaware in partial fulfillment of the requirements for the degree of Master of Science in Mechanical Engineering

Spring 2020

MULTI-CAMERA VISUAL-INERTIAL ODOMETRY:

ALGORITHM DESIGN AND PERFORMANCE EVALUATION

by

Jesse Bloecker

Approved: _____

Guoquan Huang, Ph.D.
Professor in charge of thesis on behalf of the Advisory Committee

Approved: _____

Ajay Prasad, Ph.D.
Chair of the Department of Mechanical Engineering

Approved: _____

Levi T. Thompson, Ph.D.
Dean of the College of Engineering

Approved: _____

Douglas J. Doren, Ph.D.
Interim Vice Provost for Graduate and Professional Education and
Dean of the Graduate College

ACKNOWLEDGMENTS

I feel indebted to my friends Yulin, Kevin, Patrick, Zheng, Woosik, Nate and the rest of the RPNG lab for offering me so much guidance and patience in the past few years. There is no other environment where I've felt so at ease to ask questions and discuss my ideas. I sincerely thank my advisor Paul Huang for maintaining the Robot Perception and Navigation Group, for supporting all of us, for always looking to leverage my individual skills, and for leading me in the right direction while giving me the freedom to explore on my own. Likewise, I deeply appreciate all of the help I've received from the mechanical engineering faculty at UD, both as an undergraduate and a graduate student.

I also owe a debt of gratitude to Chris Kroninger and Steve Nogar at the Army Research Lab, with whom I have spent much of the past two years— to Chris for always finding the best opportunities for me and for supporting anything I wish to pursue in any way he can, and to Steve for the huge amount of advice and guidance he has given me on the practical side of robotics. ARL has been greatly supportive of this project and other research led by RPNG— I look forward to continuing our work together.

Lastly, I would be remiss not to acknowledge all of my friends and family— throughout this project and in everything that I do, I am extremely grateful to have their consistent support.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

It is essential for micro aerial vehicles (MAVs) to be able to track their 3D motion in real time using limited onboard sensing and computational resources. To this end, the Multi-State Constraint Kalman Filter (MSCKF), a variant of the Extended Kalman Filter (EKF), is among the most popular approaches to provide real-time motion tracking using camera and IMU sensors and has been proven capable of accurate localization at reduced computational cost. As cameras are becoming ubiquitous, one of the main objectives of this work is to design an efficient multi-camera MSCKF-based visual-inertial odometry (VIO) algorithm that can utilize an arbitrary number of asynchronous cameras. While the standard MSCKF framework is well defined, it leaves available many design choices in implementation that have significant impact on both localization accuracy and computational complexity. These design choices depend strongly on the computational resources available on the platform of interest, which can vary greatly from platforms such as laptop computers to more resource-constrained system-on-chip (SoC) platforms typically found onboard MAVs. We evaluate the performance of our MSCKF-based VIO algorithm on a set of different computing platforms in order to better understand how to optimize the estimator parameters in a resource-aware way. These parameters include the number of stochastic clones to retain in the state vector and the number of visual feature points to track. In particular, in the design of our multi-camera VIO algorithm, the number of independent camera streams to process is also considered as a design parameter. The results of this study provide insight on selecting the parameters that yield the best possible performance for a given computing platform.

Chapter 1

INTRODUCTION

1.1 Micro Aerial Vehicles

The decline in cost and rise in processing power of embedded computing devices in recent years has led to a fast-growing potential for the applications of micro aerial vehicles (MAVs). In particular, quadrotor MAVs (e.g. Figure 1) are very common due to their versatility, offering better maneuverability in constrained spaces compared to fixed-wing aircraft, while still remaining mechanically simple. Sufficiently small MAVs can be operated safely indoors as well as in urban or other constrained environments, enabling the autonomous indoor mapping of buildings [1] and the exploration of caves [2]. The use of MAVs continues to extend into all different industries, with applications in agriculture for the detection of plant diseases [3], on oil refineries for the inspection of pipes [4], in transportation for traffic monitoring [5] and autonomous package delivery [6], among many others. In addition, MAVs serve an increasingly important role in military operations; many computer vision-based tasks such as surveillance and reconnaissance, inspection, and area mapping can be carried out on MAVs with increasing levels of autonomy. These capabilities reduce risk to soldiers in hostile environments and also provide situational awareness which otherwise may not be possible. Significant research interest also lies in robotic swarming/distributed intelligence systems through MAVs— these technologies hold great potential to aid in search and rescue missions [7] [8].

In any of these applications, it is critical that the MAV can accurately track its position and orientation within its environment in real-time through on-board sensors. GPS is very commonly used as a source of position data, however, MAVs must often

Figure 1: A custom-built quadrotor based on Qualcomm's Snapdragon Flight, which is one of the computing platforms evaluated in Chapter 4. This is a representative MAV platform with size suitable for many of the aforementioned applications.

be operated in an environment where GPS is unreliable (e.g. indoors) or completely denied (e.g. in hostile territory due to GPS jamming). Additionally, GPS receivers suitable for MAVs are typically only accurate to within a few meters— unless expensive RTK-GPS receivers are used, which require position data from additional ground-fixed transmitters. These limitations have motivated great research efforts in alternative localization methods. Among which, visual-inertial navigation systems (VINS) [9] are among the most popular. The decrease in cost of small, lightweight cameras and IMU sensors (see Appendix B) in recent years has lead to widespread implementation of VINS both on air vehicles and in other domains (e.g. on mobile devices). Cameras provide dense information and serve as a good complement to noise-corrupted inertial sensors. However, a major factor that limits the deployment of VINS on MAVs is the on-board computational power required to achieve acceptable performance. Even on high-end embedded computing platforms, many "off-the-shelf" VINS implementations remain too computationally demanding for real-time processing. Accordingly, it is

necessary to design an efficient VINS algorithm that can provide accurate localization in real time using limited onboard sensing and computational resources.

1.2 Visual-Inertial Navigation

Within the field of visual-inertial navigation, there are two related classes of problems: visual-inertial odometry (VIO) and visual-inertial simultaneous localization and mapping (VI-SLAM). In VIO, as illustrated in Figure 2, the objective is to track the position and orientation (pose) of a sensor platform as it moves along a 3D trajectory through onboard camera and IMU sensors.

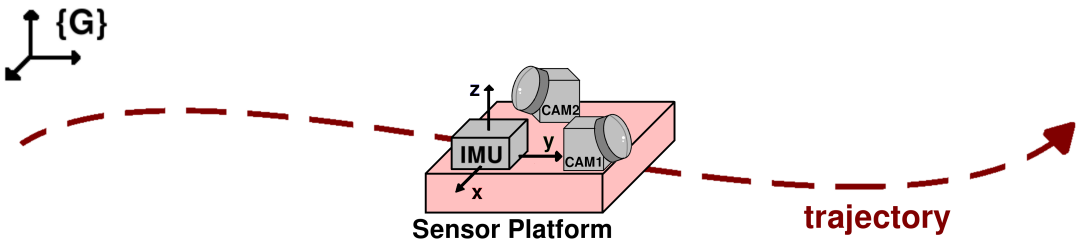


Figure 2: Sensor platform moving through 3D space.

In SLAM, the problem is not only to recover the trajectory of the sensor platform, but to jointly estimate the position of 3D points (feature points) in order to build up a map of the environment through which the platform is moving [10]. The fundamental advantage of VI-SLAM, e.g [11], is that it utilizes the correlations that exist between recognized 3D feature positions in the map and the current pose of the camera, resulting in better localization accuracy. However, this often results in prohibitively high computational complexity due to the continuously growing number of 3D feature positions that are maintained in the map. Mourikas and Roumeliotis [12] addressed this issue by introducing an efficient feature marginalization technique in the Multi-State Constraint Kalman Filter (MSCKF), which is the basis of the multi-camera VIO system presented in this work (for a breif overview of the Kalman Filter, see Appendix A). The main benefit of MSCKF is that it leverages the geometric constraints imposed by multiple observations of a 3D feature point *wthout* storing the position of the 3D

feature point, limititing the computational burden. Shortly after its original publication, MSCKF has been implemented for spacecraft decent and landing [13], and has since seen growing popularity in many visual-inertial systems. It has also served as a basis for consistency-improved estimators through enforcing observability constraints [14–16].

Much of the focus of VINS literature pertains to either monocular (one camera) or stereo (two rigidly connected, synchronized cameras observing the same space) sensor configurations —both of which have been used in implementations of MSCKF [17], but these configurations suffer from certain limitations. The most obvious of which is single-sensor failure; if a camera stops working for any reason, it is catastrophic to the whole VINS system. Barring this situation, having only a single viewing direction available can be problematic if the view falls on regions of low texture or poor lighting, as the loss of visual feature tracking (see Section 3.2) is likely. Another problem that particularly effects monocular and stereo VINS is the motion of objects in the environment; since it is assumed that the environment is static, moving objects within the field of view are falsely interpreted as motion of the sensor platform, leading to localization errors. Lastly, in open environments with a lack of objects near the camera, visual feature points behave like "points at infinity", a degenerate case when performing feature triangulation (see Section 2.6) which can severely corrupt estimation accuracy.

Having multiple camera views from which to process measurements available at all times offers some robustness to these conditions, as well as a significant gain of information that can ultimately lead to improved localization accuracy. However, as this gain of visual data carries with it an increased computational cost, it does not suffice to simply process all measurements from multiple cameras using the standard MSCKF framework (to be clear, the original MSCKF is not a multi-*camera* framework, but it leverages multiple previous states of the sensor platform to estimate its current state). Accordingly, we implement the multi-camera VINS algorithm first introduced in our recent work [18]. This estimator is an extension of the MSCKF framework to *efficiently* incorporate visual measurements from an arbitrary number of *asynchronous*

cameras pointed in arbitrary directions (no requirement of overlapping views).

The remainder of this work is outlined as follows: a detailed review of the original (monocular-case) MSCKF estimator is provided in Chapter 2. Based on this, Chapter 3 outlines our multi-camera extension of MSCKF (including the vision-processing front-end) and the results of real-world experiments. Finally, a performance evaluation of both the monocular and multi-camera MSCKF estimators is given in Chapter 4.

Chapter 2

MULTI-STATE CONSTRAINT KALMAN FILTER

The proposed multi-camera VIO system design is based on the computationally efficient MSCKF framework [12]. Before discussing the details of our *multi-camera* MSCKF system (Chapter 3), the main components of the standard (monocular-case) MSCKF are reviewed in this chapter.

2.1 Overview

Instead of considering only the *current* pose of the sensor platform as it travels through its environment, a key concept of MSCKF is to maintain a sliding window of past pose estimates, at all times. In the case of our implementation, these are past IMU poses, but camera poses could also be used, as the transformation existing between the two is assumed to be known through an offline calibration process. These past pose estimates are refereed to as "clones" as they are maintained through stochastic cloning described in [19]. The grouping of IMU clones is based on visual feature points (see Section 3.1). Specifically, any given clone window contains the set of all IMU poses from which a particular feature point was observed by the camera, with each pose time-stamped at the time of observation. Accordingly, IMU clones occur only at imaging times, although IMU measurements are received at a much higher rate (typically, camera frame rates are set to 20-30 Hz while the IMU is sampled at 200-500 Hz). All IMU clones in a given window, and the corresponding visual measurements of the feature point, are used to perform a Kalman Filter update (see Section 2.8).

The number of clones to retain in this sliding window, referred to as the *clone size*, is a freely chosen design choice. A larger clone size means more measurements of a given feature point are utilized in every Kalman Filter update, and also, that

the measurements are likely to a cover larger baseline from which the feature point is observed (see Figure 3). This results more accurate feature triangulation and thus better localization accuracy, but at the expense of added computational complexity. The impact of the clone size on the computational speed on a set of different computing platforms is evaluated in Chapter 4.

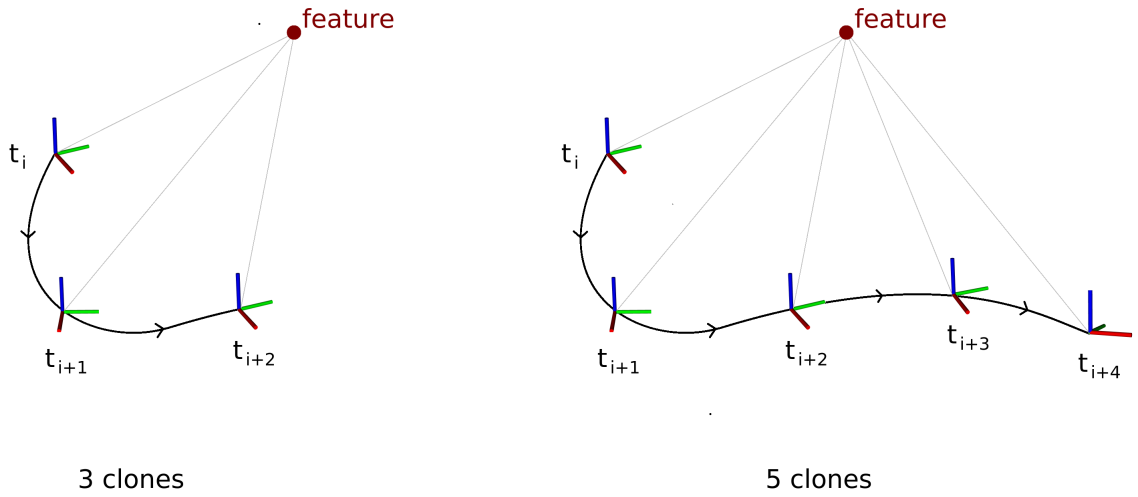


Figure 3: Visualization of MSCKF clone sizes. As the IMU-fixed coordinate frame travels along its trajectory, visual measurements of the feature occur at imaging times $t_i, t_{i+1}...t_{i+M-1}$, where M is the clone size. A larger clone size generally leads to more accurate feature triangulation.

2.2 State Vector

The state vector $\mathbf{x}$ contains all of the quantities to be estimated at every time step. Adhering to standard MSCKF, the state vector can be partitioned into two sub-vectors: the current IMU state, $\mathbf{x}_I$, and the clone state, $\mathbf{x}_M$ which contains IMU poses at the past M imaging times $\{t_k, t_{k-1}, ...t_{k-M+1}\}$. The state vector is written as:

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_I \\ \mathbf{x}_M \end{bmatrix} \quad , \quad \mathbf{x}_I = \begin{bmatrix} {}^I_G\overline{q} \\ {}^G\mathbf{p}_I \\ {}^G\mathbf{v}_I \\ \mathbf{b}_\omega \\ \mathbf{b}_a \end{bmatrix} \quad , \quad \mathbf{x}_M = \begin{bmatrix} \begin{bmatrix} {}^I_G\overline{q} \\ {}^G\mathbf{p}_I \end{bmatrix}_{t_k} \\ \vdots \\ \begin{bmatrix} {}^I_G\overline{q} \\ {}^G\mathbf{p}_I \end{bmatrix}_{t_{k-M+1}} \end{bmatrix} \tag{1}$$

where ${}^G\mathbf{p}_I$ and ${}^G\mathbf{v}_I$ are the position and velocity of the IMU, respectively, expressed in the global frame $\{G\}$, and ${}^I_G\overline{q}$ is the unit quaternion (using the JPL convention [20]) corresponding to the rotation matrix ${}^I_G\mathbf{R}$, which transforms vectors in $\{G\}$ to the IMU-affixed frame $\{I\}$. The remaining entries in the IMU state vector, $\mathbf{b}_\omega$ and $\mathbf{b}_a$, are the 3×1 gyroscope and accelerometer biases, respectively. These biases essentially describe the average output of the sensor when it is not moving (i.e. a constant offset) and tend to grow over time. The initial gyroscope bias can be determined simply through averaging measurements while the IMU is stationary. The same is true for the initial accelerometer bias, but the orientation of the IMU axes with respect to the gravity vector must be known.

The IMU state $\mathbf{x}_I$ at any time has a corresponding IMU *error state* $\widetilde{\mathbf{x}}_I$, which represents the difference between the true and estimated states. Specifically, the true value of the state, the error state, and the estimated state (denoted $\widehat{\mathbf{x}}_I$) are related as follows:

$$\mathbf{x}_I = \widehat{\mathbf{x}}_I \boxplus \widetilde{\mathbf{x}}_I \tag{2}$$

The $\boxplus$ notation indicates addition for vector quantities, but for quaternions indicates the following operation (similarly—the true, estimated, and error quaternions

are denoted $\overline{q}$, $\widehat{\overline{q}}$ and $\widetilde{\overline{q}}$, respectively):

$$\overline{q} = \widetilde{\overline{q}} \otimes \widehat{\overline{q}} \quad \approx \quad \begin{bmatrix} \frac{1}{2}\delta\boldsymbol{\theta} \\ 1 \end{bmatrix} \otimes \widehat{\overline{q}} \tag{3}$$

where $\otimes$ denotes quaternion multiplication and $\delta\boldsymbol{\theta}$ is a 3-DOF representation of the orientation error. Accordingly, the 15×1 IMU error state is given by:

$$\widetilde{\mathbf{x}}_I = \begin{bmatrix} {}^I\delta\boldsymbol{\theta}_G^\top & {}^G\widetilde{\mathbf{p}}_I^\top & {}^G\widetilde{\mathbf{v}}_I^\top & \widetilde{\mathbf{b}}_\omega^\top & \widetilde{\mathbf{b}}_a^\top \end{bmatrix}^\top \tag{4}$$

and the complete error state, with the past M IMU poses denoted $I_0 \dots I_{M-1}$, is given by:

$$\widetilde{\mathbf{x}} = \begin{bmatrix} \widetilde{\mathbf{x}}_I^\top & {}^{I_0}\delta\boldsymbol{\theta}_G^\top & {}^G\widetilde{\mathbf{p}}_{I_0}^\top & \dots & {}^{I_{M-1}}\delta\boldsymbol{\theta}_G^\top & {}^G\widetilde{\mathbf{p}}_{I_{M-1}}^\top \end{bmatrix}^\top \tag{5}$$

2.3 IMU Propagation

As is commonly done for VINS systems, IMU measurements are used to replace the system model in the Kalman Filter (see Appendix A). The equations modelling the quaternion-based inertial dynamics for state propagation are somewhat standard and this section contains only a summary of the results. All information presented in this section is consistent with [9], [12], [13], and [20].

