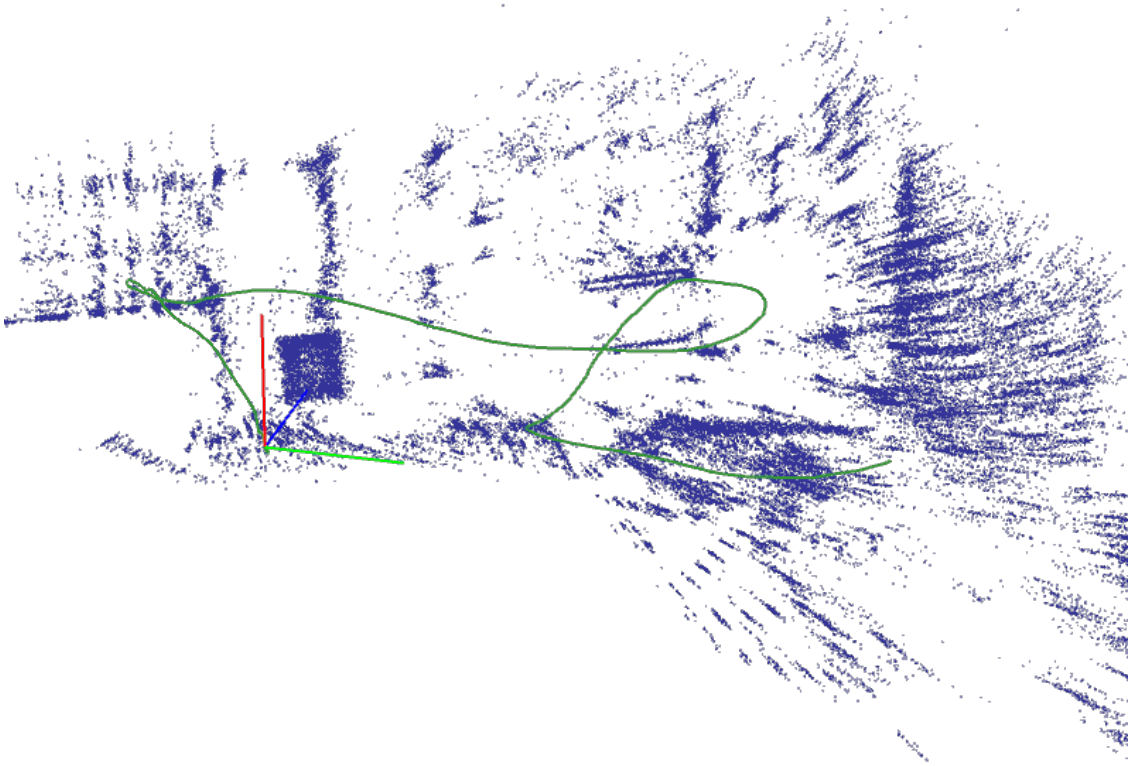
CHALMERS
UNIVERSITY OF TECHNOLOGY



Real-Time Pose Estimation by Fusing Visual and Inertial Sensors for Autonomous Driving

Master's thesis in Sustainable Energy Systems and Embedded Electronic System Design

Ruguang You
Hao Hou

Department of Mechanics and Maritime Sciences
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2019

Master's thesis in Sustainable Energy Systems and Embedded Electronic System Design

Real-Time Pose Estimation by Fusing Visual and Inertial Sensors for Autonomous Driving

RUGUANG YOU
HAO HOU

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mechanics and Maritime Sciences
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology
Gothenburg, Sweden 2019

Real-Time Pose Estimation by Fusing Visual and Inertial Sensors for Autonomous Driving

RUGUANG YOU
HAO HOU

Supervisor: Björnborg Nguyen, Department of Mechanics and Maritime Sciences
Examiner: Ola Benderius, Department of Mechanics and Maritime Sciences
Master's Thesis 2019:19
Department of Mechanics and Maritime Sciences
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2019

Abstract

The thesis works on developing a visual inertial odometry with feature-based stereo visual frontend, IMU preintegration and sliding window nonlinear optimization backend. The fusion of vision sensors and inertial sensors provide a robust and complementary pose estimation system to overcome the weaknesses of the vision-only or IMU-only systems. Stereo camera is easier to acquire the depth compared to the single view in monocular one, meanwhile has larger range and cheaper price than the RGB–D camera. When considering data association between several images, the ORB methods are applied to trade off between computational accuracy and efficiency. The inertial measurement preintegration is implemented to transform a number of measurements between selected keyframes into single relative motion constraints, which provides efficient handling of high rate sensor inputs. The problem as a whole is constructed as a bundle adjustment problem with motion constraints. It is treated as least square problems and solved with the LM method. The system performance is evaluated through EuRoC datasets, considering slow and fast motion, bright and dark scene. The average processing speed falls in the range of 19 to 26 FPS. The minimum translation and rotation error can respectively reach 0.11 m and 1.39 °. Overall this work proposes a new VIO framework, examines its performance by evaluating open datasets as well as comparing to the existing VIO algorithms, and also brings up discussions on possible directions for further improvement.

Keywords: pose estimation, Visual Inertial Odometry, stereo vision, data association, ORB, preintegration, Bundle Adjustment, least squares, nonlinear optimization

Acknowledgements

A very special thank to our examiner Ola Benderius and supervisor Björnborg Nguyen for the professional guidance and support in the whole thesis work. A big gratitude to Christian Berger for his technique help in Linux, programming and system design. Thanks to Arpit Karsolia and Fredrik von Corswant and other faculties at the Revere Lab for hosting us, offering their expertise in all areas. Also thanks to Naichen Wang, Love Mowitz, Nam Vu, Max Shvetsov, Felix Hörnschemeyer and the other Chalmers formula student driverless team members with whom we shared a great time during the project.

Ruguang You, Hao Hou, Gothenburg, Sep 2019

Contents

List of Figures

List of Tables

Acronyms

AD Autonomous Driving. 1
API Application Programming Interfaces. 2

BA Bundle Adjustment. v, 27
BRIEF Binary Robust Independent Elementary Features. 10
BRISK Binary Robust Invariant Scalable Keypoints. 1

CFSD Chalmers Formula Student Driverless. 2, 5, 47, 53, 54

EKF Extended Kalman Filter. 26

FAST Features from Accelerated Segment Test. 10
FPS Frame per Second. v, 2, 6, 43–45, 53, 54

ICP Iterative Closet Point. 21
IMU Inertial Measurement Unit. v, ix, 2, 3, 5, 11–13, 16, 17, 19–21, 25–27, 47, 48, 53

LM Levenberg–Marquardt. v, 18

MAV Micro Aerial Vehicle. 19, 30
MEMS Microelectromechanical Systems. 11

ORB Oriented FAST and Rotated BRIEF. v, ix, xi, 1, 2, 6, 9, 10, 20, 23, 24, 47, 53

PnP Perspective–n–Points. 21

RANSAC Random Sample Consensus. 24
RGB–D RGB–Depth. v, 1, 6

SfM Structure–from–Motion. 21, 22
SIFT Scale–Invariant Feature Transform. 1, 9, 47
SLAM Simultaneous Localization and Mapping. 2, 27, 47, 54
SURF Speeded–Up Robust Feature. 1, 9, 47

VIO Visual Inertial Odometry. v, 2

1
Introduction

Autonomous Driving (AD) is the topic for the future of the automotive industry. An increasing amount of companies and universities are active in this field to develop better solutions for self-driving vehicles. Research in completely automated vehicles is driven by a number of reasons, such as sustainability and safety [2]. In terms of sustainability, it is believed that integrating intelligent autonomous algorithms could improve driving efficiency and therefore reduce energy consumption, and better vehicular inter-communication could alleviate congestion and provide optimal traffic flow [3].

Pose estimation, with broad application in areas such as localization, image reconstruction, and augmented reality tracking [4], plays an important role in AD. The fusion of vision and inertial measurements has the capabilities to provide a robust and complementary pose estimation performance. The approach to this problem can be summarized into combining the vision methods and inertial sensor methods under the framework of either filtering or nonlinear optimization.

Vision methods refer to the way which estimates the position and orientation of camera from the image, where the camera could be monocular, stereo, or RGB–Depth (RGB–D) camera. The principle of vision-based method is to find the spatial relation between the 2D image points and the corresponding 3D scene points. The essential problem is to obtain the association between features in different image frames, which could be tackled with a variety of methods, such as Binary Robust Invariant Scalable Keypoints (BRISK), Scale–Invariant Feature Transform (SIFT) [5], Speeded–Up Robust Feature (SURF) [6], and Oriented FAST and Rotated BRIEF (ORB). By comparison of the same feature in two sequential frames, it is possible to find the pose change of the camera. However, due to the blurred frames under fast and unpredictable motions, the pose estimation results from the vision-based methods suffers from noticeable error sometimes. To solve this, the inertial measurement methods are introduced.

Inertial measurement units consist of sensors such as accelerometer and gyroscope. Accelerometer measures the acceleration. The velocity is then calculated from integration of the acceleration, and the position from integration of the velocity. Gyroscope measures angular velocity, whose integration gives the orientation [7]. However, inertial units suffer severely from extensive noise and accumulated drift due to integration. Fortunately it is possible to give less noisy measurements with high output rate by applying some fusion methods. Before that, the sensors are sup-

posed to be calibrated considering the physical alignment, temperature relations of the gains and offsets [8].

A well known solution is visual inertial odometry with tightly-coupled sliding window nonlinear optimization [9]. This thesis is innovated by the solution, while has different choices in some subcomponents. In terms of visual front end, the ORB method is applied to detect the features, given its fast recognition with good invariance to viewpoint in moderate computation. Also, preintegration [10] is applied to the Inertial Measurement Unit (IMU) measurements before the optimization. The cost function is formulated with combination of visual and inertial terms. In the backend, Google's Ceres solver with fast speed, high solution quality and well-supported Application Programming Interfaces (API) [11] is utilized to solve the non-linear least squares problem.

1.1 Aim

The motivation of this thesis derives from this Chalmers Formula Student Driverless (CFSD) project. The objective of the project is to design and build a reliable and safe autonomous car based on an already existing electric race car to participate the driverless formula student competitions. In order to achieve the objective, there are many crucial tasks need to be done, such as perception, control and automation. One of the key tasks is to make the vehicle able to perceive its external environment as well as the relation between itself and the surroundings, which is the common challenge known as Simultaneous Localization and Mapping (SLAM).

The thesis is going to focus on the pose estimation solution by combining data from camera and IMU as known as Visual Inertial Odometry (VIO). Afterwards the implementation is planned to be extended to OpenDLV, a software framework developed by the Chalmers Revere laboratory. The algorithm is expected to guarantee that the pose estimation task is done as quick as possible, specifically, at least 20 Frame per Second (FPS), and provides reliable vehicle egomotion to the path planning module of the CFSD project.

1.2 Limitations

Since the thesis work is based on CFSD, the functionality of the developed approaches would be designed for student formula competitions, i.e. it only considers constraints of the competition and might lose some generality.

1.3 Research questions

Question 1. How well does the proposed method estimate an egomotion state?

This investigation issue is raised to examine if the specific implementation which combines feature-based frontend, IMU preintegration theory and sliding window nonlinear optimization could successfully fuse multiple sensors, and how decent could the results be. The implementation is tested via EuRoC datasets. The evaluation criteria are based on the translational and rotational errors of the estimated trajectory relative to ground truth. In addition, the evaluation will also take the results from some well known algorithms as reference to judge the goodness of the proposed implementation.

Question 2. How fast is the implemented algorithm?

When used in practical applications, it is important to trade off the computational performance to accuracy of results due to hardware limits and other constraints. To answer this question, processing time per frame of the implemented algorithm is of concern. A set of experiments with different configurations and parameters will be evaluated.

2
Theory

This chapter first briefly introduces the microservices architecture that is used in the CFSD project, and then discusses the visual frontend from sensor selection to mathematical model, IMU sensor model and preintegration theory, as well as backend optimization methods.

Mathematical symbols such as scalar value, vectors and matrices of different dimensions follow the notation summarized in Table 2.1

Table 2.1: Typographic conventions

Typeface	Description	Meaning
X, Y, Z	normal italic	scalar values
$\mathbf{a}, \mathbf{v}, \mathbf{b}$	bold	vectors, e.g., acceleration, velocity, bias
${}_b\mathbf{v}, {}_w\mathbf{v}$	subscript before vector	the velocity is expressed in body frame or world frame
${}_b\boldsymbol{\omega}_{wb}$	subscript at both sides	the angular velocity of frame b with respect to frame w, and expressed in frame b
$\mathbf{b}^a, \mathbf{b}^g$	superscript after vector	the bias of accelerator or gyroscope
R, M	normal capital	matrices
R_{wb}	normal capital	rotation matrix from frame b to frame w

2.1 Containerized software microservices

With the growth of complexity, automation, and connectivity in vehicle realm, the vehicles are regarded as objects similar to other internet of things which are able to possess more functionalities and be updated easily from the perspective of consumers. Due to this complicated situation, a new software architecture is supposed to create in order to cope with that. According to this challenge and years of researches in autonomous vehicle area, the Chalmers Revere laboratory proposed the open source software environment called OpenDLV based on the containerized software microservices.

The microservices architecture is, when developing the application using a suite of small services, each of these services has its own process and is able to interact with lightweight mechanisms.The services are independently deployable, loosely coupled, maintained more easily, and structured in terms of business capabilities. They can be

implemented with different programming languages, software and hardware environment. For the containerized microservices in OpenDLV framework, the components are running as the Docker containers, and communicating using the C++ library libcluon, which is a single-file, header-only real time middleware [12].

2.2 Visual Frontend

This section elaborates on the visual frontend pipeline from sensor to model. Discussion includes sensor selection, pinhole camera model, epipolar geometry, the ORB approach and bundle adjustment theory.

2.2.1 Visual sensors

There are various visual sensors that can capture three dimensional images in the market, such as monocular cameras, RGB–D cameras and stereo cameras. A Mono camera is common and cheap, but it's complex to get depth information and recover the scale in real world. The depth camera also known as the RGB–D camera. It sends and receives the infrared structured light, and the Time-of-Flight is measured to compute the distance of the object. But it also has some drawbacks such as short measured distance, narrow view angle and high noise. A stereo camera is made of two mono lenses with the certain baseline. By knowing baseline, it can estimate the dimensional position in terms of each pixel, with large similarity of human binocular vision. Fig. 2.1 shows the Zed stereo camera with a baseline of 120 mm, detectable depth ranging from 0.5–20 m, and frame rate ranging from 15–100 FPS.

Figure 2.1: ZED stereo camera

The reason of using a stereo camera in this thesis work is that stereo vision has more convenience to acquire the depth compared to monocular vision, meanwhile has larger range and cheaper price than the RGB–D cameras. For the compensation of the expensive computation for pixels, an efficient feature selection method is applied and explained in section 2.2.4.

2.2.2 Pinhole camera model and geometry

Pinhole model as the most commonly used camera model represents the phenomenon when the light passes through a small hole to the room, it will project the upside-down image of the outside world to the projection plane, as seen in Fig. 2.2, where O is the optical center.

