Comparison of State Marginalization Techniques in Visual Inertial Navigation Filters

A Thesis submitted to the School of Graduate Studies in partial fulfillment of the requirements for the degree of Master of Engineering

by

Ravindu G. Thalagala, B.Sc(hons)

Supervisory Committee:

Dr. Oscar De Silva

Dr. George. K. I. Mann

Dr. Raymond G. Gosine

Faculty of Engineering and Applied Science

Memorial University of Newfoundland

October, 2019

Abstract

The main focus of this thesis is finding and validating an efficient visual inertial navigation system (VINS) algorithm for applications in micro aerial vehicles (MAV). A typical VINS for a MAV consists of a low-cost micro electro mechanical system (MEMS) inertial measurement unit (IMU) and a monocular camera, which provides a minimum payload sensor setup. This setup is highly desirable for navigation of MAVs because highly resource constrains in the platform. However, bias and noise of low-cost IMUs demand sufficiently accurate VINS algorithms. Accurate VINS algorithms has been developed over the past decade but they demand higher computational resources. Therefore, resource limited MAVs demand computationally efficient VINS algorithms.

This thesis considers the following computational cost elements in the VINS algorithm: feature tracking front-end, state marginalization technique and the complexity of the algorithm formulation. In this thesis three state-of-the-art feature tracking front ends were compared in terms of accuracy. (VINS-Mono front-end, MSCKF-Mono feature tracker and Matlab based feature tracker). Four state-of-the-art state marginalization techniques (MSCKF-Generic marginalization, MSCKF-Mono marginalization, MSCKF-Two way marginalization and Two keyframe based epipolar constraint marginalization) were compared in terms of accuracy and efficiency. The complexity of the VINS algorithm formulation has also been compared using the filter execution time.

The research study then presents the comparative analysis of the algorithms using a publicly available MAV benchmark datasets. Based on the results, an efficient VINS algorithm is proposed which is suitable for MAVs.

Acknowledgements

It is an exclusive privilege of being advised and working in *Intelligent Systems Lab* (IsLab), under the supervision of Dr.Oscar De Silva, Dr.George K.I. Mann and Dr.Raymond G. Gosine at my Master of Engineering degree. Their invaluable lessons, academic guidance as well as encouragement greatly improved my research skills. I would like to express my sincere appreciation to them and the help they gave me in order to finish my thesis.

To IsLab members, Dr.Thumeera R. Wanasinghe, Mr.Mihiran Galagedarage Don, Mr.Eranga Fernando, Mr.Mahmoud Abd El Hakim, Mr.Kusal Tennakoon, Mr.Nushen Senevirathne and Ms.Sachithra Attapattu, I would like to thank them for their great assistances, patience and inspirations.

Table of Contents

List of Tables

List of Figures

Abbreviations

BA Bundle Adjustment

BRIEF Binary Robust Independent Elementary Features

BRISK Binary Robust Invariant Scalable Keypoints

CKF Cubature Kalman filter

CMOS Complementary Metal–Oxide–Semiconductor

ESKF Error State Kalman Filter

FAST Features from Accelerated Segment Test

FPGA Field Programmable Gate Arrays

IMU Inertial measurement unit

INS Inertial Navigation Systems

ISL Intelligent Systems Lab

KLT Kanade-Lucas-Tomasi

MAV Micro-aerial vehicle

MEMS Micro Electro Mechanical Systems

MSCKF Multi-State Constraint Kalman Filter

MUN Memorial University of Newfoundland

OC-MSCKF Observability-Constrained Multi-State Constraint Kalman Filter

ORB Oriented FAST and Rotated BRIEF

RK4 Foruth Order Runge-Kutta Integration

SLAM Simultaneous localization and mapping

SR-ISWF Square-root Inverse Sliding Window Filter

SWaP Size Weight and Power

UKF Unscented Kalman filter

UWB Ultra Wide Band

VIN Visual Inertial Navigation

VINS Visual-inertial Navigation Systems

VIO Visual Inertial Odometry

Chapter 1

Introduction

In this chapter, the overall objective and motivation of this thesis are presented. An overview of visual-inertial navigation systems (VINS) along with their key sub-modules affecting VINS performance are discussed. The problem statement of the thesis is then formulated. Finally, objectives and expected contributions will be highlighted along with the organization of the thesis.

1.1 Motivation

Visual inertial navigation (VIN) is the process of combining measurements from an inertial measurement unit (IMU) and visual measurements from a camera for the purpose of estimating the 6 degrees of freedom (DOF) pose of a mobile platform.

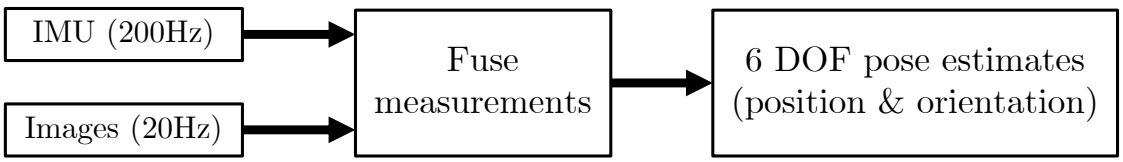

Figure 1.1: Overview of VINS

VINS has been widely popular over the past decade, particularly in GPS-denied applications such as indoor [3–5], underwater [6, 7] and planetary exploration [8, 9]. VINS has two fundamental measuring units. One is a 6-axis IMU rigidly attached on the platform which measures the local linear acceleration and angular velocity. Second is a camera that is also rigidly attached to the platform which tracks features with

the distinct appearance in the environment. Recent developments in low cost, lightweight MEMS IMUs [10, 11] and small form factor cameras [12] have made it possible to deploy VINS in a wider variety of new platforms including MAVs [4, 13–16], handheld mobile devices [17, 18], virtual reality devices [19, 20] and autonomous driving modules [21–23].

Implementing VINS algorithms on MAVs faces two main challenges. First, there is a observability constraint of VINS where the system can only provide a drifting estimate of the position and orientation of the platform [24]. Therefore, in real-world implementations on MAVs, VINS is only meant to serve as an odometer while having periodic corrections in place, such as intermittent GPS corrections [25], loop-closure updates [26], ultra wide band (UWB) ranging corrections [27], or place-recognition based updates [13] to limit the drift of the localization system. Second, MAVs have size, weight, and power (SWaP) constraints which limit the applicability of high-power, large-aperture payloads, and additional computational resources required for demanding VINS algorithms. Many state-of-the-art algorithms [26, 28] use heavy computational resources to gain marginal improvement in localization performance [29,30]. However, this thesis argues that VINS algorithms on MAVs should be designed with the objective of being highly computationally efficient and stable rather than being highly accurate. Reasonable accuracy to serve the function of an odometer is sufficient considering the nature of the application of these systems on MAVs in practice [4, 31].

In VINS algorithms, there are three main elements which dictate its computational complexity. These include the *Feature Tracking Front-End*, *State Marginalization*, and *State Estimation Algorithm*.

(1) Feature Tracking Front End

Keeps track of the pixel location of unique visual features of the environment as the platforms navigate through space. For this purpose, VINS algorithms use different feature types, descriptors, number of features, and selection approaches, which are then fed into a tracker module to match the unique features across a set of images. The combination of parameters used in this feature tracker significantly affect the performance of the VINS.

(2) State Marginalization

For navigation purposes, VINS keeps a history of camera poses in its state vector. Periodically a subset of these camera poses are selected and used for

correction/update of the filter*. This process is termed state marginalization. Many state-of-the-art marginalization techniques have been reported in literature [3, 4, 31]. These techniques have their own merits and demerits in terms of estimation accuracy, estimator stability, computational resource and time complexity. Therefore, open avenues still exist for optimal computationally efficient state marginalization in VINS for MAV applications.

(3) State Estimation Algorithm
Type of the VINS algorithm used also dictates the computational burden on resource-constrained systems [29, 30]. There are two main types of VINS algorithms, namely optimization based [26, 28] and filter based [4, 5, 32]. The optimization-based VINS methods utilize an order of a magnitude higher computational resources than the filter-based methods [29]. Among the filter-based methods, the algorithms proposed in [33] and [34] consume significantly less computational resources due to simpler measurement models used in the formulations. However, the accuracy and computational burden of these different formulations need to be quantitatively compared in establishing a custom efficient design suitable for MAVs.

The intelligent system's lab (ISL) of Memorial University of Newfoundland (MUN) is developing a building-wide navigation system for multi-robot applications. The system comprises of MAVs equipped with embedded low complexity stable VINS which serve as the odometers, and external navigation aids (UWB and place recognition) to assist with periodic corrections of the platforms. The VINS for this purpose should primarily feature low computational complexity, stable performance, and capability to embed on small form factor MAVs such as the Crazie fly MAV [35]. The main objective of this thesis is to compare the state-of-the-art subsystems used in VINS designs and identify a computationally efficient VINS design suitable for low power robotic systems [35]. The development of this thesis is specifically targeted for the building-wide multi-robot system discussed above.

*filter-based VINS algorithms are referred to as filters throughout this thesis. It is important to note that there's a separate class of VINS algorithms based on optimization techniques which are not considered in the development of this thesis. This is because the optimization-based VINS methods utilize an order of a magnitude higher computational resources.

1.2 Block Diagram of VINS

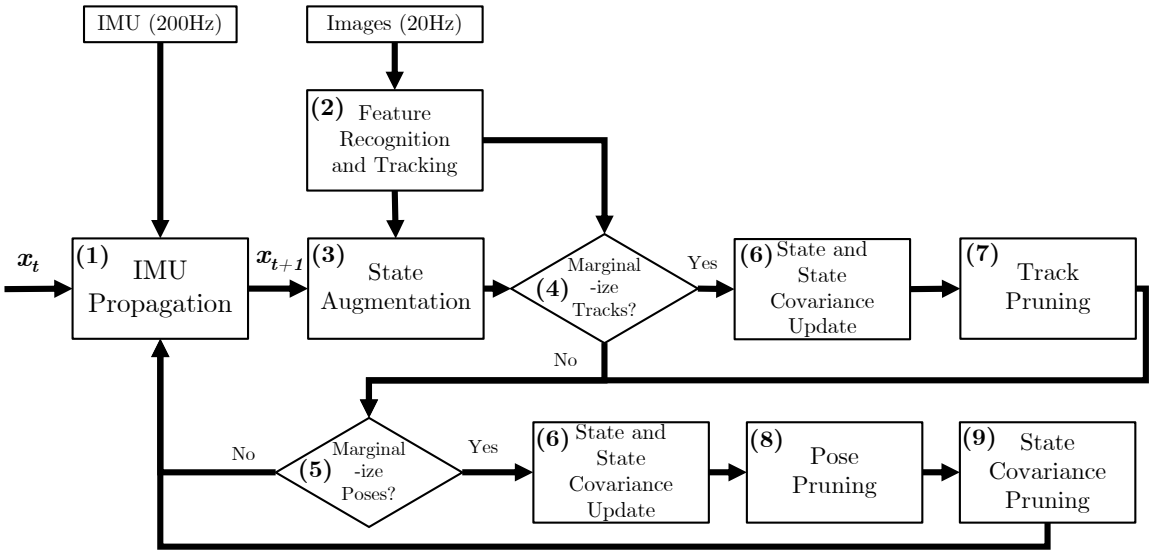

Figure 1.2: VINS filtering algorithm

The block diagram shown below provides an overview of key sub modules of a VINS. It follows a *Prediction step* and an *Update step (Marginalization step)* architecture seen in Kalman filters [36], with several additional intermediate steps to achieve robust performance. The state vector considered in VINS comprises of position ($\mathbf{p}$), orientation ($\mathbf{q}$), velocity ($\mathbf{v}$), bias of accelerometer ($\mathbf{b_a}$), bias of gyroscope ($\mathbf{b_g}$) and 30 previous camera-position ($\mathbf{p_c}$) and camera-orientation($\mathbf{q_c}$) (30 camera poses) sets. An overview of each sub-module illustrated in Figure 1.2 are provided below:

(1) **IMU Propagation:**
During this step, the measurements reading from the IMU is used to find the next state $\mathbf{x_{t+1}}$ of the filter. This is accomplished by integrating the IMU measurements with the current state estimate $\mathbf{x_t}$ of the filter, using the kinematic model of the platform.

(2) **Feature Recognition and Tracking:**
During this step, visual feature points are extracted from the image corresponding to visually unique 3D landmarks seen in the environment. If the same 3D landmark is seen in the following images, it is tracked using a Kanade-Lucas-Tomasi (KLT) [37] feature tracker. This sequence of tracked pixel positions is

called a track. All tracks correspond to different feature points are stored as a track-list in the filter.

(3) **State Augmentation:**
For VINS filters, the state vector should be expanded to keep track of the different camera poses while the robot platform moves in the environment. The different camera poses should be periodically added to the state vector. This process is called *State Augmentation*. The process is also termed as stochastic cloning in VINS literature [38].

(4) **Track Marginalization (Track Update):**
The feature tracks recorded above are used to update the filter,(i.e., track marginalization). In most implementations, this is done by selecting the tracks of features that are no longer in the view of the camera, (i.e., completed tracks).

(5) **Pose Marginalization (Pose Update):**
The state augmentation steps keep expanding the state vector as the platform moves in the environment. This is undesirable as large state vector will eventually lead to a computationally intractable update step. Therefore, a subset of camera poses are periodically selected from stored camera poses in the state vector to update and remove from the state. This is called the *Pose Marginalization*.

(6) **State and Covariance Update:**
This is the process of updating the state vector and covariance matrix as per the Kalman filter algorithm [36].

(7) **Track Pruning:**
When a track is used to update the filter, it is removed from the track-list. It is called *Track Pruning*.

(8) **Pose Pruning:**
The used camera poses to update the filter are removed from the state vector, which is termed *Pose Pruning*. Tracks which do not have any poses in the state vector, and poses which do not have any tracks associated with them as a result of the pruning steps are identified as redundant tracks and redundant poses. These are also removed from the track-list and the state vector.

(8) **State Covariance Pruning:**
When the camera poses in the state vector get removed in the pose pruning step, the covariance matrix entries correspond to those camera poses should also be removed. This process is called *Covariance Pruning*.

The complexity of a VINS algorithm mainly depends on the marginalization steps outlined above, and the feature tracking front end of the algorithm. This thesis compares the state-of-the-art algorithms and identifies the following key problems related to VINS algorithms that should be addressed in order to develop an efficient VINS algorithm for MAVs.

1.3 Problem Statement

The following section describes research gaps and/or problems that will be addressed throughout this thesis. A computationally efficient VINS algorithm is then presented with a performance comparison using a VINS benchmark dataset.

1.3.1 Problem I: Complexity of the Feature Tracking Front-End.

The robustness of VINS algorithms depends on keeping track of unique visual features (feature tracking) seen on the images. Low accuracy, unreliable visual feature tracking can downgrade the robustness of the VINS algorithm or even failure. Therefore, many recent work [14, 26, 28] focus on developing accurate and robust feature tracking algorithms.
However, these solutions become increasingly complex and computationally demanding, yet offer marginal increase in localization accuracy as outlined in [29], which makes them unsuitable for deploying in resource-constrained systems like MAVs.
Therefore, a careful selection of a feature tracking front-end is needed for an optimal trade-off between complexity and accuracy. A performance comparison highlighting the complexity and accuracy of the available feature tracking front ends is presented in this thesis.

1.3.2 Problem II: Complexity of the State Marginalization Techniques.

The state marginalization (i.e. filter update) has significant effects on the computational cost and the estimation/localization accuracy of the VINS algorithm. There are two types of marginalization methods used in VINS algorithms. One is termed *Pose Update* and the other is *Track Update*. The details of these methods will be discussed in the upcoming chapters of this thesis. The *Pose Update* has been identified as the most resource utilizing operation which uses a subset of camera poses in the state vector for marginalization.

The first MSCKF algorithm is proposed by Mourikis and Roumeliotis [3]. It uses ten (out of 30 camera poses) camera poses for the marginalization, which incurs a considerable amount of computational cost. In a subsequent variation of this algorithm, known as MSCKF-MONO [31] proposes a key-frame based update strategy which uses only two camera poses for the update. It significantly reduces the computational cost. The algorithm proposed in [4] uses a key-framing strategy termed as *Two-way Marginalization* to select two camera poses for the update. The method presented in [33] uses a constraint between two images know as *Epipolar Constraint* as the state marginalizing technique which simplifies the pose update with a marginal decrease in accuracy. However, this method has not been verified and compared throughly against generic VINS available for MAV applications.

Therefore, the trade-off between accuracy and computational cost should be optimally balanced when designing VINS algorithms. This thesis will provide another comparative analysis that can be used to identify an efficient solution.

1.3.3 Problem III: Complexity of State Estimation Algorithm versus VINS Performance.

VINS algorithms have two different state estimation problem formulations. One is optimization-based, and the other is filtering based. Optimization-based methods are more accurate but computationally demanding [29]. Therefore, resource-constrained systems like MAVs tend to use filter-based methods [4, 5, 32, 33].

In order to improve the performance of the filters, many variations of Kalman filters are proposed while Error-State Kalman Filter being the widely used method [4, 5, 32, 33]. Unscented Kalman Filter (UKF) [39] and Cubature Kalman Filter (CKF)

[34,40] were also introduced to effectively address the nonlinearity of VINS algorithms. However, these techniques have complex algorithm formulations and demand more computational power for a marginal increase in estimation accuracy [29, 30].
The MSCKF framework based VINS algorithms are computationally efficient compared to the UKF and CKF filters [40]. However, the key-framing strategies in MSCKF framework based VINS algorithms require a computationally expensive technique termed the *Null Space Trick* when calculating the measurements for state marginalization [3]. In contrast, the VINS algorithms that use epipolar-constraint or tri-focal constraints as the measurement model do not require null space mapping calculations, making them more computationally efficient.
Another important technique used to improve the consistency of the filter is the *Observability Constraint*. It makes the algorithm formulation complex. However, many recent formulations [4,32,33] use this technique, which increases filter accuracy with a marginal increase in computational cost and demonstrated the algorithm complexity is marginal. The effect of these algorithmic choices needs to be comparatively analyzed to establish complexity and accuracy implications on VINS filters quantitatively.

1.4 Objective and Expected Contributions of the Research

This study presents a comparative analysis of the state marginalization strategies used in MSCKF framework based VINS filters. Moreover, a comparison of image processing front-ends is also provided. The main objectives of the proposed study are:

Objective 1 Perform an accuracy and complexity comparison of four main state marginalization strategies used in VINS filters. The following state marginalization strategies will be evaluated using the EUROC VINS dataset [41].

- *Two-way Marginalization* [4]
- *Generic MSCKF marginalization* [3]
- *MSCKF-MONO marginalization* [31].
- *Two key-frame marginalization* [33].

Objective 2 Comparison of image processing front-ends used for VINS algorithms. The following feature trackers will be assessed using the EUROC VINS dataset [41].

- Using the feature tracker front-end used in VINS-MONO [26], MSCKF-MONO [31] and Matlab feature tracker for MSCKF [42].

Objective 3 Experimental validation of a computationally efficient VINS algorithm for mobile robot applications.

- In this thesis the EuRoC VINS benchmark dataset is used to validate the performance of the identified efficient design.
- Development of a VINS sensor module using an Intel ZR300 sensor and Intel NUC mini computer in order to experimentally validate the selected VINS designs identified in this thesis. The experimental validation using this hardware setup is not presented in this thesis as it is being completed as part of authors doctoral research work.

1.5 Organization of the Thesis

Chapter 1 presents an overview of the research area, highlights the research statement, and outlines the objectives and associated contributions of this study.