The measured angular rate $\boldsymbol{\omega}_m$ and linear acceleration $\mathbf{a}_m$ as reported by the IMU relate to the true accelerations (expressed in the IMU frame) ${}^I\mathbf{a}$ and ${}^I\boldsymbol{\omega}$ according to:

$$\boldsymbol{\omega}_m = {}^I\boldsymbol{\omega} + \mathbf{b}_\omega + \mathbf{n}_g \tag{6}$$

$$\mathbf{a}_m = {}^I\mathbf{a} + {}^I_G\mathbf{R}\,{}^G\mathbf{g} + \mathbf{b}_a + \mathbf{n}_a \tag{7}$$

where ${}^G\mathbf{g} \approx [0 \quad 0 \quad -9.81\,\mathrm{m/s^2}]^\top$ and $\mathbf{n}_g$ and $\mathbf{n}_a$ are modeled as zero-mean, Gaussian white noise. The time evolution of the IMU state (1), as well as the time evolution

of the corresponding *estimated* state are written component-wise as:

$$\begin{aligned}
&{}^{I}_{G}\dot{\bar{\mathbf{q}}}(t) = \frac{1}{2}\boldsymbol{\Omega}\big(\boldsymbol{\omega}(t)\big)\,{}^{I}_{G}\bar{\mathbf{q}}(t) && {}^{I}_{G}\dot{\hat{\bar{\mathbf{q}}}}(t) = \frac{1}{2}\boldsymbol{\Omega}\big(\hat{\boldsymbol{\omega}}(t)\big)\,{}^{I}_{G}\hat{\bar{\mathbf{q}}}(t)\\
&{}^{G}\dot{\mathbf{p}}(t) = {}^{G}\mathbf{v}(t) && {}^{G}\dot{\hat{\mathbf{p}}}(t) = {}^{G}\hat{\mathbf{v}}(t)\\
&{}^{G}\dot{\mathbf{v}}(t) = {}^{G}\mathbf{a}(t) \quad \xrightarrow[\text{current estimate}]{\text{linearize at}} && {}^{G}\dot{\hat{\mathbf{v}}}(t) = {}^{G}\hat{\mathbf{a}}(t)\\
&\dot{\mathbf{b}}_g(t) = \mathbf{n}_{wg}(t) && \dot{\hat{\mathbf{b}}}_g(t) = \mathbf{0}_{3\times 1}\\
&\dot{\mathbf{b}}_a(t) = \mathbf{n}_{wa}(t) && \dot{\hat{\mathbf{b}}}_a(t) = \mathbf{0}_{3\times 1}
\end{aligned} \tag{8}$$

where $\boldsymbol{\Omega}$ is defined as follows (see [20]), using floor brackets $\lfloor\,\rfloor$ to denote the skew-symmetric operator, in this case on vector $\boldsymbol{\omega} = [\omega_1 \quad \omega_2 \quad \omega_3]^\top$:

$$\boldsymbol{\Omega}(\boldsymbol{\omega}) = \begin{bmatrix} -\lfloor\boldsymbol{\omega}\rfloor & \boldsymbol{\omega} \\ -\boldsymbol{\omega}^T & 0 \end{bmatrix} \quad , \quad \lfloor\boldsymbol{\omega}\rfloor = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix} \tag{9}$$

Propagation of the error-state vector defined in (4) is governed by:

$$\dot{\tilde{\mathbf{x}}}_I(t) = \mathbf{F}_c(t)\tilde{\mathbf{x}}_I(t) + \mathbf{G}_c(t)\mathbf{n}_I(t) \tag{10}$$

where $\mathbf{n}_I = [\mathbf{n}_g^\top \quad \mathbf{n}_{wg}^\top \quad \mathbf{n}_a^\top \quad \mathbf{n}_{wa}^\top]^\top$ is the system noise (components $\mathbf{n}_{wg}$ and $\mathbf{n}_{wa}$ are the bias-driving noise processes on the gyroscope and accelerometer, respectively, and $\mathbf{n}_g$ and $\mathbf{n}_a$ are the measurement noises in (6) and (7)). $\mathbf{F}_c$ is the *continuous-time* (indicated by the c subscript) error-state transition matrix, and $\mathbf{G}_c$ is the corresponding noise matrix:

$$\mathbf{F}_c = \begin{bmatrix} -\lfloor\hat{\boldsymbol{\omega}}\rfloor & -\mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ -{}^{I}_{G}\hat{\mathbf{R}}^\top\lfloor\hat{\mathbf{a}}\times\rfloor & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & -{}^{I}_{G}\hat{\mathbf{R}}^\top & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \end{bmatrix} \quad \mathbf{G}_c = \begin{bmatrix} -\mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & -{}^{I}_{G}\hat{\mathbf{R}}^\top & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \end{bmatrix} \tag{11}$$

where ${}^{I}_{G}\widehat{\mathbf{R}}$ is the equivalent rotation matrix for quaternion ${}^{I}_{G}\bar{q}$ in the state vector. While (10) describes the continuous-time propagation of IMU measurements, it is necessary in any implementation to derive from it the discrete-time state transition matrix, $\mathbf{\Phi}_k = \mathbf{\Phi}(t_{k+1}, t_k)$, which is typically computed by solving the matrix differential equation:

$$\dot{\mathbf{\Phi}}(t_{k+1}, t_k) = \mathbf{F}_c(t_{k+1})\mathbf{\Phi}(t_{k+1}, t_k) \quad , \quad {}^{\text{INITIAL}}_{\text{CONDITION}} : \mathbf{\Phi}(t_k, t_k) = \mathbf{I}_{15\times 15} \tag{12}$$

This can be solved either numerically [13, 20] or analytically [14, 21–23]. Finally, after the state transition matrix is computed, the 15×15 IMU error state covariance can be propagated in the EKF using the standard equation:

$$\mathbf{P}_{II,k+1|k} = \mathbf{\Phi}_k \mathbf{P}_{II,k|k} \mathbf{\Phi}_k^\top + \mathbf{Q}_k \tag{13}$$

where $\mathbf{P}_{II,k|k}$ is a submatrix of the full state covariance $\mathbf{P}_{k|k}$, which also contains the $6M \times 6M$ clone state covariance $\mathbf{P}_{MM,k|k}$, and the cross covariance between the two:

$$\mathbf{P}_{k|k} = \begin{bmatrix} \mathbf{P}_{II,k|k} & \mathbf{P}_{IM,k|k} \\ \mathbf{P}_{IM,k|k}^\top & \mathbf{P}_{MM,k|k} \end{bmatrix} \tag{14}$$

and $\mathbf{Q}_{d,k}$ is the discrete-time system noise covariance matrix computed from $\mathbf{\Phi}$ and $\mathbf{G}_c$ according to:

$$\mathbf{Q}_{d,k} = \int_{t_k}^{t_{k+1}} \mathbf{\Phi}(t_{k+1}, \tau)\mathbf{G}_c(\tau)\mathbf{Q}_c\mathbf{G}_c^T(\tau)\mathbf{\Phi}^T(t_{k+1}, \tau)d\tau \tag{15}$$

The covariance matrix (14) is propagated according to following, where the top left entry is the result of (13).

$$\mathbf{P}_{k+1|k} = \begin{bmatrix} \mathbf{P}_{II,k+1|k} & \mathbf{\Phi}_k \mathbf{P}_{IM,k|k} \\ \mathbf{P}_{IM,k|k}^\top \mathbf{\Phi}_k^\top & \mathbf{P}_{MM,k|k} \end{bmatrix} \tag{16}$$

2.4 State Augmentation

When the estimator receives a new image from the camera, the state is propagated up to the imaging time. Stochastic cloning is then applied to the propagated state, before the EKF update, to produce the augmented state containing additional poses. The state vector can only keep growing until it contains M past poses (thus, the maximum size of the error state vector is $15+6M \times 1$). The cloning process is as follows: the current state $\mathbf{x}$ with covariance $\mathbf{P}$ is augmented with another state (denoted $\mathbf{y}$) which can be written as a function $\mathbf{y} = \mathbf{g}(\mathbf{x})$. The current estimate (mean) of the augmented state is then:

$$\widehat{\mathbf{x}}_{\text{AUG}} = \begin{bmatrix} \widehat{\mathbf{x}} \\ \widehat{\mathbf{y}} \end{bmatrix} = \begin{bmatrix} \widehat{\mathbf{x}} \\ \mathbf{g}(\widehat{\mathbf{x}}) \end{bmatrix} \tag{17}$$

and its covariance is:

$$\mathbf{P}_{\text{AUG}} = \mathbf{\Psi} \mathbf{P} \mathbf{\Psi}^\top \quad , \quad \mathbf{\Psi} = \begin{bmatrix} \mathbf{I} \\ \frac{\partial \widetilde{\mathbf{y}}}{\partial \widetilde{\mathbf{x}}} \end{bmatrix} = \begin{bmatrix} \mathbf{I} \\ \frac{\partial \mathbf{g}(\widehat{\mathbf{x}} \boxplus \widetilde{\mathbf{x}}) \boxminus \mathbf{g}(\widehat{\mathbf{x}})}{\partial \widetilde{\mathbf{x}}} \end{bmatrix} \tag{18}$$

Where $\frac{\partial \widetilde{\mathbf{y}}}{\partial \widetilde{\mathbf{x}}}$ is the Jacobian of $\widetilde{\mathbf{y}}$ with respect to the error state (5). With these quantities computed, $\widehat{\mathbf{x}}_{\text{AUG}}$ and $\mathbf{P}_{\text{AUG}}$ replace the current state and covariance:

$$\widehat{\mathbf{x}} \longleftarrow \widehat{\mathbf{x}}_{\text{AUG}}$$

$$\mathbf{P} \longleftarrow \mathbf{P}_{\text{AUG}}$$

In our MSCKF implementation, since historical *IMU poses* are stored as clones (as opposed to camera poses), $\mathbf{g}(\widehat{\mathbf{x}}) = [{}^I_G\widehat{q} \quad {}^G\widehat{\mathbf{p}}_I]^\top$, corresponding to the current IMU state.

2.5 Measurement Model

The working principle of MSCKF is to leverage *multiple* observations (measurements) of the same 3D feature point to constrain the pose estimate of the sensor platform. To this end, all measurements of a feature point are used to estimate its 3D position through triangulation (Section 2.6). Then, this position estimate is used to compute a measurement residual (Section 2.7), and ultimately to update the state.

Consider a single moving camera that observes a single feature point from M different poses (denoted $\{C_0\}, \{C_1\}...\{C_{M-1}\}$) as illustrated in Figure 4. As before, the corresponding IMU poses at these imaging times constitute the clone state for this particular feature point.

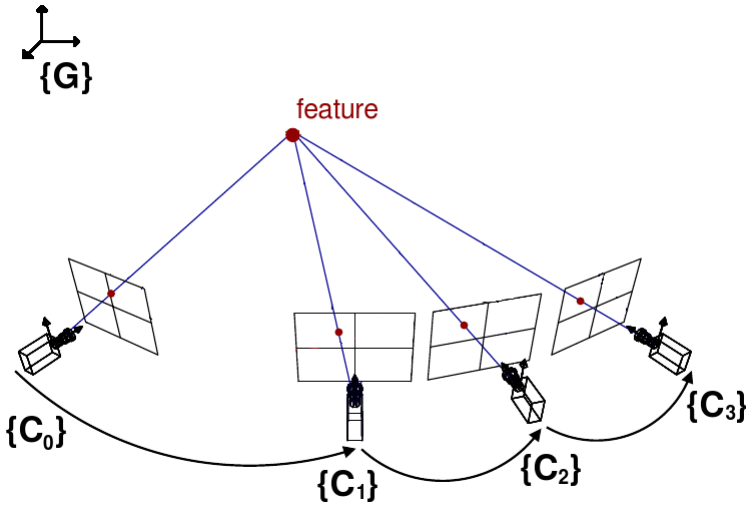


Figure 4: Single 3D feature point measured at $M = 4$ camera poses.

The measurement of the feature point in the ith camera frame ($i \in \{0, 1, \ldots, M-1\}$) within the clone state, $\mathbf{z}_i$, is realized as a 2D pixel coordinate $\begin{bmatrix} u_i & v_i \end{bmatrix}^\top$. The measurement function which maps 3D positions in any given camera frame onto the imaging plane is based on pinhole camera model (see Figure 5) with additive gaussian noise $\mathbf{n}_i$:

$$\mathbf{z}_i = \begin{bmatrix} u_i \\ v_i \end{bmatrix} = \frac{1}{{}^{C_i}Z} \begin{bmatrix} {}^{C_i}X \\ {}^{C_i}Y \end{bmatrix} + \mathbf{n}_i \tag{19}$$

where $[{}^{C_i}X \quad {}^{C_i}Y \quad {}^{C_i}Z]^\top = {}^{C_i}\mathbf{p}_f$ is the feature position in the ith camera frame, and $\mathbf{n}_i$ has covariance $\mathbf{R}_i = {\sigma^2}_{im}\mathbf{I}_2$.

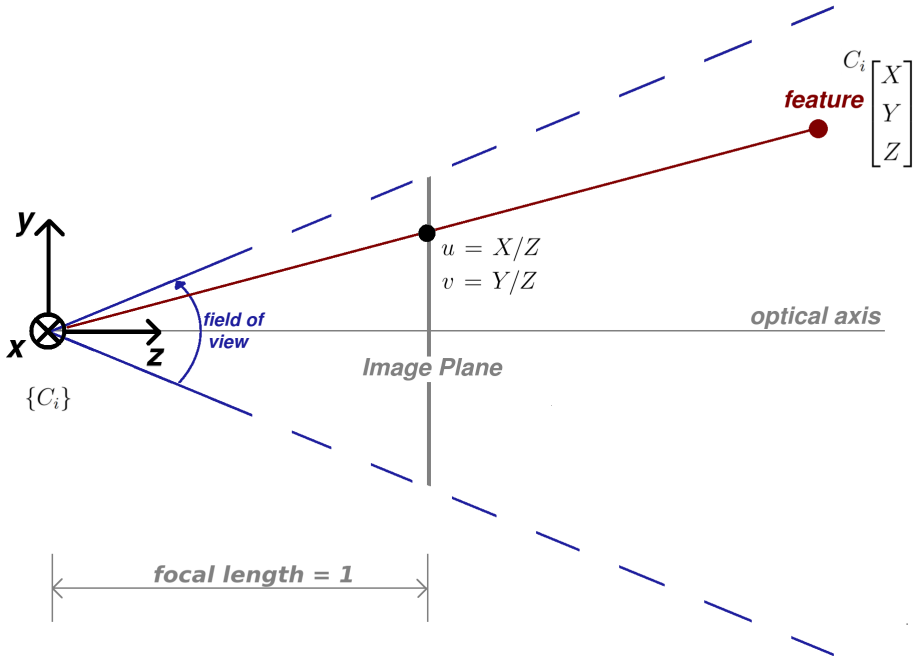


Figure 5: Pinhole camera model: The origin of the coordinate frame attached to the camera coincides with the camera center, and the z-axis aligns the optical axis. With the focal length equal to one, this figure illustrates the simplest case —in practice, camera calibration is required to normalize the measurements from an arbitrary focal length and optical center. Additional correction factors for lens distortion are typically necessary as well.

2.6 Feature Triangulation

The 3D position of a feature point observed from at least two different camera poses can be solved using the known transformations (by treating the pose estimates in the clone state as true) that exists between the camera frames. To obtain a solution, the first step is to choose any of the frames from which the feature was observed to serve as the "base frame", here it is assumed to be $\{C_0\}$ as shown in Figure 6.

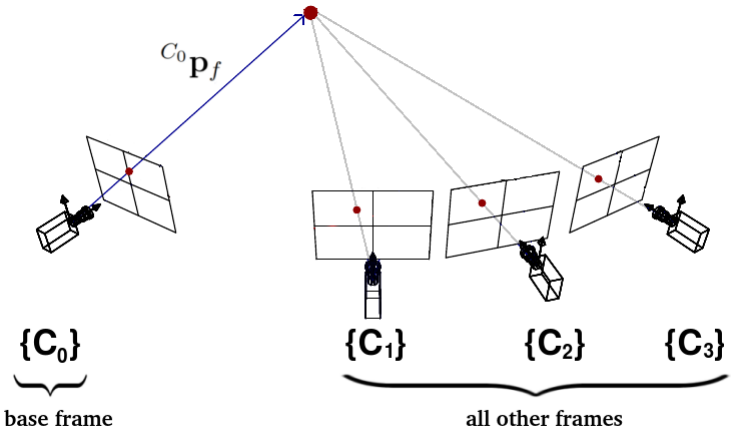


Figure 6: Multiple measurements of a 3D feature point with $\{C_0\}$ selected as the base frame to which all of the other measurements will be related.

The feature position in any non-base frame $\{C_i\}$ relates to the feature position in the base frame according to the following transformation:

$$^{C_i}\mathbf{p}_f = {}^{C_i}\mathbf{p}_{C_0} + {}^{C_i}_{C_0}\mathbf{R}\,{}^{C_0}\mathbf{p}_f \tag{20}$$

The feature position in the ith frame $^{C_i}\mathbf{p}_f$ is constrained based on the pinhole camera model illustrated in Figure 5. Specifically, the mapping $u = X/Z$ and $v = Y/Z$ yield the following two (independent) constraints on the feature position $[X \quad Y \quad Z]^\top$.

$$\begin{bmatrix} -1 & 0 & u \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} -1 & 0 & X/Z \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = 0 \tag{21}$$

$$\begin{bmatrix} 0 & -1 & v \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0 & -1 & Y/Z \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = 0 \tag{22}$$

Using the following definitions:

$$\mathbf{a}_i = \begin{bmatrix} -1 \\ 0 \\ u_i \end{bmatrix} \quad , \quad \mathbf{b}_i = \begin{bmatrix} 0 \\ -1 \\ v_i \end{bmatrix} \tag{23}$$

the constraints (21) and (22) can be expressed for each pose i as:

$$\mathbf{a}_i^{\top {}^{C_i}}\mathbf{p}_f = 0 \tag{24}$$

$$\mathbf{b}_i^{\top {}^{C_i}}\mathbf{p}_f = 0 \tag{25}$$

The definitions in (23) are not the only ones possible; any vectors constructed from the measurements that are orthogonal to the feature position could also be used. Transforming these constraints from each frame $\{C_i\}$ into the base frame $\{C_0\}$ yields:

$$\begin{aligned} \mathbf{a}_i^\top \left({}^{C_i}_{C_0}\mathbf{R}\, {}^{C_0}\mathbf{p}_f + {}^{C_i}\mathbf{p}_{C_0} \right) = 0 \\ \mathbf{b}_i^\top \left({}^{C_i}_{C_0}\mathbf{R}\, {}^{C_0}\mathbf{p}_f + {}^{C_i}\mathbf{p}_{C_0} \right) = 0 \end{aligned} \quad \Longrightarrow \quad \begin{aligned} \mathbf{a}_i^{\top C_i}_{C_0}\mathbf{R}\, {}^{C_0}\mathbf{p}_f = \mathbf{a}_i^{\top C_i}_{C_0}\mathbf{R}\, {}^{C_0}\mathbf{p}_{C_i} \\ \mathbf{b}_i^{\top C_i}_{C_0}\mathbf{R}\, {}^{C_0}\mathbf{p}_f = \mathbf{b}_i^{\top C_i}_{C_0}\mathbf{R}\, {}^{C_0}\mathbf{p}_{C_i} \end{aligned} \tag{26}$$

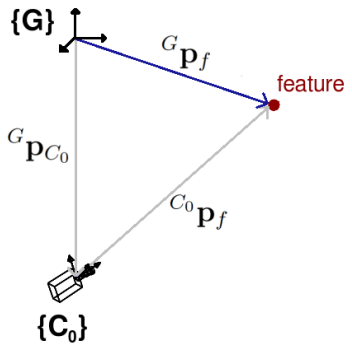
By stacking this pair of equations for each of the measurements ($i \in \{0, 1, \ldots, M-1\}$), we can construct the system (27) to solve for the feature position in the base frame ${}^{C_0}\mathbf{p}_f$. Since each of the M measurements creates two independent constraints on the feature position, the left side (denoted $\mathbf{A}$) has dimension $2M \times 3$ and the right side (denoted $\mathbf{c}$) has dimension $2M \times 1$.

$$\underbrace{\begin{bmatrix} \mathbf{a}_0^\top \\ \mathbf{b}_0^\top \\ \mathbf{a}_1^\top {}^{C_1}_{C_0}\mathbf{R} \\ \mathbf{b}_1^\top {}^{C_1}_{C_0}\mathbf{R} \\ \vdots \\ \mathbf{a}_{M-1}^\top {}^{C_{M-1}}_{C_0}\mathbf{R} \\ \mathbf{b}_{M-1}^\top {}^{C_{M-1}}_{C_0}\mathbf{R} \end{bmatrix}}_{\boldsymbol{A}} {}^{C_0}\mathbf{p}_f = \underbrace{\begin{bmatrix} 0 \\ 0 \\ \mathbf{a}_1^\top {}^{C_1}_{C_0}\mathbf{R}\,{}^{C_0}\mathbf{p}_{C_1} \\ \mathbf{b}_1^\top {}^{C_1}_{C_0}\mathbf{R}\,{}^{C_0}\mathbf{p}_{C_1} \\ \vdots \\ \mathbf{a}_{M-1}^\top {}^{C_{M-1}}_{C_0}\mathbf{R}\,{}^{C_0}\mathbf{p}_{C_{M-1}} \\ \mathbf{b}_{M-1}^\top {}^{C_{M-1}}_{C_0}\mathbf{R}\,{}^{C_0}\mathbf{p}_{C_{M-1}} \end{bmatrix}}_{\boldsymbol{c}} \tag{27}$$

Since the system (27) is overdetermined, the feature position is given by the least-squares solution:

$$ {}^{C_0}\mathbf{p}_f = (\mathbf{A}^\top \mathbf{A})^{-1}\mathbf{A}^\top \mathbf{c} \tag{28}$$

The final step is to transform the solution (28) from the base frame into the global frame according to the following geometric relation:



$$\longrightarrow \quad {}^G\mathbf{p}_f = {}^G\mathbf{p}_{C_0} + {}^G_{C_0}\mathbf{R}\,{}^{C_0}\mathbf{p}_f \tag{29}$$

Although this process produces a reasonable estimate for feature position ${}^G\widehat{\mathbf{p}}_f$, for better accuracy, it can be further optimized by formulating a non-linear least squares problem (see [24]) and treating this result as an initial guess. Then, a refined solution can be obtained iteratively (e.g. through the Gauss-Newton method). With

an estimate for the feature position, the measurement residual can be computed and ultimately used to produce a state update as discussed in the following section.