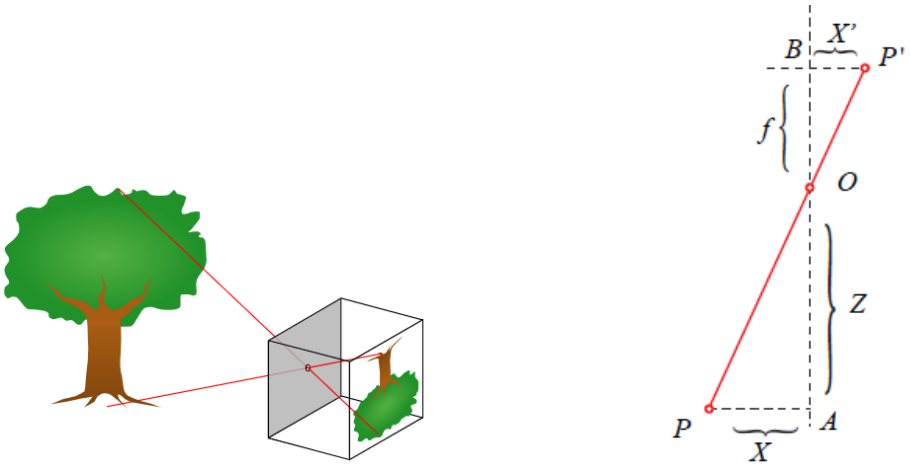


Figure 2.2: Pinhole camera phenomenon and model

The relation between the object P in the outside world and the projected point P' in the image plane is given by:

$$\frac{Z}{f} = -\frac{X}{X'} = -\frac{Y}{Y'} \tag{2.1}$$

where f is the focal length. To simplify the model, the image plane can be placed in front of the camera center:

$$\frac{Z}{f} = \frac{X}{X'} = \frac{Y}{Y'} \tag{2.2}$$

To get the spatial relationship between point P and its corresponding point P' in the image plane, the equation could be written as:

$$\begin{aligned} X' &= f\frac{X}{Z} \\ Y' &= f\frac{Y}{Z} \end{aligned} \tag{2.3}$$

Eq. 2.3 ends up with pixels in the camera system which should be sampled and discretized on the image plane. In the pixel coordinate, the origin sits at the upper left corner of the image, while the projected point sits at the center of image, therefore a translation c_x, c_y should be applied to offset. In addition, the scales f_x, f_y between the pixels and projection points should also be considered. The relation can be rewritten as:

$$\begin{aligned} u &= f_x\frac{X}{Z} + c_x \\ v &= f_y\frac{Y}{Z} + c_y \end{aligned} \tag{2.4}$$

A more compact form with the matrix and homogeneous coordinate system is:

$$Z\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \mathrm{K}\mathbf{P} \tag{2.5}$$

where K is the camera intrinsics or calibration matrix, which often is provided by the camera manufacturer, or some calibration algorithms such as the MATLAB Camera Calibrator [13].

In Eq. 2.5, $\mathbf{P}$ is in the camera coordinate. Apart from the intrinsic matrix which transforms the object from the camera coordinate perspective to the pixel coordinate perspective, there is the camera extrinsics consisting of rotation matrix R and translation vector $\mathbf{t}$ as well:

$$Z\mathbf{P}_{uv} = Z\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathrm{K}(\mathrm{R}\mathbf{P}_w + \mathbf{t}) \tag{2.6}$$

where $\mathbf{P}_{uv}$ and $\mathbf{P}_w$ are in the homogeneous pixel coordinate and world coordinate respectively. The extrinsic parameters transform the object from the world coordinate to the camera coordinate, which keep changing with the movement of the object. In this sense, the camera extrinsics essentially represent the movement or trajectory of the object, and thus is the critical target in the visual odometry.

2.2.3 Epipolar geometry and triangulation

If pairs of matched feature points are extracted from two images, the relative motion can be recovered between the two images based on epipolar geometry. Furthermore, if it is in the stereo vision, it is possible to acquire the depth information of object through triangulation based on the paired feature points.

In Fig. 2.3, relative motion between image I_1 and I_2 can be depicted by rotation R and translation $\mathbf{t}$. O_1 and O_2 are the camera centers respectively. Feature point p_2 in plane I_2 corresponds to point p_1 in plane I_1, which represents the projection of the same object in two image planes. Line O_1p_1 and O_2p_2 intersects at point P. The three points O_1, O_2, P define a plane, which is called Epipolar plane. Line O_1O_2 is the baseline, and it intersects image plane I_1, I_2 at point e_1, e_2, which are defined as Epipoles. The intersection lines l_1, l_2 are called Epipolar lines.

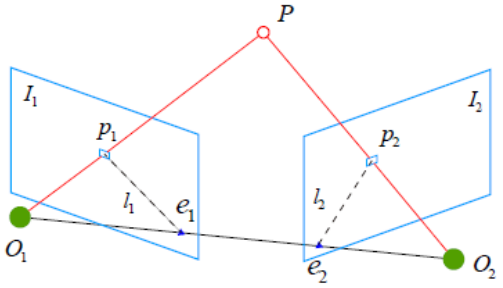


Figure 2.3: Epipolar constraint

With the knowledge of rotation R, translation $\mathbf{t}$ and feature correspondences, the depth of an object can be calculated by triangulation technique. Suppose a stereo

system with two lenses has known camera intrinsics, spatial relation between two lenses and a pair of corresponding points. Before the triangulation, it is beneficial to carry out rectification, i.e., make two images parallel. The common way to implement it is to make all epipolar lines horizontal. After rectification, it is easier to find the correspondence relations in images by searching along the single row of pixels, which makes triangulation much simpler.

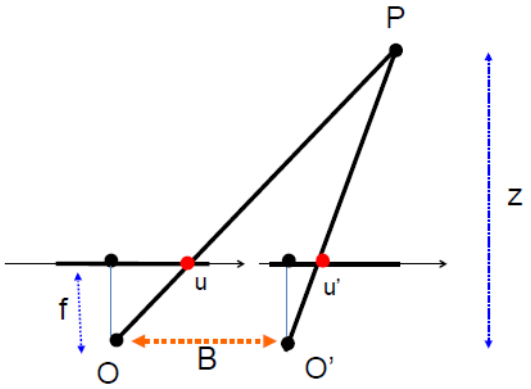


Figure 2.4: Triangulation geometry

In Fig. 2.4, the pixel displacement between u and u' is called disparity. The geometry information gives:

$$u - u' = \frac{B \cdot f}{Z} \tag{2.7}$$

where B is the baseline of the stereo camera, f is the focal length.

2.2.4 ORB features

In order to find the correspondence in the images, it is necessary to select the representative features which should remain the same after a small change in the camera pose. Afterwards, data association can be performed since the same object should keep almost the same feature descriptions when projected to different image planes. Therefore, images from different camera perspectives can be linked via these special features.

Feature points refer to pixels located at informative region in the image, such as corners and edges which are more distinguishable than other pixel blocks. However, in most applications, the natural feature points do not satisfy the needs. To this end, there are a variety of research on the artificial stable feature points, such as SIFT [5], SURF [14], and ORB [15], which have advantages in the distinctiveness and efficiency.

A feature consists of a pair of key point and descriptor. The key point means the position of the feature in the image and maybe also include information such as

size and orientation. A descriptor refers to vectors which could describe the pixel information around the key point according to certain artificially designed ways. It is regarded as the same feature point as long as the distance between two descriptors in the vector space is below some certain threshold.

ORB is a combination feature point of Features from Accelerated Segment Test (FAST) and Binary Robust Independent Elementary Features (BRIEF) feature, which can be computed efficiently. FAST is a type of corner point, which mainly detects the obvious change on local pixel gray scale. The main idea is that if the pixel differ greatly from the neighborhood pixel, either too bright or too dark, it has high likelihood to be a corner. The computation is fast because it only compares the brightness of pixels.

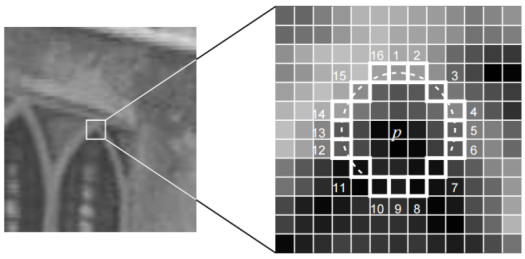


Figure 2.5: FAST feature point

The process of detection follows these rules: first, a pixel p is selected in the image with brightness assumed to be lp, and set a threshold T (for example 20% of lp). Then 16 pixels are selected on the circle with a radius of 3 centered at p. If the brightness of consecutive N points is greater than $lp + T$ or less that $lp - T$, the pixel p is considered as a feature point. These steps are performed for each pixel. To be computationally efficient, the pixels 1, 5, 9, 13 are detected. Only when three of four pixels are satisfying the rule before, the pixel may be a corner point, otherwise should be rejected. The oriented FAST adds scale and rotation to enhance the robustness.

BRIEF is a binary descriptor made of a number of 0 and 1, where 0 and 1 encode the relation between two pixels near the key point. For example, for two pixel p and q, if p is greater than p then the result is 1, otherwise 0. The ORB algorithm calculates the direction of key points and the BRIEF is acquired in use of the direction information, which gives ORB the rotation-invariant feature.

2.2.5 Bundle adjustment

In practice, there are a variety of factors which might contribute to the noise of measurements. The motion from structure problem is often regarded as the minimization problem in Eq. 2.8, where a reconstructed point $\mathbf{P}'$ that is closest to the unknown ground truth $\mathbf{P}$ is found by minimizing the reprojection error. The same

applies to localization problem in Eq. 2.8, i.e., the best approximation M′ is going to be found if the camera extrinsics M is unknown.

$$Error(\mathrm{M}, \mathbf{P}) = distance(\mathbf{p}, \mathrm{M}_1\mathbf{P}')^2 + distance(\mathbf{p}', \mathrm{M}_2\mathbf{P}')^2 \tag{2.8}$$

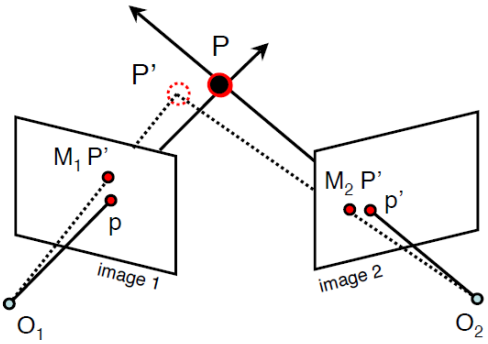


Figure 2.6: Bundle adjustment with reprojection error

Bundle adjustment is in essence a non-linear method to deal with vision related problems by constructing reprojection errors and reducing to least square forms. Afterwards, a backend typically takes over the problem and finds the solution. Details of solving will be explained in later sections.

2.3 Inertial measurement odometry

This section covers the brief introduction of inertial sensors, the sensor model and the preintegration theory.

2.3.1 Inertial measurement unit

An IMU is a device that is able to measure the acceleration and angular rate. It consists of an accelerometer and a gyroscope which is based on the Microelectromechanical Systems (MEMS) technology. Some simple mechanical principles are applied upon the MEMS sensors to get desired measurements. A spring suspended mass is used to measure the acceleration, because the mass will be displaced due to some applied acceleration. By considering the Coriolis effect in vibrating systems, the angular velocity can be measured. MEMS systems are often light, cheap and have low power consumption, but the reduced accuracy and bias stability over time lead to the drift problem in IMU [16].

2.3.2 Sensor model

The whole visual inertial system consists of different coordinate systems. As in Fig. 2.7, the world reference is an earth-fixed coordinate system which z axis is parallel to the gravity, and is denoted with W. The IMU frame is treated as body

frame that is to be tracked, denoted with B. The camera frame is denoted with C. Transformation from body frame to world frame is denoted as T_{WB}, and camera frame to body frame is T_{BC}.

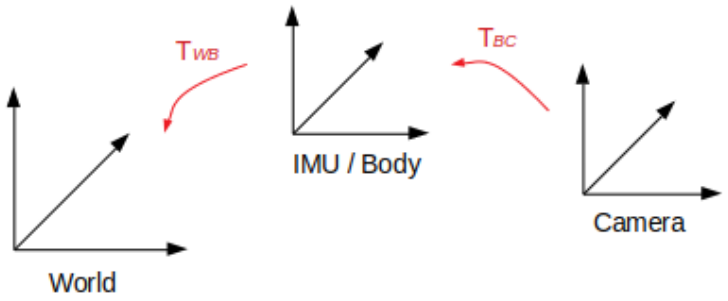


Figure 2.7: Coordinate transformation

The gyroscope measurement ${}_b\tilde{\boldsymbol{\omega}}_{wb}$ where the prefix b means the measurement is expressed in the body frame B, and wb means it's the angular velocity of frame B relative to frame W. It is modeled as:

$$ {}_b\tilde{\boldsymbol{\omega}}_{wb}(t) = {}_b\boldsymbol{\omega}_{wb}(t) + \mathbf{b}^g(t) + \boldsymbol{\eta}^g(t) \tag{2.9} $$

where ${}_b\boldsymbol{\omega}_{wb}$ is the instantaneous angular velocity of body frame with respect to world frame and expressed in body frame. $\boldsymbol{\eta}^g$ denotes white noise, whose distribution is close to Gaussian distribution. $\mathbf{b}^g$ denotes gyroscope bias, which is caused by calibration errors and temperature effects, and is slowly varying. All the variables are the funtion of time.

Similar to gyroscope, the characteristic of accelerometer can be modeled as:

$$ {}_b\tilde{\mathbf{a}}(t) = \mathrm{R}_{wb}^T(t) \cdot ({}_w\mathbf{a}(t) - {}_w\mathbf{g}) + \mathbf{b}^a(t) + \boldsymbol{\eta}^a(t) \tag{2.10} $$

where ${}_b\tilde{\mathbf{a}}$ is the measured acceleration expressed in the body frame B, ${}_w\mathbf{a}$ is the actual acceleration of body expressed in the world frame, ${}_w\mathbf{g}$ is the gravity vector in world frame, and R_{wb}^T denotes the rotation from frame B to frame W. $\mathbf{b}^a$ is a slowly varying bias of the accelerometer, and $\boldsymbol{\eta}^a$ is white noise which is assumed to have a Gaussian distribution. The accelerometer measures the specific force on the sensor. But it still need to subtract the gravity to recover the acceleration, since an accelerometer without any acceleration still have a force of 1 g due to the earth's gravitational field. The accelerometer also suffers from the sensor bias and white noise. In next chapter, the initialization of IMU will be explained in more details.

2.3.3 Preintegration

IMU Preintegration theory was first proposed in [17], and further developed in [10] with the mainfold structure of rotation group, and posterior IMU bias correction. The theory introduced in the section is mainly based on the latest preintegration formulation in [10].