Chapter 2 presents the literature review in the area of MSCKF framework based VIN algorithms and identifies research gaps in the existing literature.

Chapter 3 presents the work carried out to calibrate a sensor setup for VIN algorithm evaluation. Hardware development procedure and calibration methodology is also outlined.

Chapter 4 presents the work carried out to develop the MSCKF framework for MAV state estimation problem using Error-State Kalman Filtering method.

Chapter 5 presents the results of the comparison study.

Chapter 6 presents the conclusion and future directives of this study.

Chapter 2

Related Work

2.1 Visual Inertial Navigation Systems

Deployment of inertial navigation systems (INS) [36, 43] to estimate the 6 DOF poses of various sensing platforms (e.g. autonomous vehicles) has been widely used over the past decade. Recent developments in low cost, light-weight MEMS IMUs [10, 11] have made it possible for this usage.

The low-cost IMUs used in INS have inherent noise and bias, which make those sensors unreliable for long term navigation of MAVs when simply integrated to find pose estimates. There are tactical grade IMUs available that can provide better estimates with low bias and noise, but those sensors are highly customized and not widely available for general use [13]. As a solution to overcome the unreliability of the low-cost IMUs, environment information retrieved through a small, lightweight, energy-efficient camera is fused with the IMU measurements, thus enabling VINS [13]. Low-quality IMUs, make VINS suffer from motion drift accumulation over time. It is a challenging problem to solve. The main reason for the problem is lack of global reference information in the estimators [13]. Many algorithms have been developed to address this problem which can be categorized into two groups, namely visual-inertial simultaneous localization and mapping (SLAM) [44] and visual-inertial odometry (VIO) [4,5,14,33,45,46]. Visual inertial SLAM have additional steps to implement loop closure and map generation whereas VIO focuses on implementing an odometer (without loop closure or mapping) for navigation purposes. Many state-of-the-art VINS solutions use Filter-based VIO (Filter-based VINS) due to computational efficiency. Google ARCore [47] is one of the practical applications where motion tracking

is performed using a Filter-based VINS. Optimization-based VINS methods have also emerged as an alternative to solve the estimation accuracy problem, but requires higher computational power.

2.1.1 Optimization-based VINS

Optimization-based techniques solve a bundle-adjustment (BA) problem [48] using an iterative mechanism to achieve the state estimate, making it more accurate than Filter-based VINS [26, 28]. Although BA problem demand higher computational power, recent literature recognized its sparse structure and introduced real-time algorithms, such as graph optimization and factor graph models [49].

Inconsistency problem in BA has also been mitigated through fix-lag fusion algorithms [14]. Although these techniques enable the implementation of optimization-based VINS real-time, it needs more relaxing of tight constraints in computations when deploying on resource-constrained systems. Sometimes this leads to downgraded performance [29].

Real-time implementation of optimization-based VINS on MAVs has been accomplished in recent literature. It has been accomplished by relaxing the optimization constraints [29]. In the comparison work presented in [29] outlines the relaxed constraints. However, even with relaxed constraints it utilizes higher computational resources, where filter based techniques use less computational resources with marginal decrease in estimation accuracy [29].

2.1.2 Filter-based VINS

The first filter-based VINS algorithm was developed by Mourikis and Roumeliotis [3]. It is known as the Multi-State Constraint Kalman Filter (MSCKF). In this thesis the MSCKF will be used as the baseline and, it will be termed as Generic MSCKF (Generic-MSCKF). It uses an efficient, tightly coupled state propagation in an Error-State Kalman Filter (ESKF) update using quaternion based inertial dynamics. MSCKF keeps motion constraints that are only related to the stochastically cloned camera poses in the state vector. The filter formulation achieves this by projecting the image bearing measurements to the null space of the feature Jacobian Matrix (i.e., linear marginalization) [13]. MSCKF is highly computationally efficient since it omits the need to co-estimate thousands of point features. Due to these reasons filtering

based VINS are less computationally complex than optimization or SLAM methods and can be implemented in MAVs [29].

2.2 Variations of Filter-based VINS

Several other forms of MSCKF based VINS algorithms have been developed to achieve better performance. Observability-constrained MSCKF (OC-MSCKF) formulation has been proposed to improve the consistency of the filter [32, 50, 51], while (right) Invariant Kalman Filters has also been proposed to improve the filter consistency. Square-root inverse sliding window filter (SR-ISWF) [30] has been proposed to improve the accuracy and consistency in limited-resource mobile device level implementation. Recent developments of stereo-vision based MSCKF, also referenced as S-MSCKF, demonstrated the ability of the algorithm to deploy in MAV systems [4]. Another recent key-frame based MSCKF implementation can be found in Prof. Kotas Daniilidis's research group Github repository [31]. The algorithm is named as MSCKF-MONO [31]. Several other variations of the MSCKF based VINS have been implemented to address limitations of the filters which are summarized in [13].
Among the approached presented in [13] OC-MSCKF has been the widely accepted for MAV applications due to its consistency and computational efficiency [3, 4, 31, 32]. MAVs require consistent estimation and computational efficiency rather than higher accuracy since it acts as an odometer. Therefore, OC-MSCKF enables consistency of the algorithm with marginal increase in computational cost [29].

2.3 Visual Measurement Sensor Arrangements

Other than the monocular formulation, the MSCKF based VINS algorithms have been modified to use stereo vision cameras to include the depth image features [4]. This increases the accuracy of the estimation, but at the expense of specialized hardware and increased the payload, which can be challenging for MAVs. Rolling-shutter cameras have been incorporated to mitigate the inaccuracies of time synchronization of images and IMU measurements [13]. Some modifications include multiple cameras and multiple IMUs [52]. All these formulations increase the accuracy of the system with some specialized hardware or multiple sensors. Those applications have limited practical deployment on MAVs as their limited payload capacity restricts, the hosting

of these additional sensors.

2.4 Image Processing Front End

To keep the computational cost and the estimation accuracy at acceptable levels for the MSCKF algorithms, many state-of-the-art algorithms use indirect image measurements over direct measurements [53, 54]. Direct image measurements need robust initialization and high frame rate since they use photometric consistency assumption [13]. Although these methods require less computation, it requires accurate filter initialization and high frame rate cameras. Therefore, MSCKF based VINS algorithms prefer indirect methods [3, 4, 13, 28] to derive the measurement model. Additionally, indirect methods extract and track point features in the image, which yields sufficient measurements for the VINS algorithms to use in its measurement model [13].

VINS filters utilize a feature tracking front end to maintain a list of pixel positions which correspond to visually distinct features of an environment. Shi-Tomasi corner detector is one of the widely used feature extraction methods due to its efficiency [5, 14, 26, 28]. The work presented in [26, 28] introduced a binary robust invariant scalable keypoints (BRISK) descriptor based matching for increased accuracy and efficiency [55]. In [14], the authors have used an oriented FAST* and rotated BRIEF (ORB) descriptor-based matching approach [57], to make the descriptor matching efficient. Additionally, it uses a heuristic method to calculate the quality score for outlier rejection. However, the reduction in processing time is not significant as compared to [28] and [26].

Since VINS algorithm formulation is highly sensitive to feature outliers, many state-of-the-art algorithms use random sample consensus (RANSAC) for outliers rejection during feature matching [4, 5, 14, 26, 28]. A gyro assisted two-point RANSAC is implemented in [5] to make outlier rejection more efficient. Implementing binary robust independent elementary features (BRIEF) descriptors [58] in [26] with uniform feature distribution demonstrated higher accuracy.

Kanade-Lucas-Tomasi (KLT) algorithm is widely used to extract and track features between images in many image processing front-ends [4, 5, 14, 26, 28]. Work presented in [5] uses Gaussian thresholding and box blurring for each image before applying the KLT tracker to ensure robust tracking even when the illumination levels are changing.

*Features from Accelerated Segment Test (FAST) [56]

BRISK descriptor matching reported in [26], reports tracking of features robustly even in changing illuminations conditions.

2.5 State Marginalization Techniques in VINS

There are two types of marginalization techniques utilized in the VINS formulation presented in [3], typically named as Track Update and Pose Update.

Track Update: Update the filter using tracked features of all poses.

- When a tracked feature point go out of view in the sliding window camera frames, all tracks associated with it marginalized out
- When the tracked feature reaches a maximum track length. The maximum track length would be 30 for a feature since the filter state vector has only 30 camera poses.

Pose Update: Update the filter using a subset of camera poses stored in the state vector.

- At each state augmentation step state vector fills up with camera poses. After 30 camera poses get filled up subset of camera poses should be removed in order to keep the computation bounded.

The accuracy of the filter depends on the track update. Therefore, all the out of view features and maximum track length reached features are marginalized out. The track updates vary in length, but a minimum threshold is set. To ensure that the marginalizing features are seen by at least two camera frames to enable triangulation. The maximum track length act as a tuning parameter and is greater than or equal to the number of camera states in the state vector.
The pose marginalization step gives rise to the freedom of choosing a set of marginalization camera states. The states have to be marginalized out in such a way that it optimally balance out the trade-off between computational cost and accuracy. Many developments have been proposed since the first formulation suggested by Mourikis and Roumeliotis in [3]. Section 4.5 will discuss the methods of choosing the states to marginalize in the pose update step.

2.6 Image Bearing Measurements for Pose Update

One accurate way of defining image-bearing measurements is the geometric constraints between the images and the observing static feature. Widely used geometric constraints are listed below.

3D feature point estimation using multiple camera poses:

- Algorithm proposed in [3] 3D feature point back-projection error is considered as the measurement. This method is more accurate since it uses multiple poses to estimate the 3D point using a least-square minimization method.

3D feature point estimation using two camera poses:

- The methods proposed in [4] and [31] use the same image measurement but utilizes only two camera poses which are optimally selected using the key-framing strategies discuss in the next section. Theoretically estimation accuracy degrades in this approach because it only uses two camera poses but computationally it is the most efficient.

Epipolar constraint using two camera poses:

- Epipolar constraint between two, 2D image features can also be utilized as a geometric constraint [33]. This method is highly computationally efficient among the three methods but should have less accuracy due to the reduced number of camera poses used for the updates.

2.7 Subset of Camera Pose Selection

Selecting a subset of camera poses is key for the accuracy and efficiency of MSCKF based VINS algorithms. Using more camera states improve the accuracy of the update, but it increases the computational cost. Several methods have been evaluated in the literature to select the subset of camera poses. These methods are summarized below.

Sparse pose selection:

- The MSCKF formulation presented in [3] proposes to select the one-third of the camera states evenly spaced in time. The oldest camera pose is kept in the state vector since the geometric features involved further back in time has a larger baseline with the new measurements. These techniques exhibit higher accuracy but the computational cost of the filter increases, making it less desirable to deploy in resource-constrained systems.

Key-frame based pose selection:

- The key-framing strategy was first proposed in the work of [28] for optimization-based VINS. Rather than using the entire marginalization window in the non-linear optimization, they propose to use a set of past camera poses marked as key-frames. This method significantly reduces the computational cost in the optimization algorithm [4, 31].

The key-frame based pose selection strategies identify camera-poses as key-frames, but the pose update cannot be done with all the identified key-frames because it increases the computational complexity. Therefore, an efficient camera pose selection method is needed. *Two-Way Marginalization* approach has been used in [4].
In MSCKF-Mono [31] formulation also has a similar pose selection algorithm. It also checks for key-frames when the filter reaches its maximum camera poses in the state vector.

2.8 Key-frame Selection Strategies

Since key-frames gave valuable insights to the optimum pose selection for marginalization, it has been incorporated in pose update of MSCKF-based VINS algorithms. Since the MSCKF algorithm stochastically clone the camera poses into the state vector, many algorithms attempt to identify the key-frames in real-time. Summary of the successfully implemented key-framing strategies are outlined below.

Average feature disparity

- In [28] introduced a simple feature disparity threshold to identify key-frames. If the matched feature disparity is greater than a predefined

threshold, those camera poses are marked as key-frames and used in the pose update of the filter. This technique, however, does not offer sufficient robustness since it only uses image measurements, ignoring the relative motion between the camera poses.

Average parallax and tracking quality

- If the average parallax of the latest frame and the last key-frame is beyond a fixed threshold, and the number of tracked features between the latest frame and the last key-frame is beyond a predefined threshold, latest frame is considered as a key-frame. Rotation between the frames is removed by integrating the gyroscope reading for a short period of time. This method relys on image-feature as well as the relative camera poses [26]. Although it takes into account the relative camera poses, the computation complexity of the algorithm is high.

Relative position and orientation

- Relative position and orientation are calculated between the previous key-frame and the latest camera frame. If it goes above a certain threshold latest frame will be considered as a key-frame. This method is successfully implemented in [4] and [31].

It is evident from the above information that many algorithms have been developed in order to reduce the computational cost of the pose update in MSCKF based VINS filters. The selection of 10 poses as suggested in original MSCKF algorithm at the pose update step, incur high computational cost. A key-frame based pose update only selects two poses for update. Due to this reason VINS algorithm efficiency increases. The recent comparison work of Delmarico *at el.* [29] has shown that, the MSCKF algorithm have similar accuracy when compared to other state-of-the-art VINS algorithms (changes in the accuracy is in the order of few centimeters). However, the available comparisons are algorithm vice comparisons. Not much attention is given to compare the different state-marginalization techniques in VINS algorithms. This thesis evaluates these different marginalization techniques and associated bearing measurement models to qualitatively evaluate and identify a suitable VIN filtering solution for MAV navigation. For this purpose, an online VINS benchmark dataset is used [41] and an experimental setup is designed for experimental validation of the VINS algorithms.

Chapter 3

Calibration and Validation of VINS Sensor Setup

3.1 Motivation

When combining the IMU and camera measurements, the spacial and temporal calibration parameters of the sensor needs to be estimated with the highest possible accuracy for the state estimation algorithm to to generate pose estimates accurately. Spatial calibration is the process of estimating the *rigid-body transformation* between the camera and the IMU. Due to clock-synchronization errors, delays in transmission, irregular hardware triggering, IMU and camera data streams develop a time offset. Temporal calibration is performed to align the timestamped data streams from the IMU and camera. In this chapter, a calibration method and a validation technique is presented for a VINS sensor setup to be used in VINS experiments. For this purpose the following main steps are completed:

- Select a VINS sensor for calibration.
- Evaluation of a calibration method.
- Hardware development and calibration procedure of the selected VINS sensor.
- Experimental validation of the calibration method.

3.2 Available VINS Sensor-Units

VINS sensor-unit should be compact and light-weight, for it to be used in MAVs. Since compact VINS senors are not commonly available off the shelf, many reported work develops in-house custom experimental setups. Kelly *at el.* [59] have developed a hand-held sensor unit. The camera used is a black and white Flea FireWire model from Point Grey Research (640×480 pixel resolution), mated to a $4mm$ Navitar lens ($58°$ horizontal FOV, $45°$ vertical FOV). The IMU used was a MEMS-based $3DM - GX3$ unit, manufactured by MicroStrain [60], which provides three-axis angular rate and linear acceleration measurements at 100 Hz. The platform was built with the purpose of evaluating a calibration algorithm, therefore it lacks compactness for MAV applications.

Using the DJI A3 flight controller, Intel i7 NUC and an NVIDIA TX1, Lin *at el.* in [16] have developed a visual-inertial sensor-unit. The integration of a minicomputer has given them sufficient processing power for the algorithms. The system has been assembled together to be used as a compact module. However, the sensor-unit consists of two on-board computers (Intel i7 NUC and NVIDIA TX1), which adds more weight to the system. Additionally, it consumes more power. The system uses the IMU inside the A3 flight controller for VINS system. Many other components of A3 flight controller will be redundant if it is going to be used as a stand-alone VINS sensor-setup. Therefore, A3 flight controller will be an over-design for the requirement.

Work presented in [61], develops a sensor platform using an off-the-shelf PixHawk MAV. It has a Hardkernel Odroid-U3 mini computer. The system has been calibrated by the manufacturer. Extrinsic and intrinsic parameters have been calculated and pre-programmed in the sensor-unit. The MAV and the sensor-unit operates as a single device making it compact, light-weight and less power-consuming system. However, the sensor-unit itself is not designed to be compatible with other MAVs or computer platforms. Therefore, deploying it for general use is a complex and a time consuming task.

The work presented in [62] shows a sensor-unit development for SLAM applications. It combines an IMU with a stereo camera. The system implementation is accomplished on a System on Chip (SoC) device that synchronizes the complementary metal-oxide semiconductors (CMOS) images and IMU data time-stamped at hardware level to enable precious synchronization. It has an field programmable gate arrays (FPGA) pre-processed image processing front-end, which enables visual feature detection on

the sensor-unit itself, reducing the computational burden on VINS algorithms. The sensor-unit is also compact and light-weight. Therefore it will be an ideal sensor to be used in MAVs. Unfortunately, production of the sensor-unit is discontinued 3 years ago.

The Parrot SLAM DUNK is introduced to the market as an integrated VINS module by Parrot Inc. It claimed to have the capability of providing the sensor data for autonomous navigation of Parrot Drones. It was released with an open source license [63]. The sensor has wide FOV stereo cameras with integrated IMU and NVIDIA GPU for fast graphics processing. Due to hardware faults of the design, manufacturing of the sensor has been discontinued.

One of the successful implementations of a VINS sensor-setup has been developed by Sa *et al.* in [64]. The setup consists of a sensor and a computational unit. An Intel ZR300 sensor module is used as the sensor. The computational unit that has been selected was an Intel NUC i7 quad core 2.5 Ghz, 16GB RAM mini computer. The ZR300 sensor module has the IMU and the cameras rigidly integrated to a single circuit board. As a result, the rigid body transformation between the camera and the IMU is highly reliable. Sa *et al.* indicates that high reliability of the rigid body transformation between the camera and the IMU was the main reason for their successful sensor calibration.

In this work we use the ZR300 sensor (shown in Figure 3.1) for experimental testing of VINS algorithms. The ZR300 features a factory calibrated sensor with low power, light-weight and relatively low cost. It is important to note that there has been several newer versions of VINS capable sensor units from Intel which are in the market [65,66] . We selected the ZR300 for VINS experiments in this work as ZR300 was the most reliable and reasonably priced option available at the time of this study.

Figure 3.1: Intel ZR300 sensor

3.3 Hand-held VINS Sensor Unit

A hand-held platform, as shown in Figure 3.2, is developed using ZR300 sensor to carry-out calibration and validation. The ground truth pose data of this hand held VINS unit was captured using an Opti-track motion capture system by placing markers on top of the ZR300 sensor.

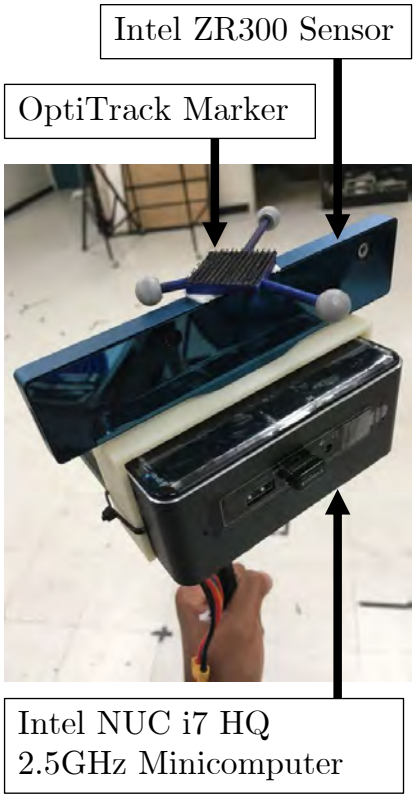

Figure 3.2: Hand-held sensor unit developed

Sensor Unit Specifications

Table 3.1: List of components used in the sensor unit

	Part	Component used	Specifications
1	Sensor	Intel RealSense ZR300	IMU, Fisheye Camera
2	Processing Unit	Intel NUC core i7	16GB Ram, 500GB SSD
3	Battery	6-cell Lipo battery	22.2V 4500 mAh
4	Voltage Regulator	10V-50V buck converter	Output 12V/10A

Intel ZR300 has one fisheye camera with a FoV of 133° and 100° horizontal and vertical respectively. It streams a 640×480 image at 60 frames per second. The components of the sensor module are shown in Figure 3.3. The on-board IMU provides accelerometer and gyroscope measurements at a rate of 250 Hz. The two IR cameras and the depth sensor of ZR300 was turned off for VINS experiments to save power.