2.7 Measurement Residual

In practice, it is necessary to apply the triangulation process of Section 2.6 to many, often hundreds, of feature points over a short time in order to achieve sufficient localization accuracy. Every observation of every feature point has a corresponding measurement residual. We use the following notation to account for each of them: the residual corresponding to the ith measurement of the jth feature point f_j is denoted $\widetilde{\mathbf{z}}_{j,i}$ and defined as:

$$\widetilde{\mathbf{z}}_{j,i} = \mathbf{z}_{j,i} - \widehat{\mathbf{z}}_{j,i} \tag{30}$$

where $\widehat{\mathbf{z}}_{j,i}$ is the predicted measurement computed based on the ith pose estimate in the clone state and the result of (29):

$$\widehat{\mathbf{z}}_{j,i} = \frac{1}{{}^{C_i}\widehat{Z}_j} \begin{bmatrix} {}^{C_i}\widehat{X}_j \\ {}^{C_i}\widehat{Y}_j \end{bmatrix} \quad , \quad \begin{bmatrix} {}^{C_i}\widehat{X}_j \\ {}^{C_i}\widehat{Y}_j \\ {}^{C_i}\widehat{Z}_j \end{bmatrix} = {}^{C_i}\widehat{\mathbf{p}}_{f_j} = {}^{C}_{I}\mathbf{R}\,{}^{I_i}_{G}\widehat{\mathbf{R}}({}^{G}\widehat{\mathbf{p}}_{f_j} - {}^{G}\widehat{\mathbf{p}}_{I_i}) + {}^{C}\mathbf{p}_I \tag{31}$$

where ${}^{C}_{I}\mathbf{R}$ and ${}^{C}\mathbf{p}_I$ (the orientation and position of the camera with respect to the IMU) are assumed to be known extrinsic parameters obtained through an offline calibration process. In order to apply an update within the EKF framework, it is necessary to use a residual $\widetilde{\mathbf{z}}$ which satisfies the general form:

$$\widetilde{\mathbf{z}} \approx \mathbf{H}\widetilde{\mathbf{x}} + \mathbf{n} \tag{32}$$

where $\mathbf{H}$ is the measurement Jacobian (derived in Section 2.8) corresponding to the $\widetilde{\mathbf{x}}$, and $\mathbf{n}$ is zero-mean Gaussian noise *uncorrelated* with $\widetilde{\mathbf{x}}$. For the single measurement residual $\widetilde{\mathbf{z}}_{j,i}$, this implies

$$\widetilde{\mathbf{z}}_{j,i} \approx \mathbf{H}_{j,i}\widetilde{\mathbf{x}} + \mathbf{n}_{j,i} \tag{33}$$

Since feature positions are not contained in the state vector, (33) is separated as follows:

$$\widetilde{\mathbf{z}}_{j,i} \approx \begin{bmatrix} \mathbf{H}_{\mathbf{x}_{j,i}} & \mathbf{H}_{f_{j,i}} \end{bmatrix} \begin{bmatrix} \widetilde{\mathbf{x}} \\ {}^{G}\widetilde{\mathbf{p}}_{f_j} \end{bmatrix} + \mathbf{n}_{j,i} = \mathbf{H}_{\mathbf{x}_{j,i}}\widetilde{\mathbf{x}} + \mathbf{H}_{f_{j,i}}{}^{G}\widetilde{\mathbf{p}}_{f_j} + \mathbf{n}_{j,i} \tag{34}$$

where $\mathbf{H}_{\mathbf{x}_{j,i}}$ and $\mathbf{H}_{f_{j,i}}$ are the Jacobians with respect to the state and feature position, respectively. Then, the residuals from *all* measurements of feature point f_j (i.e. $\{\mathbf{z}_{j,0}, \mathbf{z}_{j,1}, \ldots, \mathbf{z}_{j,M-1}\}$, assuming f_j has M measurements) and their corresponding state and feature Jacobians can be vertically stacked into a single system for feature f_j:

$$\widetilde{\mathbf{z}}_j \approx \mathbf{H}_{\mathbf{x}_j}\widetilde{\mathbf{x}} + \mathbf{H}_{f_j}{}^{G}\widetilde{\mathbf{p}}_{f_j} + \mathbf{n}_j \tag{35}$$

However, as the authors note in [12], this residual does not satisfy the form (32) because the state estimate is used to compute ${}^{G}\widehat{\mathbf{p}}_{f_j}$ and therefore the error ${}^{G}\widetilde{\mathbf{p}}_{f_j}$ is correlated with the state errors $\widetilde{\mathbf{x}}$. Equivalently, the last two terms in (35) cannot be combined and treated as noise such that (32) is satisfied. Instead, MSCKF generates a modified residual vector for each feature, denoted here as $\widetilde{\mathbf{z}}'_j$, which does *not* depend on ${}^{G}\widetilde{\mathbf{p}}_{f_j}$. This is done by projecting the system (35) onto the left nullspace of the $\mathbf{H}_{f_j}$ (i.e. the nullspace of $\mathbf{H}_{f_j}^{\top}$):

$$\mathbf{N}^{\top}\widetilde{\mathbf{z}}_j = \mathbf{N}^{\top}\mathbf{H}_{\mathbf{x}_j}\widetilde{\mathbf{x}} + \mathbf{N}^{\top}\mathbf{H}_{f_j}{}^{G}\widetilde{\mathbf{p}}_{f_j} + \mathbf{N}^{\top}\mathbf{n}_j \quad \Longrightarrow \quad \widetilde{\mathbf{z}}'_j = \mathbf{H}'_{\mathbf{x}_j}\widetilde{\mathbf{x}} + \mathbf{n}'_j \tag{36}$$

where $\mathbf{N} = \text{null}(\mathbf{H}_{f_j}^{\top})$ is a unitary matrix with columns forming a basis of the left nullspace of $\mathbf{H}_{f_j}$. Since $\mathbf{H}_{f_j}$ is a $2M_j \times 3$ matrix with full rank, $\mathbf{N}$ has dimension $2M - 3$, and both $\widetilde{\mathbf{z}}'_j$ and $\mathbf{n}'_j$ are $(2M-3) \times 1$ vectors. The covariance (denoted $\mathbf{R}'_j$)

of $\mathbf{n}'_j$ remains a scaled identity matrix since $\mathbf{N}$ is unitary: $\mathbf{R}'_j = \sigma^2\mathbf{I}$. This operation defines the multi-state constraint (the namesake of the filter itself) on the camera poses from which the feature was observed. The transformed residual is independent of errors in the feature position and can be used to compute the Kalman Gain using standard update equations.

An update is performed whenever a feature moves out of the field of view, or is lost due to tracking failure, or reaches its maximum track length (i.e. the clone size). At any given time step, all tracked features which meet one of these conditions (e.g. assuming there are N such features, the set of features $\{f_0, f_1, \ldots, f_{N-1}\}$) are combined into a single update operation by vertically stacking each of their (modified) residuals as computed from (36). This results in the combined system:

$$\widetilde{\mathbf{z}}' = \mathbf{H}'_{\mathbf{x}}\widetilde{\mathbf{x}} + \mathbf{n}' \tag{37}$$

where $\widetilde{\mathbf{z}}'$, $\mathbf{H}'_{\mathbf{x}}$ and $\mathbf{n}'$ contain block elements corresponding to each feature:

$$\begin{aligned}
\widetilde{\mathbf{z}}' &= [\widetilde{\mathbf{z}}_0'^{\top} \quad \widetilde{\mathbf{z}}_1'^{\top} \quad \ldots \quad \widetilde{\mathbf{z}}_{N-1}'^{\top}]^{\top} \\
\mathbf{n}' &= [\widetilde{\mathbf{n}}_0'^{\top} \quad \widetilde{\mathbf{n}}_1'^{\top} \quad \ldots \quad \widetilde{\mathbf{n}}_{N-1}'^{\top}]^{\top} \\
\mathbf{H}'_{\mathbf{x}} &= [\mathbf{H}'^{\top}_{\mathbf{x}_0} \quad \mathbf{H}'^{\top}_{\mathbf{x}_1} \quad \ldots \quad \mathbf{H}'^{\top}_{\mathbf{x}_{N-1}}]^{\top}
\end{aligned}$$

The dimension of $\widetilde{\mathbf{z}}'$ and $\mathbf{n}'$ are $1 \times d$ where $d = \sum_{j=0}^{N-1} 2M_j - 3$. The value of d has a significant impact on the performace of the estimator—this is assessed in Chapter 4. The covariance $\mathbf{R}'$ corresponding to $\mathbf{n}'$ again remains a scaled identity matrix $\sigma^2\mathbf{I}_{d\times d}$, since separate features tracks are uncorrelated. To reduce the complexity of computing the Kalman gain during an update, MSCKF computes the QR decomposition of the matrix $\mathbf{H}'_{\mathbf{x}}$:

$$\mathbf{H}'_{\mathbf{x}} = \begin{bmatrix}\mathbf{Q}_r & \mathbf{Q}_n\end{bmatrix} \begin{bmatrix}\mathbf{H}''_{\mathbf{x}} \\ \mathbf{0}\end{bmatrix} \tag{38}$$

The matrices $\mathbf{Q}_r$ and $\mathbf{Q}_n$ have columns forming an orthogonal basis of the range and nullspace, respectively, of $\mathbf{H}'_{\mathbf{x}}$, and are mutually orthogonal as well. The resulting "compressed" Jacobian $\mathbf{H}''_{\mathbf{x}}$ is upper triangular. Substituting (38) into (37) and pre-multiplying both sides by $[\mathbf{Q}_r \quad \mathbf{Q}_n]^\top$ gives:

$$\begin{bmatrix}\mathbf{Q}_r & \mathbf{Q}_n\end{bmatrix}^\top \widetilde{\mathbf{z}}' = \begin{bmatrix}\mathbf{Q}_r & \mathbf{Q}_n\end{bmatrix}^\top \begin{bmatrix}\mathbf{Q}_r & \mathbf{Q}_n\end{bmatrix} \begin{bmatrix}\mathbf{H}''_{\mathbf{x}} \\ \mathbf{0}\end{bmatrix} \widetilde{\mathbf{x}} + \begin{bmatrix}\mathbf{Q}_r & \mathbf{Q}_n\end{bmatrix}^\top \mathbf{n}'$$

$$= \underbrace{\begin{bmatrix}\mathbf{Q}_r^\top\mathbf{Q}_r & \mathbf{Q}_r^\top\mathbf{Q}_n \\ \mathbf{Q}_n^\top\mathbf{Q}_r & \mathbf{Q}_n^\top\mathbf{Q}_n\end{bmatrix}}_{\mathbf{I}} \begin{bmatrix}\mathbf{H}''_{\mathbf{x}} \\ \mathbf{0}\end{bmatrix} \widetilde{\mathbf{x}} + \begin{bmatrix}\mathbf{Q}_r & \mathbf{Q}_n\end{bmatrix}^\top \mathbf{n}'$$

$$\implies \begin{bmatrix}\mathbf{Q}_r^\top\widetilde{\mathbf{z}}' \\ \mathbf{Q}_n^\top\widetilde{\mathbf{z}}'\end{bmatrix} = \begin{bmatrix}\mathbf{H}''_{\mathbf{x}} \\ \mathbf{0}\end{bmatrix} \widetilde{\mathbf{x}} + \begin{bmatrix}\mathbf{Q}_r^\top\mathbf{n}' \\ \mathbf{Q}_n^\top\mathbf{n}'\end{bmatrix} \tag{39}$$

By separating (39), and defining $\widetilde{\mathbf{z}}'' = \mathbf{Q}_r^\top\widetilde{\mathbf{z}}'$ as well as $\mathbf{n}'' = \mathbf{Q}_r^\top\mathbf{n}'$, the final expression for the residual is obtained:

$$\mathbf{Q}_r^\top\widetilde{\mathbf{z}}' = \mathbf{H}''_{\mathbf{x}}\widetilde{\mathbf{x}} + \mathbf{Q}_r^\top\mathbf{n}' \quad \implies \quad \boxed{\widetilde{\mathbf{z}}'' = \mathbf{H}''_{\mathbf{x}}\widetilde{\mathbf{x}} + \mathbf{n}''} \tag{40}$$

$$\mathbf{Q}_n^\top\widetilde{\mathbf{z}}' = \mathbf{Q}_n^\top\mathbf{n}' \tag{41}$$

An EKF update can be performed based on the residual $\widetilde{\mathbf{z}}''$ in (40). Since (41) contains only noise, it is ignored and does not contribute to the update. The state correction $\Delta\mathbf{X}$ (to be added to the current state estimate, to produce $\widehat{\mathbf{x}}_{(+)}$), and the

updated covariance $\mathbf{P}_{(+)}$ are given by the standard equations:

$$\Delta\mathbf{X} = \mathbf{K}\tilde{\mathbf{z}}'' \tag{42}$$

$$\hat{\mathbf{x}}_{(+)} = \hat{\mathbf{x}} + \Delta\mathbf{X} \tag{43}$$

$$\mathbf{P}_{(+)} = (\mathbf{I} - \mathbf{K}\mathbf{H}_{\mathbf{x}}'')\mathbf{P}(\mathbf{I} - \mathbf{K}\mathbf{H}_{\mathbf{x}}'')^{\top} + \mathbf{K}\mathbf{R}''\mathbf{K}^{\top} \tag{44}$$

The form of the covariance update (44) is known as Joseph's form and is typically chosen in implementation, although other forms exist (see [25]). The Kalman Gain $\mathbf{K}$ and measurement noise covariance $\mathbf{R}''$ (corresponding to $\mathbf{n}''$) appearing in the above equations are given by:

$$\mathbf{K} = \mathbf{P}{\mathbf{H}_{\mathbf{x}}''}^{\top}\left(\mathbf{H}_{\mathbf{x}}''\mathbf{P}{\mathbf{H}_{\mathbf{x}}''}^{\top} + \mathbf{R}''\right)^{-1} \tag{45}$$

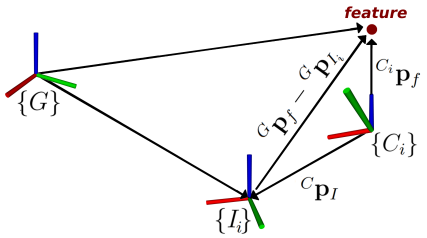
$$\mathbf{R}'' = \mathbf{Q}_r^{\top}\mathbf{R}'\mathbf{Q}_r \tag{46}$$

2.8 Measurement Jacobian

The measurement $\mathbf{H}$ can be derived through the linearization of rotations as discussed in [26]. Recalling the measurement model (19), a measurement from camera i is given by:

$$\mathbf{z}_i = \begin{bmatrix} u_i \\ v_i \end{bmatrix} = \frac{1}{{}^{C_i}Z}\begin{bmatrix} {}^{C_i}X \\ {}^{C_i}Y \end{bmatrix} + \mathbf{n}_i$$

For simplicity, the j subscript is omitted and it is assumed all equations in this section pertain to a single feature. The feature position in camera frame $\{C_i\}$ is a function of the global feature position, the IMU state, and extrinsic parameters according to the geometric relation illustrated below.



$$\longrightarrow \quad {}^{C_i}\mathbf{p}_f = {}^{C}_{I}\mathbf{R}\,{}^{I_i}_{G}\mathbf{R}({}^{G}\mathbf{p}_f - {}^{G}\mathbf{p}_{I_i}) + {}^{C}\mathbf{p}_I \tag{47}$$

Using the chain rule, from (19) and (47), the measurement Jacobian with respect to the ith clone ($i \in \{0, 1, \ldots, M-1\}$) can be computed as:

$$\frac{\partial \mathbf{z}_i}{\partial \widetilde{\mathbf{x}}_i} = \frac{\partial \mathbf{z}_i}{\partial {}^{C_i}\widetilde{\mathbf{p}}_f} \frac{\partial {}^{C_i}\widetilde{\mathbf{p}}_f}{\partial \widetilde{\mathbf{x}}_i} \tag{48}$$

$$\frac{\partial \mathbf{z}_i}{\partial {}^{C_i}\widetilde{\mathbf{p}}_f} = \frac{1}{{}^{C_i}Z} \begin{bmatrix} 1 & 0 & -\frac{{}^{C_i}X}{{}^{C_i}Z} \\ 0 & 1 & -\frac{{}^{C_i}Y}{{}^{C_i}Z} \end{bmatrix} \tag{49}$$

The second part of (48), $\frac{\partial {}^{C_i}\widetilde{\mathbf{p}}_f}{\partial \widetilde{\mathbf{x}}_i}$, can be evaluated through the first order Taylor series approximation of a rotation matrix $\mathbf{R}$, with $\delta\boldsymbol{\theta}$ representing a small angle about its axis (see [26]):

$$\mathbf{R} \approx \left(\mathbf{I}_{3\times 3} - \lfloor \delta\boldsymbol{\theta} \rfloor\right)\widehat{\mathbf{R}} \tag{50}$$

Considering ${}^{C_i}\mathbf{p}_f$ in (47) as a function of state variables and extrinsic parameters (i.e. ${}^{C_i}\mathbf{p}_f = \mathbf{f}({}^{C}_{I}\mathbf{R}, {}^{I_i}_{G}\mathbf{R}, {}^{G}\mathbf{p}_f, {}^{G}\mathbf{p}_{I_i}, {}^{C}\mathbf{p}_I) = {}^{C}_{I}\mathbf{R}\,{}^{I_i}_{G}\mathbf{R}({}^{G}\mathbf{p}_f - {}^{G}\mathbf{p}_{I_i}) + {}^{C}\mathbf{p}_I$), the Jacobian can be derived by evaluating each of the partial derivatives of this function; the partial derivatives with respect to the global IMU position ${}^{G}\mathbf{p}_{I_i}$ and global feature position ${}^{G}\mathbf{p}_f$ are given by:

$$\frac{\partial^{C_i}\widetilde{\mathbf{p}}_f}{\partial^{G}\widetilde{\mathbf{p}}_{I_i}} = -{}^{C}_{I}\mathbf{R}\,{}^{I_i}_{G}\widehat{\mathbf{R}} \tag{51}$$

$$\frac{\partial^{C_i}\widetilde{\mathbf{p}}_f}{\partial^{G}\widetilde{\mathbf{p}}_{f}} = {}^{C}_{I}\mathbf{R}\,{}^{I_i}_{G}\widehat{\mathbf{R}} \tag{52}$$

Employing the linear approximation (50) for the rotation parts, the derivatives with respect to $\delta\boldsymbol{\theta}_{G_i}$ (corresponding to ${}^{I_i}_{G}\mathbf{R}$) and $\delta\boldsymbol{\theta}_{I}$ (corresponding to ${}^{C}_{I}\mathbf{R}$) can be found as follows, using the anticommunitivity property of the cross-product (i.e. $\lfloor\mathbf{v}_1\rfloor\mathbf{v}_2 = -\lfloor\mathbf{v}_2\rfloor\mathbf{v}_1$ for vectors $\mathbf{v}_1$ and $\mathbf{v}_2$).

$$\begin{aligned}
\frac{\partial^{C_i}\widetilde{\mathbf{p}}_f}{\partial\delta\boldsymbol{\theta}_{G_i}} &= \frac{\partial}{\partial\delta\boldsymbol{\theta}_{G_i}}\Big[{}^{C}_{I}\mathbf{R}\,{}^{I_i}_{G}\widehat{\mathbf{R}}\big({}^{G}\widehat{\mathbf{p}}_f - {}^{G}\widehat{\mathbf{p}}_{I_i}\big) + {}^{C}\mathbf{p}_I\Big] \\
&= \frac{\partial}{\partial\delta\boldsymbol{\theta}_{G_i}}\Big[{}^{C}_{I}\mathbf{R}\big(\mathbf{I} - \lfloor\delta\boldsymbol{\theta}_{G_i}\rfloor\big){}^{I_i}_{G}\widehat{\mathbf{R}}\big({}^{G}\widehat{\mathbf{p}}_f - {}^{G}\widehat{\mathbf{p}}_{I_i}\big) + {}^{C}\mathbf{p}_I\Big] \\
&= \frac{\partial}{\partial\delta\boldsymbol{\theta}_{G_i}}\Big[- {}^{C}_{I}\mathbf{R}\lfloor\delta\boldsymbol{\theta}_{G_i}\rfloor{}^{I_i}_{G}\widehat{\mathbf{R}}\big({}^{G}\widehat{\mathbf{p}}_f - {}^{G}\widehat{\mathbf{p}}_{I_i}\big) + \underbrace{\ldots}_{\text{no dependance on } \boldsymbol{\delta\theta}_{G_i}}\Big] \\
&= \frac{\partial}{\partial\delta\boldsymbol{\theta}_{G_i}}\Big[{}^{C}_{I}\mathbf{R}\lfloor{}^{I_i}_{G}\widehat{\mathbf{R}}\big({}^{G}\widehat{\mathbf{p}}_f - {}^{G}\widehat{\mathbf{p}}_{I_i}\big)\rfloor\delta\boldsymbol{\theta}_{G_i}\Big] \\
&= {}^{C}_{I}\mathbf{R}\lfloor{}^{I_i}_{G}\widehat{\mathbf{R}}\big({}^{G}\widehat{\mathbf{p}}_f - {}^{G}\widehat{\mathbf{p}}_{I_i}\big)\rfloor
\end{aligned} \tag{53}$$