In order to calculate motion from IMU measurement, a kinematic model is introduced:

$$\begin{aligned} \dot{\mathrm{R}}_{wb}(t) &= \mathrm{R}_{wb}(t) \cdot [{}_b\boldsymbol{\omega}_{wb}(t)]_\times \\ {}_w\dot{\mathbf{v}}(t) &= {}_w\mathbf{a}(t) \\ {}_w\dot{\mathbf{p}}(t) &= {}_w\mathbf{v}(t) \end{aligned} \tag{2.11}$$

where the operator $[\cdot]_\times$ turns a vector into a 3×3 skew symmetric matrix:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}_\times = \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix} \tag{2.12}$$

The evolution of the body rotation R_{wb}, velocity ${}_w\mathbf{v}$ and position ${}_w\mathbf{p}$ can be calculated as follows:

$$\begin{aligned} \mathrm{R}_{wb}(t+\Delta t) &= \mathrm{R}_{wb}(t) \cdot \exp([\int_t^{t+\Delta t} {}_b\boldsymbol{\omega}_{wb}(\tau)d\tau]_\times) \\ {}_w\mathbf{v}(t+\Delta t) &= {}_w\mathbf{v}(t) + \int_t^{t+\Delta t} {}_w\mathbf{a}(\tau)d\tau \\ {}_w\mathbf{p}(t+\Delta t) &= {}_w\mathbf{p}(t) + \int_t^{t+\Delta t} {}_w\mathbf{v}(\tau)d\tau + \iint_t^{t+\Delta t} {}_w\mathbf{a}(\tau)d\tau^2 \end{aligned} \tag{2.13}$$

where $\exp(\mathrm{M})$ represents the matrix exponential of the matrix $\mathrm{M}_{n\times n}$, producing $n \times n$ matrix by the power series:

$$\exp(\mathrm{M}) = \sum_{k=0}^{\infty} \frac{1}{k!}\mathrm{M}^k \tag{2.14}$$

By making assumption that ${}_w\mathbf{a}$ and ${}_b\boldsymbol{\omega}_{wb}$ remain constant during the time interval $[t, t+\Delta t]$, and taking into account Eq. 2.9 and 2.10, as well as denoting the exponential map as:

$$\mathrm{Exp}(\mathbf{v}) = \exp([\mathbf{v}]_\times) \tag{2.15}$$

the measurements then can be related to the states:

$$\begin{aligned} \mathrm{R}_{wb}(t+\Delta t) &= \mathrm{R}_{wb}(t) \cdot \mathrm{Exp}\left(({}_b\tilde{\boldsymbol{\omega}}_{wb}(t) - \mathbf{b}^g(t) - \boldsymbol{\eta}^{gd}(t))\Delta t\right) \\ {}_w\mathbf{v}(t+\Delta t) &= {}_w\mathbf{v}(t) + {}_w\mathbf{g}\Delta t + \mathrm{R}_{wb}(t) \cdot \left({}_b\tilde{\mathbf{a}}(t) - \mathbf{b}^a(t) - \boldsymbol{\eta}^{ad}(t)\right)\Delta t \\ {}_w\mathbf{p}(t+\Delta t) &= {}_w\mathbf{p}(t) + {}_w\mathbf{v}(t)\Delta t + \frac{1}{2}{}_w\mathbf{g}\Delta t^2 + \frac{1}{2}\mathrm{R}_{wb}(t) \cdot \left({}_b\tilde{\mathbf{a}}(t) - \mathbf{b}^a(t) - \boldsymbol{\eta}^{ad}(t)\right)\Delta t^2 \end{aligned} \tag{2.16}$$

where $\boldsymbol{\eta}^{gd}$ and $\boldsymbol{\eta}^{ad}$ is discrete-time noise.

To derive an easily implemented form, Eq. 2.16 can be further discretized by iteratively integrating a number of IMU measurements. Assume the iteration starts from time i to time $j-1$, and the states at time j are computed as follows:

$$\begin{aligned} \mathrm{R}_{wb,j} &= \mathrm{R}_{wb,i} \cdot \prod_{k=i}^{j-1} \mathrm{Exp}\left(({}_b\tilde{\boldsymbol{\omega}}_{wb} - \mathbf{b}^g - \boldsymbol{\eta}^{gd})\Delta t\right) \\ {}_w\mathbf{v}_j &= {}_w\mathbf{v}_i + {}_w\mathbf{g}\Delta t_{ij} + \sum_{k=i}^{j-1} \mathrm{R}_{wb,k} \cdot ({}_b\tilde{\mathbf{a}}_k - \mathbf{b}_k^a - \boldsymbol{\eta}_k^{ad})\Delta t \\ {}_w\mathbf{p}_j &= {}_w\mathbf{p}_i + \sum_{k=i}^{j-1} \left[{}_w\mathbf{v}_k\Delta t + \frac{1}{2}{}_w\mathbf{g}\Delta t^2 + \frac{1}{2}\mathrm{R}_{wb,k} \cdot ({}_b\tilde{\mathbf{a}}_k - \mathbf{b}_k^a - \boldsymbol{\eta}_k^{ad})\Delta t^2\right] \end{aligned} \tag{2.17}$$

The drawback of Eq. 2.17 is that whenever states at time i have a change, to update all future states at time k would need recalculating the production and summation in Eq. 2.17. The preintegration technique is brought up to keep relative motion increments that are independent of the states at starting time, and thus avoiding a lot of recalculation whenever there is update at starting time. The discrete integration forms are defined as:

$$\begin{aligned}
\Delta \mathrm{R}_{ij} :&= \mathrm{R}_{wb,i}^{T} \cdot \mathrm{R}_{wb,j} \\
&= \prod_{k=i}^{j-1} \mathrm{Exp}\left(({}_b\tilde{\boldsymbol{\omega}}_{wb,k} - \mathbf{b}_k^g - \boldsymbol{\eta}_k^{gd}) \Delta t \right) \\
\Delta \mathbf{v}_{ij} :&= \mathrm{R}_{wb,i}^{T} \cdot ({}_w\mathbf{v}_j - {}_w\mathbf{v}_i - {}_w\mathbf{g}\Delta t_{ij}) \\
&= \sum_{k=i}^{j-1} \mathrm{R}_{wb,k} \cdot ({}_b\tilde{\mathbf{a}}_k - \mathbf{b}_k^a - \boldsymbol{\eta}_k^{ad}) \Delta t \\
\Delta \mathbf{p}_{ij} :&= \mathrm{R}_{wb,i}^{T} \cdot ({}_w\mathbf{p}_j - {}_w\mathbf{p}_i - {}_w\mathbf{v}_i \Delta t_{ij} - \frac{1}{2} {}_w\mathbf{g}\Delta t_{ij}^2) \\
&= \sum_{k=i}^{j-1} \left[{}_w\mathbf{v}_k \Delta t + \frac{1}{2} \mathrm{R}_{wb,k} \cdot ({}_b\tilde{\mathbf{a}}_k - \mathbf{b}_k^a - \boldsymbol{\eta}_k^{ad}) \Delta t^2 \right]
\end{aligned} \tag{2.18}$$

As mentioned before, the sensor noise is modeled as Gaussian noise. Separate out all noise related terms in Eq. 2.17:

$$\begin{aligned}
\Delta \mathrm{R}_{ij} :&= \Delta \tilde{\mathrm{R}}_{ij} \cdot \mathrm{Exp}(-\delta \boldsymbol{\phi}_{ij}) \\
\Delta \mathbf{v}_{ij} :&= \Delta \tilde{\mathbf{v}}_{ij} - \delta \mathbf{v}_{ij} \\
\Delta \mathbf{p}_{ij} :&= \Delta \tilde{\mathbf{p}}_{ij} - \delta \mathbf{p}_{ij}
\end{aligned} \tag{2.19}$$

where the preintegrated measurement terms and their iterative forms are [10]:

$$\begin{aligned}
\Delta \tilde{\mathrm{R}}_{ij} &= \prod_{k=i}^{j-1} \mathrm{Exp}\left(({}_b\tilde{\boldsymbol{\omega}}_{wb,k} - \mathbf{b}_k^g) \Delta t \right) \\
&= \Delta \tilde{\mathrm{R}}_{i,j-1} \, \mathrm{Exp}\left(({}_b\tilde{\boldsymbol{\omega}}_{wb,j-1} - \mathbf{b}_i^g) \Delta t \right) \\
\Delta \tilde{\mathbf{v}}_{ij} &= \sum_{k=i}^{j-1} \Delta \tilde{\mathrm{R}}_{wb,k} \cdot ({}_b\tilde{\mathbf{a}}_k - \mathbf{b}_k^a) \Delta t \\
&= \Delta \tilde{\mathbf{v}}_{i,j-1} + \Delta \tilde{\mathrm{R}}_{i,j-1} \cdot ({}_b\tilde{\mathbf{a}}_{j-1} - \mathbf{b}_i^a) \Delta t \\
\Delta \tilde{\mathbf{p}}_{ij} &= \sum_{k=i}^{j-1} \left[\tilde{\mathbf{v}}_{ik} \Delta t + \frac{1}{2} \tilde{\mathrm{R}}_{ik} \cdot ({}_b\tilde{\mathbf{a}}_k - \mathbf{b}_k^a) \Delta t^2 \right] \\
&= \Delta \tilde{\mathbf{p}}_{i,j-1} + \tilde{\mathbf{v}}_{i,j-1} \Delta t + \frac{1}{2} \tilde{\mathrm{R}}_{i,j-1} \cdot ({}_b\tilde{\mathbf{a}}_{j-1} - \mathbf{b}_i^a) \Delta t^2
\end{aligned} \tag{2.20}$$

The noise terms and their iterative forms are [10]:

$$\begin{aligned}
\delta\boldsymbol{\phi}_{ij} &= -\operatorname{Log}\left(\prod_{k=i}^{j-1}\operatorname{Exp}(-\Delta\tilde{\mathrm{R}}_{k+1,j}^{T}\mathrm{J}_r^k\boldsymbol{\eta}_k^{gd}\Delta t)\right) \approx \sum_{k=i}^{j-1}\Delta\tilde{\mathrm{R}}_{k+1,j}^{T}\mathrm{J}_r^k\boldsymbol{\eta}_k^{gd}\Delta t \\
&= \Delta\tilde{\mathrm{R}}_{j-1,j}^{T}\delta\boldsymbol{\phi}_{i,j-1} + \mathrm{J}_r^{j-1}\boldsymbol{\eta}_{j-1}^{gd}\Delta t \\
\delta\mathbf{v}_{ij} &\simeq \sum_{k=i}^{j-1}\left[-\Delta\tilde{\mathrm{R}}_{ik}[{}_b\tilde{\mathbf{a}}_k - \mathbf{b}_k^a]_\times\delta\boldsymbol{\phi}_{ik}\Delta t + \Delta\tilde{\mathrm{R}}_{ik}\boldsymbol{\eta}_k^{ad}\Delta t\right] \\
&= \delta\mathbf{v}_{i,j-1} - \tilde{\mathrm{R}}_{i,j-1}[{}_b\tilde{\mathbf{a}}_{j-1} - \mathbf{b}_i^a]_\times\delta\boldsymbol{\phi}_{i,j-1}\Delta t + \Delta\tilde{\mathrm{R}}_{i,j-1}\boldsymbol{\eta}_{j-1}^{ad}\Delta t \\
\delta\mathbf{p}_{ij} &\simeq \sum_{k=i}^{j-1}\left[\delta\mathbf{v}_{ik}\Delta t - \frac{1}{2}\Delta\tilde{\mathrm{R}}_{ik}[{}_b\tilde{\mathbf{a}}_k - \mathbf{b}_k^a]_\times\delta\boldsymbol{\phi}_{ik}\Delta t^2 + \frac{1}{2}\Delta\tilde{\mathrm{R}}_{ik}\boldsymbol{\eta}_k^{ad}\Delta t^2\right] \\
&= \delta\mathbf{p}_{i,j-1} + \delta\mathbf{v}_{i,j-1}\Delta t - \frac{1}{2}\Delta\tilde{\mathrm{R}}_{i,j-1}[{}_b\tilde{\mathbf{a}}_{j-1} - \mathbf{b}_i^a]_\times\delta\boldsymbol{\phi}_{i,j-1}\Delta t^2 + \frac{1}{2}\Delta\tilde{\mathrm{R}}_{i,j-1}\boldsymbol{\eta}_{j-1}^{ad}\Delta t^2
\end{aligned} \tag{2.21}$$

The logarithmic map is defined as:

$$\operatorname{Log}(\mathrm{M}) = [\log(\mathrm{M})]^\vee \tag{2.22}$$

where M is matrix, $log(\cdot)$ is matrix logarithm, and $[\cdot]^\vee$ is the inverse operator of $[\cdot]_\times$, transfering a skew symmetric matrix to a vector:

$$\begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix}^\vee = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{2.23}$$

J_r is the right Jacobians for $SO(3)$ Lie group [18]. At time $j-1$, it is:

$$\mathrm{J}_r^{j-1} = \mathrm{J}_r^{j-1}\left(({}_b\tilde{\boldsymbol{\omega}}_{wb,j-1} - \mathbf{b}_i^g)\Delta t\right) \tag{2.24}$$

i.e., the jacobian is computed when the expending point is at

$$\mathbf{x} = ({}_b\tilde{\boldsymbol{\omega}}_{wb,j-1} - \mathbf{b}_i^g)\Delta t \tag{2.25}$$

So the noise propagation could be written as matrix form:

$$\begin{aligned}
\begin{bmatrix}\delta\boldsymbol{\phi}_{ij}\\ \delta\mathbf{v}_{ij}\\ \delta\mathbf{p}_{ij}\end{bmatrix} &= \begin{bmatrix} \Delta\tilde{\mathrm{R}}_{j-1,j}^{T} & 0 & 0 \\ -\tilde{\mathrm{R}}_{i,j-1}[{}_b\tilde{\mathbf{a}}_{j-1} - \mathbf{b}_i^a]_\times\Delta t & \mathrm{I}_{3\times3} & 0 \\ -\frac{1}{2}\Delta\tilde{\mathrm{R}}_{i,j-1}[{}_b\tilde{\mathbf{a}}_{j-1} - \mathbf{b}_i^a]_\times\Delta t^2 & \mathrm{I}_{3\times3}\Delta t & \mathrm{I}_{3\times3}\end{bmatrix}\begin{bmatrix}\delta\boldsymbol{\phi}_{i,j-1}\\ \delta\mathbf{v}_{i,j-1}\\ \delta\mathbf{p}_{i,j-1}\end{bmatrix} \\
&\quad + \begin{bmatrix}\mathrm{J}_r^{j-1}\Delta t & 0 \\ 0 & \Delta\tilde{\mathrm{R}}_{i,j-1}\Delta t \\ 0 & \frac{1}{2}\Delta\tilde{\mathrm{R}}_{i,j-1}\Delta t^2\end{bmatrix}\begin{bmatrix}\boldsymbol{\eta}_{j-1}^{gd}\\ \boldsymbol{\eta}_{j-1}^{ad}\end{bmatrix} \\
&\doteq \mathrm{F}_{j-1}\begin{bmatrix}\delta\boldsymbol{\phi}_{i,j-1}\\ \delta\mathbf{v}_{i,j-1}\\ \delta\mathbf{p}_{i,j-1}\end{bmatrix} + \mathrm{G}_{j-1}\begin{bmatrix}\boldsymbol{\eta}_{j-1}^{gd}\\ \boldsymbol{\eta}_{j-1}^{ad}\end{bmatrix}
\end{aligned} \tag{2.26}$$

Therefore the noise covariance is derived as:

$$\Sigma_{ij} = \mathrm{F}_{j-1}\Sigma_{i,j-1}\mathrm{F}_{j-1}^{T} + \mathrm{G}_{j-1}\Sigma_{\eta d}\mathrm{G}_{j-1}^{T} \tag{2.27}$$

where,

$$\Sigma_{ij} = \mathrm{Cov}(\begin{bmatrix}\delta\boldsymbol{\phi}_{ij}\\ \delta\mathbf{v}_{ij}\\ \delta\mathbf{p}_{ij}\end{bmatrix}), \quad \Sigma_{i,j-1} = \mathrm{Cov}(\begin{bmatrix}\delta\boldsymbol{\phi}_{i,j-1}\\ \delta\mathbf{v}_{i,j-1}\\ \delta\mathbf{p}_{i,j-1}\end{bmatrix}), \quad \Sigma_{\eta d} = \mathrm{Cov}(\begin{bmatrix}\boldsymbol{\eta}^{gd}_{j-1}\\ \boldsymbol{\eta}^{ad}_{j-1}\end{bmatrix}) \tag{2.28}$$

and at the beginning $\Sigma_{00} = \mathbf{0}_{9\times 9}$; the $\Sigma_{\eta d}$ depends on the sampling time and continuous-time noise [19]:

$$\mathrm{Cov}(\boldsymbol{\eta}^{gd}) = \frac{1}{\Delta t}\mathrm{Cov}(\boldsymbol{\eta}^{g}), \quad \mathrm{Cov}(\boldsymbol{\eta}^{ad}) = \frac{1}{\Delta t}\mathrm{Cov}(\boldsymbol{\eta}^{a}), \quad \Delta t = \frac{1}{f} \tag{2.29}$$

where f is the sampling frepency, and the continuous-time covariance of accelerometer and gyroscope is provided by the IMU manufacturers.

