UNIVERSITÄT DES SAARLANDES

Universität des Saarlandes
Max-Planck-Institut für Informatik

MAX-PLANCK-GESELLSCHAFT

Keyframe-based Visual-Inertial Odometry for Small Workspace

Masterarbeit im Fach Informatik
Master's Thesis in Computer Science
von / by
Xi Li

angefertigt unter der Leitung von / supervised by
Dr. Roland Angst

betreut von / advised by
Dr. Roland Angst

begutachtet von / reviewers
Dr. Roland Angst
Prof. Dr. Antonio Krüger

Saarbrücken, June 2016

Acknowledgments

Keyframe-based Visual-Inertial Odometry for Small Workspace

by

Xi Li

Submitted to the Department of Computer Science
on June 1, 2016, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science

Abstract

Thesis Supervisor: Roland Angst
Title: Dr.

Contents

Chapter 1

Introduction

In past few years, the development of *Robotics* has surpassed people's expectation. The word *Robotics* has been first appeared in science fiction "Liar!" by Issac Asimov [42], it referred to science and technology of robots. By definition, *Robotics* is a research branch that related to design, control and application of robots, as well as processing feedback from robots.

Modern robots have been classified into several categories(e.g., Mobile robot, industrial robot, service robot, education robot etc.) with their usages. Among those categories, full-autonomous or semi-autonomous mobile robots attracts more and more researches. Such robots have abilities to move around in their moving space, with or without humans' control. The aim of research in mobile robots is to help us accomplish various hard tasks, whether domestically, commercially or militarily. These tasks, such as assisting disabled people, defusing bombs, or repair equipment in dangerous place is either risky or high expense for human beings.

For mobile robot, finding physical location of itself in unknown environment is normally crucial, such an ability(e.g., *Robot Navigation*) allows mobile robots avoid risky obstacles and finally arrive the goal position. Roughly speaking, *Robot Navigation* is a computing system which processes the information from external sources(e.g., sensors) and apply an algorithm to navigate robot, and sometimes build a map of environment. In *Robot Navigation*, robots sense environmental information by their sensors. These sensors, either locally (e.g., camera, inertial measurement unit (IMU)), or globally (e.g., Global Positioning System) detect events or changes in environment, and transfer their data to robots. Generally, sensors equipped in mobile robot (Local Sensor) are designed light, small, and inexpensive considering convenient movement and low expense. Camera and IMU sensor are considered most common local sensor in small mobile robot system.

A camera is a optical instrument for capturing images. Modern camera has several advantages for *Robot Navigation*. First, the core of camera chip set is cheap and easily installed in any mobile robot system; Second, camera often brings very rich information as it simulates the functioning of human eye. By recognizing key-points [32, 22, 31] in certain images, system observes the *landmarks* in environment, and those *landmarks* will localize robots by *Triangulation* [5, 17, 7].

A IMU sensor (Figure 1-1) often combines *gyroscope* and *accelerometer*, sometimes

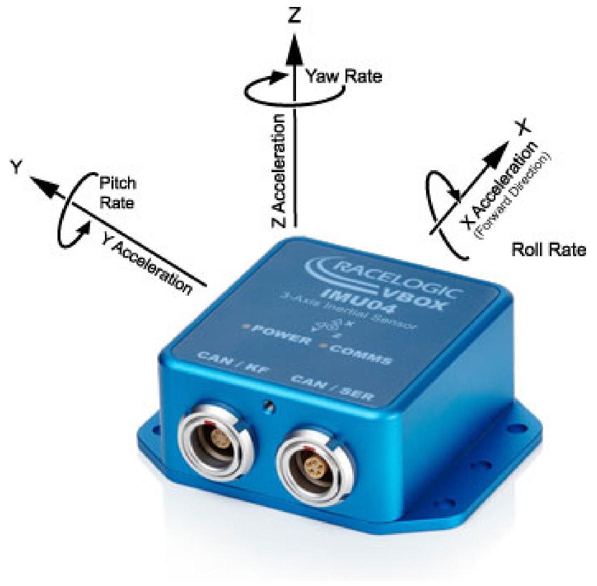


Figure 1-1: IMU sensor with gyroscope and accelerometer measures rotational rate and acceleration of X, Y, Z axis regarding its local frame. Note that IMU sensor normally has bias and irreducible noises, it is necessary to apply a calibration like cameras. The frequency of IMU output is normally larger than 100 Hz. Source: [1]

magnetometer, to measure specific force, angular rate and magnetic field regarding to its local frame. IMU is one of main component in *Inertial Navigation System*, which firstly used in air plane, spacecraft, guided missiles, and now also in mobile robot [3, 18, 26, 10]. IMU sensor utilize *Dead Reckoning* to track device's position, such technology tries to integrate IMU data over time by assuming the movement model of devices fixed(i.e., acceleration and rotation rate is constant over small period of time).

1.1 Motivation and Contribution

The main motivation of this thesis is to improve the navigation accuracy by fusing camera data and IMU sensor data.

Single sensor-based navigation system may not satisfy the requirements of high-quality localization by mobile robot. GPS-based navigation system has been used for outdoor devices for long time. However, such a system suffered from localizing in in-door environment, and also the accuracy of general GPS is not high, i.e., 3 to 5 meters error [41]. For mobile robot, which may move from in-door to out-door, and requires high-accuracy navigation, GPS is mostly not used, or as baseline of navigation [15].

Vision-based navigation system [5, 17], or simultaneous localization and mapping (SLAM) [6, 8, 27] gives an acceptable navigation result. [5] recognizes corner feature by single camera, and it uses a extended kalman filter framework to track the uncertainty and propagates the system state, however it can not handle large-scale scene. [17] utilizes key-frame based bundle adjustment to update the map, improves both accuracy and efficiency, it still met some problem in large-scale scene, becasue vision-based method often is a trade-off between computational complexity and localization accuracy due to the rich information and low output frequency by camera.

Single *Inertial Navigation System* recognize its pose by *Dead Reckoning* [24, 19]. However, the problem of *Dead Reckoning* error accumulation; Only few directions are observable [15] by IMU during whole navigation process. When object corrupts movement assumption, a correction step by external data will be needed.

Fusing camera and IMU data has many advantages. On the one hand, it can decrease computational time by making good use of high-frequency IMU data; On the other hand, it can reduce the error accumulation of IMU integration by the correction of vision-based navigation result within several turn. Fuse the camera and IMU data for robot navigation is not novel. [26] applies multi-state kalman filter (MSCKF) on state update, decrease the computational complexity by only keeping few last information of keyframe. [15] analysis the consistency of MSCKF, correct the ways of IMU integration to obtain consistency of sytem, therefore increase the accuracy. [10] exploits a way to optimize manifold information, therefore solving data fusing problem in a non-linear optimization scheme. Unfortunately, codes and data of above methods are not accessible, that motivates to explore possible methods to increase accuracy of *vision-inertial navigation system* both theoretically and experimentally.

The contributions of this thesis are as follows,

- We exploit a highly flexible, realistic software to generate synthetic IMU sensor data and corresponding vision data, that is the main source to provide experimental data in this thesis.
- We present error-state kalman filter system kinematic based on quaternion, explore the multiple ways to integrate IMU data due to different movement models.
- We propose a novel method to update fusing result based on key-framed bundle adjustment.
- We propose a real-time visual-inertial framework implemented in C++, which is stable, scalable, high-accuracy and low-latency.

1.2 Outline of the Thesis

In Chapter 2 we first overview our visual-inertial odometry, including world representation, important notations, we also compare filter method and key-frame based method in SLAM problem, and in the end we explain our choice in this thesis.

Then we enter the Chapter 3, which introduces *Quaternion Algebra*. In this chapter, we introduces the basic operations on quaternion, the relationship among quaternion, rotation matrix and rotation vector. In this chapter, we also explain how to integrate or derivative quaternion over time.

Chapter 4 is main part of this thesis. In Section 4.1, we study the Error-State Kalman Filter (ESKF), and apply it to IMU integration; Section 4.2 we summarize how to fuse camera into ESKF, and how to optimize it by key-frame based bundle adjustment. In Section 4.3, we overview the pipeline of our visual-inertial odometry system.

We show our experiment results in Chapter 5. First, the process of generating synthetic IMU and camera data is presented. Then we run several experiments to show our proposed visual-inertial odometry has higher accuracy than single IMU integration, visual SLAM, as our system is still running in real-time.

In the end, we summarize and discuss our work and analysis the potential future work in Chapter 6.

Chapter 2

Overview of Visual-inertial Odometry

In this chapter, we will overview visual-inertial odometry system. In section 2.1, we will introduce world representation(e.g., world frame, camera frame and IMU frame) together with basic notations in our odometry system. Then in section 2.2, we will discuss two important scheme, filter method and keyframe Bundle Adjustment(keyframe BA) in SLAM algorithm, and explain why we finally choose keyframe-based method.

2.1 World Representations and Notations

Visual-inertial odometry [20], literally, is an odometry system, which received environment information by visual (camera) and inertial (IMU) sensor. VIO is similar with well-known visual odometry (VO) problem [29], with an additional IMU sensor, it tries to estimate agent's pose as agent keep moving in the environment. One big difference between VIO and SLAM algorithm is that VIO does not or only build a simple map, whereas SLAM normally maintains and continuously updates a map.

To setup a VIO system, we need to first define the ways to represent the world. Globally, we have a world frame $\mathcal{W}$; World frame $\mathcal{W}$ is set to a right-handed Cartesian coordinate system that every objects has an absolute pose (translation and rotation) in it. Then we have local frame for each sensor, i.e., IMU frame $\mathcal{I}$ and camera frame $\mathcal{C}$; Every time camera and IMU sensor obtain observations within their own local frame, we need to integrate those data and estimate the pose of those sensors in world frame $\mathcal{W}$. Figure 2-1 shows the overall world representations. Both IMU frame and Camera frame are right-handed Cartesian coordinate system.

In this master thesis, we use following notations,

- We denote scalars as a, b, c, vectors as $\mathbf{a}, \mathbf{b}, \mathbf{c}$, matrices as $\mathbf{A}, \mathbf{B}, \mathbf{C}$, frames as $\mathcal{A}, \mathcal{B}, \mathcal{C}$.
- We denote measurement $\mathbf{m}$ in particular frame $\mathcal{F}$ as $\mathbf{m}_{\mathcal{F}}$. To further simplify, any parameter that is **not** in world frame shall be denoted particularly. For example, the translation $\mathbf{p}$ in camera frame will be denoted as $\mathbf{p}_{\mathcal{C}}$, and the translation $\mathbf{p}$ in world frame will be denoted as $\mathbf{p}$.

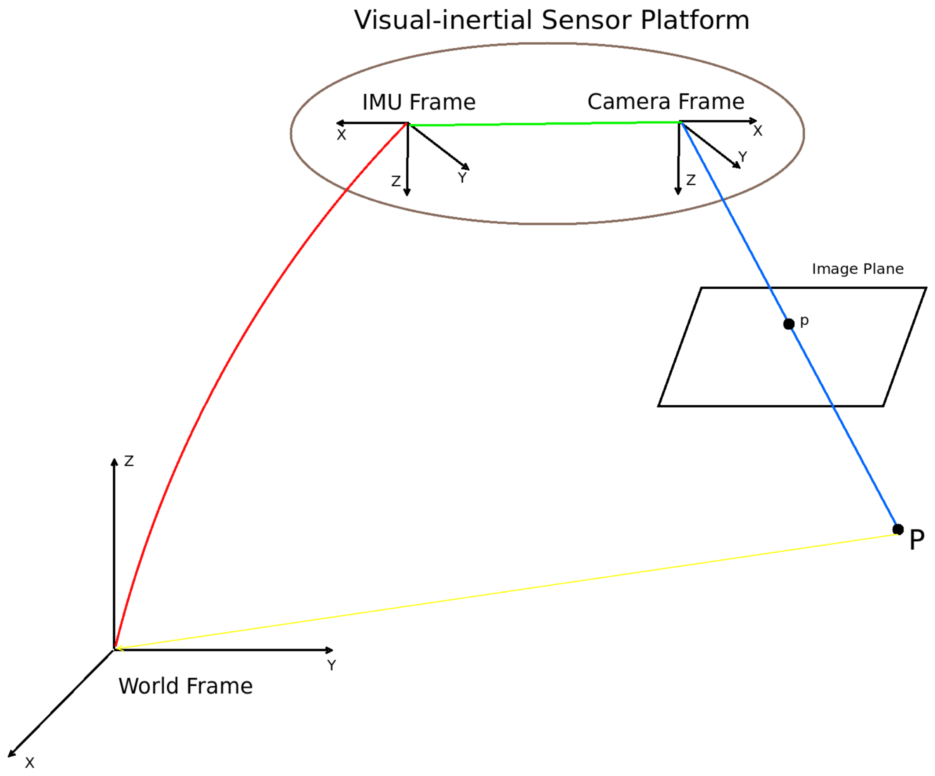


Figure 2-1: This figure shows the connection among world frame $\mathcal{W}$, IMU frame $\mathcal{I}$ and camera frame $\mathcal{C}$. Green line shows the transformation between camera and IMU, which can be pre-calibrated. Red line is the pose of IMU in world frame. Camera frame observes point $\mathbf{p}$ of object P in image plane, and connects a object P by blue line, and the coordinate of object P in world frame is presented as yellow line.