The Jacobians $\mathbf{H}_{\mathbf{x}_i}$ and $\mathbf{H}_{f_i}$ corresponding to the ith measurement of a single feature (omitting the j subscript, which appear in (34) can be constructed by horizontally stacking the results of (51) - (53):

$$\mathbf{H}_{\mathbf{x}_i} = \frac{\partial \mathbf{z}_i}{\partial {}^{C_i}\widetilde{\mathbf{p}}_f} \begin{bmatrix} \frac{\partial {}^{C_i}\widetilde{\mathbf{p}}_f}{\partial {}^{C_i}\widetilde{\mathbf{p}}_I} & \frac{\partial {}^{C_i}\widetilde{\mathbf{p}}_f}{\partial \delta\boldsymbol{\theta}_{G_i}} & \frac{\partial {}^{C_i}\widetilde{\mathbf{p}}_f}{\partial {}^{G}\widetilde{\mathbf{v}}_I} & \frac{\partial {}^{C_i}\widetilde{\mathbf{p}}_f}{\partial \widetilde{\mathbf{b}}_\omega} & \frac{\partial {}^{C_i}\widetilde{\mathbf{p}}_f}{\partial \widetilde{\mathbf{b}}_a} \end{bmatrix}$$

$$= \frac{1}{{}^{C_i}Z} \begin{bmatrix} 1 & 0 & -\frac{{}^{C_i}X}{{}^{C_i}Z} \\ 0 & 1 & -\frac{{}^{C_i}Y}{{}^{C_i}Z} \end{bmatrix} \left[\begin{array}{c|c|c|c|c} -{}^{C}_{I}\mathbf{R}\,{}^{I_i}_{G}\widehat{\mathbf{R}} & {}^{C}_{I}\mathbf{R}\lfloor {}^{I_i}_{G}\widehat{\mathbf{R}}\left({}^{G}\widehat{\mathbf{p}}_f - {}^{G}\widehat{\mathbf{p}}_{I_i}\right)\rfloor & \mathbf{0}_{3\times 3} & \mathbf{0}_{3\times 3} & \mathbf{0}_{3\times 3} \end{array}\right] \tag{54}$$

$$\mathbf{H}_{\mathbf{f}_i} = \frac{\partial \mathbf{z}_i}{\partial {}^{C_i}\widetilde{\mathbf{p}}_f} \frac{\partial {}^{C_i}\widetilde{\mathbf{p}}_f}{\partial {}^{G}\widetilde{\mathbf{p}}_f}$$

$$= \frac{1}{{}^{C_i}Z} \begin{bmatrix} 1 & 0 & -\frac{{}^{C_i}X}{{}^{C_i}Z} \\ 0 & 1 & -\frac{{}^{C_i}Y}{{}^{C_i}Z} \end{bmatrix} \left[-{}^{C}_{I}\mathbf{R}\,{}^{I_i}_{G}\widehat{\mathbf{R}} \right] \tag{55}$$

As mentioned previously, within the standard MSCKF framework, the extrinsic parameters ${}^{C}_{I}\mathbf{R}$ and ${}^{C}\mathbf{p}_I$ are considered known from offline calibration and are not included in the state vector, so the derivatives of ${}^{C_i}\mathbf{p}_f$ with respect to these parameters are not part of the measurement Jacobian. However, in our multi-camera extension of MSCKF (Chapter 3), these parameters are estimated online along with the navigation state. Accordingly, we include these calibration Jacobians here; the partial derivatives of ${}^{C_i}\mathbf{p}_f$ (the feature position in the ith camera frame within the clone state, $i \in \{0, 1, \ldots, M-1\}$) with respect to the calibration parameters ${}^{C}_{I}\mathbf{R}$ and ${}^{C}\mathbf{p}_I$ are given as follows (where $\delta\boldsymbol{\theta}_I$ is the error corresponding to ${}^{C}_{I}\mathbf{R}$):

$$\frac{\partial {}^{C_i}\widetilde{\mathbf{p}}_f}{\partial {}^{C}\widetilde{\mathbf{p}}_I} = \frac{\partial}{\partial {}^{C}\mathbf{p}_I} \left[{}^{C}_{I}\mathbf{R}\,{}^{I_i}_{G}\widehat{\mathbf{R}}\left({}^{G}\widehat{\mathbf{p}}_f - {}^{G}\widehat{\mathbf{p}}_{I_i}\right) + {}^{C}\mathbf{p}_I \right] = \mathbf{I}_{3\times 3} \tag{56}$$

$$
\begin{aligned}
\frac{\partial {}^{C_i}\widetilde{\mathbf{p}}_f}{\partial \delta \boldsymbol{\theta}_I} &= \frac{\partial}{\partial \delta \boldsymbol{\theta}_I}\Big[{}^{C}_{I}\mathbf{R}^{I_i}_{G}\widehat{\mathbf{R}}\big({}^{G}\widehat{\mathbf{p}}_f - {}^{G}\widehat{\mathbf{p}}_{I_i}\big) + {}^{C}\mathbf{p}_I \Big] \\
&= \frac{\partial}{\partial \delta \boldsymbol{\theta}_I}\Big[\big(\mathbf{I} - \lfloor \delta\boldsymbol{\theta}_I \rfloor\big) {}^{C}_{I}\mathbf{R}^{I_i}_{G}\widehat{\mathbf{R}}\big({}^{G}\widehat{\mathbf{p}}_f - {}^{G}\widehat{\mathbf{p}}_{I_i}\big) + {}^{C}\mathbf{p}_I \Big] \\
&= \frac{\partial}{\partial \delta \boldsymbol{\theta}_I}\Big[- \lfloor \delta\boldsymbol{\theta}_I \rfloor {}^{C}_{I}\mathbf{R}^{I_i}_{G}\widehat{\mathbf{R}}\big({}^{G}\widehat{\mathbf{p}}_f - {}^{G}\widehat{\mathbf{p}}_{I_i}\big) + \underbrace{\dots}_{\text{no dependance on } \boldsymbol{\delta\theta}_I} \Big] \\
&= \frac{\partial}{\partial \delta \boldsymbol{\theta}_I}\Big[\lfloor {}^{C}_{I}\mathbf{R}^{I_i}_{G}\widehat{\mathbf{R}}\big({}^{G}\widehat{\mathbf{p}}_f - {}^{G}\widehat{\mathbf{p}}_{I_i}\big) \rfloor \delta\boldsymbol{\theta}_I \Big] \\
&= \lfloor {}^{C}_{I}\mathbf{R}^{I_i}_{G}\widehat{\mathbf{R}}\big({}^{G}\widehat{\mathbf{p}}_f - {}^{G}\widehat{\mathbf{p}}_{I_i}\big) \rfloor \qquad (57)
\end{aligned}
$$

Chapter 3

MULTI-CAMERA VISUAL-INERTIAL ODOMETRY

As mentioned in Section 1.2, single view (monocular or stereo) VINS is vulnerable to failure when the view falls on textureless regions of the environment. Having multiple non-overlapping camera views recording similtanneously offers robustness to this situation as it is very unlikely (in most applications) that *all* views will contain textureless regions at the same time. Figure 7 shows an example of the loss of feature tracking (the details of the feature tracking system are provided in this section) in one out of three camera views running on a multi-camera sensor platform in an indoor environment. Similarly to textureless regions, feature tracking issues arise in camera views containing moving objects, very far away feature points, and in other situations that produce regions of "bad" visual measurements (reflections, poor lighting, etc.). While the gain of information provided by multiple camera views can lead to better localization accuracy, one of the main challenges is to reduce the computational burden introduced by the added number of feature points. The multi-camera VINS design presented here was proposed in our recent work [18] and the key aspects of it are reviewed in this section, beginning with the visual feature tracking front-end.

A vision processing system exists as a front-end to the MSCKF estimator in order to solve two different problems. The first is to *detect* the set of feature points in a given image to be used for triangulation (feature detection). The second, after feature points are initialized, is to track each feature point in the image from frame to frame (feature tracking). An overview of these steps is given in the following sections.

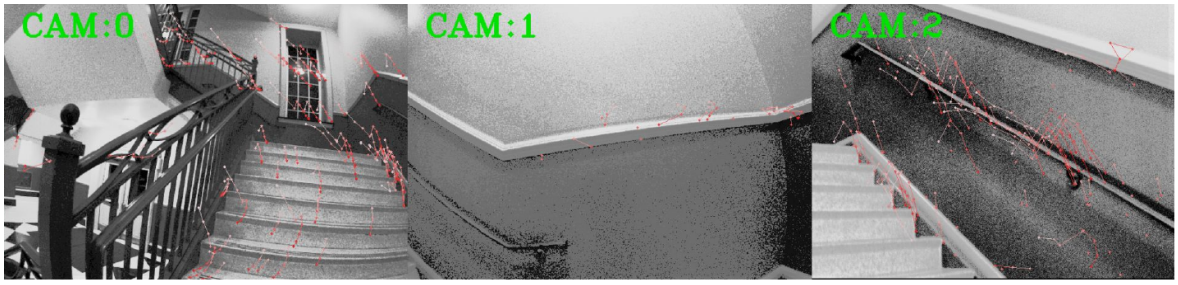


Figure 7: Textureless regions: pictured are three camera views at the same (or nearly the same, since the cameras are asynchronous) time instance. The red/white line segments indicate the KLT feature tracks up to the frame shown (the red end of a segment is the feature location at the current frame). As the view from Cam 1 (center) is mostly a blank wall, there is a visible lack of feature tracks compared to the views from Cam 0 and Cam 2 due to failure of both feature initialization and tracking. The high-contrast effect visible in all three images is due to histogram equalization, a pre-processing step that tends to improve feature detection.

3.1 Feature Detection

Good candidates for feature points are points that have a clear and definite position in the image, such that they can be tracked across image frames. These points generally correspond to local maxima or minima in image intensity, or points at which the image gradient is large along multiple directions. In terms of the physical environment observed by the camera, feature points frequently occur on the corners of objects. More generally, they occur in places of discontinuity of depth or surface orientation, changes in surface texture, illumination, or reflectance. Many different methods exist for detecting feature points in an image. One classic method is the Harris corner detector [27] which assigns a "cornerness" value to every pixel in the image based on local image gradients. The best corner points are then selected by enforcing a minimum required cornerness value and discarding all other points. Another older but widely implemented method is Smith's "smallest univalue segment assimilating nucleus" (SUSAN) [28], in which no image gradients are considered; instead, each candidate feature point is only compared to pixels in a circular area surrounding it. The candidate (nucleus) is considered a feature only if sufficiently *few* pixels in the circular area have the same (or nearly the same, within a defined tolerance) value as

the nucleus. While either method can be used to detect feature points in an image, it is usually too computationally expensive for real-time processing at the frame rates of interest (which is typically around 20-30 Hz). A more recent development, the "features from accelerated segment test" (FAST) feature detector, originally proposed in [29], has proven efficient enough to perform well at these frame rates, even on computationally limited platforms. Figure 8 describes the details of the FAST detector and its internal parameters that effect the quality and number of features detected. To illustrate the effect of varying the threshold parameter within FAST, a visual comparison of different threshold values is given in Figure 9. The FAST feature detector has also been shown capable of further speed improvements through machine learning [30], though this is not used in our system.

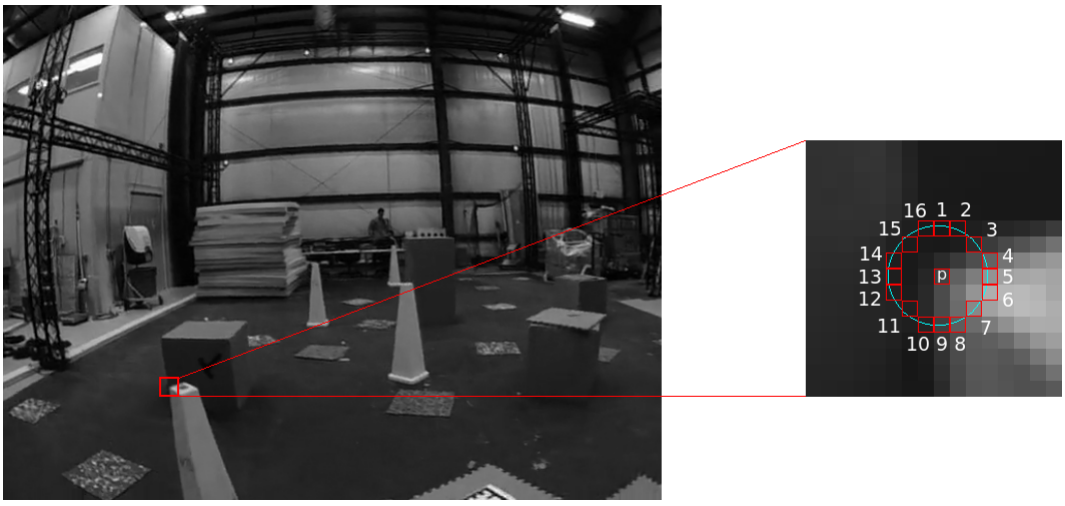


Figure 8: FAST feature detector: This image was taken from one of our own visual-inertial datasets. A given candidate pixel p is compared to all pixels intersecting a circle of radius 3 pixels centered on p. The candidate p is considered a feature if and only if N contiguous pixels of the 16 lying on the circle are all brighter than p by some threshold t or are all darker than p by the same threshold. The values N and t are ultimately design choices, but typically $N = 12$ is chosen while t is adjusted to achieve a desired number of features in the image (A larger threshold corresponds to fewer accepted feature points). A comparison of different t values is illustrated in Figure 9

Figure 9: FAST features detected at various threshold values. Detected features are shown as blue circles. Original image (left), high threshold (middle), low threshold (right). Although color images are shown, only the greyscale (average) pixel values are considered.

3.2 Feature Tracking

After feature points are initially detected, their location in the image (pixel coordinate) must be tracked as new frames are received. Otherwise, it would not be known whether different measurements correspond to the same point in space. Two fundamentally different approaches to feature tracking can be utilized: descriptor matching or optical flow. In our MSCKF implementation, these methods can be interchanged without modification to the rest of the system.

Commonly used algorithms such as SIFT [31], SURF [32], BREIF [33], and ORB [34] contain both a feature detector and a feature descriptor component. A descriptor serves to uniquely identify a feature point based on a small area of pixels surrounding it. With a descriptor value (typically represented as a vector) attached to each feature point, correspondences can be found across image frames (regardless of orientation or scale changes, ideally), by comparing descriptors. The definition of the descriptor can vary greatly and a tradeoff exists between computational efficiency and matching robustness. For example, SIFT computes a 128-dimensional descriptor vector based on a 16×16 window around the feature point while SURF computes a 64-dimensional vector based on a 20×20 window. The SIFT descriptor has proven more robust but slower than both SURF and ORB (see [35]). Of course, the appropriate choice among these depends on the particular application and the available computing resources.

After descriptors are assigned to feature points, matching can either be done by brute-force or with the aid of geometric constraints or other heuristics.

An alternative method to descriptor matching, and the method considered in the performance evaluation in Chapter 4, is to compute the optical flow, i.e. the 2D displacement vector, of each feature point from frame to frame of the image stream. Many solutions to this problem exist, Baker et. al [36] provide a survey and evaluation methods for existing techniques— however, it is noted that these evaluations are based on dense optical flow (computed at *every* pixel in the image), whereas only sparse optical flow (computed only at feature points) applies to our estimator. The Kande-Lucas Transform (KLT), originally proposed in [37], is one the most successful optical flow solutions, and is the one utilized by our system. The basic working principle of KLT is the assumption neighboring pixel values share practically the same motion from frame to frame. This constrains the problem enough to allow a least-squares solution of the optical flow vector at every point of interest (i.e. at every feature point). More specifically, optical flow is governed by the following relation, based on the "brightness constancy" assumption, which asserts that the image of a moving point in the environment will have the same brightness from one frame to the next:

$$I_x v_x + I_y v_y = -I_t \tag{58}$$

where I_x, I_y, and I_t are the two spacial derivatives and the time derivative of the image intensity, respectively, and v_x and v_y are the components of the optical flow vector. Since (58) is a scalar equation with two unknowns, there is no unique solution $[v_x \quad v_y]^\top$ at a single pixel. To resolve this, KLT assumes the optical flow will be approximately the same for all pixels in a window around the point of interest (in this case, a feature point). Assuming there are n pixels in this window, denoted $p_1 \dots p_n$, the governing equation extends to a system of n equations:

$$\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ \vdots & \vdots \\ I_x(p_n) & I_y(p_n) \end{bmatrix} \begin{bmatrix} v_x \\ v_y \end{bmatrix} = - \begin{bmatrix} I_t(p_1) \\ I_t(p_2) \\ \vdots \\ I_t(p_n) \end{bmatrix} \tag{59}$$

Then, it can be shown that the least-squares solution to the system (59) is given by:

$$\begin{bmatrix} v_x \\ v_y \end{bmatrix} = \begin{bmatrix} \sum_i I_x^2(p_i) & \sum_i I_x I_y(p_i) \\ \sum_i I_x I_y(p_i) & \sum_i I_y^2(p_i) \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i I_x I_t(p_i) \\ -\sum_i I_y I_t(p_i) \end{bmatrix} \tag{60}$$

As mentioned, in the context of the MSCKF estimator (in both the monocular and multi-camera cases), any given feature point is tracked until it is lost (e.g. by moving out of frame or becoming occluded), or it reaches a pre-set maximum track length. When a feature reaches one of these conditions, all observations of the feature are utilized to apply a state update.

3.3 Efficient Processing of Multi-Camera Measurements

The naive approach to extending the standard MSCKF (Chapter 2) to the multi-camera case is to simply retain IMU clones at all imaging times. However, unless all cameras are hardware-synchronized (it is assumed they are *not*), this would result in an exceedingly complicated state vector with a very large number of variables—and consequently, a large computational burden. To allow for asynchronous cameras, we only retain the IMU clones corresponding to the imaging times of a freely chosen "base camera" in the state vector. Then, any IMU poses at an intermediate imaging time (an imaging time of a non-base camera), is represented through linear interpolation of the clones at the two bounding base-camera times. This pose interpolation, as described in the following section, is similarly used in the alternating stereo-VINS algorithm of

[38], in which images from stereo cameras are captured in an alternating manner rather than simultaneously.

Consider a set of IMU poses existing in the clone state, each with a timestamp corresponding to an imaging time reported by the base camera. If an image is received from a non-base camera at an intermediate time t_k, we can define an interpolation factor λ_k equal to the fraction of the time interval between the two bounding base camera times that has elapsed. Denoting the bounding times t_1 and t_2:

$$\lambda_k = \frac{t_k - t_1}{t_2 - t_1} \tag{61}$$

However, there is a caveat with (61) in that it's only accurate if the hardware latency (in this case, the delay between the instant of image capture and the time at which the estimator assigns a time stamp) of the two cameras involved is identical —or equivalently, if all cameras are considered to have the same clock. In practice, there is an unknown time offset that exists not only between cameras, but also between any camera and the IMU, which can be detrimental if neglected. To compensate for this misalignment, we estimate all time offsets online, along with the navigation state, as is done in [39]. Figure 10 illustrates the time offset problem between the IMU and base camera, as well as the notation used to distinguish different clocks.

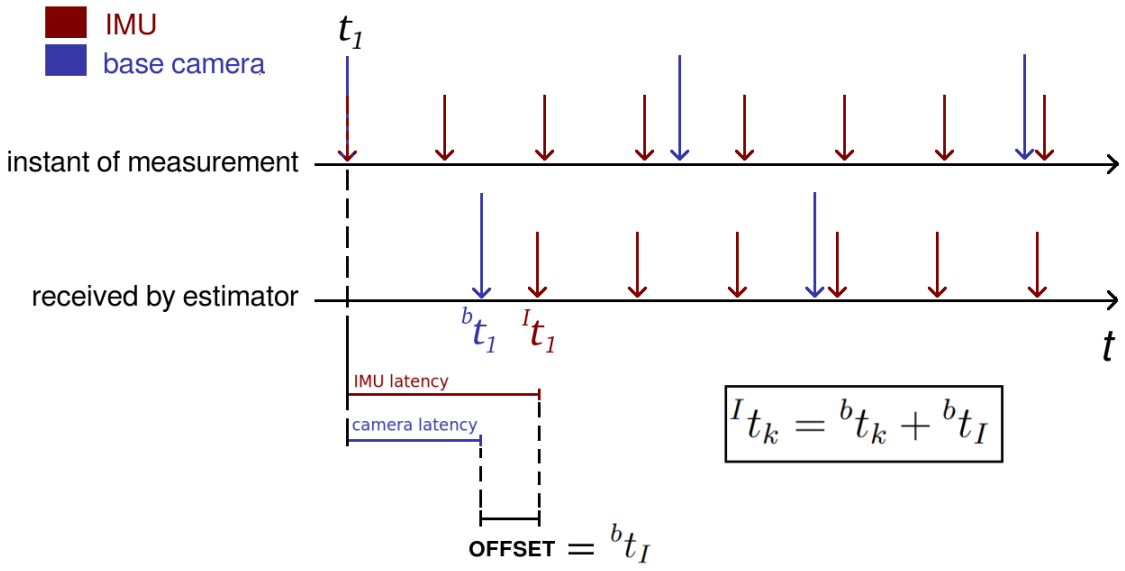


Figure 10: Time offset between IMU and base camera due to differences in hardware latency: An IMU measurement and a camera measurement occurred at the same instant t_1. This instant is represented by time ${}^{b}t_1$ in the base camera clock and by time ${}^{I}t_1$ in the IMU clock. The difference in these times (the time offset) can be expressed in either clock. The equation shown relates a general time instance in the IMU clock ${}^{I}t_k$ to that in the base camera clock through offset ${}^{b}t_I$.