When calculating the preintegrated terms, it is assumed that the biases hold constant in a short period, i.e., $\mathbf{b}_k = \mathbf{b}_i$ for all $k \in [i, j-1]$. However, the biases are actually affected by random walk noise. The noise model of biases are defined as:

$$\begin{aligned} \dot{\mathbf{b}}^g(t) = \boldsymbol{\eta}^{bg}, \quad \dot{\mathbf{b}}^a(t) = \boldsymbol{\eta}^{ba} \quad (continuous form)\\ \mathbf{b}^g_j = \mathbf{b}^g_i + \boldsymbol{\eta}^{bgd}, \quad \mathbf{b}^a_j = \mathbf{b}^a_i + \boldsymbol{\eta}^{bad} \quad (discrete form)\\ \Sigma_{bgd} \doteq \Delta t_{ij}\mathrm{Cov}(\boldsymbol{\eta}^{bg}), \quad \Sigma_{bad} \doteq \Delta t_{ij}\mathrm{Cov}(\boldsymbol{\eta}^{ba}) \end{aligned} \tag{2.30}$$

If take into account the small change of biases afterwards, instead of recalculating everything again, one efficient approach is to approximate using first order expansion:

$$\begin{aligned} \Delta\tilde{\mathrm{R}}_{ij}(\mathbf{b}^g) &= \Delta\tilde{\mathrm{R}}_{ij}(\mathbf{b}^g_i)\cdot \mathrm{Exp}(\frac{\partial\Delta\tilde{\mathrm{R}}_{ij}}{\partial\mathbf{b}^g}\delta\mathbf{b}^g_i)\\ \Delta\tilde{\mathbf{v}}_{ij}(\mathbf{b}^g,\mathbf{b}^a) &= \Delta\tilde{\mathbf{v}}_{ij}(\mathbf{b}^g_i,\mathbf{b}^a_i) + \frac{\partial\Delta\tilde{\mathbf{v}}_{ij}}{\partial\mathbf{b}^g}\delta\mathbf{b}^g_i + \frac{\partial\Delta\tilde{\mathbf{v}}_{ij}}{\partial\mathbf{b}^a}\delta\mathbf{b}^a_i\\ \Delta\tilde{\mathbf{p}}_{ij}(\mathbf{b}^g,\mathbf{b}^a) &= \Delta\tilde{\mathbf{p}}_{ij}(\mathbf{b}^g_i,\mathbf{b}^a_i) + \frac{\partial\Delta\tilde{\mathbf{p}}_{ij}}{\partial\mathbf{b}^g}\delta\mathbf{b}^g_i + \frac{\partial\Delta\tilde{\mathbf{p}}_{ij}}{\partial\mathbf{b}^a}\delta\mathbf{b}^a_i \end{aligned} \tag{2.31}$$

With all the information needed, the preintegration residuals are thus defined:

$$\begin{aligned} \mathbf{r}_{\Delta\mathrm{R}_{ij}} &\doteq \mathrm{Log}\left[\left(\Delta\tilde{\mathrm{R}}_{ij}(\mathbf{b}^g_i)\cdot\mathrm{Exp}(\frac{\partial\Delta\tilde{\mathrm{R}}_{ij}}{\partial\mathbf{b}^g}\delta\mathbf{b}^g_i)\right)^T \mathrm{R}^T_{wb,i}\mathrm{R}_{wb,j}\right]\\ \mathbf{r}_{\Delta\mathbf{v}_{ij}} &\doteq \mathrm{R}^T_{wb,i}\cdot({}_w\mathbf{v}_j - {}_w\mathbf{v}_i - {}_w\mathbf{g}\Delta t_{ij})\\ &\quad - \left[\Delta\tilde{\mathbf{v}}_{ij}(\mathbf{b}^g_i,\mathbf{b}^a_i) + \frac{\partial\Delta\tilde{\mathbf{v}}_{ij}}{\partial\mathbf{b}^g}\delta\mathbf{b}^g_i + \frac{\partial\Delta\tilde{\mathbf{v}}_{ij}}{\partial\mathbf{b}^a}\delta\mathbf{b}^a_i\right]\\ \mathbf{r}_{\Delta\mathbf{p}_{ij}} &\doteq \mathrm{R}^T_{wb,i}\cdot\left({}_w\mathbf{p}_j - {}_w\mathbf{p}_i - {}_w\mathbf{v}_i\Delta t_{ij} - \frac{1}{2}{}_w\mathbf{g}\Delta t^2_{ij}\right)\\ &\quad - \left[\Delta\tilde{\mathbf{p}}_{ij}(\mathbf{b}^g_i,\mathbf{b}^a_i) + \frac{\partial\Delta\tilde{\mathbf{p}}_{ij}}{\partial\mathbf{b}^g}\delta\mathbf{b}^g_i + \frac{\partial\Delta\tilde{\mathbf{p}}_{ij}}{\partial\mathbf{b}^a}\delta\mathbf{b}^a_i\right]\\ \mathbf{r}^g_{\mathbf{b}_{ij}} &\doteq \mathbf{b}^g_j - \mathbf{b}^g_i\\ \mathbf{r}^a_{\mathbf{b}_{ij}} &\doteq \mathbf{b}^a_j - \mathbf{b}^a_i \end{aligned} \tag{2.32}$$

So far the IMU preintegration theory developed by Forster et al [10] is introduced, readers could also refer to the origin paper for more details about the exact forms of jacobians used in optimization. In next chapter, the implementation details based on the theories of this chapter will be explained.

2.4 Backend solver

In this section, the least square problems is discussed, as well as the general numerical methods to solve this kind of problem.

2.4.1 The least square problem

Given a so called cost function or objective function $\mathbf{F} : \mathbb{R}^n \mapsto \mathbb{R}$, find an argument that gives the minimum value of $\mathbf{F}$, i.e., $\mathbf{x}^* = \underset{x}{\operatorname{argmin}}\,(\mathbf{F}(\mathbf{x}))$. The cost function $\mathbf{F}$ is formalized as [20]:

$$\mathbf{F}(\mathbf{x}) = \frac{1}{2}\sum_{i=1}^{m}(\mathbf{f}_i(\mathbf{x}))^2 \tag{2.33}$$

where $\mathbf{f}_i : \mathbb{R}^n \mapsto \mathbb{R}$, $i = 1, ..., m$ are residual functions, for example, the difference between measurements and estimations, and $m \geq n$. The global minimizer of this problem is too hard to find generally, so a local minimizer is solved instead. The cost function $\mathbf{F}$ is assumed to be differentiable, so its gradient and Hessian are valid. For $\mathbf{x}^*$ to be a local minimizer, it has to be a stationary point, i.e., $\mathbf{F}'(\mathbf{x}^*) = 0$. Besides, the Hessian matrix should be positive definite, i.e., $\mathbf{v}^T\mathbf{F}''(\mathbf{x}^*)\mathbf{v} > 0$ for all non-zero $\mathbf{v} \in \mathbb{R}^n$ [20].

2.4.2 Newton's Method

Based on the assumptions mentioned above, the cost function could be rewritten using Taylor expansion:

$$\mathbf{F}(\mathbf{x} + \Delta\mathbf{x}) \simeq \mathbf{F}(\mathbf{x}) + \mathbf{F}'(\mathbf{x})\Delta\mathbf{x} + \frac{1}{2}\Delta\mathbf{x}^T\mathbf{F}''(\mathbf{x})\Delta\mathbf{x} \tag{2.34}$$

Since the wanted x^* is stationary point, take derivative of Eq. 2.34 with respect to Δx, and it stratifies:

$$\mathbf{F}'(\mathbf{x} + \Delta\mathbf{x}) \simeq \mathbf{F}'(\mathbf{x}) + \mathbf{F}''(\mathbf{x})\Delta\mathbf{x} \equiv 0 \tag{2.35}$$

The Newton's method is to find the solution in an iterative way, $\mathbf{x} = \mathbf{x} + \Delta\mathbf{x}$, and in each iteration solve the linear equation:

$$\mathbf{F}''(\mathbf{x})\Delta\mathbf{x} = -\mathbf{F}'(\mathbf{x}) \tag{2.36}$$

2.4.3 The Gauss-Newton method

In many cases the residual functions $\mathbf{f}(\mathbf{x})$ are non-linear, so it's difficult to get the Hessian $\mathbf{F}''(\mathbf{x})$ of the cost function. The Guass-Newton method is derived to solve

the non-linear least square problems efficiently [21]. In essence, it is based on the linearization of the residual functions with Taylor expansion:

$$\mathbf{f}(\mathbf{x}+\Delta\mathbf{x}) \simeq \mathbf{f}(\mathbf{x}) + \mathbf{J}(\mathbf{x})\Delta\mathbf{x} \tag{2.37}$$

where $\mathbf{J}(\mathbf{x})$ is the Jacobian matrix, fist order derivative of $\mathbf{f}(\mathbf{x})$. And then use the linearized residuals to approximate the cost function:

$$\begin{aligned}\mathbf{F}(\mathbf{x}+\Delta\mathbf{x}) &= \frac{1}{2}\left\|\mathbf{f}(\mathbf{x}+\Delta\mathbf{x})\right\|^2 \\ &= \frac{1}{2}\left(\mathbf{f}(\mathbf{x})+\mathbf{J}(\mathbf{x})\Delta\mathbf{x}\right)^T\left(\mathbf{f}(\mathbf{x})+\mathbf{J}(\mathbf{x})\Delta\mathbf{x}\right) \\ &= \frac{1}{2}\mathbf{f}(\mathbf{x})^T\mathbf{f}(\mathbf{x}) + \mathbf{f}(\mathbf{x})^T\mathbf{J}(\mathbf{x})\Delta\mathbf{x} + \frac{1}{2}\Delta\mathbf{x}^T\mathbf{J}(\mathbf{x})^T\mathbf{J}(\mathbf{x})\Delta\mathbf{x} \\ &= \mathbf{F}(\mathbf{x}) + \mathbf{f}^T\mathbf{J}\Delta\mathbf{x} + \frac{1}{2}\Delta\mathbf{x}^T\mathbf{J}^T\mathbf{J}\Delta\mathbf{x}\end{aligned} \tag{2.38}$$

Take derivative with respect to $\Delta\mathbf{x}$:

$$\mathbf{J}^T\mathbf{f} + \mathbf{J}^T\mathbf{J}\Delta\mathbf{x} = 0 \tag{2.39}$$

The notation can be simplified to the form:

$$\mathbf{H}\Delta\mathbf{x} = \mathbf{b} \tag{2.40}$$

This equation is known as the Guass-Newton equation or Normal equation, where

$$\begin{aligned}\mathbf{H} &= \mathbf{J}(\mathbf{x})^T\mathbf{J}(\mathbf{x}) \\ \mathbf{b} &= -\mathbf{J}(\mathbf{x})^T\mathbf{f}(\mathbf{x})\end{aligned} \tag{2.41}$$

The small increment $\Delta\mathbf{x}$ is to be solved. The final solution of $\mathbf{x}$ is found by iterative update with $\Delta\mathbf{x}$.

2.4.4 The Levenberg-Marquardt Method

There are a number of problems that might occur such that the Gauss-Netwom algorithm doesn't always converge reliably. A more robust variation of Guass-newton is a damped one, the Levenberg–Marquardt method. It adds a damping parameter $\mu \geq 0$ to the normal equation [22]:

$$(\mathbf{J}^T\mathbf{J} + \mu\mathbf{I})\Delta\mathbf{x} = -\mathbf{J}^T\mathbf{f} \tag{2.42}$$

It is naturally to think of adding a trust region to the $\Delta\mathbf{x}$, and the linearization is supposed to be valid within this region since the Taylor expansion only have a good approximation near the expansion point. The Levenberg–Marquardt method is in essence a trust region approach.

The damping parameter μ has several effects. From the perspective of solving linear equation, $\mathbf{J}^T\mathbf{J}$ might be singular and thus the problem is unsolvable. Adding $\mu\mathbf{I}$ to it makes the new Hessian, $\mathbf{H} = \mathbf{J}^T\mathbf{J} + \mu\mathbf{I}$, to be full rank. In terms of numerical stability, the damping parameter penalizes a step that is too large, and thus make the iteration more smooth and consistent. When μ is a big number, the LM algorithm behaves like gradient descent method; while μ is small, it behaves more like the Gauss-Newton method. Therefore, it is a combination of two methods, and the progress is controlled by the damping parameter.

3
Methods

3.1 Datasets

The EuRoC Micro Aerial Vehicle (MAV) datasets [23] are used to assess the performance of the VIO system. The datasets consist of the IMU measurements, synchronized stereo images and ground truth.