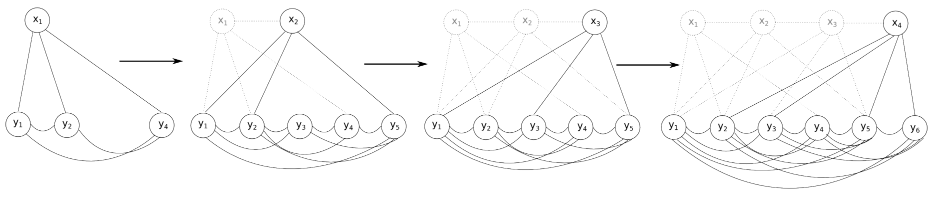
- A general translation $\mathbf{t}$ should express a translation from point A to point B in frame $\mathcal{C}$, which is denoted as $\mathbf{t}_{\mathcal{C}}^{AB}$. We simplify a point $\mathbf{p}$ in frame $\mathcal{A}$ as $\mathbf{p}_{\mathcal{A}}$, when this point is the translation $\mathbf{t}_{\mathcal{A}}^{OP}$, O is origin of frame $\mathcal{A}$, and $\mathbf{p} = P$, this holds same for vector.

- A general rotation is either expressed in quaternion $\mathbf{q}$ or rotation matrix $\mathbf{R}$. We use quaternion $\mathbf{q}$ as example. A quaternion is a orientation operation from frame $\mathcal{B}$ to frame $\mathcal{A}$, and it is denoted as $\mathbf{q}_{\mathcal{AB}}$ in this thesis. Noted that if such a operation is from world frame $\mathcal{W}$ to some frame $\mathcal{B}$, we omit both frame for simplification, i.e., $\mathbf{q}_{\mathcal{WB}} \triangleq \mathbf{q}$.

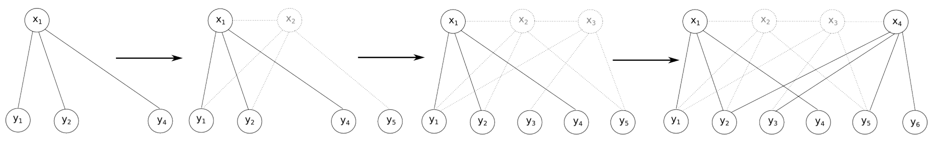
- We use hat operator $\hat{\boldsymbol{x}}$ to represent the estimation of state $\boldsymbol{x}$.

2.2 Filter Versus Keyframe

It is important to note that though this thesis focus on visual-inertial odometry for small workspace, we still tend to keep the possibility to extend our system to a general, scalable and efficient SLAM system. SLAM system usually have two parallel process,



(a) Filter



(b) Keyframe-based BA

Figure 2-2: (a) Filter method for SLAM. (b) Keyframe-based Bundle Adjustment (BA) for SLAM. We denote the i^{th} camera position as $\mathbf{x}_i$, i^{th} image feature as $\mathbf{y}_i$. We connect the line between camera and image feature if this feature is observed by this camera, the vanished observations is presented as dotted line, and the vanished camera is expressed as grey font. Both graph changes as time goes on from left to right. One can see from (a) that though only the latest camera pose is reserved, the edges between features are increasing exponentially. (b) stores some of former camera poses (keyframe) (i.e., $\mathbf{x}_1$ and $\mathbf{x}_4$) by keeping graph stay sparsity.

one is for localization and the other is for mapping, the crucial point of building such a system is to keep both processes efficient. There are two general framework (e.g., filter-based method and keyframe-based method) in SLAM. In this section, we want to discuss whether filter-based method or keyframe-based method are more suitable for our case.

Filter-based SLAM [6, 7, 5] uses *Extended Kalman Filter* (EKF) to propagate state and update the covariance of the state. In each step, system obtain the current pose estimation and map update by marginalising all former information. This marginalising step usually eliminates the former pose and adds connections to image features. As showed in Figure 2-2a, the graph will not grow fast with time since the former pose has been eliminated and features in environment is limited. However, once the system moves to large scale scene, the problem of limiting the number of features become severe as the graph tends to be fully-connected.

Keyframe-based SLAM [17, 26, 11, 8, 27] applies *bundle adjustment* (BA) for keyframes to update the map in each step. In keyframe-based SLAM, it stores some historical poses (keyframes), and combines with image feature points to do a BA step. The chosen of keyframes varies from implementations, the idea is to pick up the poses that is not very close to last keyframe, otherwise the information might be redundant, which might lead to increase the computational cost. In Figure 2-2b, the graph still stays sparsity as the number of poses and features increases. The drawback might be the behaviour of pose estimation is inadequate as it ignores some of former information.

In [33], they have shown that the computational cost for keyframe BA in $O(m^2 \cdot n)$, and for filter method is $O(n^3)$ where m is the number of key frames, and n is the number of landmarks. Therefore they conclude that keyframe-based SLAM is slightly better than filter-based SLAM in their experiment settings, especially when scale of scene becomes larger considering the number of landmarks will be much larger than key frames.

In this master thesis, we choose keyframe-based method for visual part and filter method for IMU integration part. We choose filter method for IMU integration part is that we do not keep former information (e.g., image features or landmarks) in integration step, hence each filter step can be regarded as an optimization step. The reason why we use keyframe-based method for visual part is that we want keep the scalability of our system, besides the results from IMU integration can be a good compensation in case of the lack of pose estimation in keyframe-based BA.

Chapter 3

Background on Quaternion Algebra

One important task for this master thesis is to integrate IMU data over time to estimate camera pose (e.g., position and orientation). By assuming movement model, it is straightforward to integrate position in Cartesian space, however integrating orientation in manifold space is often not a easy task. In this chapter, we introduce the quaternion algebra, and explore the way to operate quaternion over time.

3.1 Definition of Quaternion

A quaternion Q is defined as,

$$Q = q_w + q_x i + q_y j + q_z k \tag{3.1}$$

where $\{q_w, q_x, q_y, q_z\} \in \mathbb{R}$, and $\{i, j, k\}$ are three imaginary unit length, e.g., $i^2 = j^2 = k^2 = ijk = -1$.

In most cases, we represent quaternion Q as a four-element vector $\mathbf{q}$ composed by above four real number $\{q_w, q_x, q_y, q_z\}$, i.e.,

$$\mathbf{q} \triangleq [q_w \ \mathbf{q}_v]^T = [q_w \ q_x \ q_y \ q_z]^T \tag{3.2}$$

where q_w is the real part of $\mathbf{q}$, and q_v is a 3-vector to represent imaginary part of $\mathbf{q}$.

It is worth to be noted that there are two different conventions of quaternion $\mathbf{q}$, *Hamilton way* [14] and *JPL way* [4]. In Hamilton convention, the real part q_w is the first component of $\mathbf{q}$, i.e., $[q_w \ \mathbf{q_v}]$, whereas in JPL way, the real part is the fourth component, i.e., $[\mathbf{q_v} \ q_w]$. To avoid confusions, and considering Hamilton way is more common to use, especially for implementation [12, 16], we hereby claim that we use ***Hamilton way*** to represent quaternion $\mathbf{q}$ throughout this master thesis.

3.2 Properties of Quaternion

In this section, we will introduce properties of quaternion.

Summation We start with summation of two quaternions **q** and **p**,

$$\boldsymbol{q}+\boldsymbol{p} = [q_w + p_w \ \mathbf{q}_v + \mathbf{p}_v]^T = [q_w + p_w \ q_x + p_x \ q_y + p_y \ q_z + p_z \ ]^T \tag{3.3}$$

Product We use $\otimes$ to denote the product operation on quaternions, which gives,

$$\boldsymbol{q}\otimes\boldsymbol{p} = \begin{bmatrix} p_w q_w - p_x q_x - p_y q_y - p_z q_z \\ p_w q_x + p_x q_w + p_y q_z - p_z q_y \\ p_w q_y - p_x q_q + p_y q_w + p_z q_x \\ p_w q_z - p_x q_y - p_y q_z + p_z q_w \end{bmatrix} \tag{3.4}$$

The product of two quaternions can be expressed as two equivalent matrix products,

$$\boldsymbol{q_1}\otimes\boldsymbol{q_2} = Q_1^+\boldsymbol{q_2} \tag{3.5}$$

with

$$Q_{\boldsymbol{1}}^+ = q_w\mathbb{I} + \begin{bmatrix} 0 & -\boldsymbol{q_v}^T \\ \boldsymbol{q_v} & [\boldsymbol{q_v}]_\times \end{bmatrix} \tag{3.6}$$

where $[\boldsymbol{q}_v]_\times$ represents the cross-product matrix of $\boldsymbol{q}_v$, which the cross-product matrix of a vector $\boldsymbol{v}$ is defined by

$$[\boldsymbol{v}]_\times = \begin{bmatrix} 0 & -v_z & v_y \\ v_z & 0 & -v_x \\ -v_y & v_x & 0 \end{bmatrix} \tag{3.7}$$

note that the quaternion product is not commutative, i.e.,

$$\boldsymbol{p}\otimes\boldsymbol{q} \neq \boldsymbol{q}\otimes\boldsymbol{p} \tag{3.8}$$

however it is associative, and distributive over sum, i.e.,

$$\boldsymbol{p}\otimes(\boldsymbol{q}\otimes\boldsymbol{k}) = (\boldsymbol{p}\otimes\boldsymbol{q})\otimes\boldsymbol{k} \tag{3.9}$$

$$\boldsymbol{p}\otimes(\boldsymbol{q}+\boldsymbol{k}) = \boldsymbol{p}\otimes\boldsymbol{q} + \boldsymbol{p}\otimes\boldsymbol{k} \tag{3.10}$$

$$(\boldsymbol{q}+\boldsymbol{k})\otimes\boldsymbol{p} = \boldsymbol{q}\otimes\boldsymbol{p} + \boldsymbol{k}\otimes\boldsymbol{p} \tag{3.11}$$

Conjugate The conjugate $\boldsymbol{q}^*$of a quaternion is defined by

$$\boldsymbol{q}^* \triangleq q_w - \boldsymbol{q_v} = [q_w \ \ -\boldsymbol{q_v}]^T \tag{3.12}$$

Identity We call a quaternion **q** identical if, for any given quaternion **p**, $\boldsymbol{q}\otimes\boldsymbol{p} = \boldsymbol{p}\otimes\boldsymbol{q} = \boldsymbol{p}$. An identical quaternion **q** satisfies that,

$$\boldsymbol{q} = [1 \ 0 \ 0 \ 0]^T \tag{3.13}$$

In this master thesis, we denote identical quaternion as $\boldsymbol{q}_\mathbb{I}$.

Norm The norm of a quaternion $\|\boldsymbol{q}\|$ is defined similar to the norm of a vector, which is,

$$\|\boldsymbol{q}\| = \sqrt{q_w^2 + q_x^2 + q_y^2 + q_z^2} \tag{3.14}$$

Inverse The inverse of a quaternion $\boldsymbol{q}^{-1}$ is defined as,

$$\boldsymbol{q}^{-1} = \boldsymbol{q}^* / \|\boldsymbol{q}\| \tag{3.15}$$

which leads to,

$$\boldsymbol{q}^* \otimes \boldsymbol{q}^{-1} = \boldsymbol{q}^{-1} \otimes \boldsymbol{q}^* = \boldsymbol{q}_{\mathbb{I}} \tag{3.16}$$

Unit quaternion The norm of a unit quaternion $\|\boldsymbol{q}\|$ is 1, and the inverse of such a unit quaternion is equal to the conjugate of this quaternion,

$$\boldsymbol{q}^{-1} = \boldsymbol{q}^* \tag{3.17}$$

Pure quaternion A pure quaternion $\boldsymbol{q}$ is defined as,

$$\boldsymbol{q} = [0\ \boldsymbol{q}_v]^T = [0\ q_x\ q_y\ q_z]^T \tag{3.18}$$

Let pure quaternion $\boldsymbol{q} = \theta\boldsymbol{u}$, where $\theta = \|\boldsymbol{q}\|$, we can compute the exponential of $\boldsymbol{q}$ with the help of Euler formula,

$$\mathrm{e}^{\boldsymbol{q}} = \mathrm{e}^{\theta\boldsymbol{u}} = \cos\theta + \boldsymbol{u}\sin\theta = [\cos\theta\ \boldsymbol{u}\sin\theta]^T \tag{3.19}$$

which is still a quaternion, and moreover $\mathrm{e}^{\boldsymbol{q}}$ is a unit quaternion because its norm $\|\mathrm{e}^{\boldsymbol{q}}\|^2 = \cos\theta^2 + \sin\theta^2 = 1$.

3.3 Quaternions and Rotation operations

We discuss the relationship between quaternions and rotation operations by first introducing rotation vector $\boldsymbol{v}$.