Consider a multi-camera sensor platform containing L asynchronous cameras labeled $0, 1, \ldots, L-1$, where the 0-th camera is assumed to serve as the base camera. The correction illustrated in Figure 10 can be applied to each camera with respect to the base camera; specifically, a time instance expressed in the base camera's clock, ${}^{b}t_k$, relates to the *same* time instance expressed in the l-th camera's clock, ${}^{l}t_k$, according to time-offset ${}^{l}t_b$:

$$ {}^{b}t_k = {}^{l}t_k + {}^{l}t_b \tag{62} $$

and the interpolation factor (61) is adjusted accordingly:

$$ \lambda_k = \frac{{}^{l}t_k + {}^{l}t_b - {}^{b}t_1}{{}^{b}t_2 - {}^{b}t_1} \tag{63} $$

The information that is lost through this interpolation is any deviation from a straight line path that the sensor platform traversed between the two bounding time steps (see Figure 11 below) and analogously, any deviation from a constant angular rate. However, in a typical application with the sensor platform traveling on the order of 1 m/s and images captured at 20-30 Hz, we argue this is usually a good approximation.

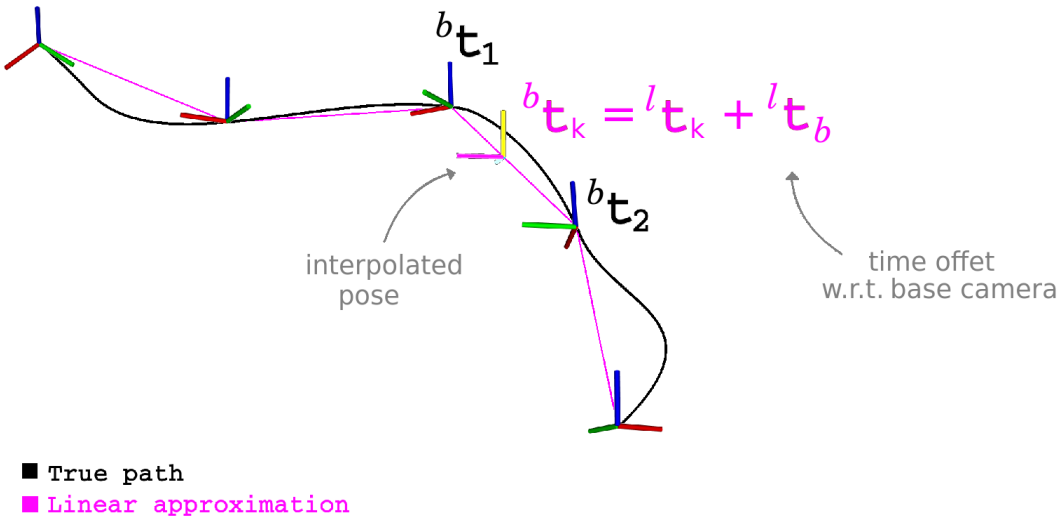


Figure 11: Pose interpolation: As an example, 5 IMU clones with time stamps corresponding to the base-camera's imaging times exist in the clone state, with the bounding times denoted ${}^bt_1$ and ${}^bt_2$. If the estimator receives an intermediate measurement from camera l (any of the non-base cameras) with arbitrary time stamp ${}^lt_k$, this time is converted to the base camera's clock according to the estimated time offset as in (62), then used to compute the interpolation factor (63). With the interpolation factor known, the approximate intermediate IMU pose can be computed from the bounding poses.

After the interpolation factor λ_k is obtained, computing the position of the IMU at the intermediate time is given by:

$$ {}^G\mathbf{p}_{I({}^bt_k)} = (1-\lambda_k){}^G\mathbf{p}_{I({}^bt_1)} + \lambda_k {}^G\mathbf{p}_{I({}^bt_2)} \tag{64} $$

To determine the rotation of the IMU at the intermediate time, the same interpolation factor λ_k can be applied to an exponential map (see [40]) using the rotations of

the IMU at the bounding times. As mentioned, this is equivalent to assuming constant angular velocity of the IMU between time steps, and can be written as:

$$ {}^{I({}^{b}t_k)}_{G}\mathbf{R} = \text{Exp}\left(\lambda_k \text{Log}\left({}^{I({}^{b}t_2)}_{G}\mathbf{R}\,{}^{G}_{I({}^{b}t_1)}\mathbf{R}\right)\right){}^{I({}^{b}t_1)}_{G}\mathbf{R} \tag{65} $$

3.4 Multi-Camera State Vector and Propagation

Recall that the time offset existing between the base camera and each other camera (as well as the IMU) on the sensor platform is assumed to be *unknown*. Therefore, we estimate the time offset associated with each camera along with the navigation state. Additionally, we estimate the spacial calibration parameters (extrinsic parameters) with respect to the IMU, for each camera. Accordingly, we must modify the state vector used in the monocular-case MSCKF (Section 2.2) to account for these new variables. The multi-camera state vector is specified as follows:

$$ \mathbf{x} = \begin{bmatrix} \mathbf{x}_I \\ \mathbf{x}_M \\ \mathbf{x}_L \end{bmatrix} \quad , \quad \mathbf{x}_I = \begin{bmatrix} {}^{I}_{G}\bar{q} \\ {}^{G}\mathbf{p}_I \\ {}^{G}\mathbf{v}_I \\ \mathbf{b}_\omega \\ \mathbf{b}_a \end{bmatrix} \quad , \quad \mathbf{x}_M = \begin{bmatrix} \begin{bmatrix} {}^{I}_{G}\bar{q} \\ {}^{G}\mathbf{p}_I \end{bmatrix}_{t_k} \\ \vdots \\ \begin{bmatrix} {}^{I}_{G}\bar{q} \\ {}^{G}\mathbf{p}_I \end{bmatrix}_{t_{k-M+1}} \end{bmatrix} \quad , \quad \mathbf{x}_L = \begin{bmatrix} \begin{bmatrix} {}^{C_0}_{I}\bar{q} \\ {}^{C_0}\mathbf{p}_I \\ {}^{0}t_b \end{bmatrix} \\ \vdots \\ \begin{bmatrix} {}^{C_{L-1}}_{I}\bar{q} \\ {}^{C_{L-1}}\mathbf{p}_I \\ {}^{L-1}t_b \end{bmatrix} \end{bmatrix} \tag{66} $$

where the IMU state $\mathbf{x}_I$ is identical to the monocular-case, but the clone state $\mathbf{x}_M$ now corresponds specifically to the *base* camera only. The calibration state $\mathbf{x}_L$ contains the time offset from the base camera ${}^{l}t_b$ and the relative transformation to the IMU frame $[{}^{C_l}_{I}\bar{q}^\top \quad {}^{C_l}\mathbf{p}_I^\top]$ for each of the L cameras onboard, where $l \in \{0, 1, \ldots, L-1\}$. We will assume $l = 0$ is chosen as the base camera, so ${}^{0}t_b = {}^{b}t_b = 0$. Thus, in place of ${}^{0}t_b$,

we instead store the base camera's time offset with respect to the IMU clock, ${}^{b}t_I$.

It is noted that in our recent work [18], the *intrinsic* parameters of each camera are similarly included in the state vector, but here we will assume the intrinsics are known.

Given the time offset existing between the IMU and the base camera, the propagation of the IMU measuremets (as discussed in Section 2.3) must now be done in terms of the *estimated* base-camera imaging times expressed in the IMU clock. Specifically, upon receiving the $(k+1)$-th image, the IMU is propagated up to time ${}^{I}\widehat{t}_{k+1} = {}^{b}t_{k+1} + {}^{b}\widehat{t}_I$, where ${}^{b}\widehat{t}_I$ is the current estimate of the IMU time offset. Correspondingly, the state transition matrix $\mathbf{\Phi}_k$ and process noise covariance $\mathbf{Q}_k$ (see Section 2.3) are computed across the time interval $[{}^{I}\hat{t}_k, {}^{I}\hat{t}_{k+1}]$. Then, the partitioned covariance matrix after propagation is given by:

$$\mathbf{P}_{k+1|k} = \begin{bmatrix} \mathbf{\Phi}_k \mathbf{P}_{II,k|k} \mathbf{\Phi}_k^\top + \mathbf{Q}_k & \mathbf{\Phi}_k \mathbf{P}_{IS,k|k} \\ \mathbf{P}_{SI,k|k} \mathbf{\Phi}_k^\top & \mathbf{P}_{SS,k|k} \end{bmatrix} \tag{67}$$

This is analogous to the monocular-case (16), except the subscript S is now introduced to include *all* state variables that are not part of the IMU navigation state (i.e. the clone state and the calibration state containing all spacial parameters and time offsets).

3.5 Multi-Camera State Augmentation

The monocular-case state augmentation step (Section 2.4) must be modified to account for the fact that times in the IMU clock are treated as estimated. It is necessary to augment the state with IMU poses corresponding to the *true* imaging time of the $(k+1)$-th image (as expressed in the IMU clock). Following the methodology of [39], we can express the IMU pose at the true imaging time as a function of the IMU pose at the estimated time ${}^{I}\widehat{t}_{k+1}$, and the error in the time offset ${}^{b}\widetilde{t}_I$:

$$
{}^{G}\mathbf{p}_{I({}^{I}t_{k+1})} \approx {}^{G}\mathbf{p}_{I({}^{I}\widehat{t}_{k+1})} + {}^{G}\mathbf{v}_{I({}^{I}\widehat{t}_{k+1})}\,{}^{b}\widetilde{t}_{I} \tag{68}
$$

$$
{}^{I({}^{I}t_{k+1})}_{G}\mathbf{R} \approx \mathrm{Exp}\left(-\boldsymbol{\omega}\,{}^{b}\widetilde{t}_{I}\right) {}^{I({}^{I}\widehat{t}_{k+1})}_{G}\mathbf{R} \tag{69}
$$

As IMU poses given by (68) and (69) are appended to the state, the covariance of this augmented state must be computed as well. This is done exactly as in (18) which is repeated here for clarity:

$$
\mathbf{P}_{\text{AUG}} = \boldsymbol{\Psi}\mathbf{P}\boldsymbol{\Psi}^{\top} \quad , \quad \boldsymbol{\Psi} = \begin{bmatrix} \mathbf{I} \\ \frac{\partial \widetilde{\mathbf{y}}}{\partial \widetilde{\mathbf{x}}} \end{bmatrix} = \begin{bmatrix} \mathbf{I} \\ \frac{\partial \mathbf{g}(\widehat{\mathbf{x}} \boxplus \widetilde{\mathbf{x}}) \boxminus \mathbf{g}(\widehat{\mathbf{x}})}{\partial \widetilde{\mathbf{x}}} \end{bmatrix}
$$

The difference now, is that the Jacobians of (68) and (69) with respect to the IMU pose at the estimated time ${}^{I}\widehat{t}_{k+1}$, and the error in the time offset ${}^{b}\widetilde{t}_{I}$ are needed to compute $\boldsymbol{\Psi}$. These Jacobians are given by:

$$
\frac{\partial {}^{G}\widetilde{\mathbf{p}}_{I({}^{I}t_{k+1})}}{\partial {}^{G}\widetilde{\mathbf{p}}_{I({}^{I}\widehat{t}_{k+1})}} = \mathbf{I}_{3\times 3} \quad , \quad \frac{\partial {}^{G}\widetilde{\mathbf{p}}_{I({}^{I}t_{k+1})}}{\partial {}^{b}\widetilde{t}_{I}} = {}^{G}\widehat{\mathbf{v}}_{I({}^{I}\widehat{t}_{k+1})} \tag{70}
$$

$$
\frac{\partial {}^{I({}^{I}t_{k+1})}\delta\boldsymbol{\theta}_{G}}{\partial {}^{I({}^{I}\widehat{t}_{k+1})}\delta\boldsymbol{\theta}_{G}} = \mathbf{I}_{3\times 3} \quad , \quad \frac{\partial {}^{I({}^{I}t_{k+1})}\delta\boldsymbol{\theta}_{G}}{\partial {}^{b}\widetilde{t}_{I}} = \boldsymbol{\omega}_{m}({}^{I}\widehat{t}_{k+1}) - \widehat{\mathbf{b}}_{w}({}^{I}\widehat{t}_{k+1}) \tag{71}
$$

After IMU propagation, the processing of visual measurements in order to produce an MSCKF update proceeds analogously to the monocular-case (Chapter 2). However, modifications to the measurement function (47) are necessary to account for and utilize the time offsets and spacial calibration parameters contained in the calibration state $\mathbf{x}_{L}$. The details of this measurement model discussed in the following section.

3.6 Multi-Camera Measurement Model and Update

Using the previously described time offset relation ${}^{b}t_k = {}^{l}t_k + {}^{l}t_b$ (Section 3.3), and the calibration parameters contained in the multi-camera state vector (66), we can express the position of a feature point in the l-th camera's frame ($l \in \{0, 1, \ldots, L-1\}$) at imaging time ${}^{b}t_k$ (time step k in the base-camera's clock) according to:

$$ {}^{C_l}\left({}^{l}t_k + {}^{l}t_b\right)\mathbf{p}_f = {}^{C_l}_{I}\mathbf{R}\,{}^{I({}^{l}t_k + {}^{l}t_b)}_{G}\mathbf{R}\left({}^{G}\mathbf{p}_f - {}^{G}\mathbf{p}_{I({}^{l}t_k + {}^{l}t_b)}\right) + {}^{C_l}\mathbf{p}_I \tag{72} $$

Recall that we *do not* store clones corresponding to imaging times of any camera except the base camera. Instead, the poses at these intermediate times are approximated through linear interpolation of the poses at the bounding base camera times ${}^{b}t_1$ and ${}^{b}t_2$, with interpolation factor λ_k. For clarity, the results from Section 3.3 defining this interpolation are repeated here:

$$ \begin{aligned} {}^{G}\mathbf{p}_{I({}^{b}t_k)} &= (1-\lambda_k)\,{}^{G}\mathbf{p}_{I({}^{b}t_1)} + \lambda_k\,{}^{G}\mathbf{p}_{I({}^{b}t_2)} \\ {}^{I({}^{b}t_k)}_{G}\mathbf{R} &= \mathrm{Exp}\left(\lambda_k \mathrm{Log}\left({}^{I({}^{b}t_2)}_{G}\mathbf{R}\,{}^{G}_{I({}^{b}t_1)}\mathbf{R}\right)\right){}^{I({}^{b}t_1)}_{G}\mathbf{R} \\ \lambda_k &= \frac{{}^{l}t_k + {}^{l}t_b - {}^{b}t_1}{{}^{b}t_2 - {}^{b}t_1} \end{aligned} $$

By substituting the above approximations into the measurement function (72), we can fully express each feature measurement as a function of variables contained in the state vector. With this representation, additional Jacobians to those specified in Section 2.8 are required in order to compute the full measurement Jacobian to be used in an MSCKF update. Namely, we need to express the Jacobians of (72) with respect to the bounding IMU poses at ${}^{b}t_1$ and ${}^{b}t_2$, as well as the time offset corresponding to camera l. These Jacobians are computed using the chain rule and are given in (73) - (85) below. Since these expressions are very dense wth notation, an interpretation of each part of (73) is provided, and subsequent expressions can be understood similarly.

$$\underbrace{\frac{\partial^{C_l({}^l t_k+{}^l t_b)}\widetilde{\mathbf{p}}_f}{\partial^{I({}^b t_1)}\delta\boldsymbol{\theta}_G}}_{\substack{\text{feature position error in camera } l \text{ frame at the true imaging time w.r.t IMU rotation error at the lower bounding time}}} = \underbrace{\frac{\partial^{C_l({}^l t_k+{}^l t_b)}\widetilde{\mathbf{p}}_f}{\partial^{I({}^l t_k+{}^l t_b)}\delta\boldsymbol{\theta}_G}}_{\substack{\text{feature position error in camera } l \text{ frame at the true imaging time w.r.t IMU rotation error at the true imaging time (see (53))}}} \underbrace{\frac{\partial^{I({}^l t_k+{}^l t_b)}\delta\boldsymbol{\theta}_G}{\partial^{I({}^b t_1)}\delta\boldsymbol{\theta}_G}}_{\substack{\text{IMU rotation error at the true imaging time w.r.t. IMU rotation error at the lower bounding time}}} \tag{73}$$

The first multiplicative term on the right side of (73), and the equations below, is essentially the same as the monocular case, except it requires the transformation specifically from the IMU to camera l, rather than to the single available camera. The right multiplicative term is evaluated through the interpolation expressions (63) - (65)—of these, the ones with respect to the IMU position are given by (78) - (80).

$$\frac{\partial^{C_l({}^l t_k+{}^l t_b)}\widetilde{\mathbf{p}}_f}{\partial^{I({}^b t_2)}\delta\boldsymbol{\theta}_G} = \frac{\partial^{C_l({}^l t_k+{}^l t_b)}\widetilde{\mathbf{p}}_f}{\partial^{I({}^l t_k+{}^l t_b)}\delta\boldsymbol{\theta}_G}\frac{\partial^{I({}^l t_k+{}^l t_b)}\delta\boldsymbol{\theta}_G}{\partial^{I({}^b t_2)}\delta\boldsymbol{\theta}_G} \tag{74}$$

$$\frac{\partial^{C_l({}^l t_k+{}^l t_b)}\widetilde{\mathbf{p}}_f}{\partial^{G({}^b t_1)}\widetilde{\mathbf{p}}_I} = \frac{\partial^{C_l({}^l t_k+{}^l t_b)}\widetilde{\mathbf{p}}_f}{\partial^{G({}^l t_k+{}^l t_b)}\widetilde{\mathbf{p}}_I}\frac{\partial^{G({}^l t_k+{}^l t_b)}\widetilde{\mathbf{p}}_I}{\partial^{G({}^b t_1)}\widetilde{\mathbf{p}}_I} \tag{75}$$

$$\frac{\partial^{C_l({}^l t_k+{}^l t_b)}\widetilde{\mathbf{p}}_f}{\partial^{G({}^b t_2)}\widetilde{\mathbf{p}}_I} = \frac{\partial^{C_l({}^l t_k+{}^l t_b)}\widetilde{\mathbf{p}}_f}{\partial^{G({}^l t_k+{}^l t_b)}\widetilde{\mathbf{p}}_I}\frac{\partial^{G({}^l t_k+{}^l t_b)}\widetilde{\mathbf{p}}_I}{\partial^{G({}^b t_2)}\widetilde{\mathbf{p}}_I} \tag{76}$$

$$\frac{\partial^{C_l({}^l t_k+{}^l t_b)}\widetilde{\mathbf{p}}_f}{\partial^{l}\widetilde{t}_b} = \frac{\partial^{C_l({}^l t_k+{}^l t_b)}\widetilde{\mathbf{p}}_f}{\partial^{I({}^l t_k+{}^l t_b)}\delta\boldsymbol{\theta}_G}\frac{\partial^{I({}^l t_k+{}^l t_b)}\delta\boldsymbol{\theta}_G}{\partial^{l}\widetilde{t}_b} + \frac{\partial^{C_l({}^l t_k+{}^l t_b)}\widetilde{\mathbf{p}}_f}{\partial^{G({}^l t_k+{}^l t_b)}\widetilde{\mathbf{p}}_I}\frac{\partial^{G({}^l t_k+{}^l t_b)}\widetilde{\mathbf{p}}_I}{\partial^{l}\widetilde{t}_b} \tag{77}$$

$$\frac{\partial^{G({}^{l}t_k+{}^{l}t_b)}\widetilde{\mathbf{p}}_I}{\partial^{G({}^{b}t_1)}\widetilde{\mathbf{p}}_I} = (1-\lambda_k)\mathbf{I} \tag{78}$$

$$\frac{\partial^{G({}^{l}t_k+{}^{l}t_b)}\widetilde{\mathbf{p}}_I}{\partial^{G({}^{b}t_2)}\widetilde{\mathbf{p}}_I} = \lambda_k\mathbf{I} \tag{79}$$

$$\frac{\partial^{G({}^{l}t_k+{}^{l}t_b)}\widetilde{\mathbf{p}}_I}{\partial^{l}\widetilde{t}_b} = \frac{1}{{}^{b}t_2 - {}^{b}t_1}({}^{G}\mathbf{p}_{I({}^{}t_2)} - {}^{G}\mathbf{p}_{I({}^{}t_1)}) \tag{80}$$

We can compute the Jacobians with respect the IMU rotation through the following approximation using small angle $\boldsymbol{\psi}$ (see [41]):

$$\begin{aligned}\mathrm{Exp}\left(\boldsymbol{\theta}+\boldsymbol{\psi}\right) &\approx \mathrm{Exp}\left(\mathbf{J}_l\left(\boldsymbol{\theta}\right)\boldsymbol{\psi}\right)\mathrm{Exp}\left(\boldsymbol{\theta}\right) &(81)\\ &\approx \mathrm{Exp}\left(\boldsymbol{\theta}\right)\mathrm{Exp}\left(\mathbf{J}_r\left(\boldsymbol{\theta}\right)\boldsymbol{\psi}\right) &(82)\end{aligned}$$

Where $\mathbf{J}_l$ and $\mathbf{J}_r$ are the left and right Jacobians of SO(3), respectively (see [42]). The full derivations of the following three Jacobians can be found in our technical report [43], but the final result for each is given below:

$$\frac{\partial^{I({}^{l}t_k+{}^{l}t_b)}\delta\boldsymbol{\theta}_G}{\partial^{I({}^{b}t_1)}\delta\boldsymbol{\theta}_G} = -\left(\lambda_k\mathbf{J}_l(\lambda_{k}{}^{2}_{1}\boldsymbol{\theta})\mathbf{J}_r^{-1}({}^{2}_{1}\boldsymbol{\theta}) - \mathrm{Exp}(\lambda_{k}{}^{2}_{1}\boldsymbol{\theta})\right) \tag{83}$$

$$\frac{\partial^{I({}^{l}t_k+{}^{l}t_b)}\delta\boldsymbol{\theta}_G}{\partial^{I({}^{b}t_2)}\delta\boldsymbol{\theta}_G} = \lambda_k\mathbf{J}_l(\lambda_{k}{}^{2}_{1}\boldsymbol{\theta})\mathbf{J}_l^{-1}({}^{2}_{1}\boldsymbol{\theta}) \tag{84}$$

$$\frac{\partial^{I({}^{l}t_k+{}^{l}t_b)}\delta\boldsymbol{\theta}_G}{\partial^{l}\widetilde{t}_b} = -\frac{1}{{}^{b}t_2 - {}^{b}t_1}{}^{2}_{1}\boldsymbol{\theta} \tag{85}$$

By horizontally stacking these results (analogously to Section 2.8) we can obtain the full measurement Jacobian corresponding to the multi-camera state vector (66). The same conditions which trigger an MSCKF update hold in the multi-camera case as they did in the monocular case (Section 2.7), namely, a feature either moves out of view, is lost due to tracking failure, or reaches its maximum track length. However, if a feature from a *non-base* camera has met one of these conditions, the feature is

triangulated based on interpolating the IMU poses that exist in the clone state as described in Section 3.3. From there, the MSCKF marginilization process (Section 2.7) remains the same, except it is likely there are significantly more features to add to the *stacked* Jacobian (37) during each update. Specifically, assuming all L independent cameras contain end-of-life features at the current time step, we can further expand the stacked Jacobian such that it contains block-elements corresponding to each camera ($l \in \{0, 1, \ldots, L-1\}$):

$$\widetilde{\mathbf{z}}' = \mathbf{H}'_{\mathbf{x}}\widetilde{\mathbf{x}} + \mathbf{n}' \Longrightarrow \begin{bmatrix} {}^{(0)}\widetilde{\mathbf{z}}' \\ {}^{(1)}\widetilde{\mathbf{z}}' \\ \vdots \\ {}^{(L-1)}\widetilde{\mathbf{z}}' \end{bmatrix} = \begin{bmatrix} {}^{(0)}\mathbf{H}'_{\mathbf{x}} \\ {}^{(1)}\mathbf{H}'_{\mathbf{x}} \\ \vdots \\ {}^{(L-1)}\mathbf{H}'_{\mathbf{x}} \end{bmatrix} \widetilde{\mathbf{x}} + \begin{bmatrix} {}^{(0)}\widetilde{\mathbf{n}}' \\ {}^{(1)}\widetilde{\mathbf{n}}' \\ \vdots \\ {}^{(L-1)}\widetilde{\mathbf{n}}' \end{bmatrix} \tag{86}$$

With this modification, computing the Kalman Gain and correction to the state proceeds identically as in Section 2.7.