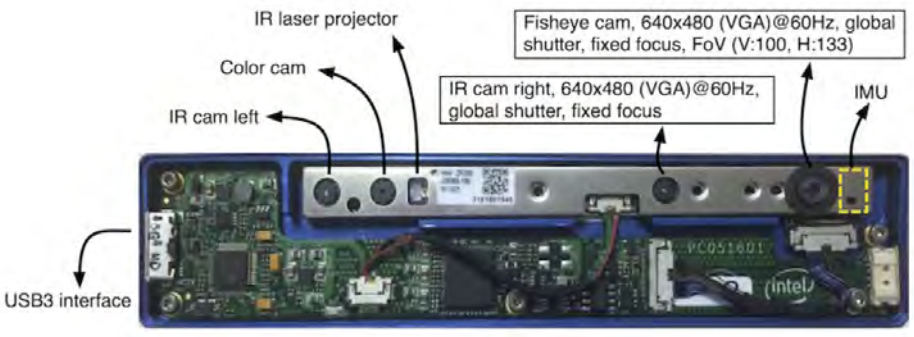

Figure 3.3: Intel ZR300 sensor configuration(front cover removed) [1]

3.4 Sensor Calibration

Finding the spatial and temporal relationships between the camera and the IMU is carried out using a VINS sensor calibration procedure. Functionality of the VINS algorithm depends on the calibration accuracy of the sensor. Therefore, both of the relationships are to be found accurately. The calibration process of the ZR300 sensor is discussed below.

3.4.1 IMU-Camera Temporal Relationship

The timestamp alignment of IMU data and camera data is found during temporal calibration. The IMU and camera data should be timestamped in order to align them. Intel Inc. has provided an open-source driver for ZR300 sensor to achieve this task. The driver timestamps data from the sensor itself and stream it out to the operating system (Ubuntu 16.04).

In the driver specification it claims that the IMU data are streaming at 250 Hz and camera data are streaming at 50 Hz. However, the timestamp between the camera-IMU were not aligning. Work presented in [2] states that due to the clock-skew, an unknown time offset t_d exists between the IMU and the camera timestamps. The time offset parameters are illustrated in Fig. 3.4 below.

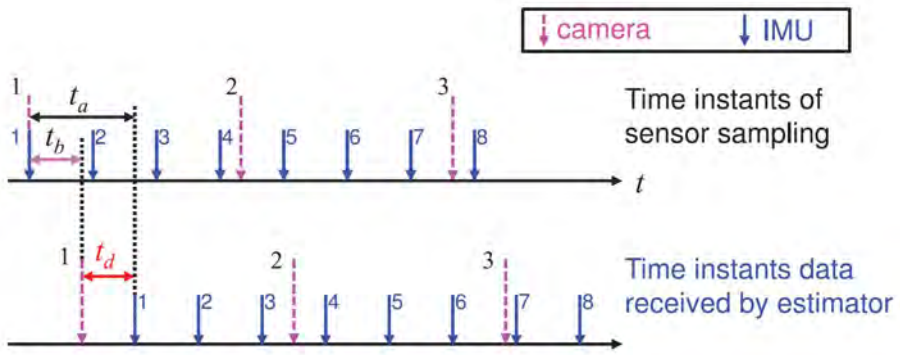

Figure 3.4: Example of time offset arising due to latency in the sensor data [2]

In this case the IMU data arrives with a latency t_a, while the camera data have a latency t_b, both of which are unknown. Since t_a and t_b are different, the sensor outputs measurements that were recorded simultaneously (e.g. the first IMU and camera measurement) with timestamps that are offset by $t_d = t_a - t_b$ [2]. Estimation of t_d should be done as part of temporal calibration.

Furthermore, the streaming frequencies of the IMU and camera data are not consistent when visualized in *ROS rqt plot* [67]. During testing, it was clearly evident that the original driver provided was not accurately capturing the timestamps of the data. In order to find t_d data should be streamed at a constant frequency. Work presented in [68], has written a custom driver for ZR300, fixing the issue of sensor data frequency inconsistency. It has changed the Fisheye image publishing frequency to 30 Hz.

3.4.2 IMU-Camera Spacial Relationship

Spatial relationship between the IMU and camera needs to be determined. The process of finding spacial relationship is also equivalent to the extrinsic camera calibration. Figure 3.5 below shows the camera extrinsic parameters* in red. It is the homogeneous transformation of camera frame $\{C\}$ to IMU frame $\{I\}$.
Static transformations were given in the Realsense datasheet provided by Intel Inc. [69], but it does only captures the nominal value which should be refined using a separate calibration. Therefore, an extrinsic calibration procedure was carried-out for the sensor unit.

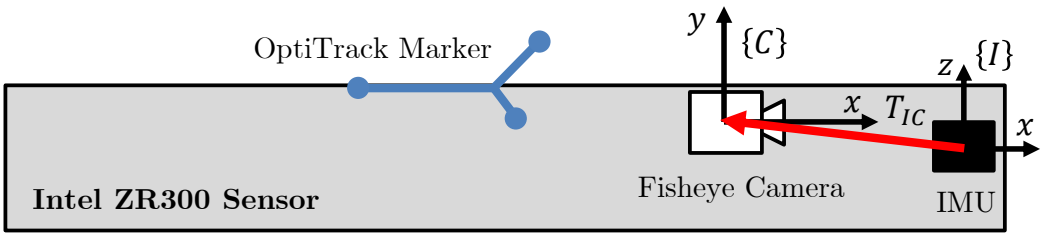

Figure 3.5: Extrinsic Calibration for ZR300 - Estimation of T_{IC}

3.5 Kalibr - Open source Camera-IMU Calibration Toolbox

In order to calibrate the camera-IMU sensor setup, an open-source calibration toolbox has been used. It is a calibration toolbox created by Furgale *et al.* [78]. Kalibr has been installed on the Intel NUC mini computer in the sensor-unit itself. Kalibr works as a ROS package in ROS Kinetic, running on Ubuntu 16.04 operating system. A summary of the Kalibr system with inputs and outputs is shown in the Figure 3.6 below.

*The rigid body transformation between the camera and the IMU is known as the camera extrinsic parameters

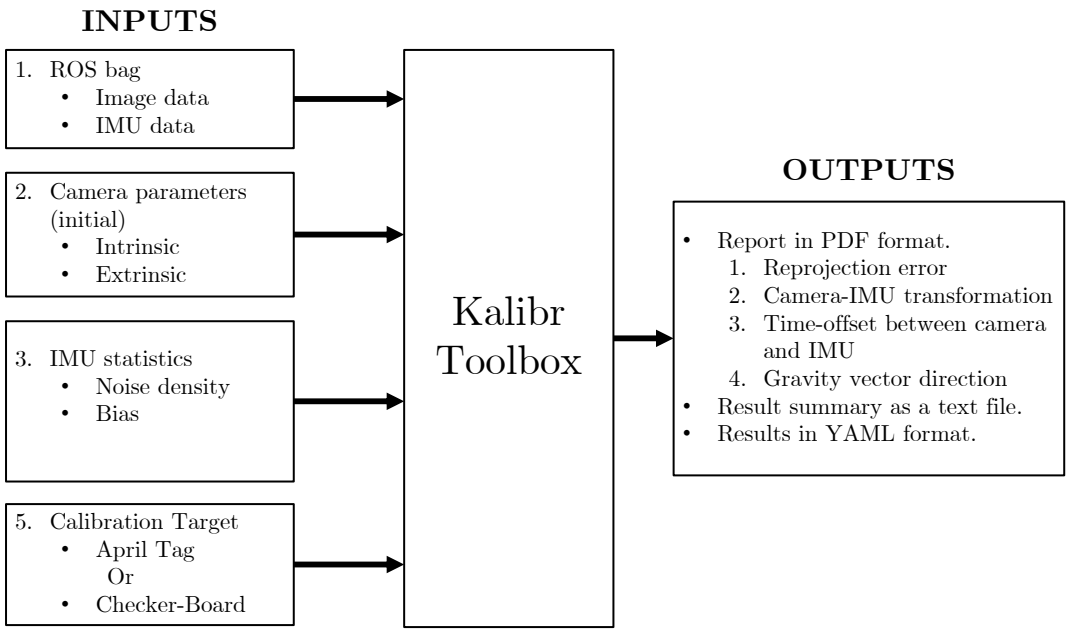

Figure 3.6: Kalibr Toolbox Overview

There are many parameters to be considered when performing IMU-Camera calibration. In the work carried out in [70], has stated that the calibration target and the excitation trajectory are critical for calibration accuracy. Most commonly used calibration target is a checker board with unequal sides. A checker board has good performance for intrinsic camera calibration but is susceptible to motion blur during fast movements [71].

Figure 3.7: Checker board target used for initial calibration

In terms of actuation patterns, work presented in [59] states that the IMU-Camera sensor-unit should be rotated while traveling along corkscrew-like trajectory to activate all the 3 DOFs of the accelerometer. The Figure 3.8 shows the excitation pattern that has been used in the calibration process of the sensor-unit. All the three axis of IMU is excited in this motion pattern. However, when the same method was used during our testing, the calibration parameters were not accurate.

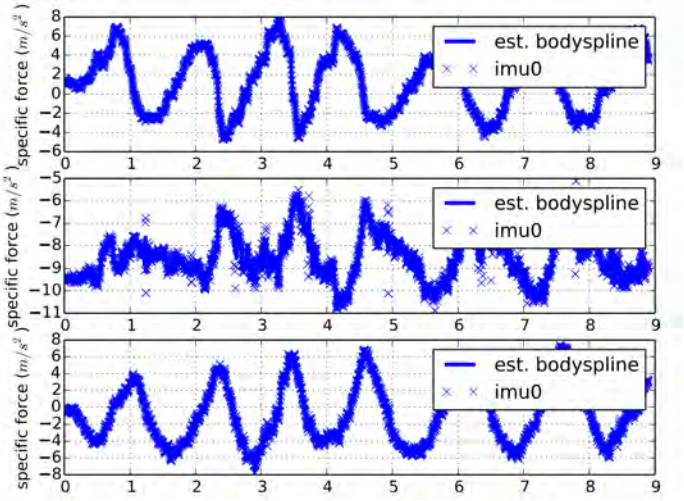

Figure 3.8: IMU actuation pattern for each x, y and z axis are shown in order, starting from top graph

3.6 Calibration Results

We initially used a checkerboard calibration target to perform calibration as this is the most common type of target used. Following is the extrinsic calibration matrix for this calibration attempt,

$$T_{IC} = \begin{bmatrix} 0.9977 & 0.02792 & -0.0610 & 0.0365 \\ -0.0267 & 0.9994 & 0.0204 & -0.0225 \\ 0.0616 & -0.0187 & 0.9979 & 0.1023 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \big[R\big]_{3\times3} & \big[t\big]_{3\times1} \\ \big[0\big]_{1\times3} & 1 \end{bmatrix}$$

The t block of the homogeneous transformation matrix gives the translation between the camera and the IMU. It comes as 3.6 cm in x-direction, -2.3 cm in y-direction and 10.2 cm in z-direction [70]. The R block corresponds to the rotation of camera

with respect to the IMU. However, the approximate physical location of camera and IMU do not lie close to these values.
This means that the calibration method using the checker board and generic screw trajectory motion profile has not performed a reasonable estimation of the calibration parameters. This can be seen in second column of Table 3.2 where the re-projection errors are at levels unacceptable for VINS systems.
To improve this result the following modifications were made.

1. Checker board was replaced by April grid calibration target
2. Sensor unit excitation pattern was modified
3. Length of the calibration ROS bag recording time was increased

An april grid calibration target [71] which is more robust to occlusion, warping, and lens distortion, was used for the calibration.

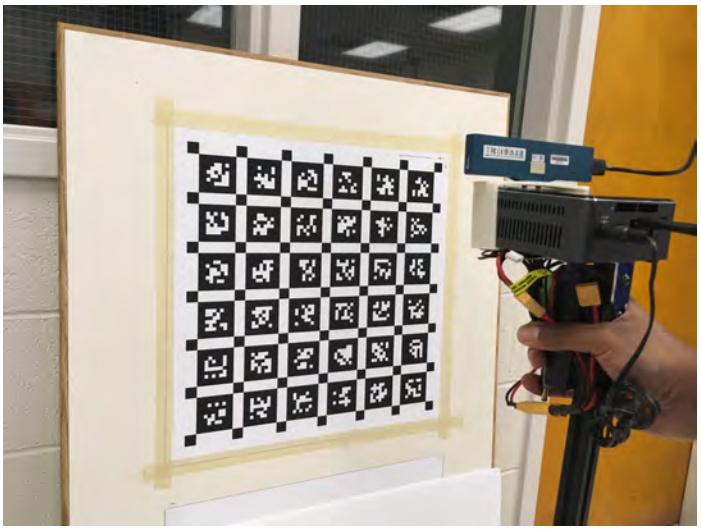

Figure 3.9: April grid calibration target used for the calibration

The excitation pattern of the sensor-unit also changed (see Figure 3.10). The corkscrew-like trajectory of the movement was not generating accurate results in Kalibr for a number of trials. Excitation of the IMU in one single axis at a time improved the estimation accuracy. The trajectory was identified by trial and error. Single DOF actuation at a time requires jerk free smooth transitions between DOFs, in order to have accurate calibration results.

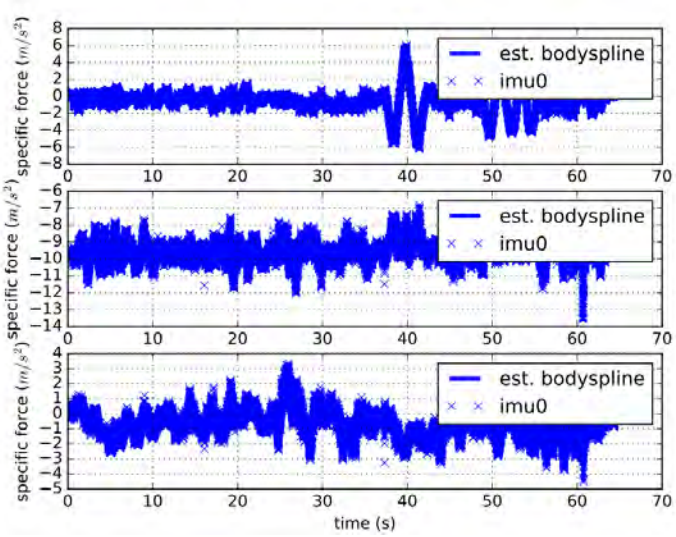

Figure 3.10: Modified excitation pattern - one axis at a time excitation (x-axis top graph, y-axis middle and z-axis bottom)

Length of calibration also has to be long enough for the IMU to be stabilized and should not be too long to the point where drift will be significant. Therefore, the optimal duration of excitation was found to between 1-2 minutes using trial and error.

After the proposed modifications the transformation matrix significantly improved,

$$T_{IC} = \begin{bmatrix} 0.9997 & 0.0235 & -0.0078 & -0.0021 \\ -0.0235 & 0.9997 & -0.0113 & -0.0078 \\ 0.0075 & 0.0114 & 0.9999 & 0.0084 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \left[R\right]_{3\times3} & \left[t\right]_{3\times1} \\ \left[0\right]_{1\times3} & 1 \end{bmatrix}$$

Table 3.2: Summary of Calibration Results

	Measurement	Attempt I	Attempt II
1	Reprojection error $[px]$	2.37409	0.3800
2	Gyroscope error $[rad/s]$	0.09078	0.0247
3	Accelerometer error $[m/s^2]$	0.20481	0.1423

The last column after the refined calibration reflects that -2.1 mm in x-direction, -7.8 mm in y-direction and 8.4 mm in z-direction [70]. Therefore, the modified cal-

ibration process yielded a better estimation with re-projection errors acceptable for VINS applications.

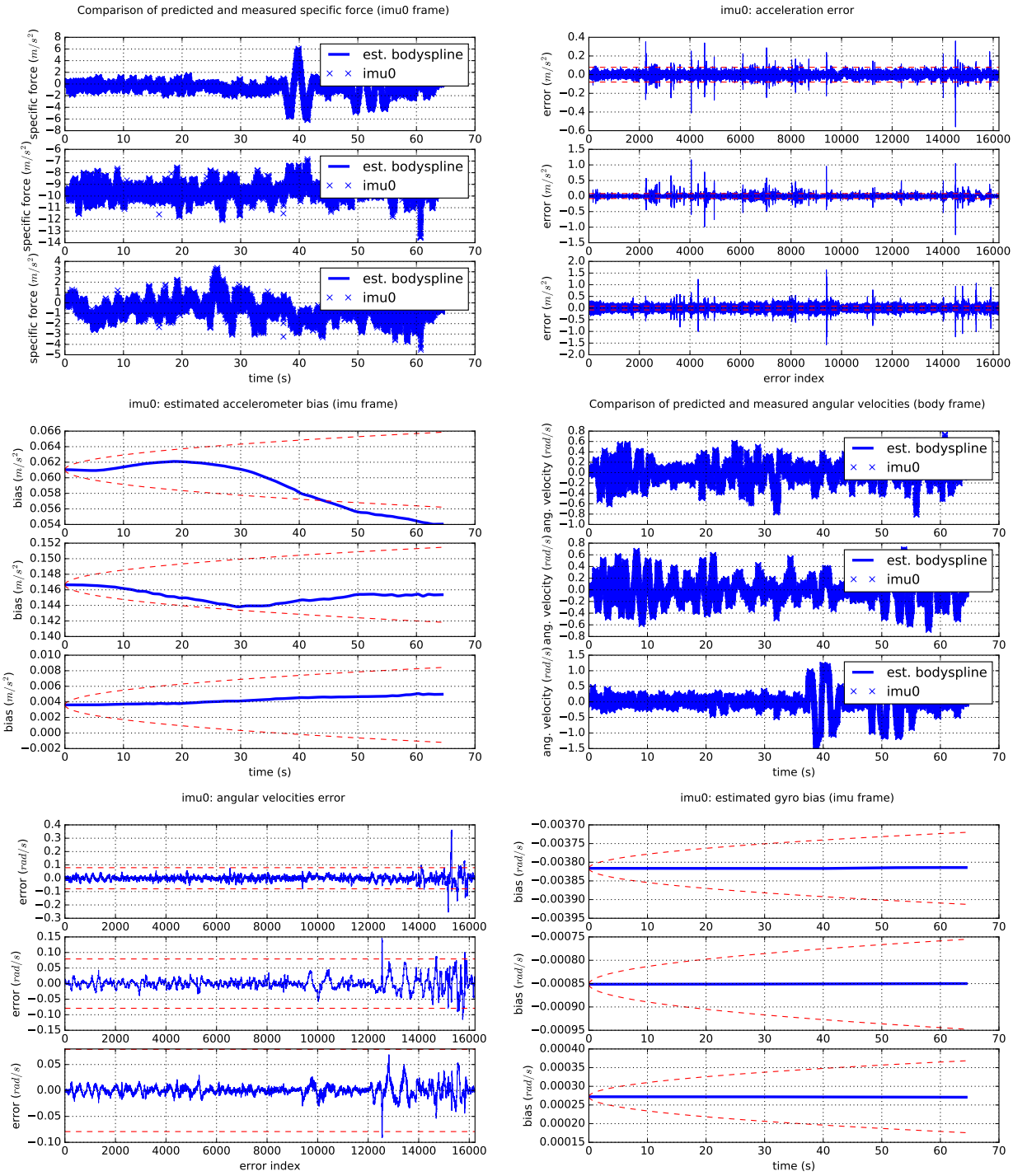

Figure 3.11: Estimation errors in blue and 3σ error bounds in red - All six graphs the x, y, and z axis are shown from top subgraph to bottom subgraph respectively

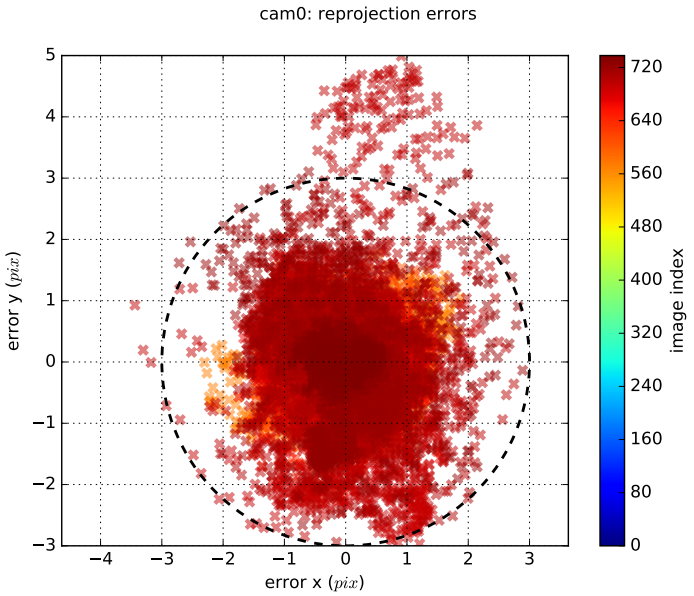

Figure 3.12: Re-projection Error in pixels

As showed in Figure 3.12 the projection error was mainly inside a three pixel error bound, which yields an acceptable error for VINS algorithms. The Figure 3.11 shows that all the state errors are inside 3σ error bounds, therefore the calibration parameters were reliable estimates.