Given a rotation vector $\boldsymbol{v} = \phi\boldsymbol{u}$, where ϕ is the norm of $\boldsymbol{v}$ and $\boldsymbol{u}$ is a unit vector, we can rotate a vector $\boldsymbol{x}$ by an angle ϕ around the axis $\boldsymbol{u}$ following right-handed rule, and obtain a new vector $\boldsymbol{x'}$,

$$\boldsymbol{x'} = \boldsymbol{x}_{||} + \boldsymbol{x}_{\perp}\cos\phi + (\boldsymbol{u} \times \boldsymbol{x})\sin\phi \tag{3.20}$$

where $\boldsymbol{x}_{||} = (\boldsymbol{x} \cdot \boldsymbol{u})\boldsymbol{u}$ is the component parallel to $\boldsymbol{x}$, and $\boldsymbol{x}_{\perp} = -\boldsymbol{u} \times (\boldsymbol{u} \times \boldsymbol{x})$ is the component perpendicilar to $\boldsymbol{x}$, therefore $\boldsymbol{x} = \boldsymbol{x}_{||} + \boldsymbol{x}_{\perp}$. This formula is known as *vector rotation formular* or *Rodrigues formula.*

We then can define a rotation matrix $\mathbf{R}$ by rotation vector $\boldsymbol{v} = \phi\boldsymbol{u}$ by the help of Equation (3.7) as,

$$\mathbf{R} = \mathrm{e}^{[\boldsymbol{v}]_\times} \tag{3.21}$$

we can rotate a vector $\boldsymbol{x}$ by an angle ϕ around the axis $\boldsymbol{u}$ using $\mathbf{R}$ in a clean way,

$$\boldsymbol{x'} = \mathbf{R}\boldsymbol{x} \tag{3.22}$$

one can show that result in Equation (3.22) is equivalent with the result in Equation (3.20) [39].

Constructing a unit quaternion $\boldsymbol{q}$ by Equation (3.19) with a rotation vector $\boldsymbol{v} = \phi\boldsymbol{u}$,

$$\boldsymbol{q} = \mathrm{e}^{\boldsymbol{v}/2} = \begin{bmatrix} \cos\phi/2 \\ \boldsymbol{v}\sin\phi/2 \end{bmatrix} \tag{3.23}$$

we can rotate a vector $\boldsymbol{x}$ by an angle ϕ around the axis $\boldsymbol{u}$ by,

$$\boldsymbol{x'} = \boldsymbol{q} \otimes \boldsymbol{x} \otimes \boldsymbol{q}^* \tag{3.24}$$

we then show $\boldsymbol{x'}$ in Equation (3.24) is equivalent with $\boldsymbol{x'}$ in Equation (3.20).

We transferred the vector $\boldsymbol{x}$ into pure quaternion form as,

$$\boldsymbol{x}_q = [0\ \boldsymbol{x}]^T \tag{3.25}$$

then we rewrite formula (3.24) as,

$$\begin{bmatrix} 0 \\ \boldsymbol{x'} \end{bmatrix} = \begin{bmatrix} \cos\phi/2 \\ \boldsymbol{v}\sin\phi/2 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ \boldsymbol{x} \end{bmatrix} \otimes \begin{bmatrix} \cos\phi/2 \\ -\boldsymbol{v}\sin\phi/2 \end{bmatrix} \tag{3.26}$$

expanding it by Equation (3.4), it is easily to show that,

$$\boldsymbol{x'} = \boldsymbol{x}_{||} + \boldsymbol{x}_{\perp}\cos\phi + (\boldsymbol{u} \times \boldsymbol{x})\sin\phi \tag{3.27}$$

which is exactly Equation (3.20).

To summarize here, we can construct a quaternion $\boldsymbol{q}$ or a rotation matrix $\mathbf{R}$ by any rotation vector $\boldsymbol{v} = \phi\boldsymbol{u}$, we denote such a quaternion as $\boldsymbol{q}\{\boldsymbol{v}\}$ and rotation matrix as $\mathbf{R}\{\boldsymbol{v}\}$ respectively. And a rotation operation of a vector $\boldsymbol{x}$ related to $\boldsymbol{v}$ can either be expressed as quaternion $\boldsymbol{q}\{\boldsymbol{v}\} \otimes \boldsymbol{x} \otimes \boldsymbol{q}\{\boldsymbol{v}\}^*$, or a rotation matrix $\mathbf{R}\{\boldsymbol{v}\}\boldsymbol{x}$. Note that we sometimes simplify $\mathbf{R}\{\boldsymbol{v}\}$ to $\mathbf{R}$, and/or $\boldsymbol{q}\{\boldsymbol{v}\}$ to $\boldsymbol{q}$ in this master thesis.

We also give conversion from rotation matrix $\mathbf{R}$ to quaternion $\boldsymbol{q}$. Knowing that,

$$\boldsymbol{q} \otimes \boldsymbol{x} \otimes \boldsymbol{q}^* = \mathbf{R}\boldsymbol{x} \tag{3.28}$$

we can construct $\mathbf{R} = \mathbf{R}\{\boldsymbol{q}\}$ by,

$$\mathbf{R} = \begin{bmatrix} q_w^2 + q_x^2 - q_y^2 - q_z^2 & 2(q_xq_y - q_wq_z) & 2(q_xq_z + q_wq_y) \\ 2(q_xq_y + q_wq_z) & q_w^2 - q_x^2 + q_y^2 - q_z^2 & 2(q_yq_z - q_wq_x) \\ 2(q_xq_z - q_wq_y) & 2(q_yq_z + q_wq_x) & q_w^2 - q_x^2 - q_y^2 + q_z^2 \end{bmatrix} \tag{3.29}$$

and a conversion from quaternion to rotation matrix can be found in [37].

3.4 Time-derivatives on Quaternion

We introduce the time-derivative on quaternion by first define,

$$\boldsymbol{q}(t + \Delta t) \triangleq \boldsymbol{q}(t) \otimes \Delta\boldsymbol{q} \tag{3.30}$$

where $\boldsymbol{q}(t)$ is the quaternion at time t and $\Delta\boldsymbol{q}$ is quaternion transformation within a small period time Δt.

One can expand $\Delta\boldsymbol{q}$ by Taylor expansions with Equation (3.23) to,

$$\Delta\boldsymbol{q} = \begin{bmatrix} 1 \\ \frac{1}{2}\Delta\boldsymbol{\theta} \end{bmatrix} + O(\|\Delta\boldsymbol{\theta}\|^2) \tag{3.31}$$

where $\Delta\boldsymbol{\theta}$ is a angular vector corresponding to $\Delta\boldsymbol{q}$. In fact, the angular rate $\boldsymbol{\omega}$ at time t is defined as,

$$\boldsymbol{\omega}(t) \triangleq \lim_{\Delta t \to 0} \frac{\Delta\boldsymbol{\theta}}{\Delta t} \tag{3.32}$$

which is one of measurements we can obtain from IMU sensor.

By definition of the derivative, we can obtain the time-derivative $\dot{\boldsymbol{q}}$ of quaternion $\boldsymbol{q}$ as,

$$\dot{\boldsymbol{q}} = \frac{d\boldsymbol{q}(t)}{dt} \triangleq \lim_{\Delta t \to 0} \frac{\boldsymbol{q}(t + \Delta t) - \boldsymbol{q}(t)}{\Delta t} \tag{3.33}$$

which follows,

$$\begin{aligned} \dot{\boldsymbol{q}} &\triangleq \lim_{\Delta t \to 0} \frac{\boldsymbol{q}(t + \Delta t) - \boldsymbol{q}(t)}{\Delta t} \\ &= \lim_{\Delta t \to 0} \frac{\boldsymbol{q} \otimes \Delta\boldsymbol{q} - \boldsymbol{q}}{\Delta t} \\ &= \lim_{\Delta t \to 0} \frac{\boldsymbol{q} \otimes (\begin{bmatrix} 1 \\ \frac{1}{2}\Delta\boldsymbol{\theta} \end{bmatrix} - \begin{bmatrix} 1 \\ 0 \end{bmatrix})}{\Delta t} \\ &= \frac{1}{2}\boldsymbol{q} \otimes \begin{bmatrix} 0 \\ \boldsymbol{\omega} \end{bmatrix} \end{aligned} \tag{3.34}$$

here we simplify $\boldsymbol{q}(t)$ to $\boldsymbol{q}$. Then we can obtain the time-derivative on quaternion by writing angular rate into pure quaternion form (3.18), which is,

$$\dot{\boldsymbol{q}} = \frac{1}{2}\boldsymbol{q} \otimes \boldsymbol{\omega} \tag{3.35}$$

3.5 Time-integration on Quaternion

To integrate quaternion over time, we explore the relationship between $\boldsymbol{q}(t_n)$ and $\boldsymbol{q}(t_{n+1})$ where $t_n = n\Delta t$. Expanding $\boldsymbol{q}(t_{n+1})$ using Taylor series, we have

$$\boldsymbol{q}(t_{n+1}) = \boldsymbol{q}(t_n) + \dot{\boldsymbol{q}}(t_n)\Delta t + \frac{1}{2!}\ddot{\boldsymbol{q}}(t_n)\Delta t^2 + \frac{1}{3!}\dddot{\boldsymbol{q}}(t_n)\Delta t^3 + \dots \tag{3.36}$$

Assume that the second order derivative of rotational rate is zero, which is $\ddot{\boldsymbol{\omega}} = 0$, we have

$$\dot{\boldsymbol{q}}(t_{n+1}) = \frac{1}{2}\boldsymbol{q}(t_n) \otimes \boldsymbol{\omega}()(t_n) \tag{3.37}$$

$$\ddot{\boldsymbol{q}}(t_{n+1}) = \frac{1}{2^2}\boldsymbol{q}(t_n) \otimes \boldsymbol{\omega}^2(t_n) + \frac{1}{2}\boldsymbol{q}(t_n) \otimes \dot{\boldsymbol{\omega}} \tag{3.38}$$

$$\dddot{\boldsymbol{q}}(t_{n+1}) = \frac{1}{2^3}\boldsymbol{q}(t_n) \otimes \boldsymbol{\omega}^3(t_n) + \frac{1}{4}\boldsymbol{q}(t_n) \otimes \dot{\boldsymbol{\omega}}\boldsymbol{\omega}(t_n) + \frac{1}{2}\boldsymbol{q}(t_n) \otimes \boldsymbol{\omega}(t_n)\dot{\boldsymbol{\omega}} \tag{3.39}$$

$$\vdots$$

and so forth and so on. We then get the result of time integration by taking Equation (3.37, 3.38, 3.39) back into Equation (3.36).

We hereby gives a stronger assumption that angular rate $\boldsymbol{\omega}(t_n)$ remains constant during a small time period Δt, which is $\dot{\boldsymbol{\omega}} = 0$. However, considering the sampling rate of IMU sensor is usually high ($>$ 100 Hz), this assumption actually is general and also gives us a cleaner expression of time integration.

Given $\dot{\boldsymbol{\omega}} = 0$, we can get

$$\boldsymbol{q}_{n+1} = \boldsymbol{q}_n \otimes (1 + \frac{1}{2}\boldsymbol{\omega}_n\Delta t + \frac{1}{2!}(\frac{1}{2}\boldsymbol{\omega}_n\Delta t)^2 + \frac{1}{3!}(\frac{1}{2}\boldsymbol{\omega}_n\Delta t)^3 + \frac{1}{4!}(\frac{1}{2}\boldsymbol{\omega}_n\Delta t)^4 + \dots) \tag{3.40}$$

here we regard $\boldsymbol{q}$ and $\boldsymbol{\omega}$ as series, which is exactly

$$\boldsymbol{q}_{n+1} = \boldsymbol{q}_n \otimes \mathrm{e}^{\boldsymbol{\omega}\Delta t/2} \tag{3.41}$$

we can rewrite it by Equation (3.23) as,

$$\boldsymbol{q}_{n+1} = \boldsymbol{q}_n \otimes \boldsymbol{q}\{\boldsymbol{\omega}_n\Delta t\} \tag{3.42}$$

which is called ***Zeroth order forward integration*** of quaternion over time.

We can obtain ***Zeroth order backward integration*** by assuming the angular rate remains $\boldsymbol{\omega}_{n+1}$ within Δt, then we have

$$\boldsymbol{q}_{n+1} = \boldsymbol{q}_n \otimes \boldsymbol{q}\{\boldsymbol{\omega}_{n+1}\Delta t\} \tag{3.43}$$

and ***Zeroth order midward integration*** by assuming the angular rate holds $\bar{\boldsymbol{\omega}}_{n+1} = (\boldsymbol{\omega}_n + \boldsymbol{\omega}_{n+1})/2$ within Δt,

$$\boldsymbol{q}_{n+1} = \boldsymbol{q}_n \otimes \boldsymbol{q}\{\bar{\boldsymbol{\omega}}_n\Delta t\} \tag{3.44}$$

Though not used in this master thesis, we notice that [35] gives ***First order integration*** by assuming angular rate is linear with time, i.e., $\dot{\boldsymbol{\omega}} = \frac{\boldsymbol{\omega}_{n+1} - \boldsymbol{\omega}_n}{\Delta t}$, which is

$$\boldsymbol{q}_{n+1} = \boldsymbol{q}_n \otimes \boldsymbol{q}\{\bar{\boldsymbol{\omega}}_n \Delta t\} + \frac{\Delta t^2}{48} \boldsymbol{q}_n \otimes (\boldsymbol{\omega}_n \otimes \boldsymbol{\omega}_{n+1} - \boldsymbol{\omega}_{n+1} \otimes \boldsymbol{\omega}_n) + \dots \tag{3.45}$$

in our quaternion convention.