3.7 Experimental Results

Figures 12 and 13 contain experimental results of the proposed multi-camera MSCKF estimator. Both figures show the estimated position of the IMU over a long indoor trajectory that begins and ends in the same location. Since no ground truth data was available, the start-end error (the difference between initial and final position estimates) is used as a performance metric. To validate the improvement over monocular MSCKF, the same trajectory is estimated using only the base camera. Details of the sensor platforms used to record data in these experiments are also given in each figure. Since we only consider the localization accuracy here, sensor data was processed *offline* on a desktop computer.

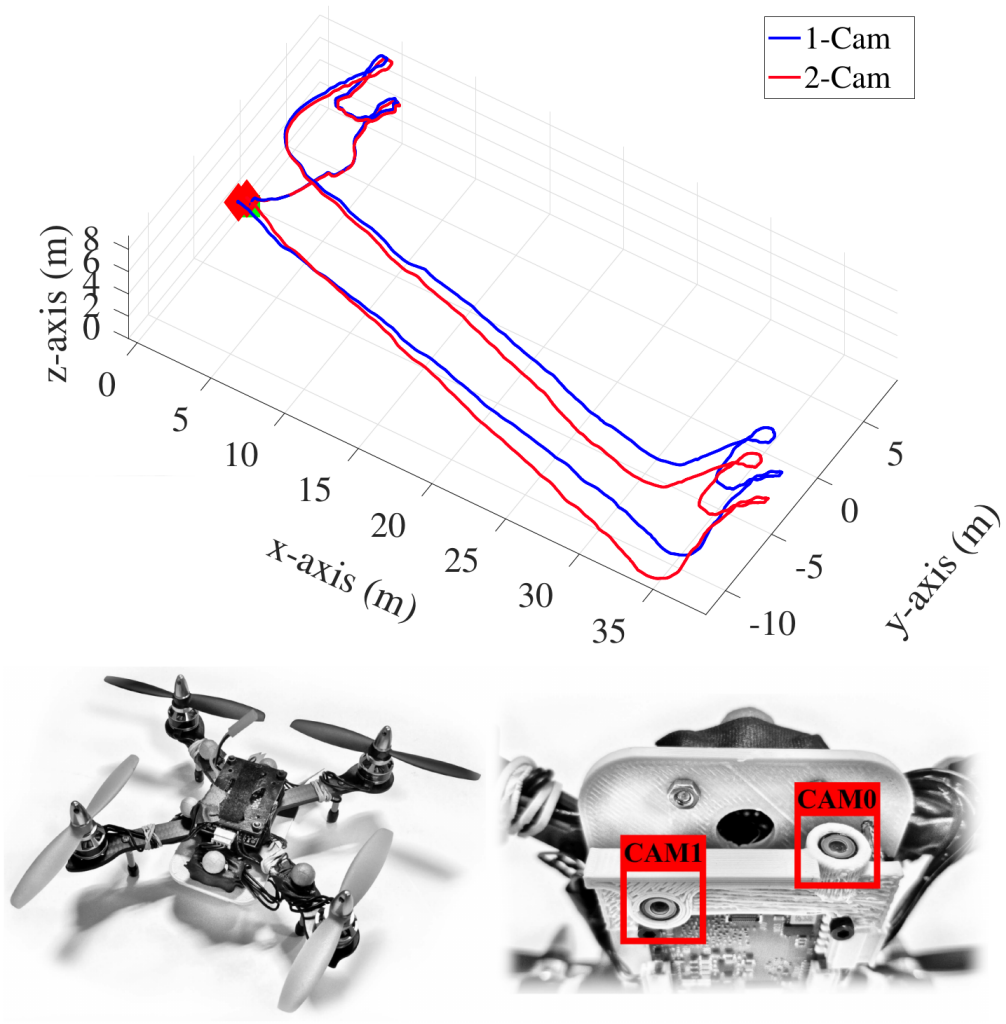


Figure 12: 2-camera sensor platform: This 143m trajectory covers the first and third floors of the University of Delaware's Spencer Lab. Data was recorded on the Snapdragon Flight attached to a quadrotor MAV as shown above. The vehicle was fitted with a custom camera mount to enable a binocular (two non-overlapping views) configuration with "CAM0" facing forward and "CAM1" facing downward. The start-end error in the 2-camera case is approximately 0.45m (0.33% of the total path), compared to 0.65m (0.48% of the total path) in the monocular case.

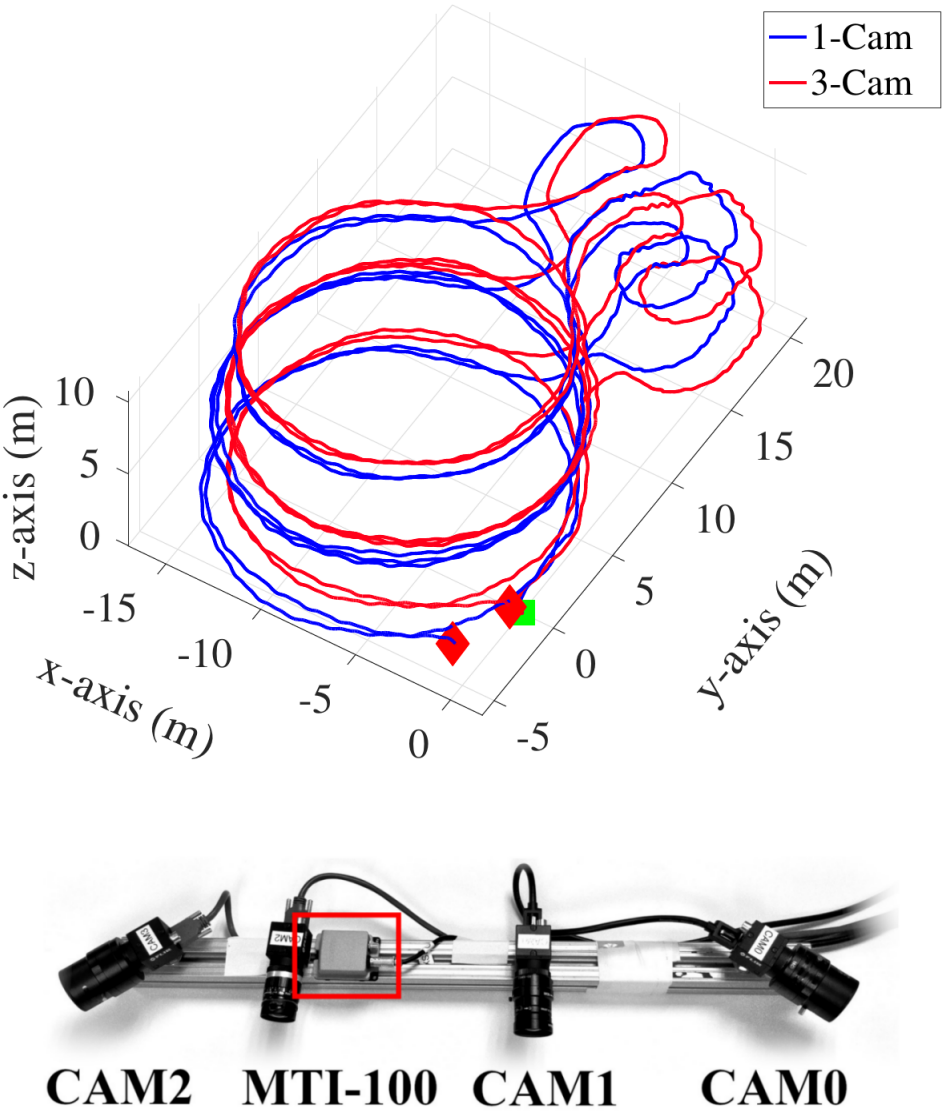


Figure 13: 3-camera sensor platform: This 440m trajectory covers 3 floors of the University of Delaware's Gore Hall. Data was recorded on custom sensor platform shown in the photo (only 3 of the 4 cameras were used). The cameras used are Point Grey global shutter cameras and the IMU is an XSENS MTI-100. The start-end error in the 3-camera case is approximately 0.61m (0.14% of the total path), compared to 3.58m (0.80% of the total path) in the monocular case.

Although not evaluated in this work, the proposed multi-camera estimator was further extended to apply to both *rolling shutter* cameras and sensor platforms with multiple *IMUs*. We designed a custom 3D printed sensor platform with 6 rolling shutter cameras (3 stereo pairs) and 2 IMUs to evaluate this system as pictured below in Figure 14. The multi-IMU multi-camera (MiMc) system is described in detail and evaluated in the recent work [44].

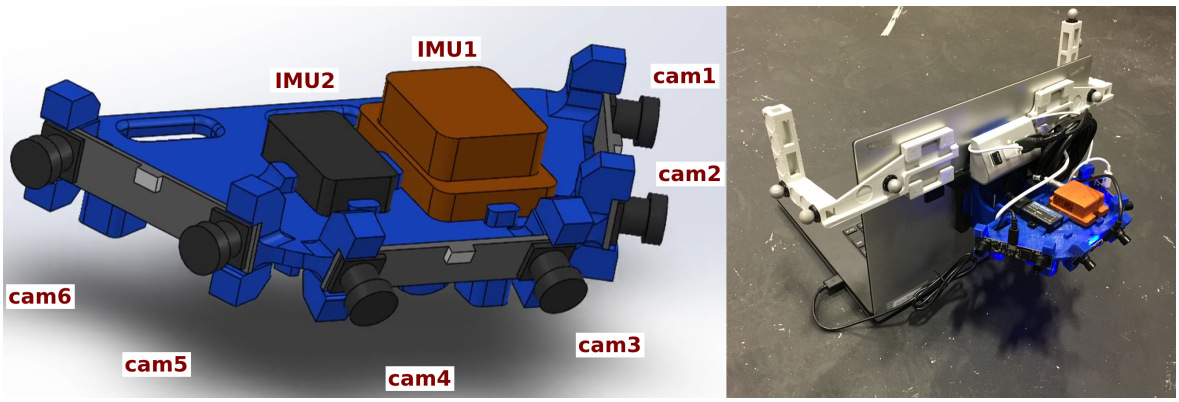


Figure 14: Custom multi-IMU multi-camera sensor platform: as shown in the model (left), 3 identical stereo (rolling-shutter) camera pairs cover a viewing angle of over 180° in total. Two IMU models ("IMU1": XSENS MT-100 and "IMU2": Microstrain 3DM-GX-25) are attached in the center. On the right is a photo showing the platform attached to the back of a laptop with Vicon motion capture markers for recording data against ground truth position and orientation.

Chapter 4

PERFORMANCE EVALUATION

The MSCKF framework as described in Chapter 2 leaves available many design choices (free parameters) in implementation which have significant impact on the performance of the VINS system. Two of the most important of these parameters are the number of stochastic clones to retain in the state vector (clone size) and the number of visual feature points to track (feature size). Additionally, in the context of our multi-camera algorithm described in Chapter 3, the number of independent camera streams to process is also considered as a design parameter. In general, increasing the value of any of these parameters results in better localization accuracy, but at the expense of greater computational cost. Therefore, selecting the "best" values for the free parameters depends strongly on the available computing platform. While this tradeoff between accuracy and computational cost is understood, little work has been done to quantitatively evaluate the effect of these parameters on MSCKF performance. To this end, we evaluate our MSCKF implementation by varying the free parameters and recording the results in two different experiments. The first experiment evaluates the effect of these parameters on localization accuracy (according to a specified error metric), without considering the computational time required. The second evaluates the effect of these parameters on the computational speed (compared to real-time) on a set of different computing platforms, without considering the accuracy. These results can be jointly used, for example, to find the set of parameters which allow real-time processing on a platform of interest, and then determining whether any part of this set also meets the accuracy requirements for the intended application. The details of these experiments are given in Sections 4.3 and 4.4.

4.1 System Implementation

Our MSCKF implementation is primarily written in C++ and is based on ROS [45] for communication with sensor data. OpenCV is heavily utilized for the vision front-end of the system, including feature detection and tracking. Most of the software used in these experiments has been open-sourced as part of the OpenVINS project [24] (early version) and is available at https://github.com/rpng/open_vins.

4.2 Evaluation Study

The three free parameters considered here are the clone size, feature size, and number of independent camera streams to process. This section provides some insight on how each of these parameters affect the performance of the estimator.

Since MSCKF updates are performed on a per-feature basis, the number of feature points to maintain in feature tracking has a significant impact on the computational speed. As each feature must be triangulated when it reaches the end of its life (as described in Section 2.6), the number of feature points set to be maintained directly impacts the number of times triangulation must be performed on every update. The optical flow for every tracked feature point must also be solved according to (59) on every new image frame. Lastly, the feature size directly affects complexity of updates, as all features that have reached their end of life at a given time step are stacked onto the system (35). Figure 15 illustrates the effect of feature size on computational speed for our monocular-case MSCKF implementation applied to one of ETH's EuRoC MAV public datasets [46].

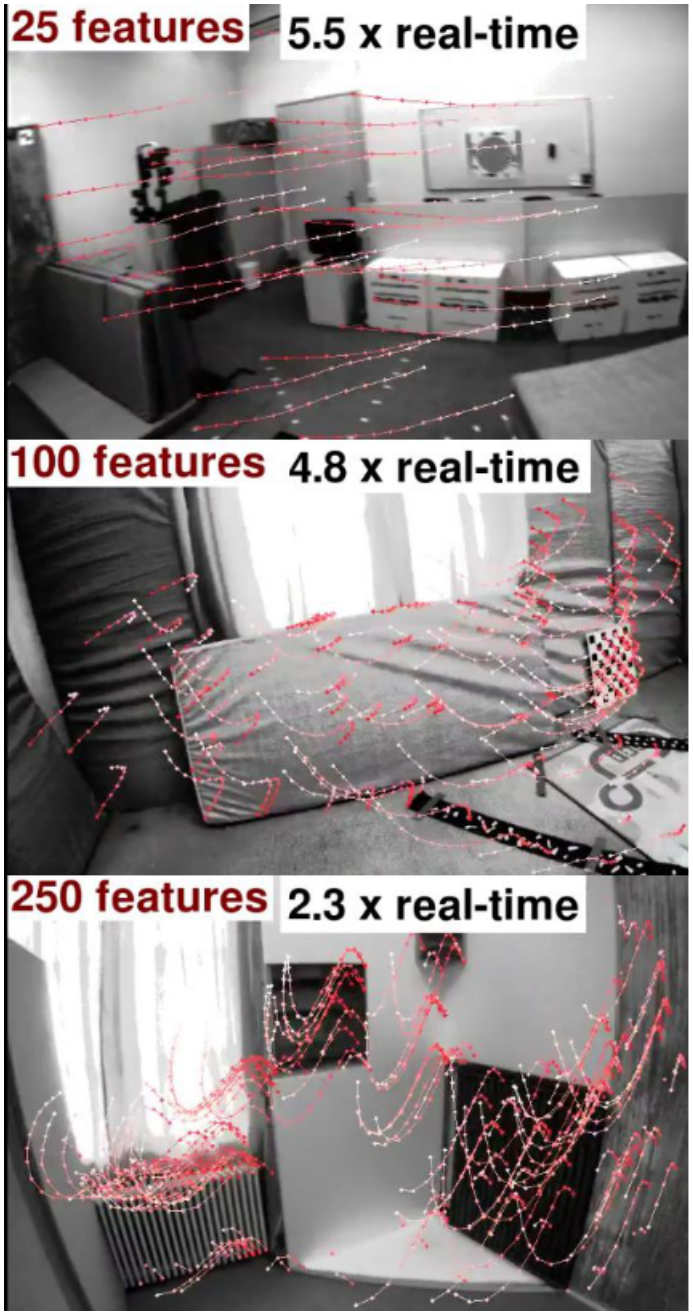


Figure 15: Comparison of feature sizes for monocular MSCKF. On each feature track, the white endpoint indicates the feature location on the current frame, and the red endpoint indicates where it was initialized— all other points correspond to intermediate frames. The dataset shown is one of ETH's EuRoC MAV datasets (Vicon Room 1 "medium"). Labeled on each image is the number of features tracked and the speed at which the system runs (i.e. the speed at which new pose estimates can be produced relative to real-time) on a Desktop computer with an Intel-i7 processor and 32 GB of RAM. The camera frame rate is 25Hz and has resolution 752×480. Although this is a more powerful computing platform than would be used in a typical VINS application, it illustrates the effect of feature size on relative computation time.

The clone size is equal to the maximum number of measurements allowed for a given feature point (however, a feature could be lost before it reaches this maximum). Having more measurements increases the number of equations in the system (27), and thus the accuracy of the least-squares solution (28), at the expense of inverting a higher-dimensional matrix. Similarly to the feature size, a larger clone size also increases the dimension of the system (35) used to compute the update.

The number of independent camera streams from which to obtain measurements is the last free parameter considered; more cameras add more information sources to the system and thus can improve localization accuracy, as well as add robustness to poor visual data as discussed in Chapter 3. While the pose interpolation scheme described in Section 3.3 prevents the need to maintain a set of clones for *each* camera, having multiple cameras still of course has an impact on computational complexity. This is mainly because the number of cameras used essentially acts as a multiplier on the number of features tracked; therefore, the feature size specified in these evaluations is considered to be *per-camera* and equal for all cameras.

4.3 Runtime Evaluation

For any practical VINS application, it is essential that pose estimates can be estimated in real time. Accordingly, it is of interest to determine the values of the free parameters (e.g. the clone size and feature size) which allow the system to operate at or above real-time on a particular computing platform. Figure 16 shows all platforms that were considered in this evaluation.