3.7 Validation of Calibration

The validity of the calibration process was further verified by running a VINS algorithm using the found calibration parameters. Many state of the art VINS algorithms were available for public use. VINS-Mono [26] was selected for the purpose as the sensor-unit can be directly connected with the package. The VINS-Mono already had a configuration that was readily available for the ZR300 sensor to be connected. The calibration parameters of this configuration was modified to the ones found for the exact sensor we used during experiment.

A dataset was recorded by navigating the developed sensor-setup around the ground floor of the MUN Engineering building as shown in Figure 3.14. The IMU topic and the fisheye image topic were recorded in a ROS bag. The dataset was 6.51 minutes long.

It is an optimization based algorithm which has pose graph optimization and loop

closure techniques. The pose graph optimization and loop closure was turned off in order to make it run as an odometer for this dataset.
The estimates were accurately generated and the deviation from the actual path used in the experiment was minimal as summarized in Table 3.3 and as shown in Figure 3.13 and 3.15. This confirms the validity of the calibration parameters found using the calibration process of this chapter.

Table 3.3: VINS-Mono estimation and the total drift for the MUN Engineering building ground floor corridor dataset.

	Direction	Drift in meters
1	X-direction	5.1800
2	Y-direction	0.7150
3	Z-direction	0.0502

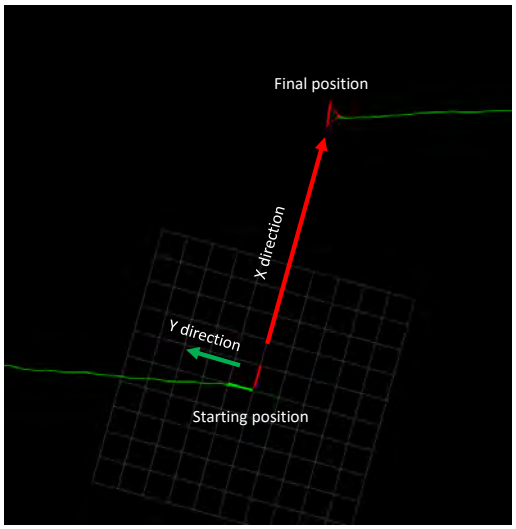

Figure 3.13: Drift on each direction after the filter run

The total length of the dataset was 252 m to the nearest whole number. Accumulated total drift was 5.95 m. The drift was only 2.36%. This results clearly validate the accuracy of the calibration.

MEMORIAL UNIVERSITY

Appendix A: Floor Plans-Level 1

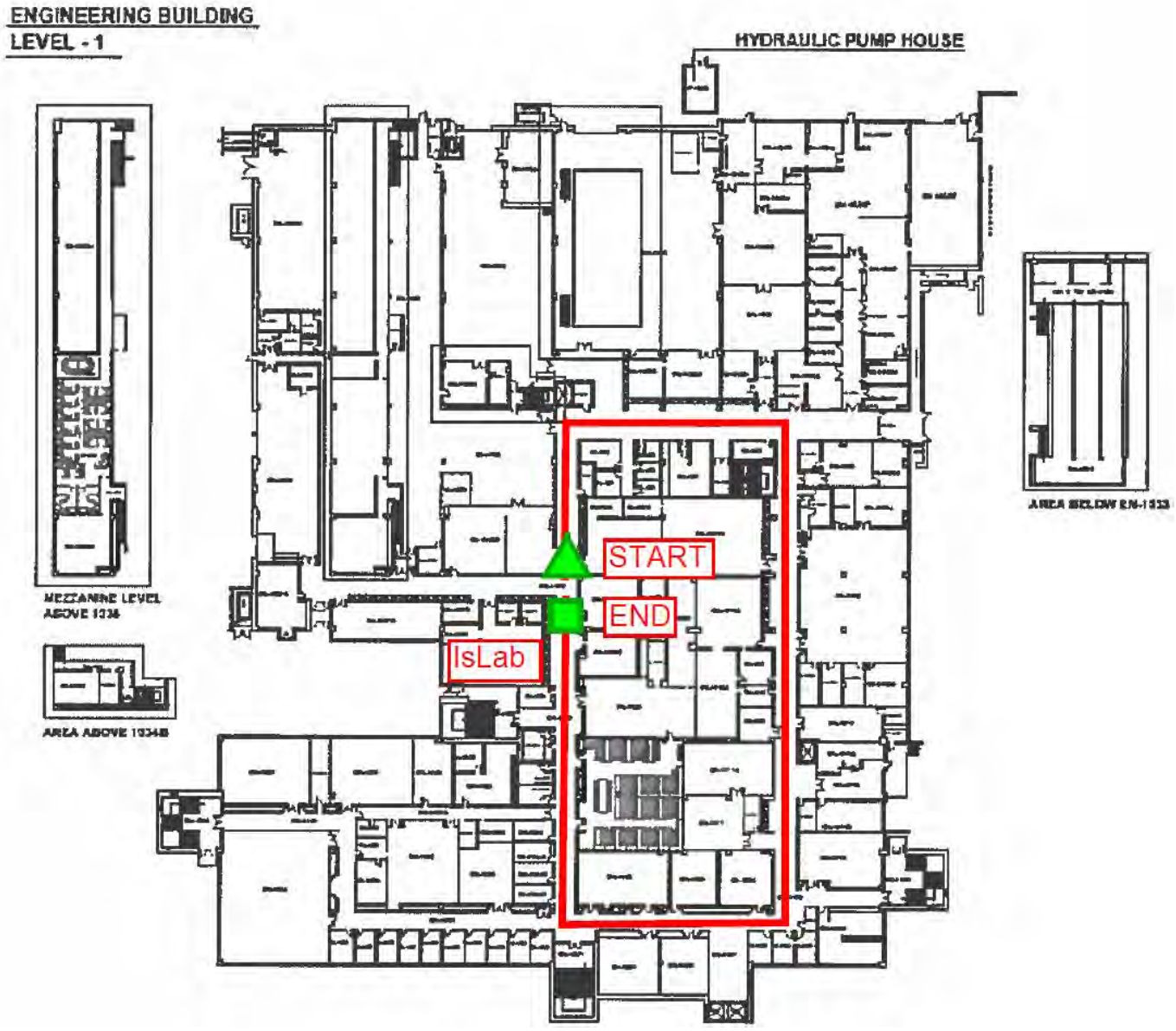

Figure 3.14: Actual dataset path for calibration dataset taken at Engineering building Corridor - Level 1

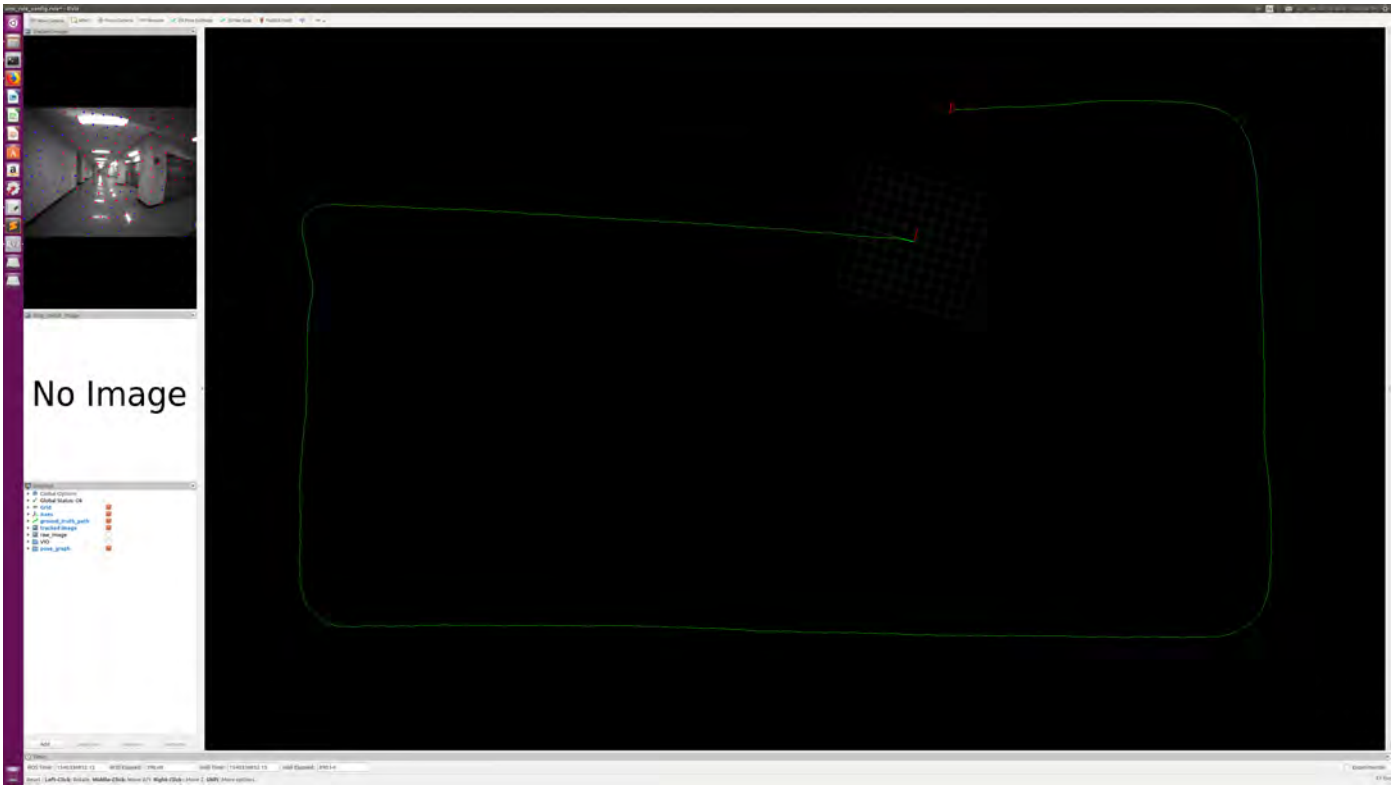

Figure 3.15: VINS-Mono estimated path of Eng Building ground floor corridor

3.8 Conclusion

This chapter performed the calibration of VINS sensor-setup developed using an Intel ZR300 sensor module and an Intel i7 NUC mini computer. The following methods were used to calibrate the sensor-unit. (1) Open-source Kalibr calibration toolbox is used with an April Tag calibration target, (2) the IMU excitation pattern used was single axis excitation at a time. The results were validated using VINS-Mono open-source ROS package. The developed sensor will be used for follow up studies and validation of the performance comparison detailed in this thesis.

Chapter 4

MSCKF based VINS

This chapter will describe the formulation of VINS algorithm in detail. The state marginalization methods, measurement models and mathematics behind the algorithms will also be presented in this chapter.

4.1 MSCKF Filter Description

The VINS algorithm introduced in Chapter 1, is MSCKF algorithm. The names derives from the fact that it keeps multiple states in the state vector and define constraints for them during the execution of the algorithm. Dynamics of the MSCKF is derived using the ESKF framework.
As outlined in Figure 1.1, the filter predict the next state of the platform by integrating the IMU measurement inputs, using the kinematic model of the platform. The image features are taken as the measurements to update the filter.
Many modifications has been introduced to the originally proposed algorithm in [3]. The modifications enables the algorithm to handle highly dynamic motions of robotic platforms like MAVs. More detailed version of the algorithm will be introduced later in the this chapter. In the next sections the filter formulation will be introduced.

4.1.1 MSCKF State Vector

The state propagation of MSCKF is performed using ESKF framework using the IMU measurements as inputs. The state vector proposed in the first formulation of MSCKF

in [3] contains the IMU states ($\mathbf{X}_{IMU}$) and N camera poses.

$$\mathbf{X}_{IMU} = \begin{bmatrix} {}^{I}_{G}\mathbf{q}^{\mathrm{T}} & \mathbf{b}_g^{\mathrm{T}} & {}^{G}\mathbf{v}_I^{\mathrm{T}} & \mathbf{b}_a^{\mathrm{T}} & {}^{G}\mathbf{p}_I^{\mathrm{T}} \end{bmatrix}^T \tag{4.1}$$

where the quaternion ${}^{I}_{G}\mathbf{q}$ represent the rotation from body frame to the inertial frame. In our formulation we define the body frame as same as the IMU frame. The vectors ${}^{G}\mathbf{v}_I$ and ${}^{G}\mathbf{p}_I$ represent the velocity and the position of body frame with respect to the inertial frame. Bias of the estimated acceleration and the gyroscope are represented by the vectors $\mathbf{b}_a$ and $\mathbf{b}_g$ respectively. $\mathbf{N}$ represent the number of camera states in the state vector. In filter $\mathbf{N}$ camera states are considered in the state vector and the complete state vector can be given as,

$$\mathbf{X} = \begin{bmatrix} \mathbf{X}_{IMU} & X_{C_1}^T & \dots & X_{C_N}^T \end{bmatrix}^{\mathrm{T}} \tag{4.2}$$

4.1.2 MSCKF Process Model

Using the dynamics of IMU, continuous time state dynamics can be given as,

$$\dot{\mathbf{X}} = f(x,u,w) = \begin{bmatrix} {}^{I}_{G}\dot{\mathbf{q}} \\ \dot{\mathbf{b}}_g \\ {}^{G}\dot{\mathbf{v}}_I \\ \dot{\mathbf{b}}_a \\ {}^{G}\dot{\mathbf{p}}_I \end{bmatrix} = \begin{bmatrix} \frac{1}{2}\boldsymbol{\Omega}(\boldsymbol{\omega}_m - \boldsymbol{b}_w + \boldsymbol{n}_w){}^{I}_{G}\mathbf{q} \\ \mathbf{n}_{wg} \\ C({}^{I}_{G}\mathbf{q})^{\mathrm{T}}(\boldsymbol{a}_m - \boldsymbol{b}_a + \boldsymbol{n}_a) + {}^{G}\mathbf{g} \\ \mathbf{n}_{wa} \\ {}^{G}\dot{\mathbf{v}}_I \end{bmatrix} \tag{4.3}$$

where $\boldsymbol{\omega}_m$ and $\boldsymbol{a}_m$ are the measurement readings from the gyroscope and the accelerometer receptively. $\boldsymbol{n}_w$ and $\boldsymbol{n}_a$ are the noise of gyroscope and accelerometer, while $\boldsymbol{b}_w$ and $\boldsymbol{b}_a$ defines the bias of gyroscope and accelerometer respectively and $\mathbf{n}_{wg}$ and $\mathbf{n}_{wa}$ defines rate of change of bias of accelerometer and gyroscope respectively. In addition to that, $\boldsymbol{\Omega}$ is defined as,

$$\boldsymbol{\Omega}(\boldsymbol{\omega}) = \begin{bmatrix} -[\boldsymbol{\omega}]_\times & \boldsymbol{\omega} \\ -\boldsymbol{\omega}^T & 0 \end{bmatrix} \tag{4.4}$$

where $[\boldsymbol{\omega}]_\times$ is the skew symmetric matrix of $\boldsymbol{\omega}$. In (4.3) $C(\cdot)$ represent the conversion from quaternion to the corresponding rotation matrix.

Noiseless version of the continuous-time state-estimate propagation equations reads as;

$$\dot{\hat{\mathbf{X}}} = f(\mathbf{x}, \mathbf{u}) = \begin{bmatrix} {}^{I}_{G}\dot{\hat{\mathbf{q}}} \\ \dot{\hat{\mathbf{b}}}_g \\ {}^{G}\dot{\hat{\mathbf{v}}}_I \\ \dot{\hat{\mathbf{b}}}_a \\ {}^{G}\dot{\hat{\mathbf{p}}}_I \end{bmatrix} = \begin{bmatrix} \frac{1}{2}\boldsymbol{\Omega}(\boldsymbol{\omega}_m - \hat{\boldsymbol{b}}_w){}^{I}_{G}\hat{\mathbf{q}} \\ \mathbf{0}_{3\times1} \\ C({}^{I}_{G}\hat{\mathbf{q}})^{\mathrm{T}}(\boldsymbol{a}_m - \hat{\boldsymbol{b}}_a) + {}^{G}\mathbf{g} \\ \mathbf{0}_{3\times1} \\ {}^{G}\hat{\mathbf{v}}_I \end{bmatrix} \tag{4.5}$$

where $\hat{\mathbf{b}}_g$ and $\hat{\mathbf{b}}_a$ are the estimated bias of the system.

4.1.3 MSCKF State Propagation

In the discrete time implementation, the IMU state stated in (4.5) is propagated using 4^{th} order Runge-Kutta (RK4) numerical integration. The RK4 integration step is given by

$$\mathbf{X}_{n+1} = \mathbf{X}_n + \frac{\Delta t}{6}\Big(k_1 + 2k_2 + 2k_3 + k_4\Big) \tag{4.6}$$

where the increment is calculated by assuming a slope which is the weighted average of the slopes $k_1, 2k_2, 2k_3, k_4$. More details can be found in [72].