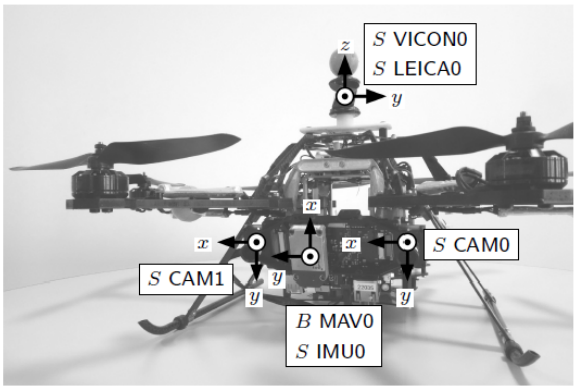


Figure 3.1: The MAV used for data collection

In Fig. 3.1, the stereo camera system is located in a front-down position, with two global shutter monocular cameras CAM0 and CAM1 both at the rate of 20 Hz. The IMU0 is output at 200 Hz. The IMU and camera are hardware-synchronized. There are two classes of datasets available, one is collected in a machine hall with 3D position ground truth from a LEICA Nova MS50 laser scanner, and the LEICA0 is in millimeter accuracy. The other is collected in a room with 6D pose ground truth from a Vicon motion capture system, VICON0 and 3D point cloud of enviroment from LEICA0. Table 3.1 shows the information about the sensors and ground truth. The datasets vary from good lighting and slow motion to dark scene and fast motion, seen in Table 3.2.

Table 3.1: Information on sensors and ground truth

Sensor	Type	Rate	Characteristics
Cameras	MT9V034	2x20 Hz	WVGA global shutter
IMU	ADIS16448	200 Hz	MEMS, intr. calibrated
Position	Leica MS50	20 Hz	Accuracy $\approx$ 1 mm
Pose	Vicon	100 Hz	6D

Table 3.2: The characteristics of the datasets

Name	Length/Duration	Avg.Vel/ Angular Vel	Note
MH_01_easy	80.6 m 182 s	0.44 m/s 0.22 rad/s	good texture bright scene
MH_02_easy	73.5 m 150 s	0.49 m/s 0.21 rad/s	good texture bright scene
MH_03_medium	130.9 m 132 s	0.99 m/s 0.29 rad/s	fast motion bright scene
MH_04_difficult	91.7 m 99 s	0.93 m/s 0.24 rad/s	fast motion dark scene
MH_05_difficult	97.6 m 111 s	0.88 m/s 0.21 rad/s	fast motion dark scene
V1_01_easy	58.6 m 144 s	0.41 m/s 0.28 rad/s	slow motion bright scene
V1_02_medium	75.9 m 83.5 s	0.91 m/s 0.56 rad/s	fast motion bright scene
V1_03_difficult	79.0 m 105 s	0.75 m/s 0.62 rad/s	fast motion motion blur
V2_01_easy	36.5 m 112 s	0.33 m/s 0.28 rad/s	slow motion bright scene
V2_02_medium	83.2 m 115 s	0.72 m/s 0.59 rad/s	fast motion bright scene
V2_03_difficult	86.1 m 115 s	0.75 m/s 0.66 rad/s	fast motion motion blur

3.2 Software Design

This section elaborates on the visual inertial odometry design. First an overview of the system structure is presented. Following discussion focuses on detailed explanations about each submodule in the system, including system initialization, feature tracking, preintegration, optimization and visulization.

3.2.1 Overview

This main work of the thesis is to design and implement a visual inertial odometry from scratch. The structure of visual inertial odometry, shown in Fig. 3.2, is composed of initialization, feature tracking, IMU preintegration and optimization. The initialization procedure aligns the body frame to gravity direction, as well as estimates initial bias of gyroscope and accelerometer. Feature tracking mainly deals with extracting the ORB features and performing feature matching. IMU measurements are preintergated between two successive image frames, and the noise of the sensor measurements is propagated. The optimizing procedure handles local optimization and global optimization. After local optimization, the frame will be

decided to be a keyframe or not based on motion change. Afterwards, the feature points in the keyframe will be stored in the map. When the whole process is done, the global optimization will be executed and then update the map according to the results.

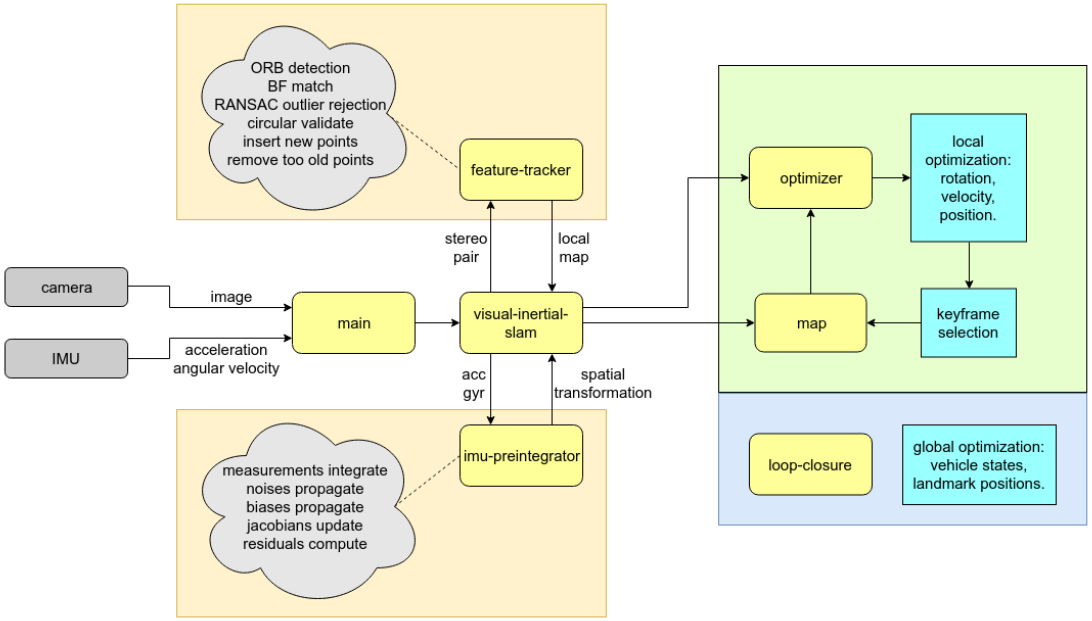


Figure 3.2: System structure of the visual inertial odometry implementation

3.2.2 Initialization

To correctly use the measurements from IMU, it's important to find out its initial pose relative to the world frame. In this section, vision-inertial fusion method is proposed to get initial estimates of gravity alignment, sensor bias and velocity. The idea is at first to perform Structure–from–Motion (SfM) to compute relative pose transformation with visual information, and then use the result to estimate IMU alignment and bias.

SfM is a technique for estimating 3D structures from 2D images that have some correspondence. The motion of camera can be estimated with three different methods [24]:

- 2D–2D: features in current image frame and previous image frame are matched, and essential matrix can be computed from corresponding pixel coordinates throu-gh 5–point or 8–point algorithm [25].
- 3D–3D: 2D feature points are triangulated to 3D points in both current and previous frames, and camera pose can be computed with Iterative Closet Point (ICP) algorithm with these two clouds of points.
- 3D–2D: feature points in previous frame are triangulated to 3D points, and matched to 2D pixels in the current frame. This problem is also known as Perspective–n–Points (PnP), and many approaches are applicable for it [26].

After collecting pose information of a set of frames from SfM, the initialization procedure starts and is divided into four steps:

Gyroscope bias estimation
Gyroscope bias can be estimation from know rotations of consecutive frames. Pose R_i and R_j is from SfM, the preintegration between frame i and j is calculated using a constant bias $\hat{b}_i^g$. But in actual the bias changes by a small amount δb^g, so it should be $\hat{b}_i^g + \delta b^g$, and the preintegrated measurement will be updated using a first-order expansion. The bias change δb^g is estimated by minimizing the rotation residual from Eq. 2.32:

$$\mathbf{r}_{\Delta \mathrm{R}_{ij}} \doteq \mathrm{Log}\left[\left(\Delta\tilde{\mathrm{R}}_{ij}\left(\mathbf{b}_i^g\right)\cdot \mathrm{Exp}\left(\frac{\partial \Delta\tilde{\mathrm{R}}_{ij}}{\partial \mathbf{b}^g}\delta\mathbf{b}^g\right)\right)^{\top}\mathrm{R}_i^{\mathrm{T}}\mathrm{R}_j\right] \tag{3.1}$$

$$\underset{\delta\mathbf{b}^g}{\mathrm{argmin}}\sum\left\|\mathbf{r}_{\Delta \mathrm{R}_{ij}}\right\|^2_{\Sigma_{\mathrm{ij}}} \tag{3.2}$$

After updating initial gyroscope bias estimation, all preintegrated measurements will be repropagated.

Gravity direction and velocity in the body frame
In this step, unit gravity vector expressed in body frame will be found, and the velocity will be estimated by minimizing the residuals as follows:

$$\mathbf{r}_{\Delta \mathbf{v}_{ij}} \doteq \mathrm{R}_i^{\top}\left(\left(\mathbf{v}_j + \delta\mathbf{v}_j\right) - \left(\mathbf{v}_i + \delta\mathbf{v}_i\right) - \delta\mathbf{g}\Delta t_{ij}\right) - \Delta\tilde{\mathbf{v}}_{ij}\left(\tilde{\mathbf{b}}_i^g, \tilde{\mathbf{b}}_i^a\right) \tag{3.3}$$

$$\mathbf{r}_{\Delta \mathbf{p}_{ij}} \doteq \mathrm{R}_i^{\top}\left(\mathbf{p}_j - \mathbf{p}_i - \left(\mathbf{v}_i + \delta\mathbf{v}_i\right)\Delta t_{ij} - \frac{1}{2}\delta\mathbf{g}\Delta t_{ij}^2\right) - \Delta\tilde{\mathbf{p}}_{ij}\left(\tilde{\mathbf{b}}_i^g, \tilde{\mathbf{b}}_i^a\right) \tag{3.4}$$

$$\underset{\delta\mathbf{v}_i,\delta\mathbf{v}_j,\delta\mathbf{g}}{\mathrm{argmin}}\sum\left\|\mathbf{r}_{\Delta \mathbf{v}_{ij}}\right\|^2_{\Sigma_{\mathrm{ij}}} + \left\|\mathbf{r}_{\Delta \mathbf{p}_{ij}}\right\|^2_{\Sigma_{\mathrm{ij}}} \tag{3.5}$$

Gravity alignment and refinement
This step is to find a rotation R_{wb} to make the unit gravity vector obtained from previous step align with gravity direction in the world frame. The information of the local gravity magnitude, e.g., $\mathrm{G} = 9.81734\ m/s^2$, is also taken into account.

Accelerometer bias estimation
In the last step, accelerometer bias is estimated using updated results from previous steps. The procedure is similar:

$$\mathbf{r}_{\Delta \mathbf{v}_{ij}} \doteq \mathrm{R}_i^{\top}\left(\mathbf{v}_j - \mathbf{v}_i - \delta\mathbf{g}\Delta t_{ij}\right) - \left[\Delta\tilde{\mathbf{v}}_{ij}\left(\tilde{\mathbf{b}}_i^g, \tilde{\mathbf{b}}_i^a\right) + \frac{\partial\Delta\tilde{\mathbf{v}}_{ij}}{\partial\mathbf{b}^a}\delta\mathbf{b}^a\right] \tag{3.6}$$

$$\mathbf{r}_{\Delta \mathbf{p}_{ij}} \doteq \mathrm{R}_i^{\top}\left(\mathbf{p}_j - \mathbf{p}_i - \mathbf{v}_i\Delta t_{ij} - \frac{1}{2}\delta\mathbf{g}\Delta t_{ij}^2\right) - \left[\Delta\tilde{\mathbf{p}}_{ij}\left(\tilde{\mathbf{b}}_i^g, \tilde{\mathbf{b}}_i^a\right) + \frac{\partial\Delta\tilde{\mathbf{p}}_{ij}}{\partial\mathbf{b}^a}\delta\mathbf{b}^a\right] \tag{3.7}$$

$$\underset{\delta\mathbf{b}^a}{\mathrm{argmin}}\sum\left\|\mathbf{r}_{\Delta \mathbf{v}_{ij}}\right\|^2_{\Sigma_{\mathrm{ij}}} + \left\|\mathbf{r}_{\Delta \mathbf{p}_{ij}}\right\|^2_{\Sigma_{\mathrm{ij}}} \tag{3.8}$$

Since the visual inertial system is highly nonlinear, it is important to have reasonable initial values to setup the system to get stable optimization results later on.

3.2.3 Feature tracking

The feature tracker aims to keep track of the image sequences and find data association among them. Before tracking, the input image has to be rectified and undistorted, so that it would be easier to perform triangulation and reprojection. As discussed in the previous chapter, ORB feature is chosen considering both accuracy and computational efficiency. Fig. 3.3 shows the detected keypoints and matching result using ORB detector provided by OpenCV library.

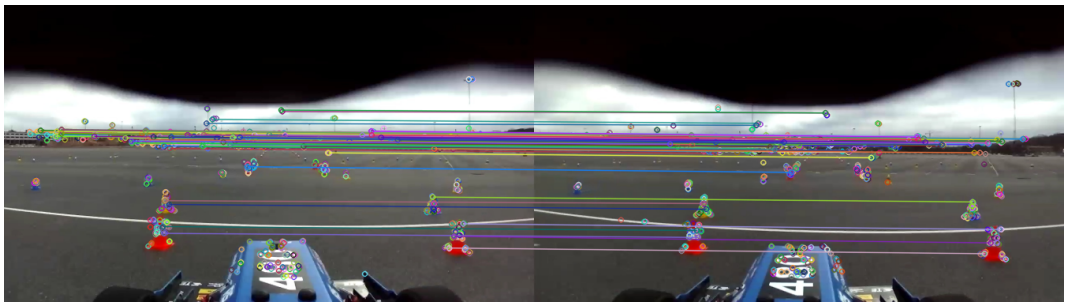


Figure 3.3: Keypoints found by ORB detector from OpenCV and filtered matches

However, in some cases many of the keypoints found by OpenCV ORB would concentrate in some parts of an image, but not evenly distribute over the entire image. It is crucial to have spatially well distributed features over the image [27] for visual tracking, which will provide more information than the stacked one, because points that are piled up in the same area add little valuable information for triangulation and will contribute less to the tracking. And stacked points are more likely to cause mismatching, which will disrupt the whole system.

One method for solving this ill distribution problem is to split the image using a grid, and detect a reasonable amount of points in each cell. The ORB extractor developed by the author of ORB–SLAM2 introduces this kind of mechanism based on grids to have the keypoints detected evenly in the whole image with a user-defined number of features. The result of the latter shows much better performance than the one provided by OpenCV, especially when there exists shadow region in the image. An example is presented in Fig. 3.4 and Fig. 3.5.