Chapter 4

Modular Sensor Fusing

In previous chapter, we learned how to represent each frame and analysis the difference between filter method and keyframe BA method for our odometry system. We also learned quaternion algebra and basic approaches for quaternion integration or derivative overtime under some general assumptions. In this chapter, we will explore the details of our sensor fusing approach, which roughly uses so called *IMU loose integration framework* [38]. In such a framework, system propagates states via Kalman filter (KF) based on IMU measurements, extrasensory (e.g., camera, GPS etc.) data are used in correction step. Computational cost for KF-styled approach is usually linear, hence *IMU loose integration framework* provides a good trade-off between computational complexity and accuracy in a real-time robotic navigation system.

4.1 Error-state Kalman Filter for IMU Integration

The error-state Kalman Filter (ESKF) followed the paradigm of Kalman filter, it also has prediction and correction step. However, ESKF separates system into three different states: true state nominal state and error state. Nominal state processes large signal, which is integrable in non-linear fashion, whereas error state keeps track of error and noise term, which can be integrated in linear way. The composition of nominal state and error state, we call it true state, which is the final guess of system.

The ESKF has some nice properties when building a visual-inertial odometry:

1. The computation of Jacobian may be very fast, because the error state is small and all second order products are negligible. This is very important since we want the system run in real-time.
2. Integration of vision data with IMU data is straightforward in KF correction step. One can utilize the result of tracking to correct the IMU integration state.
3. Large signal has been integrated in nominal state, so that we can apply the correction step in a lower rate than prediction step.

The procedure of ESKF in this system can be explained as follows. IMU data first is integrated into nominal state via numerical integration methods, note that nominal state does not take noise term or error term into account, hence nominal state will accumulate errors. The error state then predict the error and noise using normal extended KF paradigm, meaning it will predict the mean and covariance of system's error. In parallel a correction step is performed at a lower rate, the results of visual tracking are used to correct error state, the error state then is injected into nominal state, which nominal state become the final guess of system at that time point. The system goes on until the criterion condition has been met.

We explain the ESKF for IMU integration in this section, and visual sensor as correction data in Section 4.2.

4.1.1 System Kinematics

We denote our true state $\boldsymbol{x}_t$ as,

$$\boldsymbol{x}_t = \boldsymbol{x}_n \oplus \boldsymbol{x}_e \tag{4.1}$$

where $\boldsymbol{x}_n$ is the nominal state for large signals, and $\boldsymbol{x}_e$ is error state for small error/noise signal, we use $\oplus$ to denote a general composition step.

We then introduce position $\boldsymbol{p}$, velocity $\boldsymbol{v}$, quaternion $\boldsymbol{q}$, accelerometer bias $\boldsymbol{a}_b$, gyroscope bias $\boldsymbol{\omega}_b$ and gravity vector $\boldsymbol{g}$ into true state, nominal state and error state respectively. The general composition step can be shown as,

$$\boldsymbol{p}_t = \boldsymbol{p}_n + \boldsymbol{p}_e \tag{4.2}$$

$$\boldsymbol{v}_t = \boldsymbol{v}_n + \boldsymbol{v}_e \tag{4.3}$$

$$\boldsymbol{q}_t \approx \boldsymbol{q}_n \otimes \begin{bmatrix} 1 \\ \boldsymbol{\theta}_e/2 \end{bmatrix} \tag{4.4}$$

$$\boldsymbol{a}_{bt} = \boldsymbol{a}_{bn} + \boldsymbol{a}_{be} \tag{4.5}$$

$$\boldsymbol{\omega}_{bt} = \boldsymbol{\omega}_{bn} + \boldsymbol{\omega}_{be} \tag{4.6}$$

$$\boldsymbol{g}_t = \boldsymbol{g}_n + \boldsymbol{g}_e \tag{4.7}$$

note that we derive Equation (4.4) by small angle approximation (Equation (3.31)), we apply angular error $\boldsymbol{\theta}_e$ instead of quaternion error in error state following classical approaches.

We then construct kinematic equations for true state, which are

$$\begin{aligned}
\dot{\boldsymbol{p}}_t =& \boldsymbol{v}_t & (4.8)\\
\dot{\boldsymbol{v}}_t =& \mathbf{R}_t(\boldsymbol{a}_m - \boldsymbol{a}_{bt} - \boldsymbol{a}_n) + \boldsymbol{g}_t & (4.9)\\
\dot{\boldsymbol{q}}_t =& \frac{1}{2}\boldsymbol{q}_t \otimes (\boldsymbol{\omega}_m - \boldsymbol{\omega}_{bt} - \boldsymbol{\omega}_n) & (4.10)\\
\dot{\boldsymbol{a}_{bt}} =& \boldsymbol{a}_w & (4.11)\\
\dot{\boldsymbol{\omega}_{bt}} =& \boldsymbol{\omega}_w & (4.12)\\
\dot{\boldsymbol{g}}_t =& 0 & (4.13)
\end{aligned}$$

where $\boldsymbol{a}_m$, $\boldsymbol{\omega}_m$ are the measurements from accelerometer and gyroscope respectively within **local frame**, $\boldsymbol{a}_n$, $\boldsymbol{\omega}_n$ are noises with those measurements, $\boldsymbol{a}_w$, $\boldsymbol{\omega}_w$ are white Gaussian noise together with accelerometer and gyroscope bias, and $\mathbf{R}_t$ is the rotation matrix corresponding to true state quaternion, i.e., $\mathbf{R}_t \triangleq \mathbf{R}_t\{\boldsymbol{q}\}$ regarding to Equation (3.29). We use similar notations in nominal state and error state.

We obtain kinematic equations for nominal state by cutting off all small signals, which leads to

$$\begin{aligned}
\dot{\boldsymbol{p}}_n =& \boldsymbol{v}_n & (4.14)\\
\dot{\boldsymbol{v}}_n =& \mathbf{R}_n(\boldsymbol{a}_m - \boldsymbol{a}_{bt}) + \boldsymbol{g}_n & (4.15)\\
\dot{\boldsymbol{q}}_n =& \frac{1}{2}\boldsymbol{q}_n \otimes (\boldsymbol{\omega}_m - \boldsymbol{\omega}_{bn}) & (4.16)\\
\dot{\boldsymbol{a}_{bn}} =& 0 & (4.17)\\
\dot{\boldsymbol{\omega}_{bn}} =& 0 & (4.18)\\
\dot{\boldsymbol{g}}_n =& 0 & (4.19)
\end{aligned}$$

and error state with small signals,

$$\begin{aligned}
\dot{\boldsymbol{p}}_e =& \boldsymbol{v}_e & (4.20)\\
\dot{\boldsymbol{v}}_e =& \mathbf{R}_n\left[\boldsymbol{a}_m - \boldsymbol{a}_{bn}\right]_\times - \mathbf{R}_n\boldsymbol{a}_{be} + \boldsymbol{g}_e - \mathbf{R}_n\boldsymbol{a}_n & (4.21)\\
\dot{\boldsymbol{\theta}}_e =& \left[\boldsymbol{\omega}_m - \boldsymbol{\omega}_{bn}\right]_\times - \boldsymbol{\omega}_{be} - \boldsymbol{\omega}_n & (4.22)\\
\dot{\boldsymbol{a}_{be}} =& \boldsymbol{a}_w & (4.23)\\
\dot{\boldsymbol{\omega}_{be}} =& \boldsymbol{\omega}_w & (4.24)\\
\dot{\boldsymbol{g}}_e =& 0 & (4.25)
\end{aligned}$$

it is trivial to derive Equation (4.20, 4.23, 4.24, 4.25), see Appendix A for derivation of Equation (4.21 and 4.22).

4.1.2 State Time-integration and Error-state Jacobian

We then gives time-integration equations between any two time stamp t_n and t_{n+1} where we measure the time difference Δt as $\Delta t = t_{n+1} - t_n$. In order to simplify our notations, we denote last state parameters as $\boldsymbol{x}$, and denote current state parameters as $\boldsymbol{x}'$, where current state is measured at time stamp t_n, and last state is measured at t_{n-1}. Same notations for error state. Therefore, time-integration equations for nominal state for one updating are

$$\begin{aligned}
\boldsymbol{p}'_n &= \boldsymbol{p}_n + \boldsymbol{v}_n \Delta t + \frac{1}{2}(\mathbf{R}_n(\boldsymbol{a}_m - \boldsymbol{a}_{bt}) + \boldsymbol{g}_n)\Delta t^2 && (4.26)\\
\boldsymbol{v}'_n &= \boldsymbol{v}_n + (\mathbf{R}_n(\boldsymbol{a}_m - \boldsymbol{a}_{bt}) + \boldsymbol{g}_n)\Delta t && (4.27)\\
\boldsymbol{q}'_n &= \boldsymbol{q}_n \otimes \boldsymbol{q}\{(\boldsymbol{\omega}_m - \boldsymbol{\omega}_{bn})\Delta t\} && (4.28)\\
\boldsymbol{a}'_{bn} &= \boldsymbol{a}_{bn} && (4.29)\\
\boldsymbol{\omega}'_{bn} &= \boldsymbol{\omega}_{bn} && (4.30)\\
\boldsymbol{g}'_n &= \boldsymbol{g}_n && (4.31)
\end{aligned}$$

we use ***Zeroth order forward integration*** explained in Section 3.5 to integrate our state over time, this is also called Euler method in Runge-Kutta numerical integration methods (see Appendix B.1).

We integrate our error state in same manner, except truncating second-order signal out. Hence we obtain the integration equations for error state

$$\begin{aligned}
\boldsymbol{p}'_e &= \boldsymbol{p}_e + \boldsymbol{v}_e \Delta t && (4.32)\\
\boldsymbol{v}'_e &= \boldsymbol{v}_e + (\mathbf{R}_n [\boldsymbol{a}_m - \boldsymbol{a}_{bn}]_\times - \mathbf{R}_n \boldsymbol{a}_{be} + \boldsymbol{g}_e)\Delta t + \boldsymbol{v}_i && (4.33)\\
\boldsymbol{\theta}'_e &= (\mathbf{R}_n^T\{\boldsymbol{\omega}_m - \boldsymbol{\omega}_{bn}\}\boldsymbol{\theta}_e - \boldsymbol{\omega}_{be})\Delta t + \boldsymbol{\theta}_i && (4.34)\\
\boldsymbol{a}'_{be} &= \boldsymbol{a}_{be} + \boldsymbol{a}_i && (4.35)\\
\boldsymbol{\omega}'_{be} &= \boldsymbol{\omega}_{be} + \boldsymbol{\omega}_i && (4.36)\\
\boldsymbol{g}'_e &= \boldsymbol{g}_e && (4.37)
\end{aligned}$$

where $\boldsymbol{v}_i$, $\boldsymbol{\theta}_i$, $\boldsymbol{a}_i$ and $\boldsymbol{\omega}_i$ are random impulses for velocity, angular error, accelerometer bias and gyroscope. Those impulses can be modelled by Gaussian process. We derive Equation (4.34) by close-formed integration methods described in Appendix B.2.

We then give the Jacobian of error state $\mathbf{J}_{x_e}$ for ESKF prediction step usage,

$$\mathbf{J}_{e'e} = \frac{\partial \boldsymbol{x}'_e}{\partial \boldsymbol{x}_e} = \begin{bmatrix} \mathbf{1} & \mathbf{1}\Delta t & 0 & 0 & 0 & 0 \\ 0 & \mathbf{1} & \mathbf{R}_n [\boldsymbol{a}_m - \boldsymbol{a}_{bn}]_\times \Delta t & \mathbf{R}_n \Delta t & 0 & \mathbf{1}\Delta t \\ 0 & 0 & \mathbf{R}_n^T\{\boldsymbol{\omega}_m - \boldsymbol{\omega}_{bn}\}\Delta t & 0 & -\mathbf{1}\Delta t & 0 \\ 0 & 0 & 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 & 0 & 0 & \mathbf{1} \end{bmatrix} \tag{4.38}$$

Noted that partial derivative between true state $\boldsymbol{x}_t$ and $\boldsymbol{x}_e$ is not identity because we use different parameters to represent orientations, e.g.quaternion in true state, angular error in error state. We then give the Jacobian of true state with respect to error state by

$$\mathbf{J}_{te} = \frac{\partial \boldsymbol{x}_t}{\partial \boldsymbol{x}_e} = \begin{bmatrix} \mathbf{1} & 0 & 0 & 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{\partial \boldsymbol{q}_t}{\partial \boldsymbol{\theta}_e} & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 & 0 & 0 & \mathbf{1} \end{bmatrix} \tag{4.39}$$

and by Equation (3.6), we have

$$\frac{\partial \boldsymbol{q}_t}{\partial \boldsymbol{\theta}_e} = \frac{1}{2} Q^+(\boldsymbol{q}) \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.40}$$

$$= \frac{1}{2} \begin{bmatrix} -q_x & -q_y & -q_z \\ q_w & -q_x & q_y \\ q_z & q_w & -q_x \\ -q_y & q_x & q_w \end{bmatrix} \tag{4.41}$$

$$\tag{4.42}$$

4.1.3 State Propagation

Initially, nominal state $\boldsymbol{x}_n$ has been set to a initial guess based on some prior knowledge, and there is no error at start, i.e., error state is set to zero. We assume error state $\boldsymbol{x}_e$ as a normal distribution, i.e., $\boldsymbol{x}_e = \mathcal{N}(\hat{\boldsymbol{x}}_e, \boldsymbol{\Sigma})$, where $\boldsymbol{\Sigma}$ denotes the covariance matrix for error state, which helps us to track the uncertainty of error state. Note that $\boldsymbol{\Sigma}$ is initialized to a very small diagonal matrix.