4.1.4 MSCKF Covariance

Covariance of the MSCKF filter is calculated by using,

$$\mathbf{P} = \mathbf{F}\mathbf{P}\mathbf{F}^T + \mathbf{G}\mathbf{Q}\mathbf{G}^T \tag{4.7}$$

$\mathbf{F}$ is calculated as the partial derivative of the system model with respect to the error state evaluated at current system state.
The $\mathbf{F}$ is,

$$\mathbf{F} = \frac{\partial f(\mathbf{x}, \mathbf{u})}{\partial \delta\tilde{\mathbf{x}}}\bigg|_{\mathbf{x}=\hat{\mathbf{x}}} = \begin{bmatrix} -[\hat{\boldsymbol{\omega}}_\times] & -\mathbf{I}_3 & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ -C({}^{I}_{G}\hat{\mathbf{q}})^{\mathrm{T}}[\hat{\mathbf{a}}_\times] & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & -C({}^{I}_{G}\hat{\mathbf{q}})^{\mathrm{T}} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{I}_3 & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \end{bmatrix} \tag{4.8}$$

in a similar way $\mathbf{G}$ is calculated as,

$$\mathbf{G} = \frac{\partial f(\mathbf{x}, \mathbf{u})}{\partial \mathbf{n}_I}\bigg|_{\mathbf{n}_I=0} = \begin{bmatrix} -\mathbf{I}_3 & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{I}_3 & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & -C({}^I_G\hat{\mathbf{q}})^{\mathrm{T}} & \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{I}_3 & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \end{bmatrix} \tag{4.9}$$

where, $\mathbf{n}_I = [\mathbf{n}_g \quad \mathbf{n}_{wg} \quad \mathbf{n}_a \quad \mathbf{n}_{wa}]$. Details of the derivation and covariance propagation can be found in the work presented in [4].

4.1.5 MSCKF State Augmentation

Upon receiving an image the state should be augmented with the new camera state. The new camera pose can be computed from the last IMU state using,

$${}^C_G\hat{\mathbf{q}} = {}^C_I\hat{\mathbf{q}} \otimes {}^I_G\hat{\mathbf{q}} \qquad {}^G\hat{\mathbf{p}}_C = {}^G\hat{\mathbf{p}}_C + C({}^I_G\hat{\mathbf{q}})^{\mathrm{T}} {}^I\hat{\mathbf{p}}_C \tag{4.10}$$

and the augmented covariance matrix is derived as [3],

$$\mathbf{P} = \begin{bmatrix} \mathbf{I}_{6N+15} \\ \mathbf{J} \end{bmatrix} \mathbf{P} \begin{bmatrix} \mathbf{I}_{6N+15} \\ \mathbf{J} \end{bmatrix}^{\mathrm{T}} \tag{4.11}$$

where the Jacobian $\mathbf{J}$ is derived as,

$$\mathbf{J} = \begin{bmatrix} \mathbf{C}({}^C_I\hat{\mathbf{q}}) & \mathbf{0}_{3\times9} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times6N} \\ -\mathbf{C}({}^C_I\hat{\mathbf{q}})[{}^I\hat{\mathbf{p}}_C]_\times & \mathbf{0}_{3\times9} & \mathbf{I}_3 & \mathbf{0}_{3\times6N} \end{bmatrix} \tag{4.12}$$

4.1.6 Measurement Model

In this work, two measurement models are implemented to compare the accuracy and the performance. One being the Generic-MSCKF measurement model and the other is the Epipolar constraint based measurement model. In this section the measurement model descriptions will be introduced.

4.1.6.1 MSCKF Measurement Model

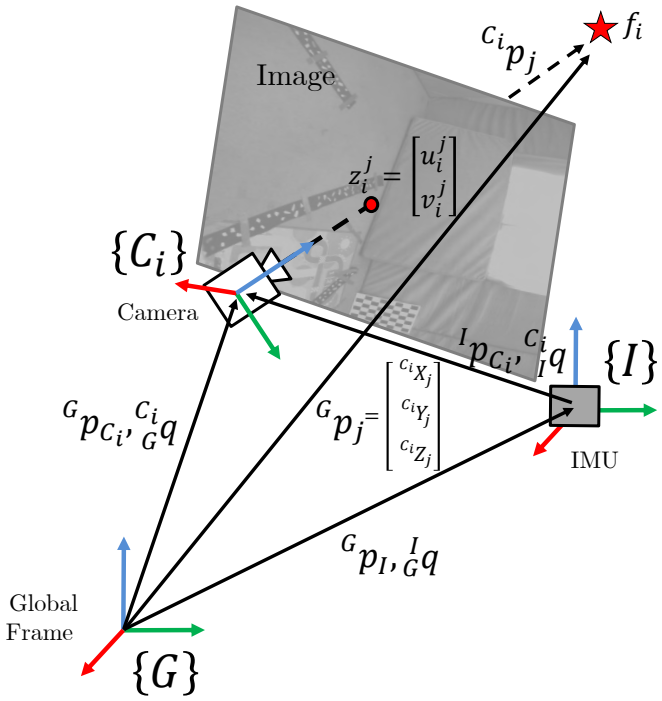

Figure 4.1: Measurement model for MSCKF filter

A single feature f_i can be observed by the camera $\left({}^{C_i}_{G}\hat{\mathbf{q}}, {}^{G}\hat{\mathbf{p}}_{C_i}\right)$. The camera measurement, $\mathbf{z}_i^j$ (image pixel measurement) is represented as,

$$\mathbf{z}_i^j = \begin{bmatrix} u_i^j \\ v_i^j \end{bmatrix} = \left(\frac{1}{{}^{C_i}Z_j}\right) \begin{bmatrix} {}^{C_i}X_j \\ {}^{C_i}Y_j \end{bmatrix} \tag{4.13}$$

The relationship between position of the feature to the camera pose $({}^{C_i}\mathbf{p}_j)$ can be given by,

$${}^{C_i}\mathbf{p}_j = \begin{bmatrix} {}^{C_i}X_j \\ {}^{C_i}Y_j \\ {}^{C_i}Z_j \end{bmatrix} = C\left({}^{C_i}_{G}\hat{\mathbf{q}}\right)\left[{}^{G}\mathbf{p}_j - {}^{G}\mathbf{p}_{C_i}\right] \tag{4.14}$$

Using the currently estimated camera pose, the feature position in the world frame ${}^{G}\mathbf{p}_j$ is calculated using the least-square minimization as suggested in [3] and [4]. The measurement residual can be approximated by linearizing at the current estimate as,

$$\mathbf{r}_i^j = \mathbf{z}_i^j - \hat{\mathbf{z}}_i^j = \mathbf{H}_{C_i}^j \tilde{\mathbf{x}}_{C_i} + \mathbf{H}_{f_i}^j {}^{G}\tilde{\mathbf{p}}_j + \mathbf{n}_i^j \tag{4.15}$$

where the $\mathbf{n}_i^j$ is the noise of the measurement. The covariance matrix of n^j can be given as $\mathbf{R}^j = \sigma_{im}^2 \mathbf{I}_{2M_j}$ ($2M_j$ represent the number of camera poses) since feature observations of different images are independent. The Jacobian with respect to the pose and feature position is given by $\mathbf{H}_{C_i}^j$ and $\mathbf{H}_{f_i}^j$ respectively. The details of the Jacobian matrix and the null-space calculations can be found in [4].

4.1.6.2 Epiploar Constraint Measurement Model

Epipolar constraint is defined between image pairs.

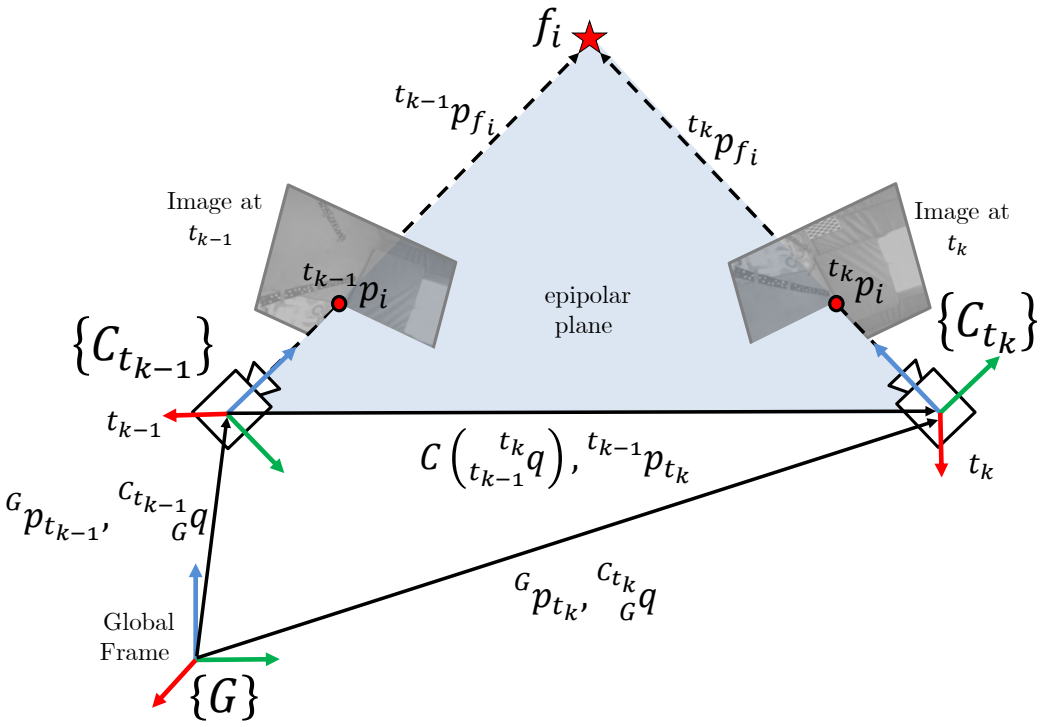

Figure 4.2: Measurement model for Epipolar Constraint

It can be defined for two images taken at t_k and t_{k-1} two time steps (current and previous) can be given following the notation of [33],

$$\begin{aligned} ({}^{t_k}\mathbf{p}_i)^T \mathbf{K}^{-T} \mathbf{E} \mathbf{K}^{-1} ({}^{t_{k-1}}\mathbf{p}_i) = 0 \\ ({}^{t_k}\mathbf{p}_i)^T \mathbf{K}^{-T} [\bar{\mathbf{p}}]_\times \bar{\mathbf{R}} \mathbf{K}^{-1} ({}^{t_{k-1}}\mathbf{p}_i) = 0 \end{aligned} \tag{4.16}$$

where $\bar{\mathbf{p}} = C\left({}^{C_{t_k}}_{G}\hat{\mathbf{q}}\right)\left({}^{G}_{C}\mathbf{p}(t_{k-1}) - {}^{G}_{C}\mathbf{p}(t_k)\right)$ and $\bar{\mathbf{R}} = C\left({}^{G}_{C_{t_k}}\hat{\mathbf{q}}\right)C\left({}^{C_{t_{k-1}}}_{G}\hat{\mathbf{q}}\right)$ where $\mathbf{K}$ is the camera intrinsic matrix. All the image coordinates are defined as homogeneous coordinates including the image pixel points ${}^{t_k}\mathbf{p}_i$ and ${}^{t_{k-1}}\mathbf{p}_i$.
The implementation can be done by simplifying essential matrix*.

$$\begin{aligned} \mathbf{p}_d &= {}^{G}\mathbf{p}_{C_{t_{k-1}}} - {}^{G}\mathbf{p}_{C_{t_k}} \\ \mathbf{E} = [\bar{\mathbf{p}}]_\times \mathbf{R} &= C\left({}^{G}_{C_{t_k}}\hat{\mathbf{q}}\right)[\mathbf{p}_d]_\times C\left({}^{C_{t_{k-1}}}_{G}\hat{\mathbf{q}}\right) \end{aligned} \tag{4.17}$$

The system is linearized at the current best estimate as suggested in ESKF formulations, which yields the following equation for to be used in the update. The $\hat{\mathbf{E}}$ is the essential matrix constructed from the estimated state.

$$\hat{h}_v = \left({}^{t_k}\mathbf{p}_i\right)^T \mathbf{K}^{-T}\hat{\mathbf{E}}\mathbf{K}^{-1}\left({}^{t_{k-1}}\mathbf{p}_i\right) \tag{4.18}$$

Following the derivation in [33], the image measurement covariance is defined as,

$$\mathbf{R}_{im} = \begin{bmatrix} \sigma_i^2 & \sigma_i^2 & 0 \end{bmatrix} \tag{4.19}$$

where σ_i^2 is the image measurement uncertainty standard deviation.

4.2 Statistical Measurement Validation

Due to the highly dynamics nature of the MAV systems sudden motions can generate outlying measurements. In addition to that sensor inaccuracies also contribute to outliers in the measurements. The MSCKF framework assumes the measurements to be Gaussian but this assumption is violated by the outliers. In [73] an automatic outliers rejection method was introduced. It has been developed taking into consideration the measurement and its covariance definitions.
Since the linearization is done with respect to the pose $\mathbf{H}^j_{C_i}$ and feature position $\mathbf{H}^j_{f_i}$, the measurement covariance is defined as follows, with $\mathbf{r}$ being the residual, $\mathbf{S}$ being the residual covariance $\mathbf{P}$ being the current covariance of the state estimate and $\mathbf{R_j}$

*The essential matrix generally defined as

$$\mathbf{E} = [{}^{t_{k-1}}\mathbf{p}_{t_k}]_\times C\left({}^{C_{t_{k-1}}}_{C_{t_k}}\mathbf{q}\right)$$

is the measurement covariance,

$$\begin{aligned} \mathbf{r} &= \mathbf{z} - \hat{\mathbf{z}} \\ \mathbf{S} &= \mathbf{H}_{C_i}^{j} \mathbf{P} \mathbf{H}_{C_i}^{j} + \mathbf{R_j} \end{aligned} \tag{4.20}$$

The automatic outlier rejection method in [73] is a hypothesis based validation test,commonly referred to as gating test. The formulation is based on the assumption that residual $\mathbf{r}$ is a multivariate Gaussian distribution with the covariance matrix $\mathbf{S}$. Therefore the sum of squares of $\mathbf{r}$ should follow a Chi-Squared ($\boldsymbol{\chi}^2$) distribution, with as many degrees of freedoms as the measurements. The test is to determine whether the residual falls inside the confident bounds of 95%, in the Chi-Square distribution as given in 4.21, where the parameter $\mathbf{r}^T\mathbf{S}^{-1}\mathbf{r}$ is known as the Mahalanobis distance squared.

$$\mathbf{r}^T\mathbf{S}^{-1}\mathbf{r} < \boldsymbol{\chi}^2(0.95) \tag{4.21}$$

When implementing in the MSCKF framework, gating test can be incorporated in two ways. Since the pixel error is the residual, one can define the test for each pixel in the *Pose Update* or *Track Update*. Using this approach would discard the erroneous pixel measurements. The null-space of the feature Jacobian matrix will then be evaluated for the entire set of measurements extracted from the image at that time-step.
The other method is to evaluate the gating test for the entire set of pixel errors (i.e for the entire track or the set of all marginalizing poses) after calculating the null space for each measurement independently. This method is used in the filter implemented in this work.

4.3 Observability Constraint

Consistency of the MSCKF depends on the system observability. System observability is used to determine whether the available measurements is sufficient to estimate the state of the system without any ambiguity [13]. If the observability matrix of the system is invertible, the system is proved to be observable. The directions spanned by the null space of observability matrix can be used to determine the measurements directions of the system.
Performing an observability analysis for the linearized VINS system summarized in

[13], shows four directions that are not observable. The translation with respect to a global frame along orthogonal coordinate axes and the global rotation about the gravity vector axis. Therefore MSCKF system model should be designed in a such a way that it also should have an unobservable subspace spanned by those four directions.

The MSCKF VINS has three unobservable states. Current state linearization of system and measurement model, makes the MSCKF to have three unobservable dimensions even though it has four dimensions. Therefore, the filter assumes information that are not available in the measurement. In MSCKF VINS, this is the rotation about z-axis (i.e. yaw angle).

Many solutions for this problem has been formulated, such as first estimate jacobian EKF (FEJ-EKF) [74], observability constrained EKF (OC-EKF) [24, 75, 76] and robocentric visual odometry (R-VIO) [5]. In the implementation of the MSCKF consistency of the filter is maintained by using observability-constrained method. The FEJ-EKF depends on the initialization of the filter, therefore accurate initialization is needed for the filter to maintain its consistency. In the R-VIO formulation the camera poses are expressed in the latest IMU frame. The drawback of this approach is that the uncertainty of the IMU-frame propagate the uncertainty of the camera-poses.

The original algorithm overview proposed in **Algorithm** 4.1 has been modified to improve its robustness and accuracy. The following will be a comprehensive description of the widely implemented MSCKF algorithm that has been used in the back end of [31].

4.4 Epipolar MSCKF Filtering Algorithm

As discussed in the preceding chapters, the selection of two poses allows the incorporation of essential matrix constraints between image pairs as measurements. The epipolar constraint is one of those methods derived to simplify the update rule in VINS filtering algorithms.

The formulation of the algorithm was different to the usual MSCKF formulation. It only keep one camera pose in the state vector. Therefore, the algorithm simplifies upto a greater extent. Many of the features remains the same as MSCKF algorithm, but the state and the covariance pruning is not required since it has only one pose which updates at each iteration. Only one type of update taking place in the filter.

Algorithm 4.1 Modified OC-MSCKF Algorithm

Initialization: Initialize using gravity direction approximation.
for $t = 1, \cdots, \infty$ **do**
    Propagation: State vector and covariance matrix using IMU measurements.
        • State vector propagation using RK5 integration.
        • Covariance propagation using OC-EKF method.
    if Image Registered **then**
        Feature Tracking and Warping: Recognition, Matching and Tracking
        State Augmentation: State vector and state covariance matrix with the current IMU position and orientation.
        Measurement Update:
        if Out of View or Max Track length **then**
            if 3D point estimation cost < max cost **then** Pass the tracks for the EKF track update
                if Measurement Cov < Mahalanobis distance threshold **then**
                    • ESKF Track Update
                end if
            end if
        end if
        if Pose Max **then**
            if 3D point estimation cost < max cost **then**
                • Pass the set of **keyframe poses** for the ESKF pose update.
                if Measurement Cov < Mahalanobis distance threshold **then**
                    • ESKF Pose Update
                end if
            end if
        end if
        Track Pruning: Remove feature tracks from the tracking history that has been used for Track Update.
        State Pruning: Remove the camera poses and the associated feature tracks that has been used for Pose Update.
        Covariance Pruning: Remove the respective camera covariance entries of the removing camera states.
    end if
end for

The revised algorithm is sanitized in **Algorithm** 4.2.

Algorithm 4.2 Epipolar OC-MSCKF Algorithm

Initialization: Initialize using gravity direction approximation.
for $t = 1, \cdots, \infty$ **do**
    Propagation: State vector and covariance matrix using IMU measurements.
        - State vector propagation using RK5 integration.
        - Covariance propagation using OC-EKF method.
    if Image Registered **then**
        Measurement Update:
        if First Image **then**
            - Update previous features from current features.
            - Update the previously stored IMU pose in state vector to the current IMU pose.
            - State Augmentation
            - Covariance Conditioning
        else
            Image Features: Recognition and Matching
            if Disparity > Disparity Thresh **then**
                Keyframe = **ture**
            else
                Keyframe = **false**
            end if
            if Keyframe **then**
                - ESKF Epipolar Constraint Update
                - Update the previously stored IMU pose in state vector to the current IMU pose.
                - State Augmentation
                - Covariance Conditioning
            end if
        end if
    end if
end for

4.5 State Marginalization Techniques

The keyframe selection strategies used in *Two Way Marginalization*, *MSCKF-MONO Marginalization* and *Two Keyframe Marginalization* are introduced in this section.