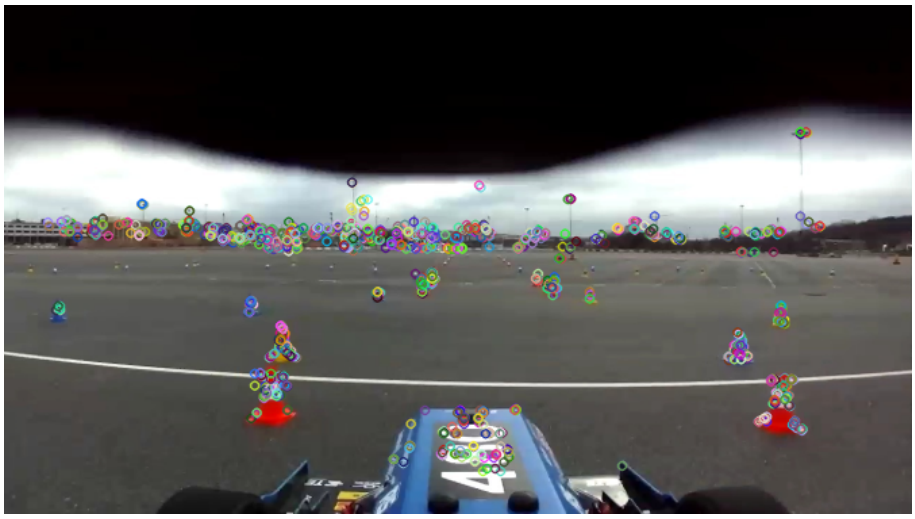


Figure 3.4: Keypoints found by ORB detector from OpenCV

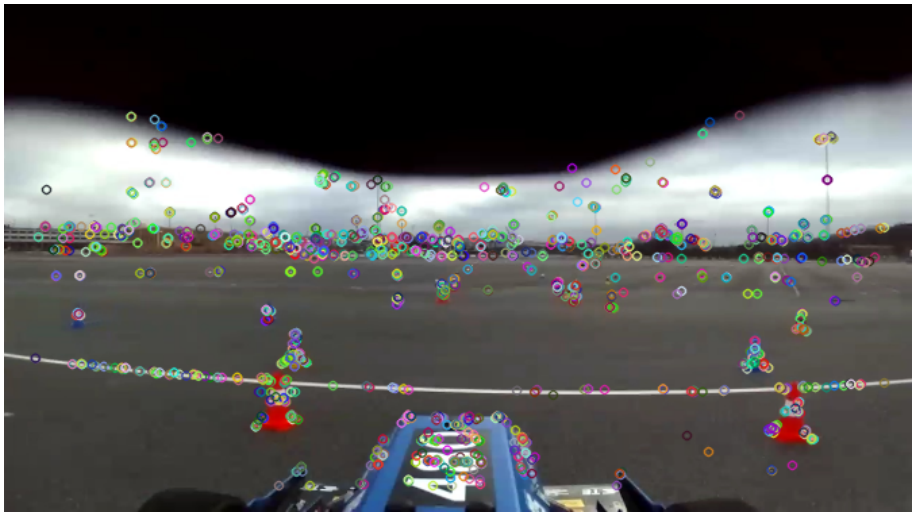


Figure 3.5: Keypoints found by ORB detector from ORB–SLAM2

After finding keypoints, a brute force matcher in OpenCV is used to match keypoints in two frames. It is quite efficient to compare Hamming distance of 32–bit binary descriptors even using brute force approach [15]. To improve the matching quality, some extra steps are performed to filter the outliers. One is to set a minimum matching distance, i.e., any matching with distance higher than the threshold will be removed. For stereo pair matching, the y coordinate of pixel can also be a criteria. Since images have been rectified, if a feature in the left image matches to another feature in the right image, their y coordinates should be the same. Random Sample Consensus (RANSAC) is also a good choice for eliminating outliers. It randomly samples observed data and uses voting scheme to find the optimal inliers. OpenCV provides a function to find fundamental matrix with RANSAC scheme, which could be applied for the purpose of filtering matches.

Features will be added to a feature pool, where the ID of the frame that observes this feature, pixel coordinate, descriptors and the age of this feature are saved. Newly coming in images will have to search matches in the feature pool. If a new feature matches to an old feature, the old feature will be kept, because it is assumed that the earlier measurements contain less error or drift. Whereas, too old features will be removed from the pool since they are less likely to be matched and removing them will avoid the size of pool becoming unreasonably large. The property, *age*, is designed for deciding features that are too old. Currently it uses a simple scheme: the age of features that are matched will increment by one, otherwise by two; if the age exceeds some threshold, the feature will be removed. Additionally, a method called circular matching [28] is applied. As shown in Fig. 3.6, detected features in current feature will have a stereo matching firstly, and then match with left and right images of history frames in respective. If the matching result shows a closed circle presented in Fig. 3.6, the corresponding feature will be considered as stable and kept for further analysis, otherwise it will be filtered out.

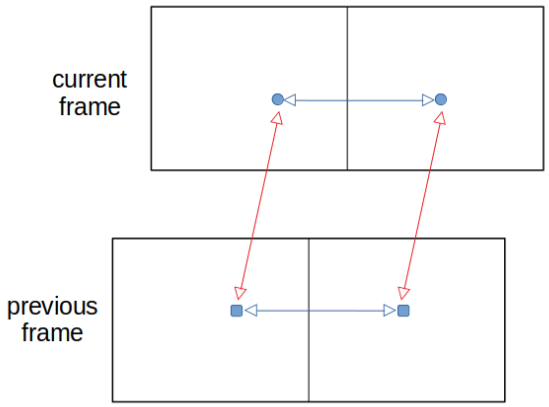


Figure 3.6: Circular matching, features in the stereo pair in current and previous frame should match to each other

After all these steps, the outcome would be a set of high-quality features with few outliers. And features in the current frame can be divided into two categories: matched to old features and not matched. For the former, information from old features will be kept as discussed before. For the latter, 3D point positions are computed through triangulation. These 3D points that are expressed in current body frame will be transformed to world frame based on current pose T_{wb} and saved to map for bundle adjustment later.

3.2.4 IMU preintegration

The aim of IMU preintegrator is to compute the preintegrated measurements in Eq. 2.20, noise covariance, and jacobians of first-order expansion of the residual terms. Since IMU runs at higher sampling rate than camera, typically 200 Hz, a queue is created in the implementation to store IMU measurements between two consecutive camera frames. Synchronization between IMU and camera is achieved

by simply comparing measurement timestamps. Here it is assumed that timestamps from these two sensors are perfectly matched at the same time point to simplify the problem, even though they would be slightly different in real case.

After collecting enough measurements, relative motion $\Delta \mathrm{R}_{ij}$, $\Delta \mathbf{v}_{ij}$, $\Delta \mathbf{p}_{ij}$, and noise covariance Σ_{ij}, and partial derivative of delta information with respect to bias $\frac{\partial \Delta \bar{\mathrm{R}}_{ij}}{\partial \mathbf{b}^g}$, $\frac{\partial \Delta \bar{\mathrm{R}}_{ij}}{\partial \mathbf{b}^a}$, $\frac{\partial \Delta \bar{\mathbf{v}}_{ij}}{\partial \mathbf{b}^g}$, $\frac{\partial \Delta \bar{\mathbf{v}}_{ij}}{\partial \mathbf{b}^a}$, $\frac{\partial \Delta \bar{\mathbf{p}}_{ij}}{\partial \mathbf{b}^g}$, $\frac{\partial \Delta \bar{\mathbf{p}}_{ij}}{\partial \mathbf{b}^a}$ are iteratively computed with constant bias $\mathbf{b}_i^g$ and $\mathbf{b}_i^a$ from frame i to frame j. The information will be saved to map and serve as motion constraints during the nonlinear optimization.

In the optimization problem, good initial values would help it converge faster and lead to a stable solution, while bad initial values might cause divergence and numerical instability. The states of frame i and frame j are to be optimized, if the former is known then the latter should be initialized. In vision-only system, a constant velocity model is typically used to give initial states estimate of next frame. When coupling IMU measurements, states of frame j can be estimated with the delta information, which could give better estimation than constant velocity assumption.

3.2.5 Optimization

The optimizer aims to minimize residuals from IMU motion constraints and errors of reprojecting 3D points to 2D pixels for the tracked features using least-square methods.

The techniques for solving the general problem of SLAM contain filtering and smoothing. The standard algorithm for filtering using Gaussian probability distribution is the Extended Kalman Filter (EKF) which is the nonlinear version of the Kalman filter that linearizes around an estimate of the current mean and covariance [29]. In this approach, all past poses are marginalized out, and features that are likely to be observed in the future are retained, so the estimation is an inference process to the latest state of the system. The inter-connections between features are stored in the form of a mean vector and covariance matrix [30].

The smoothing, also known as optimization methods, estimate multiple states other than just the latest state. This approach generally keeps a small subset of all the states, which includes keyframes selected by heuristic methods. Since the number of keyframes and size of map grows over time, it would be difficult to estimate all states while still guarantee real-time operation. The sliding window optimization methods only estimate the most recent states that fall within a fixed-size window, while marginalizing out older states and absorb the corresponding information in a Gaussian prior [31][32]. For nonlinear problem, optimization approaches generally outperforms filtering, since they relinearize past measurements during optimizing iteration [33].

In the implementation, the information of keyframes, IMU motion constraints and feature correspondence is stored in a map. Cost functions are created according to

Eq. 2.32 and Eq. 2.8. In the literature of SLAM and graph optimization, popular C++ numerical packages include g2o [34] and Ceres [11]. In this work, Ceres solver is picked due to the high solution quality, extensively optimized codes, and specifically, detailed documentations. It provides automatic derivatives, but analytical jacobians are also applied for computational efficiency. Moreover, the following Huber loss function is applied to reduce the influence of outliers:

$$\rho(s) = \begin{cases} s & s \leq 1 \\ 2\sqrt{s} - 1 & s > 1 \end{cases} \tag{3.9}$$

For real-time purpose, motion-only Bundle Adjustment (BA) is performed, as shown in Fig. 3.7. In the motion-only BA, the 3D positions of landmarks and old states are assumed to be fixed, only pose, velocity and sensor bias in the sliding window are estimated. In a factor graph, a constraint is also called a factor. Each frame can observe many landmarks, and one landmarks might be seen by more than one frames. To increase the consistency, only those that are observed by at least two frames will be constructed as reprojection factors. Two consecutive frames are connected by IMU factors. All frames in the sliding window is keyframes except the last one, which will be decided to be a keyframe or not in next step.

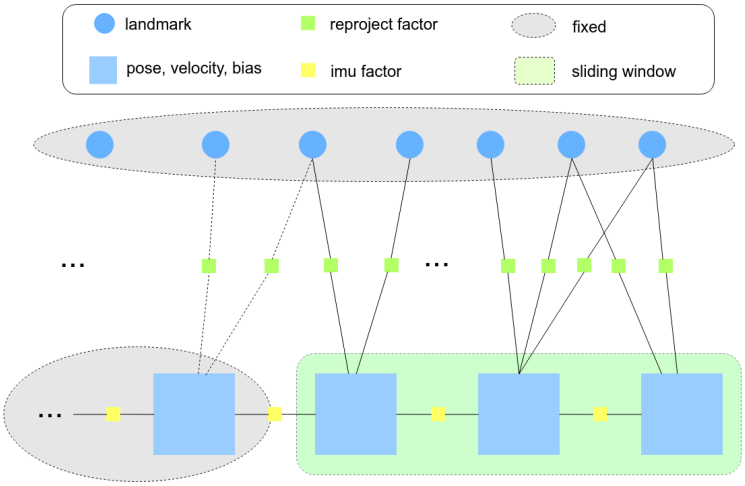


Figure 3.7: Factor graph of motion-only bundle adjustment

After solving the least-squares problem, states of frames in the window will be updated. For the last frame to be a keyframe, it should satisfy one of the conditions:

- translation from the second last frame exceeds some threshold
- rotation from the second last frame exceeds some threshold
- preintegration time is longer than some threshold, since IMU measurements are easy to drift over long time

If the last frame is a keyframe, it will be saved to map and the window slides one step forward, otherwise it will be replaced by the next coming frame.

Efficient though, motion-only BA still has long-term drift because in each optimization it only finds the local optima. One way to resolve this is to perform a global

optimization at final. As seen in Fig. 3.8, global optimization also estimates landmark positions, and it takes all past frames as input. However it would need much longer time to find solution, so it can be performed in an independent thread or offline.

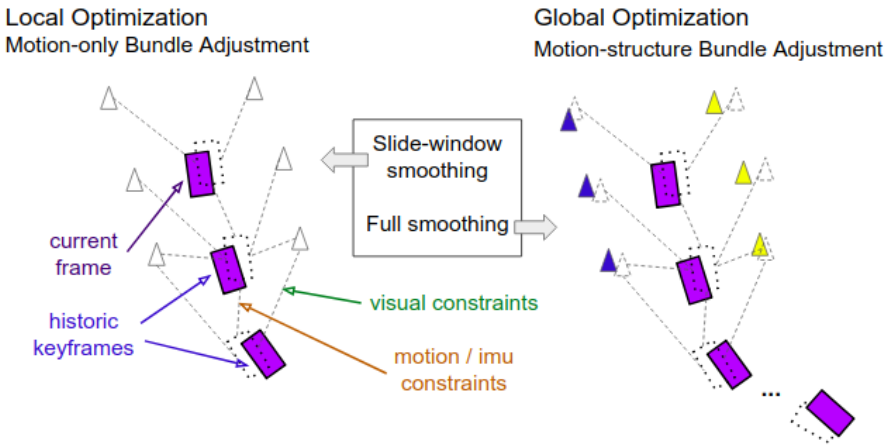


Figure 3.8: Local optimization and global optimization

3.2.6 Visualization

A visualization program is developed based on Pangolin to present the trajectory and environment. The Pangolin is selected since it is a lightweight portable rapid development library for managing OpenGL display, interaction and abstracting video input. In Fig. 3.9, the green line shows the path of the body, and red pyramid presents to which direction the camera faces. The blue dots presents the observed features which are projected to 3D space using triangulation.

Besides the trajectory viewer, image is displayed with detected feature positions in black square and reprojected ones in white circle, which gives an intuitive feeling of the reprojection errors. The frame per second, number of keypoints and feature points are also showed in Fig. 3.10.