At certain round, we first obtain the measurements from accelerator and gyroscope, and compute a new nominal state estimation $\hat{\boldsymbol{x}'_n}$ by Equation (4.14) to (4.19). We then compute error state Jacobian $\mathbf{J}'_{e'e}$ by Equation(4.38), then update the error state and covariance matrix of current error state by,

$$\hat{\boldsymbol{x}'_e} = \mathbf{J}'_{e'e} \hat{\boldsymbol{x}}_e \tag{4.43}$$

$$\boldsymbol{\Sigma}' = \mathbf{J}'_{e'e} \boldsymbol{\Sigma} (\mathbf{J}'_{e'e})^T \tag{4.44}$$

which is called *prediction step* in ESKF. We omit prime symbol in next step, i.e.current state $\boldsymbol{x}$ is replaced by $\boldsymbol{x}'$.

We then assume correction measurement $\boldsymbol{y}$ from extrasensory data is a non-linear function with additional white Gaussian noise $w = \mathcal{N}(0, \mathbf{W})$ of our true state, i.e.,

$$\boldsymbol{y} = h(\boldsymbol{x}_t) + w \tag{4.45}$$

and the *correction step* of ESKF are as follows,

$$\mathbf{K} = \mathbf{\Sigma}\mathbf{H}^T(\mathbf{H}\mathbf{\Sigma}\mathbf{H}^T + \mathbf{W})^{-1} \tag{4.46}$$

$$\hat{\boldsymbol{x}_e'} = \mathbf{K}(\boldsymbol{y} - h(\hat{\boldsymbol{x}}_t)) \tag{4.47}$$

$$\mathbf{\Sigma}' = (\mathbf{1} - \mathbf{K}\mathbf{H})\mathbf{\Sigma} \tag{4.48}$$

where $\mathbf{H}$ is the Jacobian matrix of measurement function $h()$ with respect to error state $\boldsymbol{x}_e$ (see Section 4.2.2). Noted the estimation of true state here is the nominal state since we have not observed the mean of error state. The true state is estimated by Equation (4.1) and Equation (4.8) to (4.13). Depending on the output frequency of extra sensor, *correction step* often happens on a lower rate than *prediction step*. As always, we omit prime symbol in next few steps as state has been refreshed.

Before system enters into next round, we reset error state to initial state, i.e., $\hat{\boldsymbol{x}}_e = 0$ in our case. We update covariance matrix $\mathbf{\Sigma}$ by

$$\mathbf{\Sigma}' = \mathbf{J}_{ge}\mathbf{\Sigma}(\mathbf{J}_{ge})^T \tag{4.49}$$

where $\mathbf{J}_{ge}$ is Jacobian matrix of updated error state with respect to old error state, and it is given by

$$\mathbf{J}_{ge} = \begin{bmatrix} \mathbf{1}_6 & 0 & 0 \\ 0 & \mathbf{1} - \left[\frac{1}{2}\hat{\boldsymbol{\theta}}_e\right]_\times & 0 \\ 0 & 0 & \mathbf{1}_9 \end{bmatrix} \tag{4.50}$$

we derive it as similar way with the one shows in Appendix A.

4.2 Camera as Complementary Sensory Data

As we discussed in Section 4.1, we somehow need extrasensory data in ESKF *correction step*, we modelled this sensor as a non-linear measurement of true state $\boldsymbol{x}_t$ plus a white Gaussian noise as showed in Equation (4.45). This sensor in our case should satisfy following conditions,

- This sensor should carry rich information as we need to obtain accurate camera pose estimation from it.
- This sensor, unlike GPS, should work both inside and outside environment as our system is designed for mobile robots.
- This sensor should be light-weight, low-expense and better easy to handle as this is the general requirements for mobile robots.

which lead to our choice — camera. It is worthy to note that though we treat data in correction step as black box, it is also possible to choose a multiple sensory platform. However on the one hand, we choose camera not only it satisfies all above conditions, also the camera as most commonly-used sensor has been well-investigated and well-understanding, this could help us to find multiple possible approaches to solve our problem, e.g., camera pose estimation in our case; on the other hand, multiple extra sensors (i.e., camera+GPS) probably meet the some problems such as synchronizing between multiple sensory platforms. Therefore we choose camera as complementary data in this work.

4.2.1 Introduction to Monocular Visual Odometry

Visual odometry estimates the camera pose , and it can provide the measurements (e.g., global translation and rotation) we need in ESKF prediction step as the transformation between camera and IMU has been pre-calibrated. We here briefly introduce different types of visual odometry system. There are stereo-based odometry [25], depth-camera odometry [28], however considering we only use one camera in this work, we here mainly introduce monocular visual odometry.

A monocular odometry usually works as follows. Initially, system obtain the first guess of camera's pose by Homography [9] between two images. After a new image is acquired, system then tries to find the correspondences among common *landmarks*, where a landmark is defined as the most distinctive object in real world. Depending on the types of correspondences, we have

- **Feature-based methods** which is proposed in [6, 7, 17, 27], feature-based methods use image features to denote these correspondences, e.g., points are same if the features corresponds to it are same. System then reproject similar points from last image to current one using estimated camera pose transformation, error between points in current visual frame is called reprojection error.
- **Direct methods** which is proposed in [8], it uses image intensity (e.g., photometric error) to find such correspondences.
- **Semi-direct methods** which is showed in [11], it uses half image feature and half image intensity to find correspondences.

After the points connection between two images has been established, camera pose is estimated by minimizing the re-projection error or image intensity error.

Building a map based on such odometry is straightforward. A point is recognized as a part of landmark if it has been frequently tracked, and then will be inserted into the map as a recognized "map point"; A map is then constructed by such map points, normally a batch processing (e.g., bundle adjustment) is used for optimizing the map during mapping process. However such a technique can also be used for improving the pose estimation quality as we introduced in Section 4.2.3.

4.2.2 Self-adapt Map Scale

Estimation of camera pose from monocular visual odometry (VO) usually is a good compensation to ESKF IMU integration since global translation is unobservable to IMU integration. However since camera is a angle-sensor, it is impossible to obtain the scale of map [11], i.e., global translation from monocular VO has been scaled up/down with real world scale. People used to estimate map scale factor by aligning first few frames with ground-truth data [11], or initialize map scale by a standard object (e.g., a A4 paer) [6], it is still likely to accumulate scale drift with time goes on. In our framework, since IMU measures under a real world scale, we can propagate the map scale in our ESKF framework by introducing a scale factor in our state.

We first add scale factor $\boldsymbol{\lambda}$ which 3-vector to represent the scalability with x-axis, y-axis and z-axis respectively, now our true state, nominal state and error state has been changed into,

$$\boldsymbol{x}_t = [\boldsymbol{p}_t\ \boldsymbol{v}_t\ \boldsymbol{q}_t\ \boldsymbol{a}_{bt}\ \boldsymbol{\omega}_{bt}\ \boldsymbol{g}_t\ \boldsymbol{\lambda}_t]^T \tag{4.51}$$

$$\boldsymbol{x}_n = [\boldsymbol{p}_n\ \boldsymbol{v}_n\ \boldsymbol{q}_n\ \boldsymbol{a}_{bn}\ \boldsymbol{\omega}_{bn}\ \boldsymbol{g}_n\ \boldsymbol{\lambda}_n]^T \tag{4.52}$$

$$\boldsymbol{x}_e = [\boldsymbol{p}_e\ \boldsymbol{v}_e\ \boldsymbol{\theta}_e\ \boldsymbol{a}_{be}\ \boldsymbol{\omega}_{be}\ \boldsymbol{g}_e\ \boldsymbol{\lambda}_e]^T \tag{4.53}$$

the derivative with $\boldsymbol{\lambda}$ with respect to time is easily derived since we assume scale factor is independent with time, therefore we have

$$\dot{\boldsymbol{\lambda}}_t = 0 \tag{4.54}$$

$$\dot{\boldsymbol{\lambda}}_n = 0 \tag{4.55}$$

$$\dot{\boldsymbol{\lambda}}_e = 0 \tag{4.56}$$

it is then trivial to make corresponding changes to system kinematic equations (e.g., Equation 4.14 to 4.8) and error state Jacobian (e.g., Equation 4.39 and 4.40). The measurement function $h()$ of estimated true state $\dot{\boldsymbol{x}}_t$ can then be derived as

$$h(\boldsymbol{p}_t) = \boldsymbol{\lambda}_t \odot (\boldsymbol{p}_t - \boldsymbol{p}_{t0}) + \boldsymbol{p}_{t0} \tag{4.57}$$

$$h(\boldsymbol{v}_t) = \boldsymbol{v}_t \tag{4.58}$$

$$h(\boldsymbol{q}_t) = \boldsymbol{q}_t \tag{4.59}$$

$$h(\boldsymbol{a}_{bt}) = \boldsymbol{a}_{bt} \tag{4.60}$$

$$h(\boldsymbol{\omega}_{bt}) = \boldsymbol{\omega}_{bt} \tag{4.61}$$

$$h(\boldsymbol{g}_t) = \boldsymbol{g}_t \tag{4.62}$$

$$h(\boldsymbol{\lambda}_t) = \boldsymbol{\lambda}_t \tag{4.63}$$

where $\odot$ is point-wise vector multiplication and $\dot{\boldsymbol{p}}_{t0}$ is the camera/IMU position in reference frame, i.e., the position obtained from Homography of first two keyframes. Noted here, we assume the camera sensor and IMU sensor has identical position for simplification, i.e., $\dot{\boldsymbol{p}}_t$ is the estimated camera position in world frame. A more complicated case can be found in [23].

In Section 4.1, we did not give the explicit expression of $\mathbf{H}$ in Equation (4.46). $\mathbf{H}$ is defined as the Jacobian matrix of extrasensory measurement function $h()$ with respect to error state at nominal state $\boldsymbol{x}_n$, as $\boldsymbol{x}_n$ is the estimation of true state $\boldsymbol{x}_t$

here. By chain rule, $\mathbf{H}$ can be written as

$$\mathbf{H} \triangleq \frac{\partial h}{\partial \boldsymbol{x}_e}\bigg|_{\boldsymbol{x}_n} = \frac{\partial h}{\partial \boldsymbol{x}_t}\bigg|_{\boldsymbol{x}_n} \frac{\partial \boldsymbol{x}_t}{\partial \boldsymbol{x}_e}\bigg|_{\boldsymbol{x}_n} = \mathbf{J}_H \mathbf{J}_{te} \tag{4.64}$$

we already give $\mathbf{J}_{te}$ in Equation (4.39), we then derive $\mathbf{J}_H$, which leads to

$$\mathbf{J}_H = \begin{bmatrix} \mathbf{J}_p & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \mathbf{1}_3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1}_4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1}_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1}_3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \mathbf{1}_3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1}_3 \end{bmatrix} \tag{4.65}$$

where $\mathbf{J}_p$ is a 3×3 matrix where diagonal is three elements of $\boldsymbol{\lambda}_n$ respectively.

4.2.3 Keyframe-based Bundle Adjustment

At last, a keyframe-based bundle adjustment is used to further increase pose estimation accuracy.

A general bundle adjustment tries to optimize bunch of camera poses and 3D points together by minimizing the reprojection error between reprojected 3D points and predicted image points. Mathematically, we obtain our optimized camera poses vector $\boldsymbol{c}$ and 3D points vector $\boldsymbol{p}$ by

$$\{\boldsymbol{c}, \boldsymbol{p}\} = \arg\min_{\boldsymbol{c}_i, \boldsymbol{p}_j} \sum_{i=1}^{n} \sum_{j=1}^{m} Obj(CamProj(\boldsymbol{c}_i, \boldsymbol{p}_j), \mathbf{I}_{ij})^2 \tag{4.66}$$

where n, m is the number of camera poses and 3D points respectively, $\mathbf{I}_{ij}$ is the predicted 2D point position corresponding to j^{th} 3D point in i^{th} image frame, function $CamProj()$ reprojects the 3D point from global frame into camera frame and $Obj()$ is a general function (e.g., Euclidean distance) that measures the error between two 2D points.

As long as we assume our camera is pin-hole model, we then can give function $CamProj()$ as

$$CamProj(\boldsymbol{c}_i, \boldsymbol{p}_j) = K(\mathbf{R}\{\boldsymbol{c}_i\}\boldsymbol{p}_j) \tag{4.67}$$

$$K(\boldsymbol{P}_{\mathcal{C}}) = [u_0 \ v_0]^T + \begin{bmatrix} f_u & 0 \\ 0 & f_v \end{bmatrix} r \begin{bmatrix} \frac{P_x}{P_z} & \frac{P_y}{P_z} \end{bmatrix}^T \tag{4.68}$$

where $\mathbf{R}\{\boldsymbol{c}_i\}$ is the rotation matrix corresponding to camera pose, function K transforms a point P in camera frame $\mathcal{C}$ into 2D point in image plane, parameters u_0, v_0 are principle point, f_u, f_v are focal length, and r is the distortion factor, which those parameters are obtained by camera calibration.