4.5.1 Two Way Marginalization

In the original formulation presented in [77] dictates two ways of adding the next camera pose to the sliding window of poses in the MSCKF algorithm. If one of the below criteria are satisfied:, the next pose will be added to the state vector.

1. Time between two images is larger than a certain predefined threshold.
2. After eliminating the relative rotation, the average parallax of all common features between the most recent two images is larger than a predefined threshold.

The first condition ensures that the error in the integrated IMU measurement between two camera states is bounded, while second condition ensures that the new camera state is added when translation motion of the vehicle with respect to the scene is significant.
The most recent implementation of this algorithm in [4] uses this technique to remove the poses. The camera states were added to the sliding window without checking the above criteria. After the sliding window of poses are filled up, two camera poses which do not satisfy this criteria will be selected and removed. This ensure that the sliding window satisfies the above mentioned criteria.
In the practical implementation the authors in [4] have used relative position and orientation as the criteria for pose removal, oppose to the originally posed criteria mentioned above. Practically there can be three scenarios for the pose selection. They are outlined below.

4.5.2 MSKCF-MONO Marginalization

The technique described here, check for keyframes in the camera poses at each iteration after the 30 poses filled up in the state vector. The non-keyframe poses are then recorded. If the non-keyframes reach two, the filter will be updated using these two poses. If all the poses in the state vector becomes keyframes, then the oldest two poses will be used to update the filter. After each update the two camera poses used for the update will be removed from the state vector.
The keyframe selection criteria is similar was the previous *Two Way Marginalization*. The relative translation and rotation between two consecutive camera poses should be lower than a certain pre-defined threshold for the poses to be removed.

4.5.3 Two Keyframe Marginalization

The two keyframes are selected based on the feature disparity between the two images. If the feature disparity is higher than a certain threshold, the current frame is considered as a key-frame and used to update the filter as discussed in Algorithm 4.2. Higher feature disparity simply means that the two camera frames are in higher relative translational distance.

This completes the background details related to the MSCKF formulation and its modifications considered in this thesis. The next chapter implements the generic and modified versions of the algorithm and establish performance implications of each to identify an efficient implementation for MAV applications.

Chapter 5

Results of Comparison Study

The main objective of this chapter is to provide the experimental evaluation results of the three modules (feature tracking-front end, state marginalization technique and complexity of the VINS algorithm) that dictates the complexity of the VINS algorithm. Thereby, the optimum combination of the modules to achieve an efficient VINS algorithm is proposed, which can be implemented in MAVs. Two different state-of-the-art feature-tracking front ends and four different state-of-the-art state marginalization techniques were compared.

5.1 Evaluation Setup

5.1.1 Hardware Platform

The evaluation was carried out using a Dell D7 laptop with a i5-7300HQ Quad Core processor with multi-threading, operating at 2.5 GHz with 16GB of RAM, which is a commonly available workstation range laptop computer. It is important to note that after development, the algorithms should be implemented on embedded computing platforms for MAV deployment.

5.1.2 Dataset

The EuroC MAV dataset [41] is used for this study since it includes full spectrum of 6 DOF flying trajectories A manually piloted MAV is used to record eleven visual-inertial sequences, in three different indoor environments. Each environment sequence

has increasing qualitative complexity. "Vicon room 1-difficult" dataset is more challenging than "Vicon room 1-easy" dataset, with aggressive motions and varying illumination conditions. More aggressive motions of the flying robot makes it difficult to generate accurate image measurements.
The sensor data captured comprises of stereo WVGA monochrome images at 20 Hz, and temporally synchronized IMU data at 200 Hz. In the work presented here only uses the left camera since this is a monocular algorithm evaluation (the dataset has stereo images recorded from left and right cameras both). For all the datasets the ground-truth positioning data is available in two formats. In Vicon room data sequences, it is provided using a vicon motion capture system while in Machine Hall sequences it is given by a Leica MS50 laser tracker.

5.1.3 Software Environment

The laptop is setup with Matlab 2018a in windows 10. The feature-tracker data were imported to matlab environment from open-source ROS packages available online. The sequences were presented to the filters the same way MAV receive data real-time, when its flying. The track data were also populated and presented to the filter the same way, the features are recognized and tracked in real-time.
The receiving of an image was simulated at image receiving frequency in the dataset, and feature tracking was also simulated as happening in real-time from the imported track data from ROS. Upon receiving an image, the update takes place and the pose estimate was recorded after each update step.

5.1.4 Visualization and Debugging Figures

During the implementation, debugging of the filtering algorithm was challenging. Therefore, many visualizations and debugging figures have been generated.

5.1.4.1 Track Re-projection Visualizer

The feature track re-projection error was used as the measurement residual in the filtering algorithm. Therefore, a visualization figure for track projection and re-projection has been implemented. The marginalized tracks were highlighted before the update, after that they were re-projected back to check whether the estimation

algorithm work as expected. The re-projected track should closely resemble the original track used for the update. The Figure 5.1 below shows an accurate re-projected track.

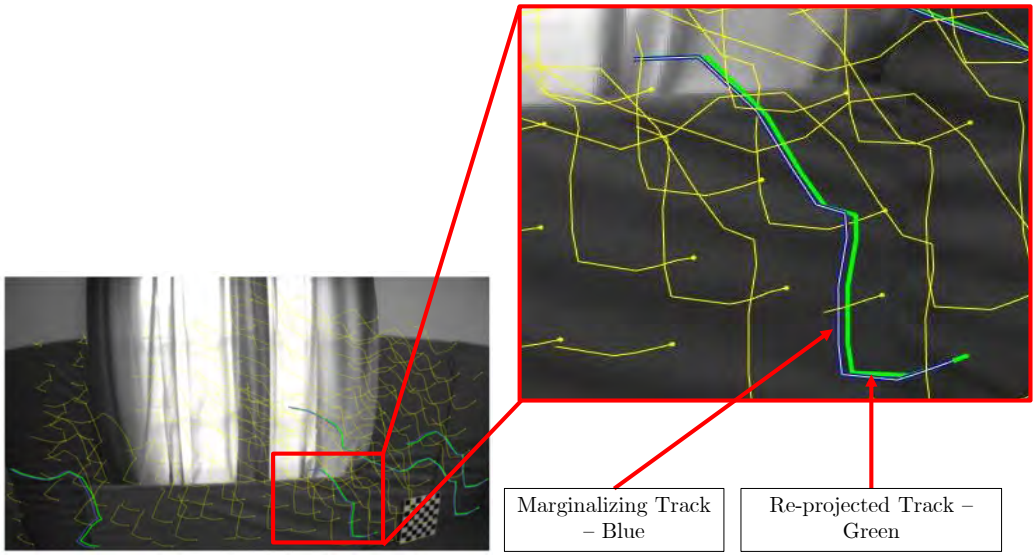

Figure 5.1: Re-projection visualization

5.1.4.2 Trajectory Visualizer

The trajectory of the MAV is visualized in Matlab (refer to Figure 5.2). A visualization robot frame was created and animated to represent the MAV. The robot frame is capable of following a trajectory upon receiving position and orientation states. The ground-truth path of the MAV is plotted and a visualization robot is introduced in red color. A blue robot frame is used to visualize the estimated path by the algorithm. A green robot is used to visualize the camera poses stored in the state vector. All the robot frames were updated at each iteration to their respective next state. The blue robot frames help to visualize and debug whether the estimates were accurate. The camera pose frames were used to visualize and debug the pose update steps. If 30 poses are kept in the state vector, 30 frames were plotted in the visualization figure. This was instrumental in state marginalization technique comparison. If 10 poses are marginalized out, from the figure one can easily visualize that 1/3 of the bots are getting deleted.

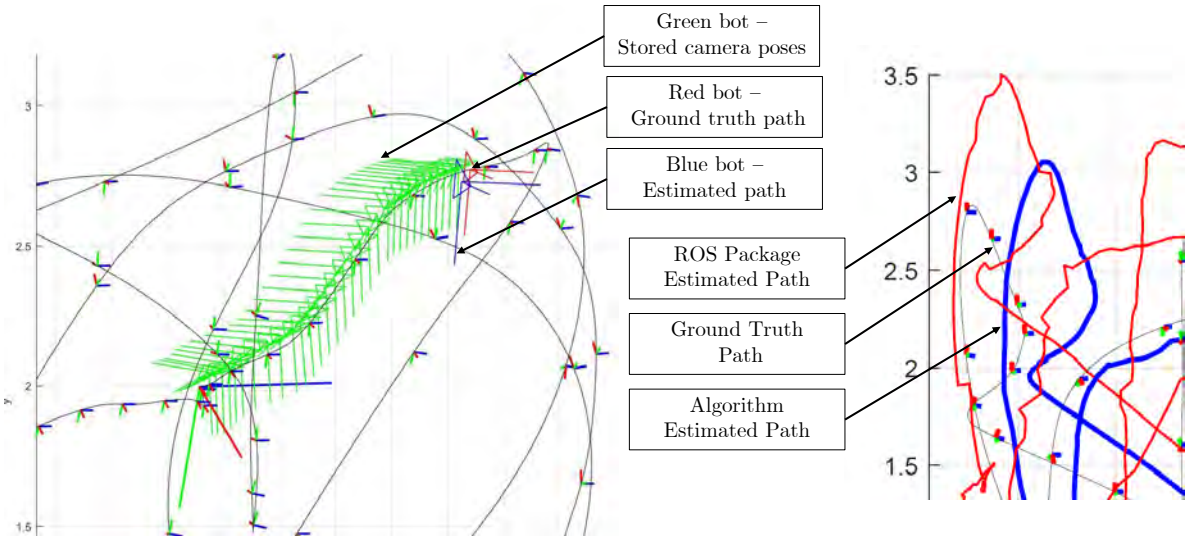

Figure 5.2: Estimated, ground-truth and stored camera pose path visualizations [11]

From the ground-truth trajectory plotted in the figure, one can easily visualize whether the estimates lie close to the ground-truth or drifting away. The Figure 5.2 shows the trajectories visualized. The benchmark for the comparison was the open-source ROS package. Therefore, estimated path from the ROS package was also plotted in the same figure in red color. After the estimation has been completed for the entire dataset, the estimated path was also plotted in the same figure.

The MSCKF algorithm estimation is an odometer therefore drifting is unavoidable. The tuning parameters (**P** and **Q** matrices) and the accuracy of the feature tracking front end reduce this drift. The blue color trajectory shows lesser drift due to the highly accurate feature tracker than the red color trajectory generated by ROS package.

5.1.5 Accuracy and Performance Indicators

The accuracy of the estimation was compared using Root Mean Squared Error (RMSE) for position and orientation. Using the recorded pose estimates after each update step and the ground-truth provided for each sequence, RMSE values were calculated. The time averaged RMSE for each dataset sequence is then calculated and arranged in a tabular format for ease of comparison in Table 5.2.

The time to populate the features and the time to run the filter was recorded using Matlab timing functions. The data was arranged in a tabular format for the ease of

comparison in Table 5.3.

5.2 Comparison of Feature Tracking Front-Ends

The three methods compared here are VINS-MONO [26], MSCKF-MONO [31] and Matlab feature tracker for MSCKF [42]. For VINS-MONO and MSCKF-MONO, open-source software packages available in ROS environment. Therefore, the feature tracker data were imported from ROS to matlab for ease of comparison with [42], which was implemented in Matlab.

All the tracker data were imported to Matlab and visualized. The image below captures feature tracking at the same image time-stamp in V1_02_medium dataset. This shows that the VINS-Mono has dense features (i.e. more features per image) compared to the MSCKF-Mono feature tracker.

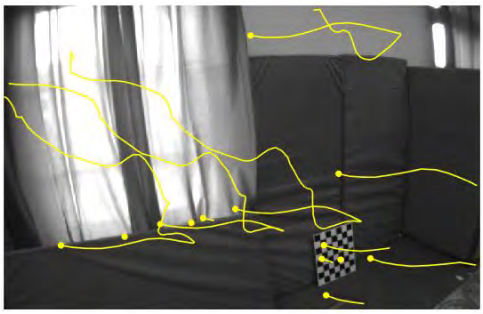

Figure 5.3: MSCKF-Mono Tracker image

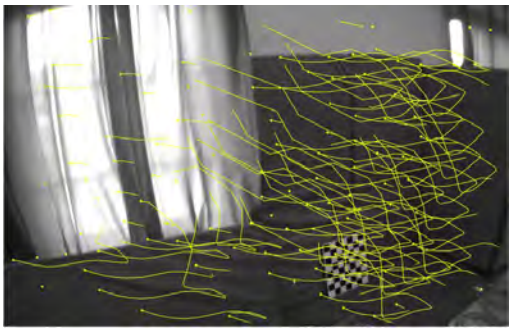

Figure 5.4: VINS-Mono Tracker image

The yellow dots indicated the recognized corner feature and the lines indicate the track of a feature. A video captured in the comparison of the entire tracker running for the V1_02_medium dataset can be found in this video link. The RMSE values for V1_02_medium dataset is plotted in the Figure 5.5 below. It can be seen that the dense feature tracker gives higher performance.

The comparison of the feature-trackers has been done for one dataset to validate the performance improvement of the dense feature tracker. The other datasets also showed a similar performance when comparing the results generated by the dense filter with the ones generated by the MSCKF mono ROS package.

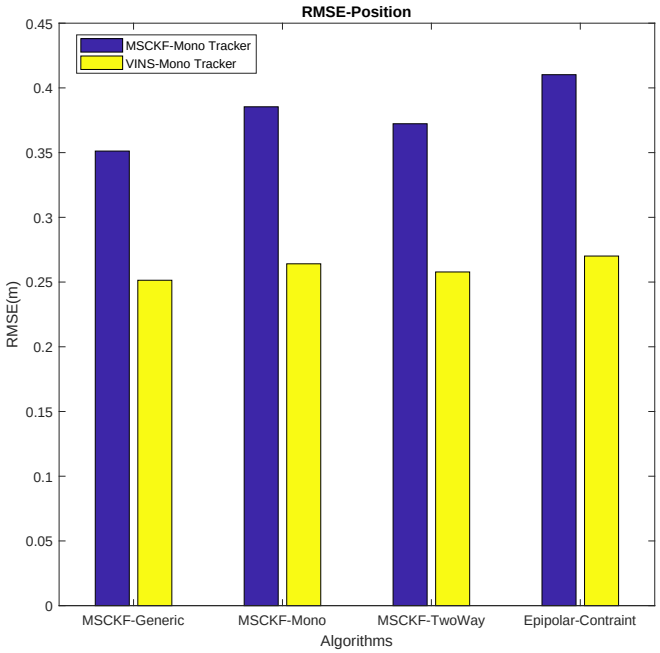

Figure 5.5: RMSE of position for feature trackers using V1_02_medium dataset

As outlined in the table below highest performance was achieved using the VINS-MONO [26] feature tracker. A complete comparison is shown for one dataset capturing the tracker accuracy for each VINS algorithms implemented.

Table 5.1: Feature tracker accuracy comparison for the three trackers using RMSE-position measured in cm and RMSE-orientation measured in degrees

Tracker Name	EuroC dataset Vicon room 1 "medium"							
	MSCKF-Generic		MSCKF-Mono		MSCKF-Two Way		Epipolar Constraint	
	Pos.	Orien.	Pos.	Orien.	Pos.	Orien.	Pos.	Orien.
VINS-Mono	25.143	0.101	26.414	0.160	25.781	0.103	27.012	0.089
MSCKF-Mono	35.122	0.198	38.540	0.234	37.235	0.253	41.021	0.125
Matlab Tracker	×	×	×	×	×	×	×	×

The feature tracker used in MSCKF-Mono [31] uses Harris corner detector and matching, which was an efficient state-of-the-art feature recognition algorithm, but the matching features are concentrated in some areas of the images and the number of features recognized were less. High number of features and distributed features across the image increased the accuracy of the VINS algorithms as shown in Table 5.1.

However, the more robust brisk descriptor based matching used in VINS-Mono [26] has a higher number of matched features distributed over the image. This achieved higher performance. The distribution of features also handled the varying illumination conditions of the sequences. Due to this reason the algorithms achieved higher performance in the challenging datasets. It can be further observed in the results outlined in the following sections of this chapter.
The matlab feature tracker which uses a FAST feature detection and matching algorithm failed to capture the highly dynamic motion of the MAV and the varying lighting conditions. Therefore, non of the parameter combinations worked with the algorithm for the implemented datasets.

5.3 Comparison of State Marginalization Strategies

Once the estimation is completed, the ground truth path, open source ROS package path (data imported from ROS environment to Matlab and plotted as a reference) and estimated path using the algorithm was plotted in the same figure. The Figure 5.6 shows the result generated for MSCKF-Mono algorithm estimation of V1_02_medium dataset. All the results were also generated from Matlab. A video of the implementation can be found in this video link.
The accuracy of the estimation is compared using position-RMSE and orientation-RMSE (refer Figure 5.7). The estimated path points and ground truth points were used to calculate the RMSE. It can be seen from the Figure 5.6 that dense-feature tracker estimate is better than the open-source ROS package estimate that uses low-dense tracker as shown previously.
The performance of the algorithm was evaluated by calculating the 3σ error bounds for each state in the state vector. Figure 5.8 shows the results of the calculation.