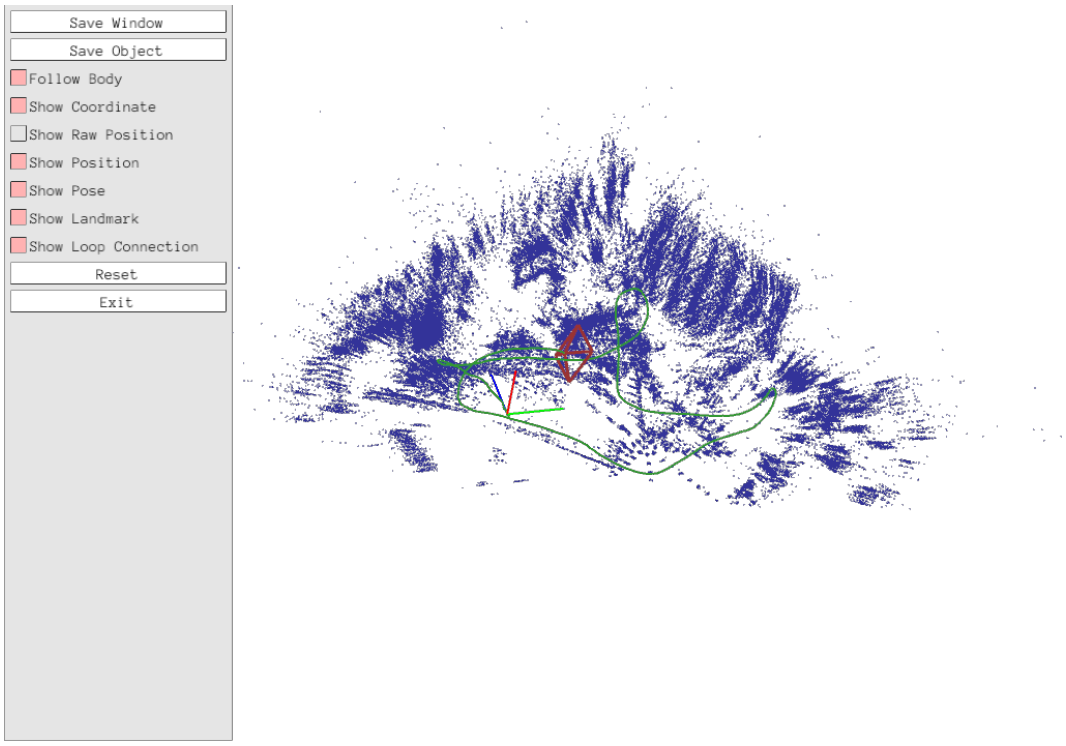


Figure 3.9: Visualizer using Pangolin

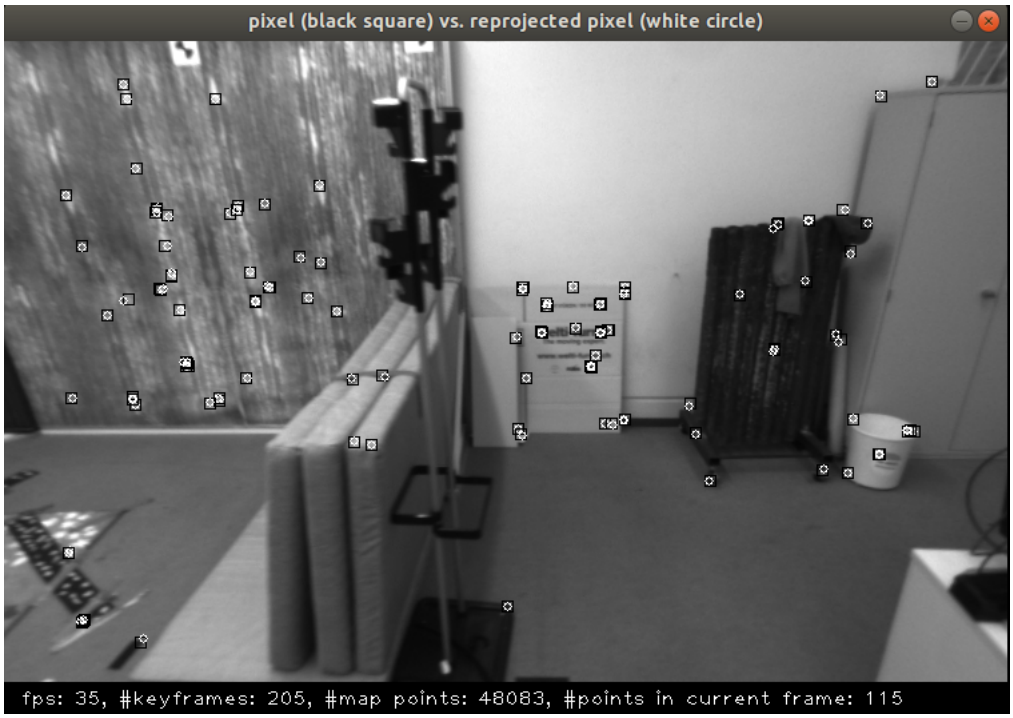


Figure 3.10: Display pixels, reprojected pixels, FPS, number of keyframes and points

3.3 Evaluation

In the experiment, EuRoC MAV datasets are used. Three representative datasets, varying from easy to difficult in terms of speed and lightning, are selected to perform detailed analysis:

- `V1_01_easy` recorded in a room has slow motion and bright scene
- `V1_02_medium` recorded in a room has fast motion and bright scene
- `MH_05_difficult` recorded in a machine hall has fast motion, dark scene

To evaluate the algorithm's outputs, the EVO package which provides executable tools for handling, evaluating and comparing the trajectory output of odometry and SLAM algorithms [35], is applied. Several commonly used formats are supported, such as TUM trajectory, KITTI pose, and EuRoC MAV groundtruth. And it has algorithmic options for association, alignment and scale adjustment. These features make it a suitable tool package for the purpose of evaluation. Next chapter will cover the running results and data analysis.

4
Results

This chapter elaborates on the testing results with various settings in terms of motion speed and brightness. All tests run in Ubuntu 18.04, RAM@32 GB and Intel Xeon CPU@2.00 GHz. The qualitative and quantitative comparisons between the estimation and ground truth will be presented.

4.1 Accuracy

This section mainly focuses on running results of the proposed method with respect to accuracy.

Slow motion and bright scene
Since the proposed method is key-frame-based, the testing results only contains poses of key frames, and thus is not as informative as the ground truth. The number of poses, length of path and duration time are shown in Table 4.1.

Table 4.1: Trajectory information of `V1_01_easy`

	Ground truth	Estimation
Number of poses	28712	868
Path length [m]	58.592	56.320
Duration [s]	143.555	143.150

In Fig. 4.1, the trajectories of both the ground truth and testing result of `V1_01_easy` are sketched out in the three dimensional space for a qualitative comparison.
The estimated trajectory shows similar shape with the ground truth, while it has drifts in some parts. To make the difference clearer, the trajectory is represented in x, y and z direction in respective, as depicted in Fig. 4.2. It can be seen that the trajectory is quite close to the ground truth, despite that in the first few seconds there are some drift due to the system has not been well initialized. The drifts approximately range from 0 to 0.5 m.

To dig more information out of the data, the absolute trajectory error is used. It is implemented in the `evo_ape` tool, where corresponding poses are directly compared between the estimation and the reference given a pose relation, and then statistics such as max, min, mean and median are calculated [35].

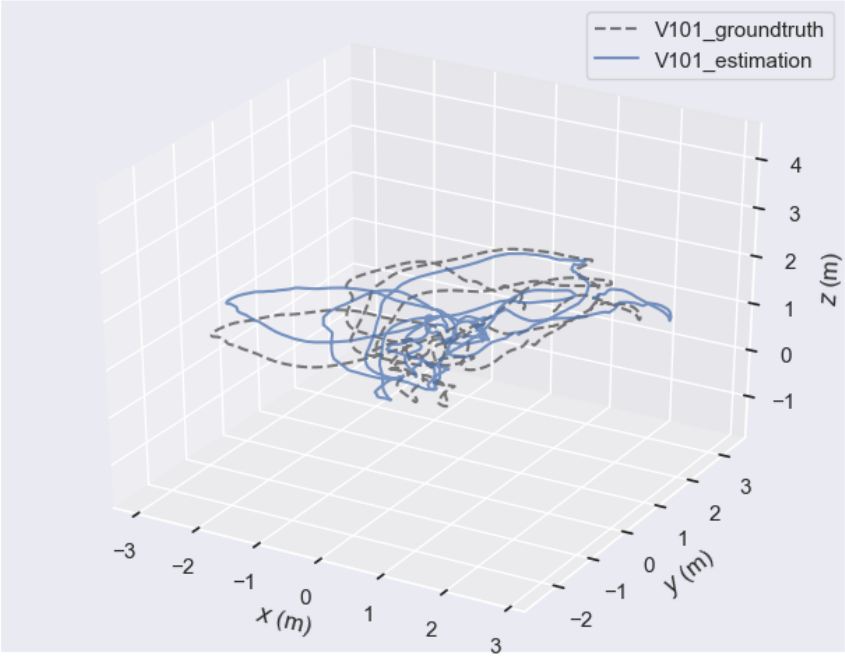


Figure 4.1: The ground truth and estimated trajectory of `V1_01_easy`

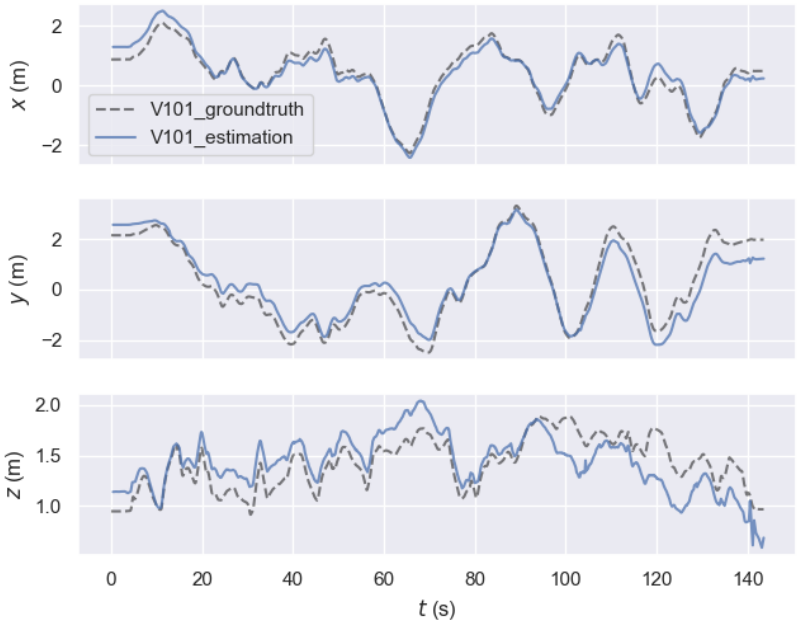


Figure 4.2: x, y, z translation of `V1_01_easy`

Fig. 4.3 shows a color map of the translation error. A more precise quantitative error analysis is present in Fig 4.4. The average error is about 0.55 m, with standard deviation around 0.22 m. At about $t = 85$ s, it has lowest error 0.12 m, while in the worst case the error reaches 1.05 m. Overall, the estimation achieves small translation error in some places, but most of the time the trajectory is drifted to

some extent.

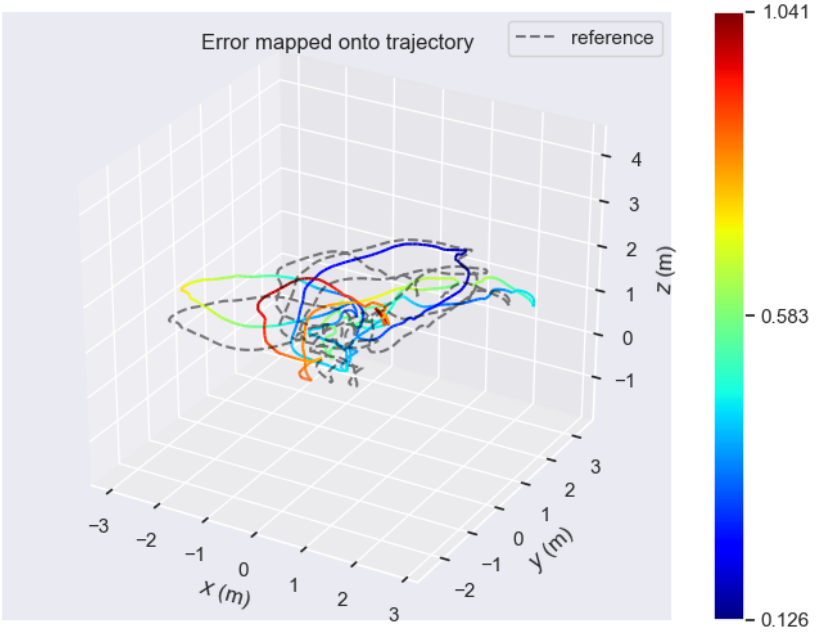


Figure 4.3: Translation error mapped onto trajectory of `V1_01_easy`

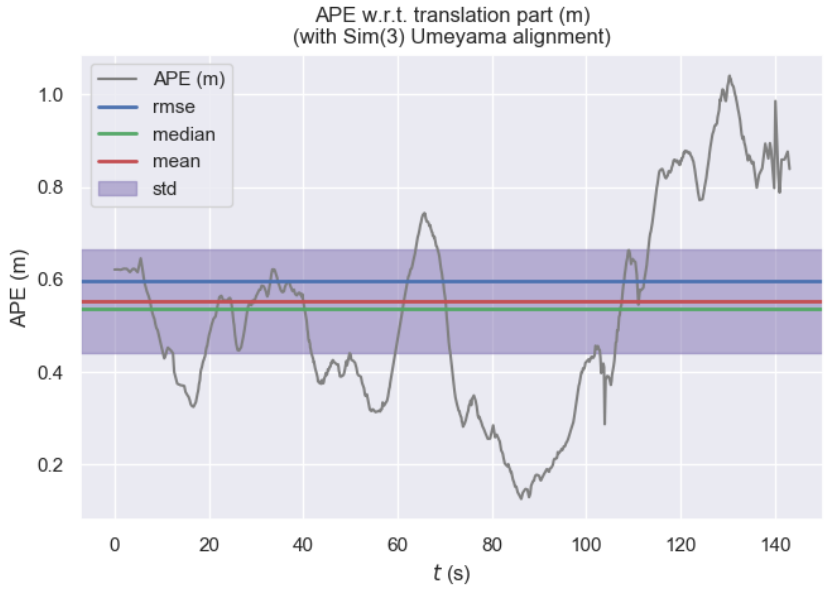


Figure 4.4: Absolute pose error with respect to translation part of `V1_01_easy`

When it comes to rotation evaluation, the same method is applied. Rotation is presented using roll, pitch and yaw angle. From the comparison in Fig. 4.5, the yaw angle estimation outperforms the other two. The estimated roll angle sometimes

has the right value but wrong direction, for example, the estimation is 180 ° while the ground truth is −180 °. However, to some extent rotating 180 ° or −180 ° has the same effect. The roll angle flipping is possibly due to the numerical oscillation. The pitch angle has error less than 10 °, and oscillates around the ground truth.