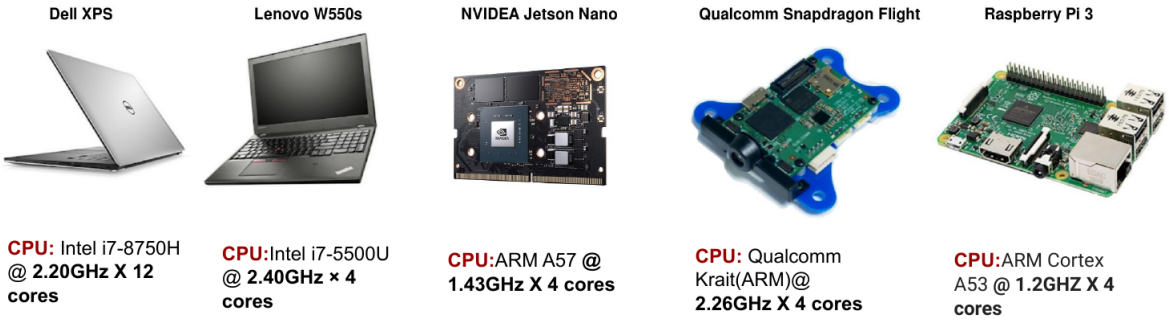


Figure 16: All computing platforms on which our VINS system was evaluated, with computational power roughly decreasing from left to right. The SoC platforms are all small enough to be viable as an on-board MAV computer. CPU information is listed under each platform.

The plots in Figures 17 - 21 were generated by running our monocular-case MSCKF estimator on one of ETH's EuRoC MAV datasets and recording the total time required to estimate the entire trajectory for a particular combination of clone size and feature size. This time was then divided by the actual duration of the trajectory and denoted the real-time ratio (a real-time ratio of 1 indicates exactly real-time). Care was taken to ensure the runtime recorded was an an accurate representation of the estimator alone; time required for loading data from file into memory and initializing the system in ROS were rejected, as these times would not be present in the "real" scenario of online data processing. Such corrections were most significant on the embedded (SoC) computing platforms. Every data point shown in Figures 17 - 22 reflects the *average* runtime over a set of repetitions of the entire trajectory. Evidently, some of the lower-end platforms could not achieve real-time operation for any combination of these parameters, while the higher-end platforms could operate well beyond real-time.

It is noted that two different ranges of the feature sizes and clone sizes evaluated are shown in the figures below; specifically, a range of 100-500 features with 5-25 clones is computed for the laptop platforms, while a range of 25-100 features with 2-10 clones is computed for the SoC platforms. Additionally, the number of runs that were averaged to obtain a final result was also reduced; however, since the runtimes for a

particular parameter configuration generally remained consistent within 1 or 2 seconds, we argue that this has little impact on the result. The reduced experiment size for the SoC platforms became necessary due to the impractical amount of time required to compute the trajectory multiple times for larger feature and clone sizes. For example, the results shown for the Rasperry Pi in Figure 21 (a total of 120 runs of the dataset) required roughly 16 hours to compute.

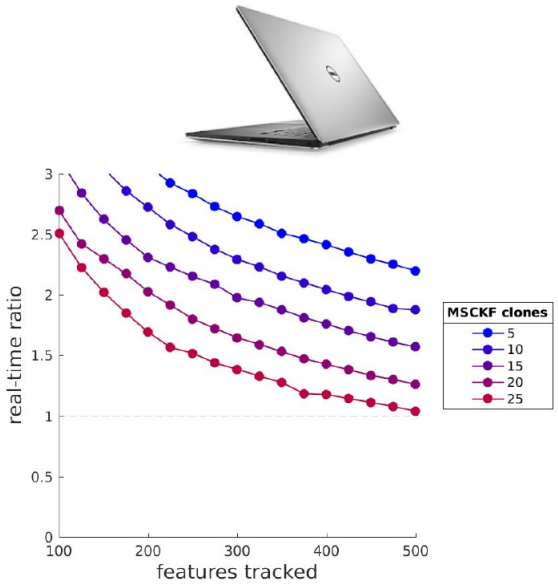


Figure 17: EuRoC MAV dataset on Dell XPS laptop. Each data point represents an average of 10 runs.

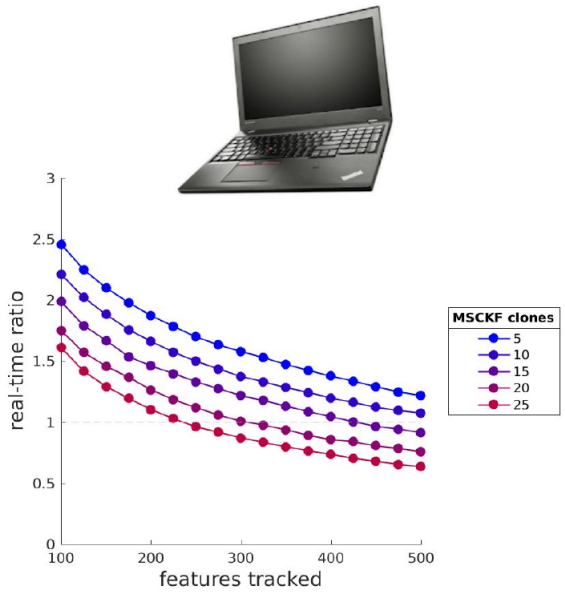


Figure 18: EuRoC MAV dataset on Lenovo W550s laptop. Each data point represents an average of 10 runs.

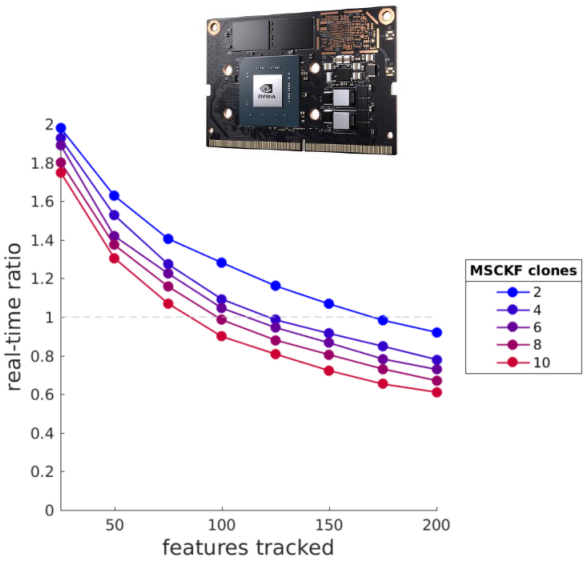


Figure 19: EuRoC MAV dataset on NVidea Jetson Nano SoC. Each data point represents an average of 5 runs.

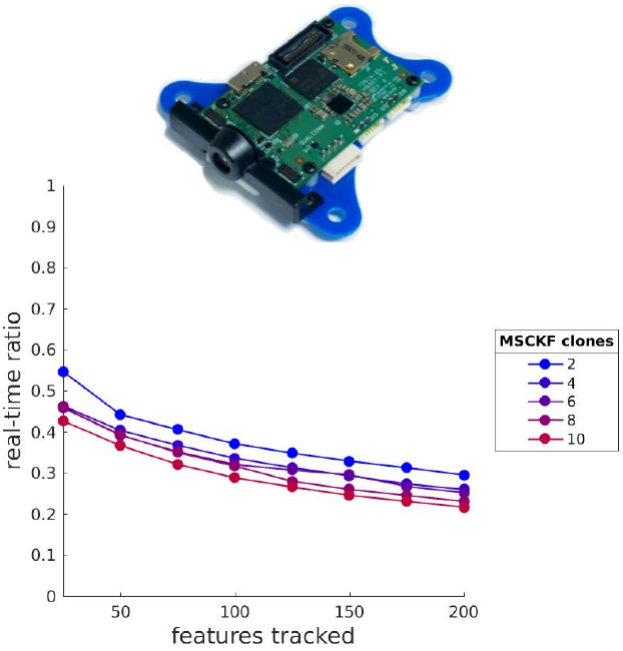


Figure 20: EuRoC MAV dataset on Qualcomm Snapdragon Flight SoC (based on Snapdragon 801 processor). Each data point represents an average of 3 runs.

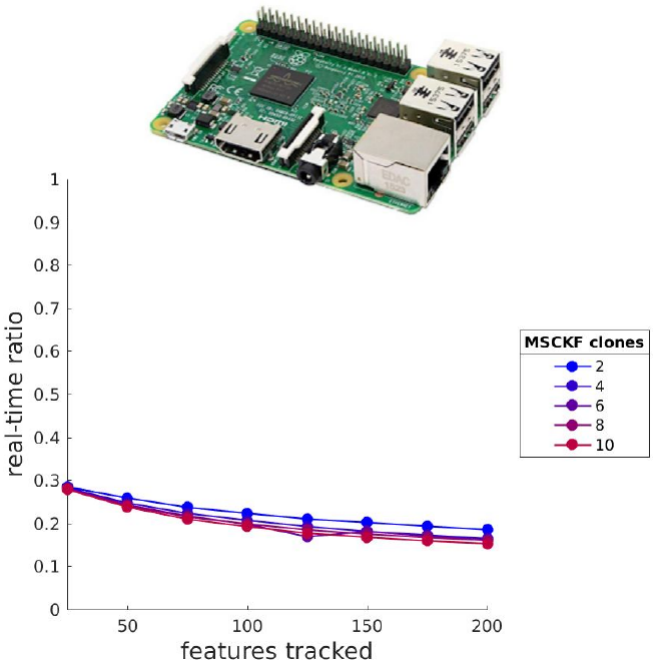


Figure 21: EuRoC MAV dataset on Raspberry Pi 3 SoC. Each data point represents an average of 3 runs.

As expected, the larger computing platforms (Figures 17 and 18) can perform well above real time for nearly all parameter configurations considered; however, since our primary interest is in achieving accurate localization in real time on MAV platforms, the results for the SoC platforms are of greater interest. Evidently the Snapdragon Flight (Figure 20) and Raspberry Pi (Figure 21) are unable to track their pose estimate in real time through our MSCKF implementation, even using the lightest parameter settings. This is expected for the low-grade Raspberry Pi, but is likely due to a lack of utilization of all computing resources onboard the Snapdragon Flight (e.g. the GPU and DSP— specifications can be found in [47]). By contrast, the NVidea Jetson Nano is able to track the IMU pose in real time (or above) through the current implementation for any of the parameter combinations at or above the dotted line in Figure 19.

Lastly, Figure 22 below shows a different runtime evaluation pertaining specifically to the multi-camera estimator described in Chapter 3. This plot was generated by running our multi-camera estimator on one of our University of Delaware Gore Hall datasets (the dataset shown in Figure 7 and Figure 13). Here, a *fixed* clone size is evaluated for a set of feature sizes and the number of cameras used, in order to illustrate the computational cost of adding more cameras.

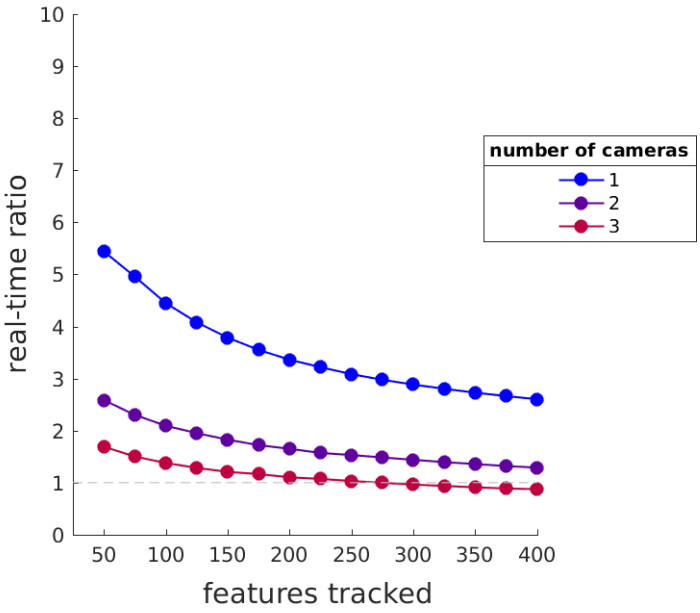


Figure 22: University of Delaware Gore Hall dataset (316s, 443m closed loop trajectory with images of size 648×488 at 20 Hz and IMU measurements at 400 Hz). Each data point represents the average result over 10 runs. The computing platform used in this plot not shown in Figure 16, but it is a desktop with an intel-i7 CPU comparable to the laptop platforms shown.

While these results give insight into the sets of free MSCKF parameters that are computationally viable on some common platforms, they do not indicate the corresponding localization accuracy; this evaluation is provided in the next section.

4.4 Accuracy Evaluation

On any computing platform, running the estimator multiple times using the *same* input sensor data, and the *same* design parameters will still give varying results in terms of the localization accuracy. This is due to an inherent randomness in the feature tracking implementation; the tracking consistency is based on Random Sample Consensus (RANSAC) [48], an outlier rejection method which involves taking random subsets of the data to generate a model. RANSAC can be applied to essentially any model fitting problem, and in this context it is applied to computing the Fundamental

Matrix (see [49]) from a set of image correspondences—this is then used to reject outlying feature tracks. To capture this variability, the results reported in this accuracy evaluation are the average result of many runs of the same dataset.

We measured the localization error in these experiments in two ways, both of which only consider the position of the IMU frame rather than both the position and orientation; we argue this is still a very good indicator of overall estimator performance. When ground truth position data is available (which it is in the EuRoC MAV datasets), the position error reported for a single run is itself an average; specifically, it is the difference between the estimated position and the ground truth position, evaluated at at every time step and then averaged over the entire trajectory— this is the error metric on the vertical axis of Figures 23 and 24. When ground truth position data is *not* available (as in Figure 25), we consider the start-end error, i.e. the final distance from the starting position as reported by the estimator (in this case, the sensor platform executed a closed-loop trajectory, starting and ending in the same location).

In most cases, when running these evaluations, we found that an average final result over 10 runs showed reasonable convergence. However, to ensure a conclusive result, every data point shown in Figures 23 and 24 represent an average over *100 runs*. Both of these trajectories are part of the EuRoC MAV datasets and the evaluations were done using our monocular MSCKF implementation.

A video showing how the plots 23 and 24 change with the increasing number of runs, and many of the other results from this chapter, is available at `https://www.youtube.com/watch?v=FuWLygVV9zM`.

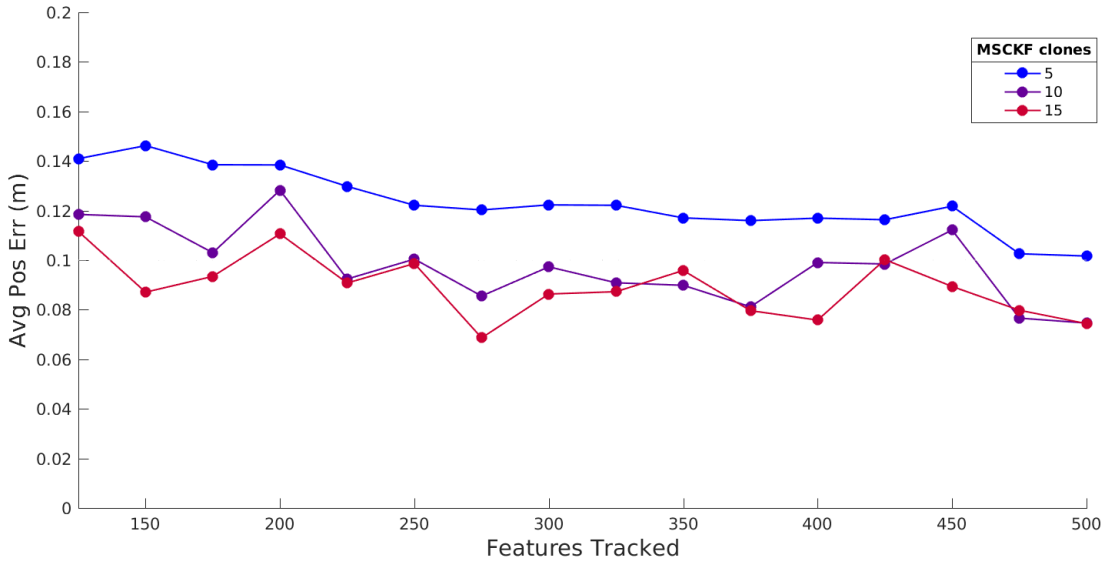


Figure 23: EuRoC MAV dataset Vicon Room 1 "medium" (88s trajectory with images of size 752×480 at 20 Hz and IMU measurements at 200 Hz): average position error (monocular MSCKF) relative to ground truth data, with each data point representing an average over 100 runs.

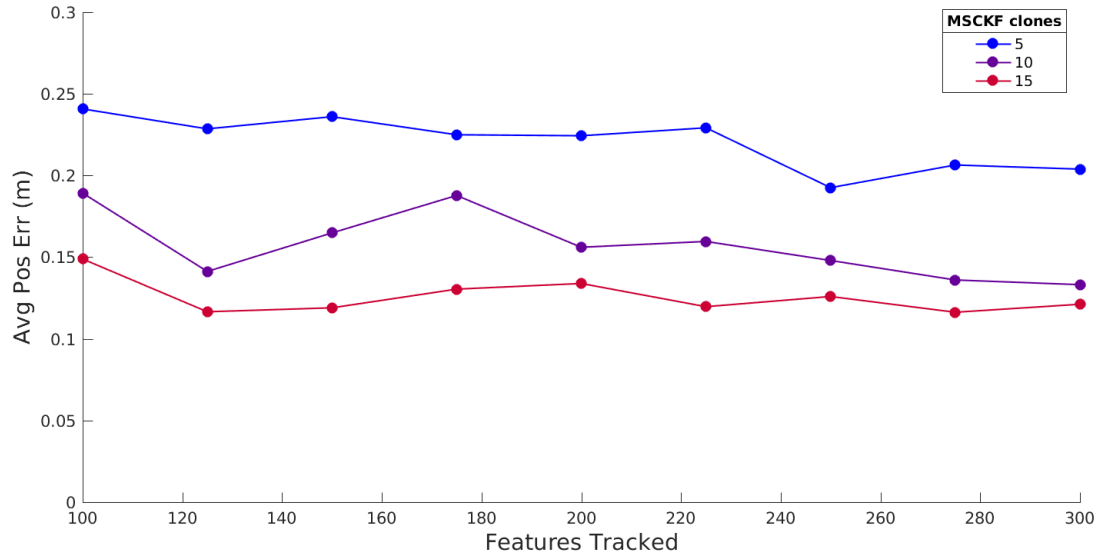


Figure 24: EuRoC MAV dataset Vicon Room 2 "easy" (113s second trajectory with images of size 752×480 at 20 Hz and IMU measurements at 200 Hz): average position error (monocular MSCKF) relative to ground truth data, with each data point representing an average over 100 runs.

The accuracy gain that results from using larger clone sizes is clear in both of the preceding figures. Looking at the results from the first dataset (Figure 23), a larger clone size reduces error for all feature sizes up to 325— beyond this, the result becomes less consistent. Additionally, the downward trend in error as feature size increases (for all clone sizes) is most visible up to about 300 features; evidently there is little to no benefit of increasing the feature size over 300. Accordingly, in the second dataset (Figure 24), we reduce the range of feature sizes such that the maximum is 300, but performed the evaluation in finer increments.

As a practical example of using these results, consider Figure 24 along side the runtimes for the Jetson Nano (Figure 19): evidently, choosing a feature size no larger than 100 in order to remain real-time would *not* cause a significant performance loss in a typical MAV application (at 10 clones, for example, the reduction of the average position error by moving from 100 to 200 features is only about 3cm).

The previous evaluations consider the monocular-case MSCKF with varying clone size and feature size; to evaluate our multi-camera estimator as described in Chapter 3, we consider a *fixed* clone size and vary both the feature size and the number of independent camera streams. Figure 25 shows the results from the same multi-camera dataset shown in Figure 7 and Figure 13. We compare the start-end position error resulting from the 2-camera versus the 3-camera sensor configuration—the independence of the image streams implies that no stereo constraints are enforced in either case. For all feature sizes considered, the 3-camera case proves to significantly reduce the start-end position error over the 2-camera case.

Figure 25: University of Delaware Gore Hall dataset (316s, 443m closed loop trajectory with images of size 648×488 at 20 Hz and IMU measurements at 400 Hz): start-end position error (multi-camera MSCKF) with each data point representing the average result over 10 runs.

Chapter 5

CONCLUDING REMARKS

5.1 Summary

The work presented here outlines the design and evaluation of a multi-camera visual-inertial estimation system based on the efficient MSCKF. After reviewing the standard (monocular/stereo) MSCKF in detail in Chapter 2, we introduce a method to extend MSCKF to incorporate an arbitrary number of asynchronous, non-overlapping cameras. This extension is motivated by the fact that single view (monocular or stereo) VINS is vulnerable to failure when viewing textureless or dynamic regions of the environment, and that adding information sources to the system can improve localization accuracy. One of the key ideas that enables the multi-camera estimator to remain computationally efficeient, as described in Section 3.3, is to apply linear pose interpolation on $SO(3) \times \mathbb{R}^3$ based on the imaging times of only a single camera. We present results to validate the multi-camera estimator in Section 3.7, where the start-end position error using a 3-camera sensor rig was measured to be less than 1% of the 440m path traveled.

Both the standard and multi-camera MSCKF leave available several free parameters (e.g. the number of features to track, the number of stochastic clones to retain in the state vector, and the number of independent cameras to use), which can have significant impact on the estimation accuracy and computational complexity. To gain a quantitative understanding of the free parameters on the performance of the VINS system, we carry out a set of experiments in Section 4.3 and 4.4. First, we vary the free parameters and evaluate their impact on the computational speed on a set of common computing platforms (including some resource-constrained SoC platforms, since the

main target application is localization onboard MAVs). Then, we investigate how the same free parameters impact the localization accuracy on both public visual-inertial datasets and our own.