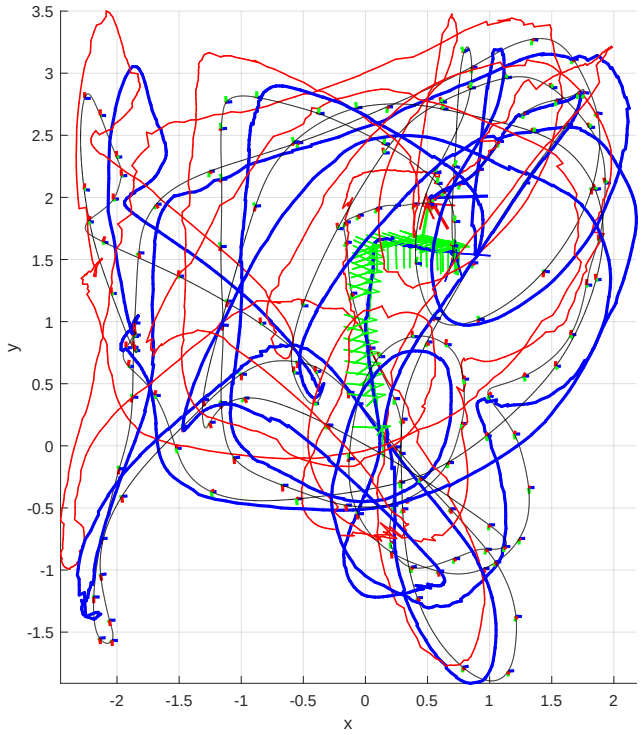

Figure 5.6: MSCKF-Mono algorithm: Black-ground truth, Blue-estimate, Red-ROS package estimate

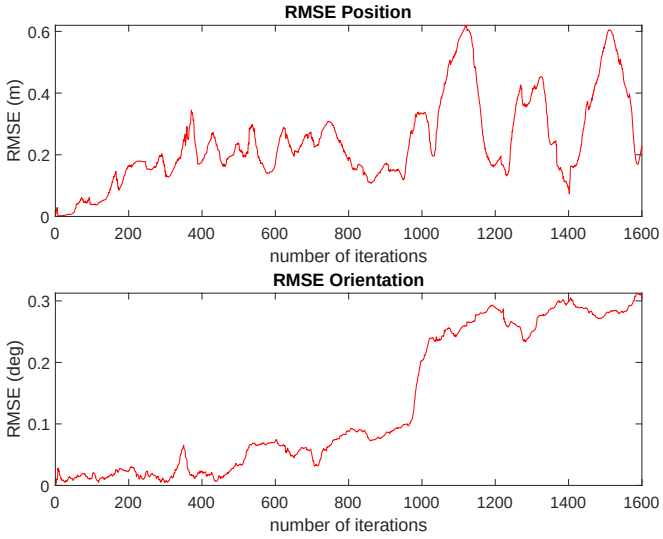

Figure 5.7: MSCKF-Mono algorithm: Time averaged RMSE

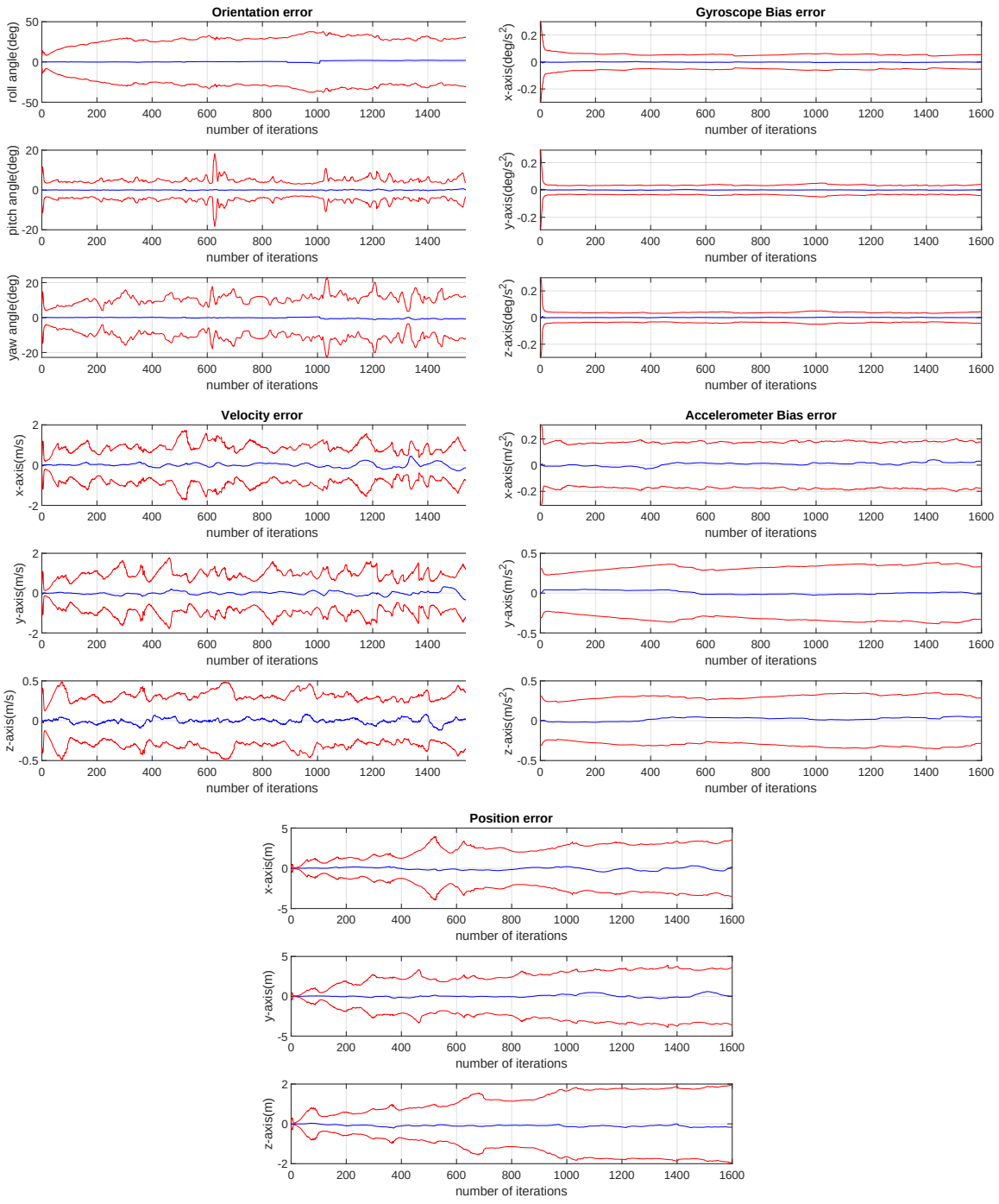

Figure 5.8: Performance of MSCKF-Mono algorithm - 3σ error bounds for states

Image pixel noise, measurement noise and the process noise parameters were tuned for each algorithm, for each dataset. Mainly, the image noise increased when the difficulty level of the dataset increases. In a Kalman filter algorithm when the measurement

noise increased, it becomes biased towards the process model. In difficult datasets the tracking was not highly accurate due to aggressive motions and illumination changes. Therefore, the estimation should be biased towards the process model rather than the measurement model. Once an algorithm successfully completed, the estimation for a particular dataset, parameters were re-tuned to increase accuracy if possible. The highlighted values in bold indicate higher accuracy than the open-source ROS package.

Table 5.2: Time averaged absolute translation and orientation RMSE in centimeters and degrees

Dataset Sequence	Dell D7 i5-7300HQ Quad Core 2.5 GHz workstation									
	Mono-ROS		Generic		Mono		Two Way		Epipolar	
	Pos.	Orien.	Pos.	Orien.	Pos.	Orien.	Pos.	Orien.	Pos.	Orien.
V1_01_easy	39.366	0.094	44.692	0.125	45.856	0.109	46.511	0.108	40.484	0.075
V1_02_medium	40.936	0.247	**25.143**	**0.101**	**26.414**	**0.160**	**25.781**	**0.103**	**27.012**	**0.089**
V1_03_difficult	147.986	0.335	**93.936**	**0.423**	**28.507**	**0.068**	**52.207**	**0.209**	**62.856**	**0.176**
MH_01_easy	64.717	0.136	**34.047**	**0.100**	96.971	**0.125**	98.387	**0.136**	**40.325**	**0.235**
MH_03_medium	70.515	0.233	**53.352**	**0.108**	**57.356**	**0.122**	**58.981**	**0.200**	**60.115**	**0.105**
MH_04_difficult	186.725	0.165	152.545	0.142	154.877	0.324	×	×	×	×

The datasets were tested in Maltab 2018a environment in Windows 10 on Dell D7 workstation with a i5-7300HQ Quad Core 2.5 GHz processor, 16GB RAM and Nvidia GTX 1050 4GB graphics driver. The ROS package was installed and used in the same laptop in Ubuntu 16.04 to generate results for the Table 5.2. The parameters in the ROS package were not tuned explicitly for each sequence, but the initialization position was changed to the time-stamp where first ground-truth measurement is available, for the ease of comparison.

The feature tracker for the algorithms were imported from open-source VINS-Mono ROS package. The number of features per frame was kept unchanged at 150. In the MSCKF-Mono the Minimum Track length is set to five. The number of camera poses in the state vector was kept at 30. The gating test for each track evaluated for MSCKF-Mono and MSCKF-TwoWay. Only one camera-pose is stored in the state vector for the epipolar-constraint method.

Table 5.3: Execution time in seconds for front-end and back-end of the filter

Dataset Sequence	Dell D7 i5-7300HQ Quad Core 2.5 GHz workstation							
	Generic		Mono		Two Way		Epipolar	
	front-end	back-end	front-end	back-end	front-end	back-end	front-end	back-end
V1_01_easy	37.32	708.64	40.29	754.97	40.79	784.47	0.421	37.159
V1_02_medium	17.75	395.11	19.93	478.99	19.58	617.38	0.321	17.590
V1_03_difficult	23.41	391.95	19.99	402.25	17.55	444.31	0.355	17.970
MH_01_easy	39.37	724.20	35.43	735.89	30.65	750.74	0.411	38.429
MH_03_medium	32.35	678.49	35.87	714.80	34.15	827.52	0.435	34.746
MH_04_difficult	34.44	660.80	38.44	706.79	×	×	×	×

The execution time was recorded using Matlab timing functions. The execution time for feature tracker propagation was recorded separately and deducted from the over all algorithm execution time. The optimization of the code effects the execution time of the filter. However, all filters were programmed by the author in Matlab which followed similar coding practices followed by the author. Therefore, it is reasonable to assume that the results are not biased towards code optimizations.

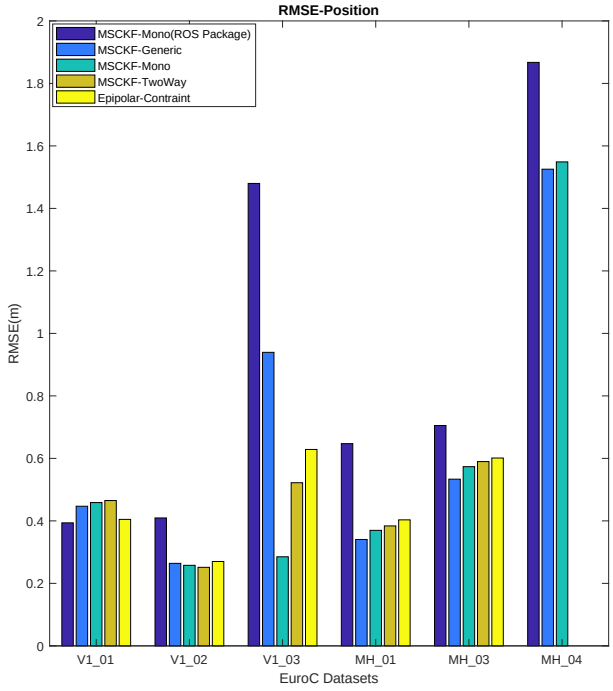

Figure 5.9: RMSE of position estimates

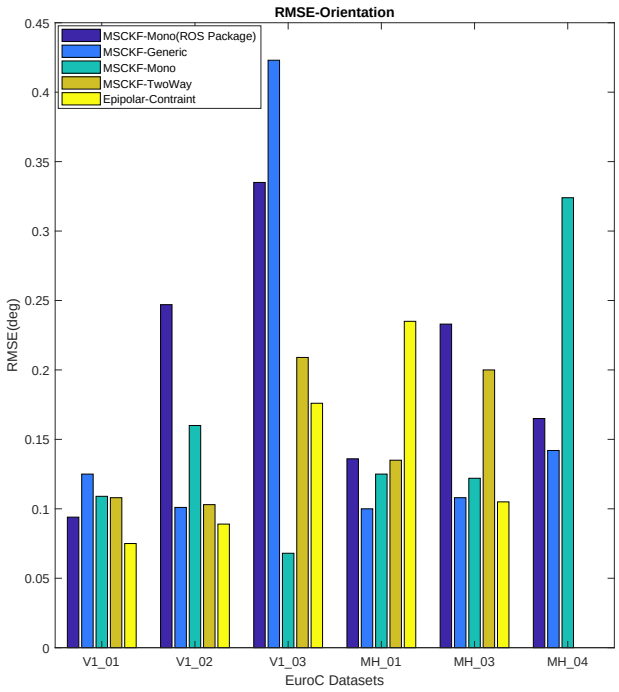

Figure 5.10: RMSE of orientation estimates

Graphical representation of the numbers in the Table 5.2 is shown in the Figure 5.9 and Figure 5.10.

5.4 Complexity of State Estimation Algorithm

Computational efficiency was dependent on complexity of the state estimation algorithm used. There is significantly less execution time recorded for epipolar-constraint formulation as shown in Figure 5.11. It clearly shows that simpler formulations are highly computationally efficient (i.e. takes less time for execution).

The MSCKF formulation uses more than two camera poses for state marginalization, while incorporating null-space trick for measurement updates, hence taking more time for execution. On the other hand, Epipolar-constraint only uses two camera poses and no additional calculations are performed like null-space trick for measurement updates.

As shown in the Figure 5.9 the accuracy of the filter is only reduced by around 10% when Epipolar-constraint is used. The computational time was reduced by more than 90%. Therefore, the gain in efficiency is clearly significant.

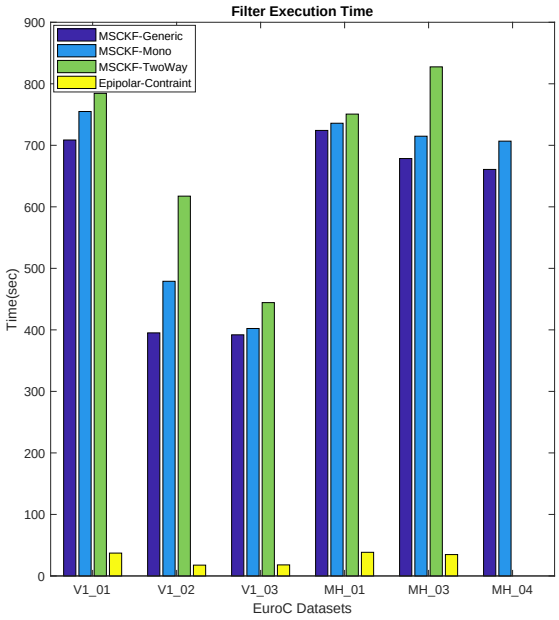

Figure 5.11: Execution time for filter algorithm for each dataset

5.5 Conclusion

This chapter performed a comparative evaluation of several VINS filters which used different state marginalization techniques and associated measurement model simplifications. The performance improvement over a state of the art algorithm MSCKF-MONO was evaluated in detail.

All the algorithms show improved accuracy in position RMSE and orientation RMSE when compared to the open-source ROS package trails. The main reason for this can be identified as the feature-dense feature tracker. The filter estimates become more accurate when more tracks are available for the updates of the filter.

The keyframing strategy used in MSCKF mono and Two-way marginalization methods did not show much difference in terms of accuracy and efficiency. Since both algorithms only remove two poses depending on the relative motion of the camera poses made it similar to each other. However, two way marginalization technique change the selection of frames to remove depending on the motion of the platform while MSCKF-Mono marginalization strategy was trying to remove the non-keyframes from the camera poses. The non-keyframe removal makes the tracks to have a higher

baseline between camera poses, thus making the track update more accurate. Therefore, MSCKF-Mono marginalization recorded higher accuracy in more challenging datasets like V1_03_difficult.

Two key frame marginalization using epipolar-constraints only compare two camera poses, which eliminates the need to keep a track of features over a number of images, saving computational cost. The inaccurate updates can also be handled by the key-framing strategy that uses a feature disparity threshold. As shown in the results in Figure 5.11 and Figure 5.2, the marginal decrease around 10% in algorithm estimation accuracy, gives more than 90% gain in efficiency which is highly desirable for the MAVs.

From the available eleven datasets only six datasets are selected for the evaluation since the Vicon Room 2 dataset had similar feature environment compared to Vicon Room 1 dataset. MH_05_difficult was failed in the two key-frame epipolar-constraint and MSCKF-TwoWay. It may be because of the faster motions and illumination changes in the tracker. In the future these two algorithms should be studied further to find a solution for this failure.

Chapter 6

Conclusion and Future Directives

The focus of this research study was to find an optimal VINS algorithm suitable for implementation on MAVs. MAVs are resource constrained (limited on-board memory, computational power and sensor payload) systems therefore highly computationally demanding algorithms are not suitable. Since VINS is only meant to serve as an odometer during navigation, stability and computational efficiency is critical than accuracy. Therefore, highly computationally demanding elements (state marginalization technique, feature tracking front-end and complexity of the VINS algorithm) in the VINS algorithm were comparatively analyzed to design an efficient algorithm for MAVs. The study formed the following research objectives:

1. Accuracy and performance comparison of four main state marginalization strategies used in VINS filters.
2. Accuracy comparison of image processing front-ends used for VINS algorithms.
3. Complexity comparison of the VINS algorithms in-terms of execution time.

6.1 Research summary based on Objective I

The first objective of this study was to compare the state marginalization techniques in VINS algorithms. The state marginalization techniques were implemented and evaluated on EuRoC MAV VINS benchmark dataset. The VINS algorithm accuracy findings indicated that all the algorithms performs in par with each other. The difference in accuracy levels with respect to position-RMSE were in the order of **few centimeters** (e.g. MSCKF-Generic - 25.14 cm , MSCKF-Mono - 26.41 cm and Epipolar

constraint - 27.01 cm for V1_02_medium dataset). On the other hand, the accuracy levels of orientation-RMSE was only changing in sub-degree level.

6.2 Research summary based on Objective II

The second objective of this study was to compare the feature tracking front ends for VINS algorithms. Two state of the art feature trackers were evaluated. MSCKF-Mono feature tracker and VINS-Mono feature tracker are compared.

The VINS-Mono tracker had a dense (i.e. more features per image) feature tracker than MSCKF-Mono tracker. Due to this reason, VINS-Mono feature tracker based VINS algorithms gave higher performance compared to the MSCKF-Mono tracker based VINS algorithms. The increase in accuracy is around 30%. Therefore, having a feature tracker front end with higher feature density makes the algorithms more accurate.

6.3 Research summary based on Objective III

The third objective of this study is to compare the complexity of the VINS algorithm formulation, using the filter execution time. The comparison of algorithm execution time had shown significant changes. Three algorithms (namely, MSCKF-Generic, MSCKF-Mono and MSCKF-Two Way) had similar execution times. The execution time difference between those algorithms were less than 30% of the total execution time (e.g. MSCKF-Generic 395.11 s and MSCKF-Mono 478.99 s for V1_02_medium dataset). However, the Epipolar constraint algorithm formulation reported more than 90% reduction in execution time (e.g. MSCKF-Generic 395.11 s and Epipolar constraint 17.95 s for V1_02_medium dataset). with respect to the other three methods compared.

Therefore, Epipolar constraint based VINS had significant efficiency compared to the other methods. The reduction in accuracy was only around 10% when the gain in efficiency was more than 90%. Due to this reason, Epipolar constraint based VINS algorithm is deemed most suitable for MAVs.

6.4 Contribution

The following contributions resulted from this thesis:

- Comparative evaluation of feature tracking front-ends, different state marginalization techniques, and different measurement models of VINS algorithms.
- Identification of an efficient VINS algorithm using epipolar constraint state marginalization technique which achieves more that 90% improvement in computational complexity for minimal (10%) decrement in algorithm accuracy.
- Design and calibration of a sensor system suitable for experimental evaluation of VINS algorithms.

The following articles are a result of this thesis:

- **R. Thalagala**, O. De Silva, G. K. I. Mann, and R. G. Gosine, "Calibration and Validation of a Hand-Held Visual Inertial Sensor Set-up for Visual Inertial Odometry Filters," in Proceedings, *Newfoundland Electrical and Computer Engineering Conference*, 2018.
- **R. Thalagala**, O. De Silva, G. K. I. Mann, and R. G. Gosine, "Two key frame state marginalization for highly efficient Visual odometry on Micro Aerial vehicles," in preparation, *American Control Conference*, 2020. (The results of this thesis are being prepared for submission in this conference)

6.5 Future Directives

As future work and the author is experimentally validating the state marginalization algorithms on the VINS sensor-setup developed. This work is being conducted as part of the article preparation for the American Control Conference 2020.