At each key frame turn after correction step, we employ such bundle adjustment

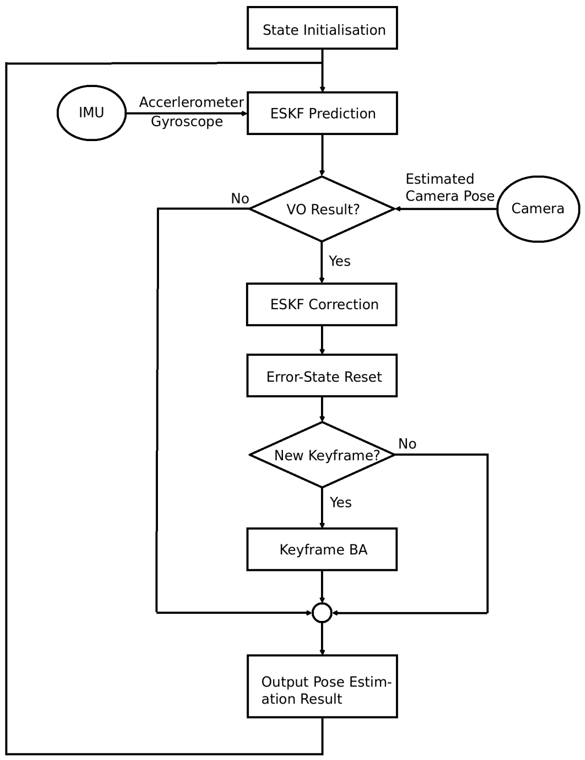


Figure 4-1: Pipeline of our visual-inertial odometry. Noted that measurement frequency from camera is 4 times lower than measurement frequency of IMU sensor in our experiment setting, therefore we do not have visual odometry (VO) result in some certain turn. Also whether this frame is keyframe is provided by VO.

step by inputting the camera pose in nominal state and 3D points obtained by VO; We will do an extra correction step using the camera pose we obtain from BA to further improve the estimation quality. In order to ensure the efficiency, we throw out the oldest key frame after key frame queue has been reached maximum number, we set this number to 20 in all our experiments. We observe this number is suitable and keep our visual-inertial odometry runs in real-time, and at the same improves the estimation accuracy.

Solutions to general bundle adjustment varies [44, 36, 21], we here roughly follow the work by [44] since it is efficient for sparse bundle adjustment problem.

4.3 Visual-inertial Odometry Pipeline Summary

In this section we summarize our visual-inertial odometry pipeline (see Figure 4-1). We also analysis the computational cost of our visual inertial odometry system in this section.

An initialisation step is needed for both nominal state and error state. After system obtains measurements (e.g., 3-vector from accelerometer, 3-vector from gyroscope) from IMU sensor, nominal state and error state are then predicted using

Equation (4.14) to (4.25), and system will update corresponding covariance matrix using Equation (4.43).

Now if camera gives estimated camera pose in this turn, system then applies a *correction step* as we describe in Section 4.1.3. After injecting error state from nominal state, we obtain the estimated true state which contains camera pose. System then reset the error state, and further improving the odometry estimation quality by applying a keyframe bundle adjustment (keyframe BA) if this camera frame is specified as key frame by visual odometry as we discuss in Section 4.2.3. Finally system output the estimated camera pose and start next round after receiving IMU measurements.

In our ESKF framework, computational complexity remains constant since we only keep the current states and covariance, and the size of states and covariance matrix are fixed during estimation process. To certify, we design an experiment (see Section 5.2.2) to show that computational time remains unchanged when running a single IMU integration process using ESKF. As we discuss in Section 2.2, the computational complexity is

$$O(m^2 \cdot n) \tag{4.69}$$

where m is the number of key frame, and n is the number of landmarks. We claim that our odometry can also be extended to large scale since the number of landmarks can be enlarged and this in general is the key factor of limiting the scalability of SLAM-like system. Moreover it is also more beneficial to obtain high estimation accuracy by increasing the number of landmarks than number of key frames [33].

We do not build a environment map due to the time limitation of this master thesis. However though a few more map optimization steps are needed, the key frame BA we propose in Section 4.2.3 gives a direct result of optimized landmarks, which are the main components of mapping. We explore this in future work chapter (Chapter 6).

All together, in this chapter we suggest a error-state kalman filter (ESKF) based visual-inertial odometry. The system accurately estimates the camera-IMU platform pose by fusing the measurements from IMU and camera sensors. By using a loosely-coupled approach, system runs ESKF in a constant computational complexity and needs no special initialization step, therefore it is suitable for localization of mobile robot in real-time. This odometry can also be extended to large scaled scene, and/or an efficient SLAM system.

Chapter 5

Experiments

In this chapter, we first introduce our synthetic dataset, this dataset contains IMU measurements and corresponding visual frames, it also provides the ground truth IMU-camera pose to be used in evaluation stage. We then perform some experiments to show that our visual-inertial odometry has good accuracy and low computational cost in several different experimental settings.

5.1 Synthetic Dataset

We use a synthetic IMU-camera dataset in this master thesis because it has following advantages rather than real dataset

- Noises in synthetic dataset is controllable. Though we have considered the noise model in our work, the types of noises in real data varies. Besides, denoising from such as IMU sensor is beyond the content of this master thesis.
- Synchronization between IMU and camera is annoying. Though we know the frequency of camera and IMU from datasheet or introduction, the real output might still differ, e.g., the first image frame might correspond to fourth IMU measurement in first trial but changes to fifth IMU measurement in next trial. In synthetic data, we could use aligned IMU and camera data.
- Synthetic data is more flexible. We could generate the special trajectory, i.e., a pure rotation sequence. However it is hard to create such a sequence in real platform.
- Synthetic data can provide accurate ground truth data at each timestamp. In real platform, a evaluation of position drift usually only happens at end point, e.g., setting the start point as same as end point and measure the drift. Motion capture system can provide ground truth data at each timestamp, however accuracy of such system usually is not high as synthetic data.
- Numerous calibration steps are saved. In real equipment, it is normal to calibrate the device again after several uses. We can ignore the calibration process in simulated platforms.

Name	Overall Duration $[s]$	Travelled Distance $[m]$	Description
001	30.0	165.97	Pure movement without rotation
002	30.0	0.0	Pure rotation without translation
003	30.0	118.99	Random straight trajectory
004	120.0	257.71	Random circle-like trajectory

Table 5.1: Trajectories we create for experiments, the datasets are named after the corresponding trajectories. Noted that **001** and **002** are ideal trajectories to test the correctness and performance of ESKF IMU integration. The trajectory **003** and **004** try to simulate the trajectories of micro helicopter by setting the similar velocity and pose.

Also to best of our knowledge, there is no open sourced IMU-camera synthetic data set, hence we decide to generate our own IMU-camera synthetic dataset.

Trajectory We start to generate synthetic data by first defining the trajectory of our IMU-camera platform. As we explain before, we assume the camera and IMU sensor share an identical pose in all our experiments for simplification. We have created four different types of trajectory showed in Table 5.1. Dataset **001** and Dataset **002** is used to evaluate the performance of ESKF IMU integration as we use ground-truth data in *correction step*; And Dataset **003** and Dataset **004** test the performance of visual-inertial odometry which **004** has longer travelled distance. We have made following several general assumptions of trajectories,

- We assume the initial position of trajectory as $(0, 0, 0)$ in global frame, and initial orientation as quaternion $(1, 0, 0, 0)$ which points at the positive z axis of global coordinate system.
- We assume the second derivative of position and orientation remains constant within two successive sensor samplings.
- We assume the second derivative of position and orientation obeys a normal distribution with additional white Gaussian noise.

Synthetic IMU data We then use IMUSim [45] to simulate IMU data by our customized trajectory. IMUSim is a powerful python-based open-sourced IMU simulation tool, which models a wide range of real-world environments and/or external noises. To obtain more realistic IMU data, we set the output frequency of IMU as 100 [Hz], sensitivity of gyroscope is 1200 [deg/s] and sensitivity of accelerometer is 4 gravity. The whole IMU model is noise-free with some additional white Gaussian noise. Overall we have 3-vector gyroscope readings, 3-vector accelerometer readings, 3-vector ground truth position, 4-vector ground truth orientation from IMUSim for single IMU sampling.

Dataset	fx	fy	cx	cy	d0	d1	d2	d3	d4
003	315.5	315.5	376.0	240.0	0.0	0.0	0.0	0.0	0.0
004	315.5	315.5	376.0	240.0	0.0	0.0	0.0	0.0	0.0

Table 5.2: We use the similar annotation for intrinsic matrix as in [34], such a parameter setting is similar to default ROS [30] camera.

Synthetic visual data Blender [40] is the main tool that creates virtual scene for our experiments. We use high-frequency grass-like texture and sun light to simulate the outdoor environment. Thanks to Python interface of Blender, we can import our defined trajectory, and then render a video clip as our visual data set. Here we use linear interpolation for position, e.g., a position $\boldsymbol{p}$ between two successive ground truth positions $\boldsymbol{p}_t$ and $\boldsymbol{p}_{t+1}$ can be computed as

$$\Delta t = \frac{t_p - t}{t_1 - t} \tag{5.1}$$

$$\boldsymbol{p} = \boldsymbol{p}_t + (\boldsymbol{p}_{t+1} - \boldsymbol{p}_t)\Delta t \tag{5.2}$$

where t_p, t_1 and t is time when $\boldsymbol{p}$, $\boldsymbol{p}_t$ and $\boldsymbol{p}_{t+1}$ has been measured. We use *Slerp* to interpolate between two quaternion $\boldsymbol{q}_t$ and $\boldsymbol{q}_{t+1}$ using Equation (5.1), we obtain the result $\boldsymbol{q}$ as

$$\boldsymbol{q} = \frac{\boldsymbol{q}_t \sin\left((1 - \Delta t)\theta\right) + \boldsymbol{q}_{t+1} \sin\left(\Delta t \theta\right)}{\sin\theta} \tag{5.3}$$

where θ is the half angle between quaternion $\boldsymbol{q}_t$ and $\boldsymbol{q}_{t+1}$. The resolution of image is 752×480, and intrinsic matrix of our virtual pin-hole camera is pre-defined, which is showed in Table 5.2. The output frequency is set to 25 [Hz], three times less than IMU output frequency.

5.2 Experimental Results

5.2.1 Implementation Details

Our implementation consists of a visual odometry based on SVO [11] and a ESKF framework. The goal of visual odometry is to track landmarks (for keyframe BA), estimate camera poses and select keyframes as we discuss in Section 4.2, and ESKF aims to integrate IMU measurements and fuse the visual result in correction step.

SVO [11] is a fast semi-direct monocular visual odometry. Instead of using image features, they estimate camera pose by minimizing photometric error between successive images. SVO is very efficient because high-cost feature extraction step has been skipped, besides a probabilistic mapping method has been used, therefore more reliable 3D landmarks can be obtained. Overall SVO provides a efficient and robustness way to obtain camera pose, keyframes and 3D landmarks.

We implement C++ based ESKF framework as we explain in Section 4.1. Mathematical operations such as quaternion operation, matrix operation are implemented with the help of efficient C++ template linear algebra library Eigen [13]. In keyframe BA part, the basic BA solver is provided by Google Ceres Solver [2], we select the maximum number of keyframes to 20 for efficiency reasons, the BA step runs in parallel with ESKF. IMU measurements has been already aligned with visual measurements manually (every four IMU data corresponds to one image frame).

All the experiments run in a Intel Core i7 4700MQ laptop, and all codes related to this master thesis will publish on `https://github.com/OscarLiXi/MasterThesis` afterwards.

5.2.2 Experiment 1: IMU Integration

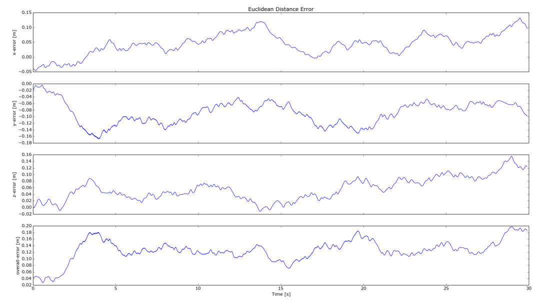
In this experiment, we examine the correctness and efficiency of IMU integration. since **001** and **002** are special cases that many VO system [5, 8] fails, and dataset **004** is a general simulated micro helicopter trajectory, we have used dataset **001**, **002**, and **004** (see Table 5.1) to test the correctness of our IMU integration. Moreover we assume the VO provides us a very accurate camera pose estimation in correction step at a lower rate since we only focus on IMU integration part.

We report the results in Figure 5-1 and Table 5.3. For trajectory **001**, the average position drift is approximately 0.125 [m], which is less than 0.08% of total travelled distance (see Table 5.1); for trajectory **002**, the average attitude drift is approximately 0.0134 [rad], which is less than 0.8 [deg]; for more realistic and longer trajectory **004**, the average position drift is larger because error accumulation is inevitable but still less than 0.55% of overall distance travelled, the result of average attitude error is also acceptable but has larger variance as we can see from Figure 5-1f. For processing time, one can see that time for processing a single IMU measurement does not grow as time goes by, which supports the claim that ESKF IMU integration runs under a constant time computational complexity; also ESKF needs approximately 0.019 [ms] in a single run, which runs in a real-time considering the output frequency of IMU is 100 [Hz].