5.2 Path Forward

The work accomplished here serves as a good foundation for further extensions of the presented multi-camera MSCKF estimator. One of which, as mentioned in Section 3.7, is to introduce multiple IMUs to the sensor platform. Similarly, this introduces more information sources and adds robustness to the estimator. As mentioned, the multi-IMU, multi-camera research direction is the subject of the recent work [44], where a proposed estimator (MiMc) is presented in detail. The same performance evaluation strategy presented here could apply to this estimator— where the number of IMUs would be treated as a free parameter.

While the performance evaluations provided here consider three of the important free parameters within our estimator, many other free parameters could be evaluated in the same way— among which, the image frame rate, the image resolution, and the IMU rate are typically available. One could also go further and consider the parameters within the vision front-end (e.g. the FAST feature detector threshold value, or the KLT optical flow window size) as design parameters of the overall system.

The multi-camera estimator in its present state has no mechanism to recognize previously seen feature points. An interesting improvement for future work could include a SLAM-like feature recognition in order to perform short-term loop closures when the same feature point is observed at different time's by two or more cameras. This could be helpful in the likely scenario where, due to the sensor platform rotating as it travels, a feature point that moves out of view of one camera quickly moves *into* view of another.

Lastly, since the performance evaluation presented here only considers *general-purpose* computing hardware, another research direction to continue this study is toward hardware-software co-design as done in [50]; here, VIO algorithm and parameter

choices are tightly coupled to hardware design choices. As a result, a specialized chip (e.g. an FPGA) can be implemented to greatly reduce the power consumption and form-factor of the computing platform. This type of design strategy, as the authors of [50] argue, is necessary to implement VIO on extremely small MAV platforms (typically called nano or pico-UAVs, with mass on the order of 10 grams). Some examples of this class of MAVs include Bitcraze's Crazyflie [51] and FLIR's Black Hornet [52].

BIBLIOGRAPHY

[1] S. Shen. "Autonomous navigation in complex indoor and outdoor environments with micro aerial vehicles". PhD thesis. University of Pennsylvania, 2014.

[2] G. Zhang, B. Shang, Y. Chen, and H. Moyes. "SmartCaveDrone: 3D cave mapping using UAVs as robotic co-archaeologists". In: *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE. 2017, pp. 1052–1057.

[3] A.-K. Mahlein. "Plant disease detection by imaging sensors–parallels and specific demands for precision agriculture and plant phenotyping". In: *Plant disease* 100.2 (2016), pp. 241–251.

[4] A. Ollero, G. Heredia, A. Franchi, G. Antonelli, K. Kondak, A. Sanfeliu, A. Viguria, J. R. Martinez-de Dios, F. Pierri, J. Cortes, A. Santamaria-Navarro, M. A. Trujillo Soto, R. Balachandran, J. Andrade-Cetto, and A. Rodriguez. "The AEROARMS Project: Aerial Robots with Advanced Manipulation Capabilities for Inspection and Maintenance". In: *IEEE Robotics Automation Magazine* 25.4 (Dec. 2018), pp. 12–23. ISSN: 1070-9932.

[5] K. Kanistras, G. Martins, M. J. Rutherford, and K. P. Valavanis. "Survey of unmanned aerial vehicles (UAVs) for traffic monitoring". In: *Handbook of unmanned aerial vehicles* (2015), pp. 2643–2666.

[6] M. Kim and E. T. Matson. "A cost-optimization model in multi-agent system routing for drone delivery". In: *International Conference on Practical Applications of Agents and Multi-Agent Systems*. Springer. 2017, pp. 40–51.

[7] R. D. Arnold, H. Yamaguchi, and T. Tanaka. "Search and rescue with autonomous flying robots through behavior-based cooperative intelligence". In: *Journal of International Humanitarian Action* 3.1 (2018), p. 18.

[8] R. Bähnemann, D. Schindler, M. Kamel, R. Siegwart, and J. Nieto. "A decentralized multi-agent unmanned aerial system to search, pick up, and relocate objects". In: *2017 IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*. IEEE. 2017, pp. 123–128.

[9] G. Huang. "Visual-inertial navigation: A concise review". In: *arXiv preprint arXiv:1906.02650* (2019).

[10] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. Leonard. "Past, Present, and Future of Simultaneous Localization And Mapping: Towards the Robust-Perception Age". In: *IEEE Transactions on Robotics* 32.6 (2016), pp. 1309–1332.

[11] R. Mur-Artal and J. D. Tardós. "Visual-inertial monocular SLAM with map reuse". In: *IEEE Robotics and Automation Letters* 2.2 (2017), pp. 796–803.

[12] A. I. Mourikis and S. I. Roumeliotis. "A multi-state constraint Kalman filter for vision-aided inertial navigation". In: *Proceedings of the IEEE International Conference on Robotics and Automation*. Rome, Italy, Apr. 2007, pp. 3565–3572.

[13] A. Mourikis, N. Trawny, S. Roumeliotis, A. Johnson, A. Ansar, and L. Matthies. "Vision-Aided Inertial Navigation for Spacecraft Entry, Descent, and Landing". In: *Robotics, IEEE Transactions on* 25.2 (Apr. 2009), pp. 264–280. ISSN: 1552-3098.

[14] M. Li and A. I. Mourikis. "High-Precision, Consistent EKF-based Visual-Inertial Odometry". In: *International Journal of Robotics Research* 32.6 (2013), pp. 690–711.

[15] G. Huang. *Towards Consistent Visual-Inertial Navigation*. Tech. rep. MIT Computer Science and Artificial Intelligence Laboratory (CSAIL), Sept. 2013. URL: http://people.csail.mit.edu/ghuang/paper/tr/stocvins.pdf.

[16] G. Huang, K. Eckenhoff, and J. Leonard. "Optimal-State-Constraint EKF for Visual-Inertial Navigation". In: *Proc. of the International Symposium on Robotics Research*. Sestri Levante, Italy, Sept. 2015.

[17] 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". In: *IEEE Robotics and Automation Letters* 3.2 (2018), pp. 965–972.

[18] K. Eckenhoff, P. Geneva, J. Bloecker, and G. Huang. "Multi-Camera Visual-Inertial Navigation with Online Intrinsic and Extrinsic Calibration". In: *Proc. International Conference on Robotics and Automation*. (in review). Montreal, Canada, May 2019.

[19] S. I. Roumeliotis and J. W. Burdick. "Stochastic cloning: A generalized framework for processing relative state measurements". In: *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*. Vol. 2. IEEE. 2002, pp. 1788–1795.

[20] N. Trawny and S. I. Roumeliotis. *Indirect Kalman Filter for 3D Attitude Estimation*. Tech. rep. University of Minnesota, Dept. of Comp. Sci. & Eng., Mar. 2005.

[21] J. Hesch, D. Kottas, S. Bowman, and S. Roumeliotis. "Towards Consistent Vision-Aided Inertial Navigation". In: *Algorithmic Foundations of Robotics X*. Ed. by E. Frazzoli, T. Lozano-Perez, N. Roy, and D. Rus. Vol. 86. Springer Tracts in Advanced Robotics. Springer Berlin Heidelberg, 2013, pp. 559–574.

[22] J. Hesch, D. Kottas, S. Bowman, and S. Roumeliotis. "Consistency Analysis and Improvement of Vision-aided Inertial Navigation". In: *IEEE Transactions on Robotics* PP.99 (2013), pp. 1–19.

[23] M. Li and A. I. Mourikis. "Improving the Accuracy of EKF-based Visual-Inertial Odometry". In: *Proc. of the IEEE International Conference on Robotics and Automation*. St. Paul, MN, May 2012, pp. 828–835.

[24] P. Geneva, K. Eckenhoff, W. Lee, Y. Yang, and G. Huang. "OpenVINS: A Research Platform for Visual-Inertial Estimation". In: (Nov. 2019). URL: https://github.com/rpng/open_vins.

[25] P. S. Maybeck. *Stochastic Models, Estimation, and Control*. Vol. 1. Academic press, 1979, p. 217.

[26] T. Barfoot, J. R. Forbes, and P. T. Furgale. "Pose estimation using linearized rotations and quaternion algebra". In: *Acta Astronautica* 68.1-2 (2011), pp. 101–112.

[27] C. G. Harris, M. Stephens, et al. "A combined corner and edge detector." In: *Alvey vision conference*. Vol. 15. 50. Citeseer. 1988, pp. 10–5244.

[28] S. M. Smith and J. M. Brady. "SUSAN—a new approach to low level image processing". In: *International journal of computer vision* 23.1 (1997), pp. 45–78.

[29] E. Rosten and T. Drummond. "Fusing points and lines for high performance tracking". In: *Proc. of the IEEE International Conference on Computer Vision*. Beijing, China, Oct. 2005, pp. 1508–1515.

[30] E. Rosten and T. Drummond. "Machine learning for high-speed corner detection". In: *European conference on computer vision*. Springer. 2006, pp. 430–443.

[31] D. G. Lowe. "Object recognition from local scale-invariant features". In: *In Proceedings of the International Conference on Computer Vision*. 1999, pp. 1150–1157.

[32] H. Bay, T. Tuytelaars, and L. V. Gool. "SURF: Speeded Up Robust Features". In: *Proc. of the European Conference on Computer Vision*. Vol. 3951. 1. 2006, pp. 404–417.

[33] M. Calonder, V. Lepetit, C. Strecha, and P. Fua. "BRIEF: Binary Robust Independent Elementary Features". In: *Proc. of the European Conference on Computer Vision*. Crete, Greece, Sept. 2010, pp. 778–792.

[34] E. Rublee, V. Rabaud, K. Konolige, and G. R. Bradski. "ORB: An efficient alternative to SIFT or SURF." In: *ICCV*. Vol. 11. 1. Citeseer. 2011, p. 2.

[35] E. Karami, S. Prasad, and M. Shehata. "Image matching using SIFT, SURF, BRIEF and ORB: performance comparison for distorted images". In: *arXiv preprint arXiv:1710.02726* (2017).

[36] S. Baker, D. Scharstein, J. Lewis, S. Roth, M. J. Black, and R. Szeliski. "A database and evaluation methodology for optical flow". In: *International Journal of Computer Vision* 92.1 (2011), pp. 1–31.

[37] B. D. Lucas, T. Kanade, et al. "An iterative image registration technique with an application to stereo vision". In: (1981).

[38] M. K. Paul and S. I. Roumeliotis. "Alternating-Stereo VINS: Observability analysis and performance evaluation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 4729–4737.

[39] T. Qin and S. Shen. "Online temporal calibration for monocular visual-inertial systems". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 3662–3669.

[40] T. D. Barfoot. *State Estimation for Robotics*. Cambridge University Press, 2017, p. 209.

[41] C. Forster, L. Carlone, F. Dellaert, and D. Scaramuzza. "IMU preintegration on manifold for efficient visual-inertial maximum-a-posteriori estimation". In: Georgia Institute of Technology. 2015.

[42] T. D. Barfoot. *State Estimation for Robotics*. Cambridge University Press, 2017, p. 222.

[43] URL: http://udel.edu/~ghuang/papers/tr_mc-vins.pdf.

[44] K. Eckenhoff. "Towards Robust Visual-Inertial State Estimation". PhD thesis. University of Delaware, 2019.

[45] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. "ROS: an open-source Robot Operating System". In: *ICRA workshop on open source software*. Vol. 3. Kobe. 2009, p. 5.

[46] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik, and R. Siegwart. "The EuRoC micro aerial vehicle datasets". In: *The International Journal of Robotics Research* 35.10 (2016), pp. 1157–1163.

[47] URL: https://ardupilot.org/copter/docs/common-qualcomm-snapdragon-flight-kit.html.

[48] M. A. Fischler and R. C. Bolles. "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography". In: *Communications of the ACM* 24.6 (1981), pp. 381–395.

[49] R. Hartley and A. Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003, p. 239.

[50] Z. Zhang, A. A. Suleiman, L. Carlone, V. Sze, and S. Karaman. "Visual-inertial odometry on chip: An algorithm-and-hardware co-design approach". In: *Robotics: Science and Systems* (2017).

[51] W. Giernacki, M. Skwierczyński, W. Witwicki, P. Wroński, and P. Kozierski. "Crazyflie 2.0 quadrotor as a platform for research and education in robotics and control engineering". In: *2017 22nd International Conference on Methods and Models in Automation and Robotics (MMAR)*. IEEE. 2017, pp. 37–42.

[52] URL: https://www.flir.com/products/black-hornet-prs/.

[53] K. Li, B. Hu, L. Chang, and Y. Li. "Comparison of direct navigation mode and indirect navigation mode for integrated SINS/GPS". In: *Transactions of the Institute of Measurement and Control* 38.1 (2016), pp. 3–13.

[54] Kazusuke Maenaka. "MEMS inertial sensors and their applications". In: *2008 5th International Conference on Networked Sensing Systems*. June 2008, pp. 71–73.

[55] D. Titterton, J. L. Weston, and J. Weston. *Strapdown inertial navigation technology*. Vol. 17. IET, 2004.

[56] M. Brossard, A. Barrau, and S. Bonnabel. "AI-IMU Dead-Reckoning". In: *CoRR* abs/1904.06064 (2019). arXiv: 1904.06064. URL: http://arxiv.org/abs/1904.06064.

Appendix A

KALMAN FILTER OVERVIEW

The vast majority of visual-inertial navigation systems are based on the Extended Kalman Filter (EKF), an estimation framework which has seen decades of implementation. In any context (not limited to VINS), the Kalman Filter is an optimal recursive data processing algorithm used to keep track of the estimated state of a system together with its variance or uncertainty. In principle, EKF is no different than an ordinary Kalman Filter with the exception of an additional a linearization step which enables the standard equations (see [25]) to be applied to non-linear system models and/or measurement models. The form of the EKF implemented in this work is an *indirect* (error-state) filter (see [53]), as an estimate of the *error state* is what is maintained by the filter, and then used to correct the value of the true state.

The key idea of any Kalman filter, as illustrated by the block diagram in Figure 1, is its predictor-corrector structure; at each time step, the state of the system is predicted by stepping system model forward in time, then corrected (updated) using new measurements.

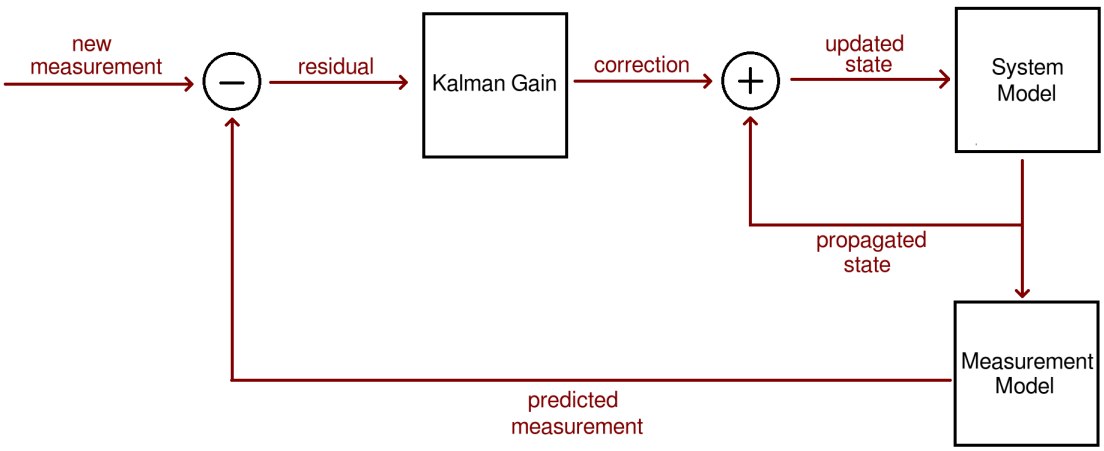


Figure 1: Kalman Filter structure (general): A predicted measurement (based on system and measurement models) is subtracted from each new measurement to obtain the measurement residual. This residual is weighted by the Kalman Gain (a factor determined by the relative uncertainties in the new and predicted measurements) and then added to the propagated (predicted) state to produce the updated state. The updated state is then used to compute a new predicted measurement and the process is repeated.

However, there is an important distinction in the case many VINS applications of the Kalman Filter, including the one presented here, which is that *no* system model or dynamic constraints are imposed in the prediction step. Instead, the IMU measurements are used in place of the system model, and the predicted state is given by propagation of the IMU measurements (see Section 2.3). Then, the update step depends only on visual measurements.

Appendix B

INERTIAL SENSORS

As inertial sensors may be less familiar to many than are digital cameras, some background is included here. An inertial measurement unit (IMU) is a device that contains (possibly multiple) accelerometers and gyroscopes which together are used to measure the linear acceleration and angular rate of the device as it moves along a 3D trajectory. Depending on the sensitivity of the IMU, the frame of reference in which the measurements are reported can be considered Earth-fixed or to include the effects of the Earth's rotation (we assume Earth-fixed in our implementation). For VINS applications on MAVs, the relevant class of IMU sensors are those based on micro-electromechanical systems (MEMS). Improved manufacturing processes in recent years have enabled the mass production of small, inexpensive MEMS-based IMU sensors and their incorporation into many types of consumer products, including (in addition to MAVs) gaming systems, pedometers (e.g Fitbit), and mobile devices.

The basic operating principle behind one common form of MEMS-based accelerometer is illustrated in Figure 1. Measurements are based on the detection of a change in electrical capacitance caused by the displacement of a very small mass inside the device. This mass is suspended by internal spring-like structures, isolating it from the device housing, which is rigidly attached to the sensor platform.

MEMS-based gyroscopes also have a similar structure, but the measurement of angular rates is less simple; in MEMS-gyroscopes, a mass is continually vibrated by internal actuators along a fixed axis. Then, the angular rate of the IMU can be determined through the resulting Coriolis forces which drive the mass along an axis perpendicular to the actuation. Detailed overviews of MEMS inertial sensors and their applications are given in [54] and [55].

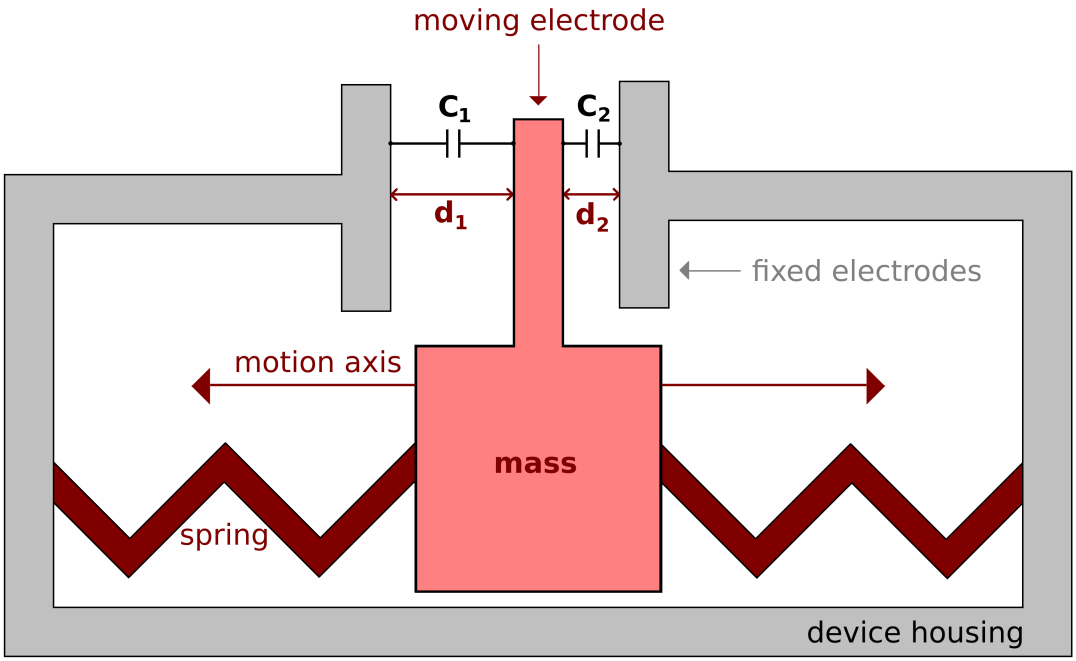


Figure 1: Working principle of capacitive accelerometer: As the device accelerates along the motion axis in either direction, the changing ratio d_1/d_2 causes a corresponding change in the parallel-plate capacitance ratio c_1/c_2. This change in capacitance drives an internal measuring circuit to produce a signal proportional to the acceleration of the device.

In the context of VINS applications, the pose of an IMU-affixed reference frame can be tracked through the double integration of linear and angular acceleration measurements over time. However, using *only* the integration of inertial measurements (an example of what is known as dead-reckoning) will result in substantial drifting of the pose estimate as the error grows unboundedly. Interestingly, there have been recent efforts to improve IMU dead-reckoning through machine learning [56]. In VIO, the visual measurements from the camera(s) serve to correct the IMU drift and reduce the rate at which the error in the pose estimate grows. The equations governing the step-wise time integration of inertial measurements (IMU propagation) are given in Section 2.3.