6.5.1 Preliminary work completed

The sensor-setup outlined in Chapter 3 has been integrated to ROS for VINS experimentation. It was held by hand and walked inside the IsLab, while recording a ROS bag. IMU topic, fisheye image topic and ground-truth topic from the OptiTrack

motion capture system were recorded into the ROS bag. The OptiTrack system publishes the 6 DOF pose of the sensor-unit with respect to the marker placed on top of the ZR300 sensor. The dataset was captured for a duration of **6.5 minutes**.
The data were streaming at three different frequencies. The IMU data were published at 250Hz. Fisheye images were published at 30Hz while ground truth data were published at 120Hz.
The fisheye image stream was time stamped and saved as separate image files using the Kalibr bag extractor ROS node [78]. Similarly, the IMU topic and the ground-truth data were also time stamped and saved as csv files. After that all the data were imported to Matlab. The circular path traversed during this experiment is shown in Figure 6.1. The experimental setup used to collect the dataset is shown in Figure 6.2
The next step is to generate the feature tracks for the images and experimentally validate the VINS algorithms using this sensor setup.

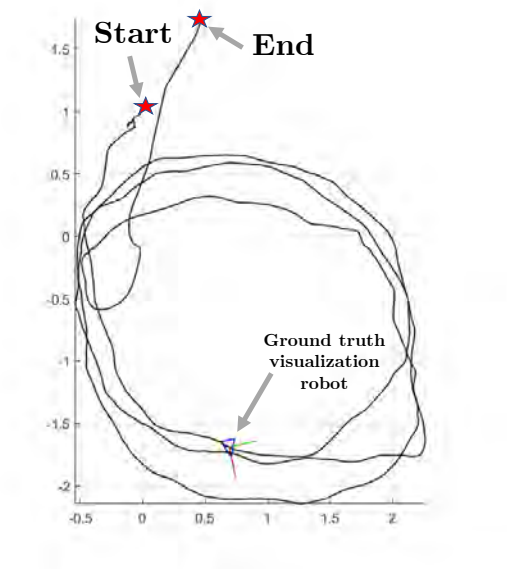

Figure 6.1: Dataset ground truth plot

Figure 6.2: VINS sensor setup

Figure 6.3: The above dataset was only the visualization of the ground truth

(a) image1 (b) image2 (c) image3

(d) image4 (e) image5 (f) image6

(g) image7 (h) image8 (i) image9

Figure 6.4: IsLab Dataset images - Raw fisheye images

Bibliography

[1] ZR300 Datasheet, "Datasheet for the Intel® RealSense™ Camera ZR300." [Online]. Available: https://www.intel.me/content/www/xr/ar/support/articles/000023563/emerging-technologies/intel-realsense-technology.html

[2] M. Li and A. I. Mourikis, "Online temporal calibration for camera–IMU systems: Theory and algorithms," *The International Journal of Robotics Research*, vol. 33, no. 7, pp. 947–964, jun 2014.

[3] A. I. Mourikis and S. I. Roumeliotis, "A Multi-State Constraint Kalman Filter for Vision-aided Inertial Navigation," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*. IEEE, apr 2007, pp. 3565–3572.

[4] K. Sun, K. Mohta, B. Pfrommer, M. Watterson, S. Liu, Y. Mulgaonkar, C. J. Taylor, and V. Kumar, "Robust Stereo Visual Inertial Odometry for Fast Autonomous Flight," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 965–972, apr 2018.

[5] Z. Huai and G. Huang, "Robocentric Visual-Inertial Odometry," in *IEEE International Conference on Intelligent Robots and Systems*. IEEE, oct 2018, pp. 6319–6326.

[6] B. Joshi, S. Rahman, M. Kalaitzakis, B. Cain, J. Johnson, M. Xanthidis, N. Karapetyan, A. Hernandez, A. Q. Li, N. Vitzilaios, and I. Rekleitis, "Experimental Comparison of Open Source Visual-Inertial-Based State Estimation Algorithms in the Underwater Domain," apr 2019.

[7] S. Rahman, A. Q. Li, and I. Rekleitis, "An Underwater SLAM System using Sonar, Visual, Inertial, and Depth Sensor," oct 2018.

[8] D. S. Bayard, D. T. Conway, R. Brockers, J. H. Delaune, L. H. Matthies, H. F. Grip, G. B. Merewether, T. L. Brown, and A. M. San Martin, "Vision-Based Navigation for the NASA Mars Helicopter," in *AIAA Scitech 2019 Forum*. Reston, Virginia: American Institute of Aeronautics and Astronautics, jan 2019.

[9] C. Bamann and P. Henkel, "Visual-Inertial Odometry with Sparse Map Constraints for Planetary Swarm Exploration," in *2019 IEEE International Con-*

ference on Industrial Cyber Physical Systems (ICPS). IEEE, may 2019, pp. 290–295.

[10] J. Collin, P. Davidson, M. Kirkko-Jaakkola, and H. Leppäkoski, "Inertial Sensors and Their Applications," in *Handbook of Signal Processing Systems*. Cham: Springer International Publishing, 2019, pp. 51–85.

[11] R. Osiander, M. A. G. Darrin, J. L. Champion, M. A. G. Darrin, and J. L. Champion, *MEMS and Microstructures in Aerospace Applications*, R. Osiander, M. A. G. Darrin, and J. L. Champion, Eds. CRC Press, oct 2018.

[12] D. Honegger, L. Meier, P. Tanskanen, and M. Pollefeys, "An open source and open hardware embedded metric optical flow CMOS camera for indoor and outdoor applications," in *2013 IEEE International Conference on Robotics and Automation*. IEEE, may 2013, pp. 1736–1741.

[13] G. Huang, "Visual-Inertial Navigation: A Concise Review," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, jun 2019, pp. 9572–9582.

[14] X. Qiu, H. Zhang, W. Fu, C. Zhao, Y. Jin, X. Qiu, H. Zhang, W. Fu, C. Zhao, and Y. Jin, "Monocular Visual-Inertial Odometry with an Unbiased Linear System Model and Robust Feature Tracking Front-End," *Sensors*, vol. 19, no. 8, p. 1941, apr 2019.

[15] K. Wu, T. Zhang, D. Su, S. Huang, and G. Dissanayake, "An invariant-EKF VINS algorithm for improving consistency," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, sep 2017, pp. 1578–1585.

[16] Y. Lin, F. Gao, T. Qin, W. Gao, T. Liu, W. Wu, Z. Yang, and S. Shen, "Autonomous aerial navigation using monocular visual-inertial fusion," *Journal of Field Robotics*, vol. 35, no. 1, pp. 23–51, jan 2018.

[17] R. Institute of Control and Institute of Electrical and Electronics Engineers, *ICCAS 2018 : 2018 18th International Conference on Control, Automation and Systems : proceedings : October 17 (Wed)-20 (Sat), 2018, YongPyong Resort, PyeongChang, Korea.*

[18] H. Hellmers, Z. Kasmi, A. Norrdine, A. Eichhorn, H. Hellmers, Z. Kasmi, A. Norrdine, and A. Eichhorn, "Accurate 3D Positioning for a Mobile Platform in Non-Line-of-Sight Scenarios Based on IMU/Magnetometer Sensor Fusion," *Sensors*, vol. 18, no. 2, p. 126, jan 2018.

[19] G. Tsaramirsis, S. Buhari, M. Basheri, M. Stojmenovic, G. Tsaramirsis, S. M. Buhari, M. Basheri, and M. Stojmenovic, "Navigating Virtual Environments Using Leg Poses and Smartphone Sensors," *Sensors*, vol. 19, no. 2, p. 299, jan 2019.

[20] T. K. Chan, Y. K. Yu, H. C. Kam, and K. H. Wong, "Robust Hand Gesture Input Using Computer Vision, Inertial Measurement Unit (IMU) and Flex Sensors," in *2018 IEEE International Conference on Mechatronics, Robotics and Automation (ICMRA)*. IEEE, may 2018, pp. 95–99.

[21] J. Gim, C. Ahn, J. Gim, and C. Ahn, "IMU-Based Virtual Road Profile Sensor for Vehicle Localization," *Sensors*, vol. 18, no. 10, p. 3344, oct 2018.

[22] X. Guang, Y. Gao, H. Leung, P. Liu, G. Li, X. Guang, Y. Gao, H. Leung, P. Liu, and G. Li, "An Autonomous Vehicle Navigation System Based on Inertial and Visual Sensors," *Sensors*, vol. 18, no. 9, p. 2952, sep 2018.

[23] X. Xia, L. Xiong, W. Liu, and Z. Yu, "Automated Vehicle Attitude and Lateral Velocity Estimation Using a 6-D IMU Aided by Vehicle Dynamics," in *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, jun 2018, pp. 1563–1569.

[24] M. Li and A. I. Mourikis, "High-precision, consistent EKF-based visual-inertial odometry," *The International Journal of Robotics Research*, vol. 32, no. 6, pp. 690–711, may 2013.

[25] D. Dusha and L. Mejias, "Error analysis and attitude observability of a monocular GPS/visual odometry integrated navigation filter," *The International Journal of Robotics Research*, vol. 31, no. 6, pp. 714–737, may 2012.

[26] T. Qin, P. Li, and S. Shen, "VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator," *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 1004–1020, aug 2018.

[27] X. Li, Y. Wang, and K. Khoshelham, "Comparative analysis of robust extended Kalman filter and incremental smoothing for UWB/PDR fusion positioning in NLOS environments," *Acta Geodaetica et Geophysica*, vol. 54, no. 2, pp. 157–179, jun 2019.

[28] S. Leutenegger, S. Lynen, M. Bosse, R. Siegwart, and P. Furgale, "Keyframe-based visual–inertial odometry using nonlinear optimization," *The International Journal of Robotics Research*, vol. 34, no. 3, pp. 314–334, mar 2015.

[29] J. A. Delmerico and D. Scaramuzza, "A Benchmark Comparison of Monocular Visual-Inertial Odometry Algorithms for Flying Robots," *Memory*, vol. 10, p. 20, 2018.

[30] M. K. Paul, K. Wu, J. A. Hesch, E. D. Nerurkar, and S. I. Roumeliotis, "A comparative analysis of tightly-coupled monocular, binocular, and stereo VINS," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, may 2017, pp. 165–172.

[31] A. Z. Zhu, N. Atanasov, and K. Daniilidis, "Event-Based Visual Inertial Odometry," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, jul 2017, pp. 5816–5824.

[32] M. Bloesch, S. Omari, M. Hutter, and R. Siegwart, "Robust visual inertial odometry using a direct EKF-based approach," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, sep 2015, pp. 298–304.

[33] D. Abeywardena, Shoudong Huang, B. Barnes, G. Dissanayake, and S. Kodagoda, "Fast, on-board, model-aided visual-inertial odometry system for quadrotor micro aerial vehicles," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, may 2016, pp. 1530–1537.

[34] T. Nguyen, G. K. I. Mann, A. Vardy, and R. G. Gosine, "Developing a Cubature Multi-state Constraint Kalman Filter for Visual-Inertial Navigation System," in *2017 14th Conference on Computer and Robot Vision (CRV)*. IEEE, may 2017, pp. 321–328.

[35] A. Taffanel, "Crazyflie nano quadcopter." [Online]. Available: https://www.bitcraze.io/crazyflie-2-1/

[36] J. A. Farrell, *Aided Navigation Systems: GPS and High Rate Sensors*, 1st ed. New York: McGraw-Hill, 2008.

[37] Bruce D. Lucas and Takeo Kanade, "An Iterative Image Registration Technique with an Application to Stereo Vision (DARPA)," in *Proceedings of the 1981 DARPA Image Understanding Workshop*, 1981, pp. 121–130.

[38] S. Roumeliotis and J. 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, pp. 1788–1795.

[39] M. Brossard, S. Bonnabel, and A. Barrau, "Unscented Kalman Filter on Lie Groups for Visual Inertial Odometry," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, oct 2018, pp. 649–655.

[40] T. Nguyen, G. K. I. Mann, A. Vardy, and R. G. Gosine, "Developing Computationally Efficient Nonlinear Cubature Kalman Filtering for Visual Inertial Odometry," *Journal of Dynamic Systems, Measurement, and Control*, vol. 141, no. 8, p. 081012, mar 2019.

[41] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik, and R. Siegwart, "The EuRoC micro aerial vehicle datasets," *The International Journal of Robotics Research*, vol. 35, no. 10, pp. 1157–1163, sep 2016.

[42] L. E. Clement, V. Peretroukhin, J. Lambert, and J. Kelly, "The Battle for Filter Supremacy: A Comparative Study of the Multi-State Constraint Kalman Filter and the Sliding Window Filter," in *2015 12th Conference on Computer and Robot Vision*. IEEE, jun 2015, pp. 23–30.

[43] T. D. Barfoot, *State Estimation for Robotics*. Cambridge: Cambridge University Press, 2017.

[44] M. Zaffar, S. Ehsan, R. Stolkin, and K. M. Maier, "Sensors, SLAM and Long-term Autonomy: A Review," in *2018 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*. IEEE, aug 2018, pp. 285–290.

[45] M. Bloesch, M. Burri, S. Omari, M. Hutter, and R. Siegwart, "Iterated extended Kalman filter based visual-inertial odometry using direct photometric feedback," *The International Journal of Robotics Research*, vol. 36, no. 10, pp. 1053–1072, sep 2017.

[46] M. He, C. Zhu, Q. Huang, B. Ren, and J. Liu, "A review of monocular visual odometry," *The Visual Computer*, pp. 1–13, jun 2019.

[47] Google, "Google ARCore." [Online]. Available: https://developers.google.com/ar/discover/

[48] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon, "Bundle Adjustment — A Modern Synthesis." Springer, Berlin, Heidelberg, 2000, pp. 298–372.

[49] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. J. Leonard, and F. Dellaert, "iSAM2: Incremental smoothing and mapping using the Bayes tree," *The International Journal of Robotics Research*, vol. 31, no. 2, pp. 216–235, feb 2012.

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

[51] G. Huang, M. Kaess, and J. J. Leonard, "Towards consistent visual-inertial navigation," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, may 2014, pp. 4926–4933.

[52] K. Eckenhoff, P. Geneva, and G. Huang, "Sensor-Failure-Resilient Multi-IMU Visual-Inertial Navigation," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, may 2019, pp. 3542–3548.

[53] L. von Stumberg, V. Usenko, and D. Cremers, "Direct Sparse Visual-Inertial Odometry using Dynamic Marginalization," apr 2018.

[54] V. Usenko, J. Engel, J. Stuckler, and D. Cremers, "Direct visual-inertial odometry with stereo cameras," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, may 2016, pp. 1885–1892.

[55] S. Leutenegger, M. Chli, and R. Y. Siegwart, "BRISK: Binary Robust invariant scalable keypoints," in *2011 International Conference on Computer Vision*. IEEE, nov 2011, pp. 2548–2555.

[56] E. Rosten and T. Drummond, "Machine Learning for High-Speed Corner Detection." Springer, Berlin, Heidelberg, 2006, pp. 430–443.

[57] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: An efficient alternative to SIFT or SURF," in *2011 International Conference on Computer Vision*. IEEE, nov 2011, pp. 2564–2571.

[58] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "BRIEF: Binary Robust Independent Elementary Features." Springer, Berlin, Heidelberg, 2010, pp. 778–792.

[59] J. Kelly and G. S. Sukhatme, "Visual-Inertial Sensor Fusion: Localization, Mapping and Sensor-to-Sensor Self-calibration," *The International Journal of Robotics Research*, vol. 30, no. 1, pp. 56–79, jan 2011.

[60] Kellysensor, "3DM-GX3® -35 All-In-One Navigation Solution | LORD Sensing Systems." [Online]. Available: https://www.microstrain.com/inertial/3dm-gx3-35

[61] M. Faessler, F. Fontana, C. Forster, E. Mueggler, M. Pizzoli, and D. Scaramuzza, "Autonomous, Vision-based Flight and Live Dense 3D Mapping with a Quadrotor Micro Aerial Vehicle," *Journal of Field Robotics*, vol. 33, no. 4, pp. 431–450, jun 2016.

[62] J. Nikolic, J. Rehder, M. Burri, P. Gohl, S. Leutenegger, P. T. Furgale, and R. Siegwart, "A synchronized visual-inertial sensor system with FPGA preprocessing for accurate real-time SLAM," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, may 2014, pp. 431–437.

[63] Thibaut Rouffineau, "Welcoming the Parrot S.L.A.M.dunk: the new drone development kit | Ubuntu." [Online]. Available: https://ubuntu.com/blog/welcoming-the-parrot-s-l-a-m-dunk-the-new-drone-development-kit

[64] I. Sa, M. Kamel, M. Burri, M. Bloesch, R. Khanna, M. Popovic, J. Nieto, and R. Siegwart, "Build Your Own Visual-Inertial Drone: A Cost-Effective and Open-Source Autonomous Drone," *IEEE Robotics & Automation Magazine*, vol. 25, no. 1, pp. 89–103, mar 2018.

[65] IntelD435i, "Depth Camera D435i – Intel® RealSense™ Depth and Tracking Cameras." [Online]. Available: https://www.intelrealsense.com/depth-camera-d435i/

[66] IntelT265, "Tracking camera T265 – Intel RealSense Depth and Tracking Cameras." [Online]. Available: https://www.intelrealsense.com/tracking-camera-t265/?{_}ga=2.208021524.706492624.1566521139-364711539.1566521139

[67] R. rqt_plot, "rqt_plot - ROS Wiki." [Online]. Available: http://wiki.ros.org/rqt{_}plot

[68] T. Schneider, M. Dymczyk, M. Fehr, K. Egger, S. Lynen, I. Gilitschenski, and R. Siegwart, "Maplab: An Open Framework for Research in Visual-Inertial Mapping and Localization," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1418–1425, jul 2018.

[69] IntelRealSense, "Intel® RealSense™ Camera ZR300 | Intel® Software." [Online]. Available: https://software.intel.com/en-us/realsense/zr300

[70] J. Rehder, J. Nikolic, T. Schneider, T. Hinzmann, and R. Siegwart, "Extending kalibr: Calibrating the extrinsics of multiple IMUs and of individual axes," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, may 2016, pp. 4304–4311.

[71] E. Olson, "AprilTag: A robust and flexible visual fiducial system," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, may 2011, pp. 3400–3407.

[72] J. Sola, "Quaternion kinematics for the error-state KF," Institut de Robòtica i Informàtica Industrial, Barcelona, Tech. Rep., 2016. [Online]. Available: http://www.iri.upc.edu

[73] K. Hausman, S. Weiss, R. Brockers, L. Matthies, and G. S. Sukhatme, "Self-calibrating multi-sensor fusion with probabilistic measurement validation for seamless sensor switching on a UAV," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, may 2016, pp. 4289–4296.

[74] G. P. Huang, A. I. Mourikis, and S. I. Roumeliotis, "A First-Estimates Jacobian EKF for Improving SLAM Consistency," 2009, pp. 373–382.

[75] ——, "Observability-based Rules for Designing Consistent EKF SLAM Estimators," *The International Journal of Robotics Research*, vol. 29, no. 5, pp. 502–528, apr 2010.

[76] J. A. Hesch, D. G. Kottas, S. L. Bowman, and S. I. Roumeliotis, "Consistency Analysis and Improvement of Vision-aided Inertial Navigation," *IEEE Transactions on Robotics*, vol. 30, no. 1, pp. 158–176, feb 2014.

[77] S. Shen, Y. Mulgaonkar, N. Michael, and V. Kumar, "Initialization-Free Monocular Visual-Inertial State Estimation with Application to Autonomous MAVs." Springer, Cham, 2016, pp. 211–227.

[78] P. Furgale, J. Rehder, and R. Siegwart, "Unified temporal and spatial calibration for multi-sensor systems," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, nov 2013, pp. 1280–1286.