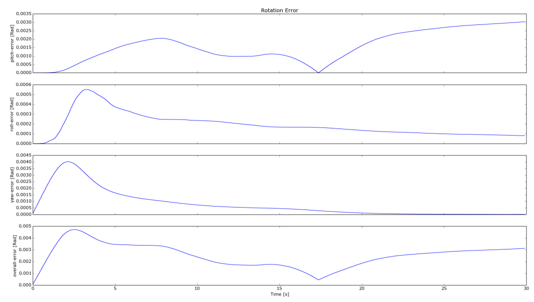
We conclude from this experiment that our ESKF IMU integration can deal with various types of trajectory and it has a precise pose estimation result **if VO system gives a accurate correction**, in next few experiments we will abandon this assumption and propose other comparisons. Furthermore, experimental results show that ESKF IMU runs in real-time, which its computational complexity remains constant as we point out in Section 4.3.

5.2.3 Experiment 2: VIO Versus VO

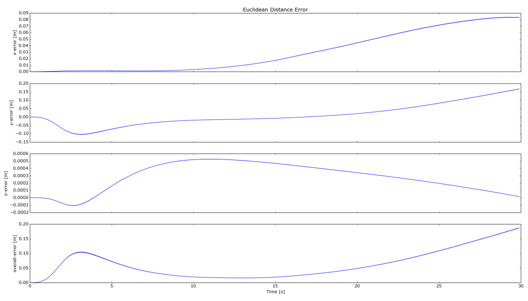
In this experiment, we will compare our suggested VIO with single VO using same dataset sequences. As we have discussed before, correction data by visual sensor has a large impact on our final pose estimation since global translation and angle rotated around gravity vector (yaw) are unobservable for IMU, ESKF somehow needs those results if it is not aware of exact movement model. Here we run experiments roughly



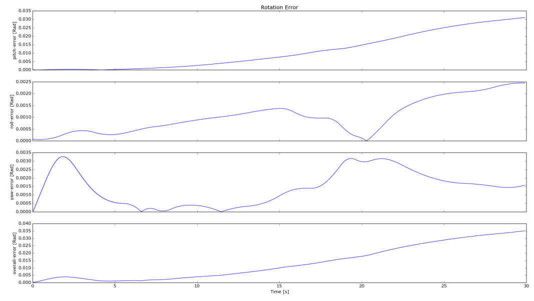
(a)



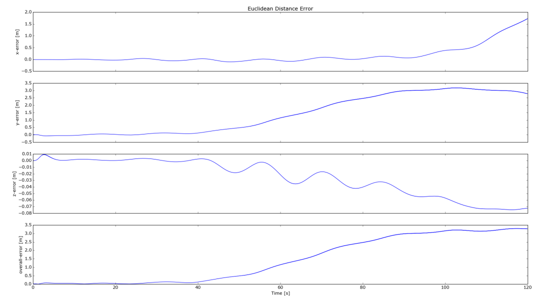
(b)



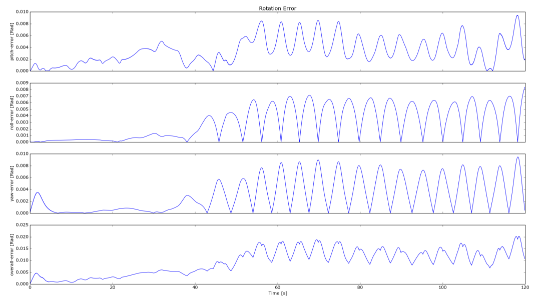
(c)



(d)



(e)

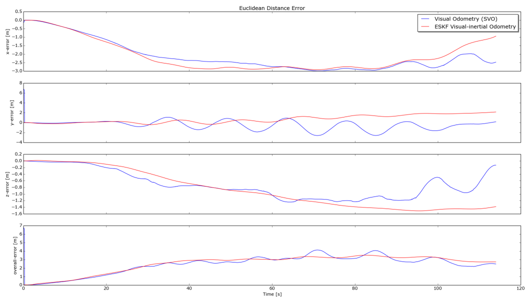


(f)

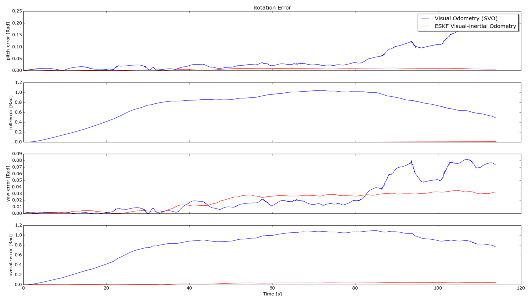
Figure 5-1: Experiment 1: (a)(b) Position and attitude drift of Dataset **001** (c)(d) Position and attitude drift of Dataset **002** (e)(f) Position and attitude drift of Dataset **004**. Position error is measured using Euclidean distance, and attitude drift is measured by transforming quaternion into Euler angle (see Appendix C). Ground truth data is provided by synthetic IMU data generator.

Dataset	APD [m]	AAD [Rad]	TPD [ms]
001	0.1251	0.0025	0.0184
002	0.0622	0.0134	0.0188
004	1.4319	0.0094	0.0187

Table 5.3: Experiment 1: APD (Average Position Drift) and AAD (Average Attitude Drift) is measured by averaging overall position and attitude drift. For TPD (Time-elapsed Per Data), we run the experiment 100 times to measure a overall elapsed time, then divide by number of experiments and number of IMU measurements to obtain the single data processing time.



(a)



(b)

Figure 5-2: Experiment 2: (a) Position drift of VIO and VO. (b) Attitude drift of VIO and VO. Both VIO and VO report an average position drift less than 1% of overall travelled distance, however VIO performs better than VO in attitude drift as VIO has 0.027 [rad] and VO has 0.787 [rad] average attitde drift.

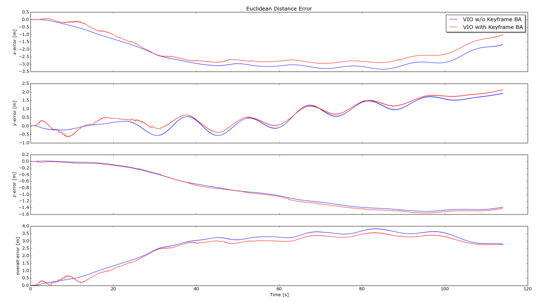
under the pipeline in Section 4.3 except we exclude the keyframe BA procedure, which we will discuss in next experiment.

The results of experiment 2 on trajectory **004** have been shown in Figure 5-2. One can see from figure that VIO has lower variance than VO in translation though they have similar drift. VIO is better than VO in attitude drift since VIO gains much lower error than VO in two observable parameters (pitch and roll), we can also see that VIO and VO report the similar error trends in yaw. Overall such results are competitive among similar systems [26, 10]. We also evaluate the processing time using similar way as in Section 5.2.2, VIO finally has an average time per IMU data as 4.570 [ms], which still runs in real-time.

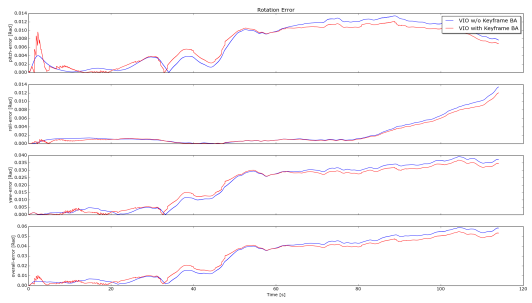
We conclude that the performance of VO has a large impact on 3-vector global translation and rotational angle around gravity vector. However ESKF frame has lower variance and less drift in pitch and roll.

5.2.4 Experiment 3: Keyframe Bundle Adjustment

We further show keyframe BA improves the estimation quality by performing experiment 3. In experiment 3, we run our VIO on trajectory **004** with and without kerframe BA. Each BA step only happens when new keyframe is inserted, it stops



(a)



(b)

Figure 5-3: Experiment 3: (a) Position drift of VIO with and without BA. (b) Attitude drift of VIO with and without BA. The curve shows the error between estimation and ground truth, more close to zero is better.

	Time Consuming [ms]
ESKF Prediction	0.0146
Visual Odometry	4.4656
ESKF Correction	0.0422
Keyframe BA	0.2671
Total Visual-Inertial Odometry	4.7913

Table 5.4: Experiment 4: Break-up timing results.

when meeting the maximum iteration number (we set this number to 10) or a threshold value. After BA step, an extra correction step will be applied on error state via the results from keyframe BA.

In Figure 5-3, we report the results of VIO with BA and without BA. One can see that red curve (VIO with keyframe BA) is more close to zero for most time, which shows that keyframe BA improves the camera pose estimation quality. Moreover blue curve reports an average position drift of 2.67 [m], where red curve reports 2.49 [m], which concludes that VIO with keyframe BA have 10 % less position drift than VIO without BA on average. For attitude, VIO with keyframe BA also have less drift than VIO without keyframe BA, which former reports an average attitude drift of 0.028 [rad] and the other is 0.029 [rad]. For processing time, keyframe BA increases approximately 0.5 [ms] for single IMU measurement on average, which leads to 5.08 [ms] per IMU measurement.

We conclude that keyframe BA as a motion optimization step improves the estimation quality in our case, and within 10 maximum iteration for single BA solution turn, the whole system still runs in real-time.

5.2.5 Experiment 4: Runtime Evaluation

Table 5.4 shows a break-up of average time consumption per IMU measurement to estimate camera motion in our synthetic datasets. In this experiment, we follow the parameter setting in former experiments, which explicitly shows in Table 5.5. We run

Name	Parameter Name	Value
ESKF IMU	Nominal state integration	Euler
	Error state integration	Truncated Euler
VO	Max number of features per image	120
	Max number of keyframes	20
Keyframe BA	Max number of camera poses	20
	Max number of iterations	10

Table 5.5: Experiment 4: parameter settings in experiment 4, which follows settings as we explained in experiment 1, experiment 2 and experiment 3.

experiments on Dataset **003** and Dataset **004** 100 times respectively to obtain the average processing time.

Though we have shown that our ESKF framework runs in constant time complexity, the whole VIO spends most of time on VO part. For VO part, we use *Fast* parameter setting as [11] suggests except we compromise the maximum keyframe number to 20 to obtain a more accurate estimation results. The whole system runs smoothly in real-time with our virtual IMU sensor and camera settings, which needs 4.7913 [ms] to process single IMU measurement. The potential time left could be used to build and optimize a environmental map.

We conclude that the VIO we suggest in this master thesis have the ability to accomplish a real-time navigation task. It will further be significantly speed-up when more efficient and robust VO has been exploited.

5.3 Conclusion

We first explain the reasons and ways we generate synthetic datasets in this chapter. Then we demonstrate four experiments on our synthetic dataset. In experiment 1, we use ground truth data in correction step, showing that if the extrasensory data gives us a good camera pose estimation, our visual-inertial odometry will obtain very little position drift (less than 0.55% of overall distance travelled) and attitude drift (average 0.0134 [rad]); In experiment 2, we compare our visual-inertial odometry with state-of-art visual odometry SVO, the final results show that our method have less variation in position drift and huge improvement (0.027 [rad] vs 0.787 [rad]) in average attitude drift with regarding to SVO; In experiment 3, we further show that keyframe BA we propose decrease the overall position drift and attitude drift; We examine the time consumption in experiment 4, which shows our odometry runs in real-time with our virtual IMU (100 HZ) and camera (25 Hz).

Due to the time limits, we are not available to perform more experiments, e.g., using different visual odometry or on more realistic dataset. However the results we obtain in above experiments have shown that the keyframe-based visual-inertial odometry we suggest is robust, efficient and high accuracy.

Chapter 6

Summary, Discussion and Future Works

In conclusion, we suggest a robust sensory fusing framework to be applied on visual-inertial odometry. This framework is lead by integrating IMU measurement from local frame to global frame, with additional extrasensory data (vision data) to complement unobservable parameters of IMU. We further explore a self-adapt map scale method and keyframe-based local bundle adjustment to increase the estimation accuracy.

The advantages of this framework is that system does not keep visual landmarks and IMU measurement, therefore runs in a constant time complexity, which can be easily extended a large scaled scene. The drawback is that it is more reliable to the performance of visual odometry, meaning that if the correction data gives bad feedback, the whole system is easily collapsed. However we have provided experimental results that our framework can give more stable and accuracy estimation of camera pose than current state-of-art visual odometry on same data sequence, thus we argue that our work is meaningful and the results of our system will further be increased with the development of visual odometry.

Another way to resolve this drawback is to apply feature-based visual-inertial methods (intense IMU integration), which could be a potential future work. More recent literatures show that feature-based VIO performs well and can be used commercially. [10] uses pre-integration theory and manifold optimization to optimize IMU and visual information together; [15], which is considered the foundation algorithm of Google Project Tango, runs a consistency analysis and modify some terms of traditional IMU integration to obtain less variation results. Unfortunately, we could not compare those latest algorithm due the un-release of their codes, it is definitely worth to try their ideas of VIO in the future. Because of the lack of resources and time, we are not available to build a real IMU-camera platform in this master thesis. A more rigorous denoise technique and more complicated noise model might have to be applied when using real data, which is also a possible way to work with in the future.

Appendix A

The Derivation of Error-state Kinematic Equations

We here derive Equation (4.21) and Equation (4.22) in Chapter 4.

In order to simplify our notation, we define

$$\boldsymbol{a}_n \triangleq \boldsymbol{a}_m - \boldsymbol{a}_{bn} \tag{A.1}$$

$$\boldsymbol{\omega}_n \triangleq \boldsymbol{\omega}_m - \boldsymbol{\omega}_{bn} \tag{A.2}$$

$$\boldsymbol{a}_e \triangleq -\boldsymbol{a}_{be} - \boldsymbol{a}_n \tag{A.3}$$

$$\boldsymbol{\omega}_e \triangleq -\boldsymbol{\omega}_{be} - \boldsymbol{\omega}_n \tag{A.4}$$

We derive Equation (4.21) by

$$\dot{\boldsymbol{v}}_e = \dot{\boldsymbol{v}}_t - \dot{\boldsymbol{v}}_n \tag{A.5}$$

$$= (\mathbf{R}_t \boldsymbol{a}_t + \boldsymbol{g}_t) - (\mathbf{R}_n \boldsymbol{a}_n + \boldsymbol{g}_n) \tag{A.6}$$

we then uses small signal approximation for R_t by Equation (3.29) and Equation (3.31), which leads to

$$\mathbf{R}_t = \mathbf{R}_n(\mathbf{1} + [\boldsymbol{\theta}_n]_\times) + O(\|\Delta\boldsymbol{\theta}_e\|^2) \tag{A.7}$$

we omit the second order of angular term and followed by Equation (4.15), we obtain

$$\dot{\boldsymbol{v}}_e = (\mathbf{R}_n(\mathbf{1} + [\boldsymbol{\theta}_n]_\times)\boldsymbol{a}_t + \boldsymbol{g}_t) - (\mathbf{R}_n \boldsymbol{a}_n + \boldsymbol{g}_n) \tag{A.8}$$

$$= \mathbf{R}_n(\mathbf{1} + [\boldsymbol{\theta}_n]_\times)(\boldsymbol{a}_n + \boldsymbol{a}_e) + \boldsymbol{g}_e \tag{A.9}$$

By applying the property of skew-symmetric matrix $[\boldsymbol{a}]_\times \boldsymbol{b} = [\boldsymbol{b}]_\times \boldsymbol{a}$, and recalling (A.1), (A.2). We have

$$\dot{\boldsymbol{v}}_e = \mathbf{R}_n \left[\boldsymbol{a}_m - \boldsymbol{a}_{bn}\right]_\times - \mathbf{R}_n \boldsymbol{a}_{be} + \boldsymbol{g}_e - \mathbf{R}_n \boldsymbol{a}_n \tag{A.10}$$

which is exactly Equation (4.21).

We then derive Equation (4.22) by

$$\dot{\boldsymbol{q}}_e = \frac{1}{2}(\boldsymbol{q}_e \otimes \boldsymbol{\omega}_t - \boldsymbol{\omega}_n \otimes \boldsymbol{q}_e) \tag{A.11}$$

and we have pure quaternion $\dot{\boldsymbol{\theta}}_e = 2\dot{\boldsymbol{q}}_e$. By expanding all terms in above equations, we have

$$\dot{\boldsymbol{\theta}}_e = -\left[\boldsymbol{\omega}_n\right]_\times \boldsymbol{\theta}_e + \boldsymbol{\omega}_e + O(\|\Delta\boldsymbol{\theta}_e\|^2) \tag{A.12}$$

By omitting all second-oder terms and recalling (A.3), (A.4), we obtain

$$\dot{\boldsymbol{\theta}}_e = \left[\boldsymbol{\omega}_m - \boldsymbol{\omega}_{bn}\right]_\times - \boldsymbol{\omega}_{be} - \boldsymbol{\omega}_n \tag{A.13}$$

which is Equation (4.22).

Appendix B

Integration Methods

B.1 Runge-Kutta Numerical Integration Methods

Runge-Kutta methods aims to give a approximate solutions of ordinary differential equations, e.g., our time-derivative state $\dot{\boldsymbol{x}}$, which is

$$\dot{\boldsymbol{x}} = f(t, \boldsymbol{x}) \tag{B.1}$$

The solution give in Equation (4.14) - (4.19) by simplest version of Runge-Kutta methods, e.g., assuming $\dot{\boldsymbol{x}}$ over each time period Δt, which gives us

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + f(t_n, \boldsymbol{x}_n)\Delta t \tag{B.2}$$

More complicated Runge-Kutta methods please refers to [43].

B.2 Closed-form Integration Methods

We first gives a clean version of Equation (4.22), which is

$$\dot{\boldsymbol{\theta}}_e = -\left[\boldsymbol{\omega}\right]_\times \boldsymbol{\theta}_e \tag{B.3}$$

we update $\boldsymbol{\theta}_e$ by

$$\begin{aligned}
\boldsymbol{\theta}_e' &= \boldsymbol{\theta}_e + \dot{\boldsymbol{\theta}}_e \Delta t && \text{(B.4)}\\
&= \boldsymbol{\theta}_e - \left[\boldsymbol{\omega}\right]_\times \boldsymbol{\theta}_e \Delta t && \text{(B.5)}\\
&= \mathbf{R}\{-\boldsymbol{\omega}\Delta t\} && \text{(B.6)}\\
&= \mathbf{R}\{\boldsymbol{\omega}\Delta t\}^T && \text{(B.7)}
\end{aligned}$$

which we apply Equation (3.29) and Equation (3.31) from Equation (B.5) to (B.6). This integration is a closed-form integration.

Appendix C

Conversion from Quaternions to Euler Angles

We introduce conversion from quaternions to Euler angles in this appendix. Though there are many conventions of Euler angles, we hereby specifically use so-called *Tait Bryan angles* to represent Euler angle. Tait Bryan angles is composed by Roll, Pitch and Yaw, which is the rotation angle around X-axis, Y-axis and Z-axis respectively. A quaternion $\boldsymbol{q}$ might be represented as

$$\boldsymbol{q} = [q_w \; q_x \; q_y \; q_z]^T \tag{C.1}$$

then Roll ϕ, Pitch θ and Yaw ψ can be obtained by

$$\phi = \arctan \frac{2(q_w q_x + q_y q_z)}{1 - 2(q_x^2 + q_y^2)} \tag{C.2}$$

$$\theta = \arcsin\left(2(q_w q_y - q_z q_x)\right) \tag{C.3}$$

$$\psi = \arctan \frac{2(q_w q_z + q_x q_y)}{1 - 2(q_y^2 + q_z^2)} \tag{C.4}$$

$$\tag{C.5}$$

which is precisely the attitude expression we use in experiment parts.

Bibliography

[1] Imu sensor https://www.vboxautomotive.co.uk/index.php/en/products/modules/inertial-measurement-unit, 2016. [Online; accessed 22-March-2016].

[2] Sameer Agarwal, Keir Mierle, and Others. Ceres solver. http://ceres-solver.org.

[3] Maxim A Batalin, Gaurav S Sukhatme, and Myron Hattig. Mobile robot navigation using a sensor network. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 1, pages 636–641. IEEE, 2004.

[4] WG Breckenridge. Quaternions proposed standard conventions. *Jet Propulsion Laboratory, Pasadena, CA, Interoffice Memorandum IOM*, pages 343–79, 1999.

[5] Andrew J Davison. Real-time simultaneous localisation and mapping with a single camera. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 1403–1410. IEEE, 2003.

[6] Andrew J Davison, Ian D Reid, Nicholas D Molton, and Olivier Stasse. Monoslam: Real-time single camera slam. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(6):1052–1067, 2007.

[7] Ethan Eade and Tom Drummond. Monocular slam as a graph of coalesced observations. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8. IEEE, 2007.

[8] Jakob Engel, Thomas Schöps, and Daniel Cremers. Lsd-slam: Large-scale direct monocular slam. In *Computer Vision–ECCV 2014*, pages 834–849. Springer, 2014.

[9] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[10] Christian Forster, Luca Carlone, Frank Dellaert, and Davide Scaramuzza. Imu preintegration on manifold for efficient visual-inertial maximum-a-posteriori estimation. In *Robotics: Science and Systems (RSS)*, 2015.

[11] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. Svo: Fast semi-direct monocular visual odometry. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 15–22. IEEE, 2014.

[12] G Guennebaud and B Jacob. The eigen c++ template library for linear algebra. *http:/eigen. tuxfamily. org*, 2010.

[13] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[14] William Rowan Hamilton. Ii. on quaternions; or on a new system of imaginaries in algebra. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 25(163):10–13, 1844.

[15] Joel A Hesch, Dimitrios G Kottas, Sean L Bowman, and Stergios I Roumeliotis. Consistency analysis and improvement of vision-aided inertial navigation. *Robotics, IEEE Transactions on*, 30(1):158–176, 2014.

[16] Roland Hess. *The essential Blender: guide to 3D creation with the open source suite Blender*. No Starch Press, 2007.

[17] Georg Klein and David Murray. Parallel tracking and mapping for small ar workspaces. In *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*, pages 225–234. IEEE, 2007.

[18] Taehee Lee, Joongyou Shin, and Dongil Cho. Position estimation for mobile robot using in-plane 3-axis imu and active beacon. In *Industrial Electronics, 2009. ISIE 2009. IEEE International Symposium on*, pages 1956–1961. IEEE, 2009.

[19] Robert W Levi and Thomas Judd. Dead reckoning navigational system using accelerometer to measure foot impacts, December 10 1996. US Patent 5,583,776.

[20] Mingyang Li and Anastasios I Mourikis. Consistency of ekf-based visual-inertial odometry. *University of California Riverside, Tech. Rep*, 2011.

[21] Manolis IA Lourakis and Antonis A Argyros. Sba: A software package for generic sparse bundle adjustment. *ACM Transactions on Mathematical Software (TOMS)*, 36(1):2, 2009.

[22] David G Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999.

[23] Simon Lynen, Markus W Achtelik, Steven Weiss, Maria Chli, and Roland Siegwart. A robust and modular multi-sensor fusion approach applied to mav navigation. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 3923–3929. IEEE, 2013.

[24] BL McNaughton, Lijiang Chen, and EJ Markus. âĂIJdead reckoning,âĂİ landmark learning, and the sense of direction: a neurophysiological and computational hypothesis. *Cognitive Neuroscience, Journal of*, 3(2):190–202, 1991.

[25] Christopher Mei, Gabe Sibley, Mark Cummins, Paul M Newman, and Ian D Reid. A constant-time efficient stereo slam system. In *BMVC*, pages 1–11, 2009.

[26] Anastasios Mourikis, Stergios Roumeliotis, et al. A multi-state constraint kalman filter for vision-aided inertial navigation. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 3565–3572. IEEE, 2007.

[27] Raul Mur-Artal, JMM Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *Robotics, IEEE Transactions on*, 31(5):1147–1163, 2015.

[28] Richard A Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*, pages 127–136. IEEE, 2011.

[29] David Nistér, Oleg Naroditsky, and James Bergen. Visual odometry. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 1, pages I–652. IEEE, 2004.

[30] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5, 2009.

[31] Edward Rosten, Reid Porter, and Tom Drummond. Faster and better: A machine learning approach to corner detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(1):105–119, 2010.

[32] Jianbo Shi and Carlo Tomasi. Good features to track. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on*, pages 593–600. IEEE, 1994.

[33] Hauke Strasdat, JMM Montiel, and Andrew J Davison. Real-time monocular slam: Why filter? In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 2657–2664. IEEE, 2010.

[34] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of rgb-d slam systems. In *Proc. of the International Conference on Intelligent Robot Systems (IROS)*, Oct. 2012.

[35] Nikolas Trawny and Stergios I Roumeliotis. Indirect kalman filter for 3d attitude estimation. *University of Minnesota, Dept. of Comp. Sci. & Eng., Tech. Rep*, 2:2005, 2005.

[36] Bill Triggs, Philip F McLauchlan, Richard I Hartley, and Andrew W Fitzgibbon. Bundle adjustmentâĂŤa modern synthesis. In *Vision algorithms: theory and practice*, pages 298–372. Springer, 1999.

[37] JMP Van Waveren. From quaternion to matrix and back. *Id Software Inc*, 2005.

[38] Stephan M Weiss. *Vision based navigation for micro helicopters.* PhD thesis, Citeseer, 2012.

[39] Wikipedia. Rodrigues' rotation formula — wikipedia, the free encyclopedia, 2015. [Online; accessed 6-April-2016].

[40] Wikipedia. Blender (software) — wikipedia, the free encyclopedia, 2016. [Online; accessed 21-April-2016].

[41] Wikipedia. Global positioning system — wikipedia, the free encyclopedia, 2016. [Online; accessed 24-March-2016].

[42] Wikipedia. Robotics — wikipedia, the free encyclopedia, 2016. [Online; accessed 22-March-2016].

[43] Wikipedia. RungeâĂŞkutta methods — wikipedia, the free encyclopedia, 2016. [Online; accessed 11-April-2016].

[44] Changchang Wu, Sameer Agarwal, Brian Curless, and Steven M Seitz. Multicore bundle adjustment. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 3057–3064. IEEE, 2011.

[45] Alexander D Young, Martin J Ling, and Damal K Arvind. Imusim: A simulation environment for inertial sensing algorithm design and evaluation. In *Information Processing in Sensor Networks (IPSN), 2011 10th International Conference on*, pages 199–210. IEEE, 2011.