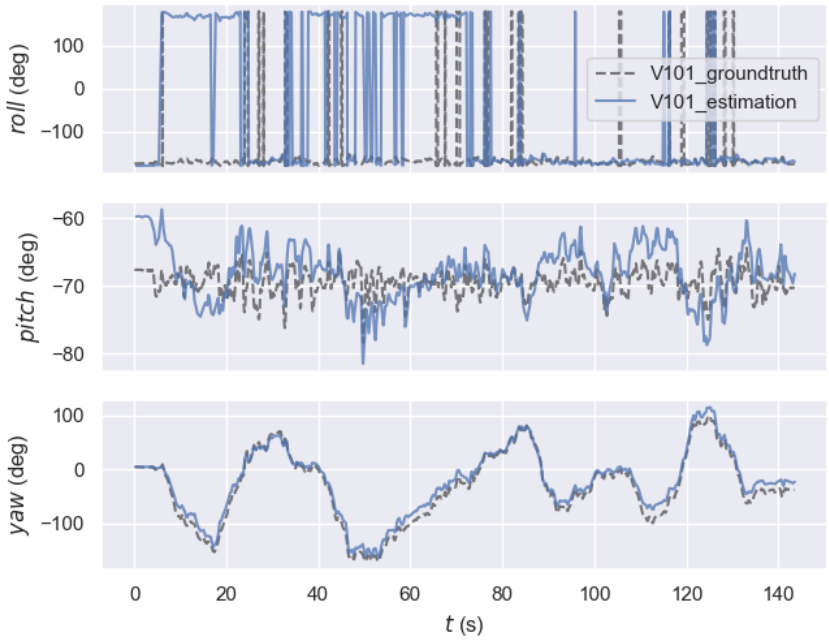


Figure 4.5: Roll, pitch and yaw angle of `V1_01_easy`

In Fig. 4.6 and Fig. 4.7, the quantitative result is presented. The average error is 9.11 °. It has lowest error 3.17 ° at around $t = 82$ s, and goes worse after $t = 100$ s.

Fast motion and bright scene

The number of poses, length of path and duration time are shown in Table 4.2. Compared to the trajectory in Table 4.1, this one has longer path but shorter duration, meaning that the drone moves faster.

Table 4.2: Trajectory information of `V1_02_medium`

	Ground truth	Estimation
Number of poses	16702	757
Path length [m]	75.891	70.649
Duration [s]	83.505	82.800

In Fig. 4.8, the ground truth and the estimated trajectories of `V1_02_medium` are sketched out. Similar to the case above, the trend of the estimated trajectory and ground truth look alike, though drifts in some parts are obvious.

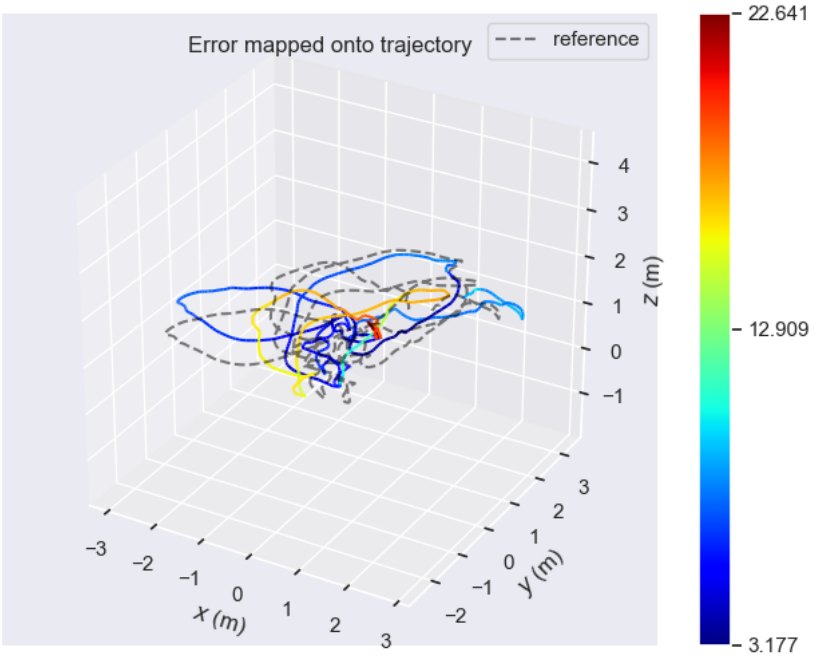


Figure 4.6: Rotation error mapped onto trajectory of `V1_01_easy`

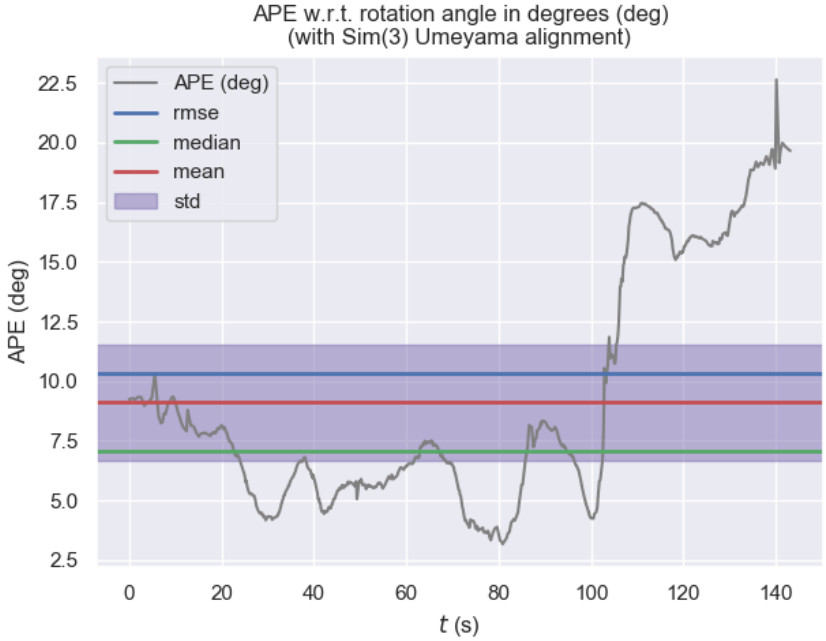


Figure 4.7: Absolute pose error with respect to rotation part of `V1_01_easy`

In Fig. 4.9, the trajectory is represented in x, y and z direction. It can be seen that the trajectory is quite close to the ground truth. The drift approximately ranges from 0 to 0.9 m.

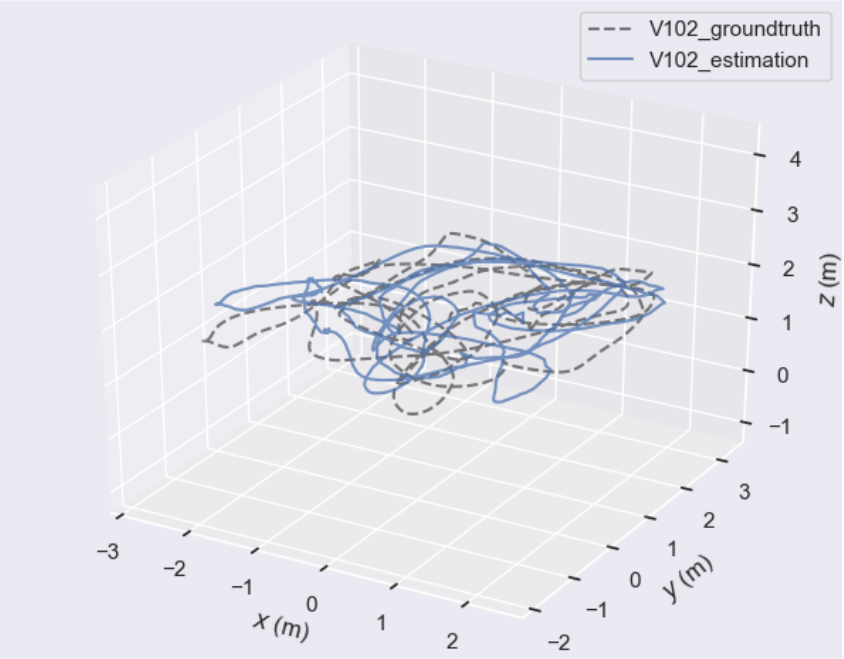


Figure 4.8: The ground truth and estimated trajectory of `V1_02_medium`

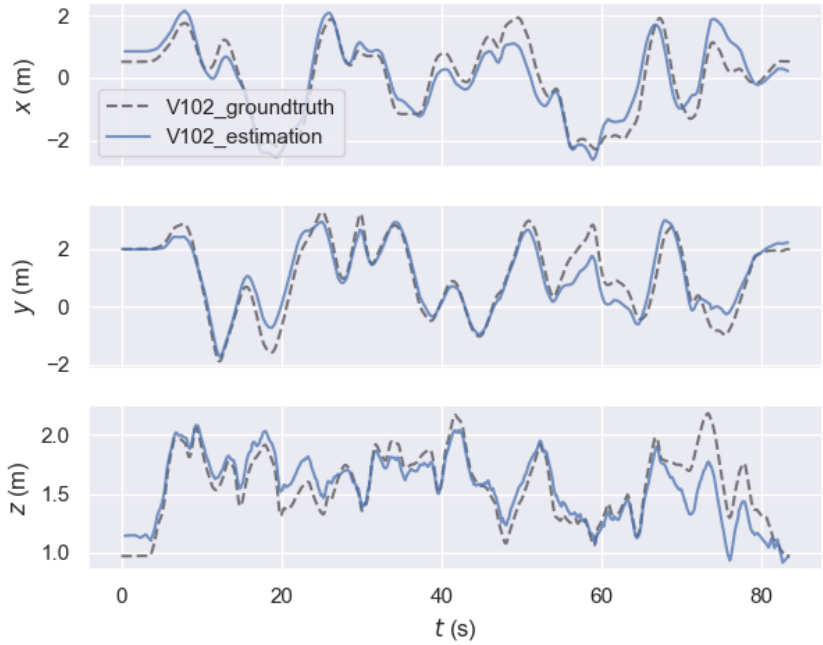


Figure 4.9: x, y, z translation of `V1_02_medium`

Fig. 4.10 is the translation error map, where the red circle part drifts heavily, more than one meter away from the ground truth. The quantitative error analysis is given in Fig 4.11. The average error is about 0.61 m, with standard deviation around 0.28 m. In the best case, the error can be as low as 0.11 m. While in the

worst case the error reaches 1.34 m.

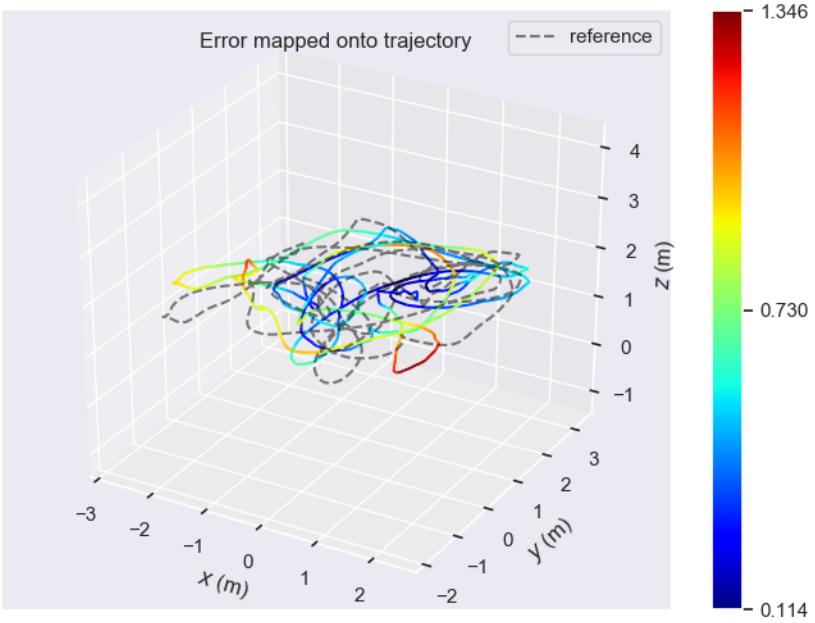


Figure 4.10: Translation error mapped onto trajectory of `V1_02_medium`

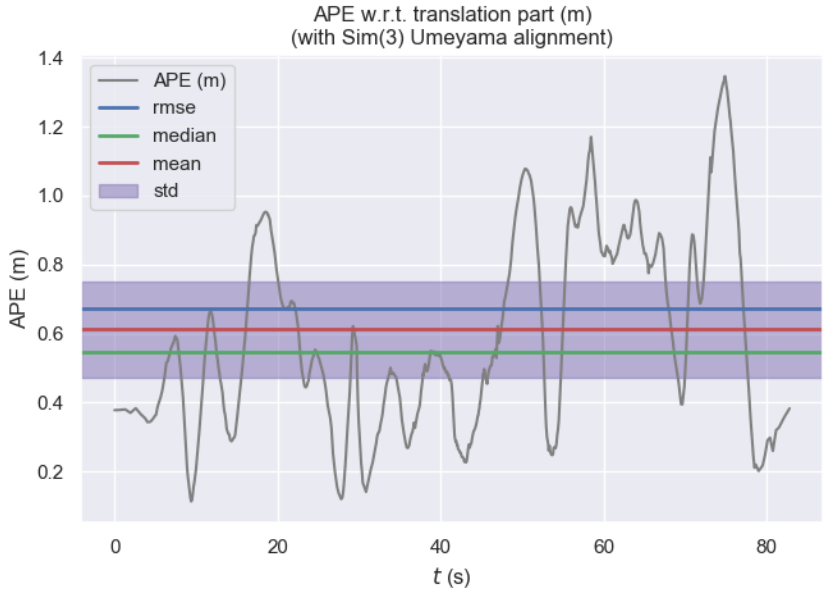


Figure 4.11: Absolute pose error with respect to translation part of `V1_02_medium`

For rotation, the comparison is shown in Fig. 4.12. Like the case in `V1_01_easy` the roll angle flips sometimes. But overall the rotation angles match well with the ground truth from the figure.

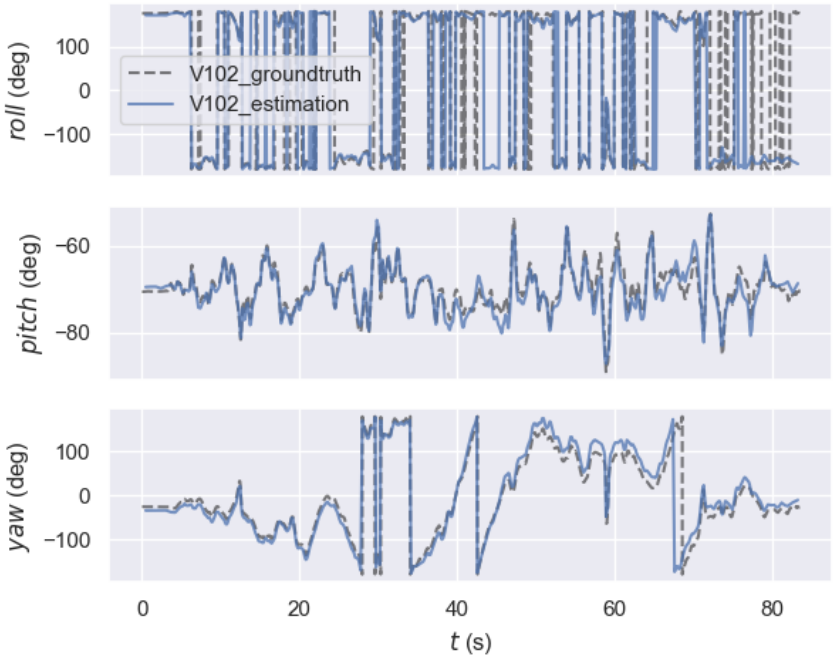


Figure 4.12: Roll, pitch and yaw angle of `V1_02_medium`

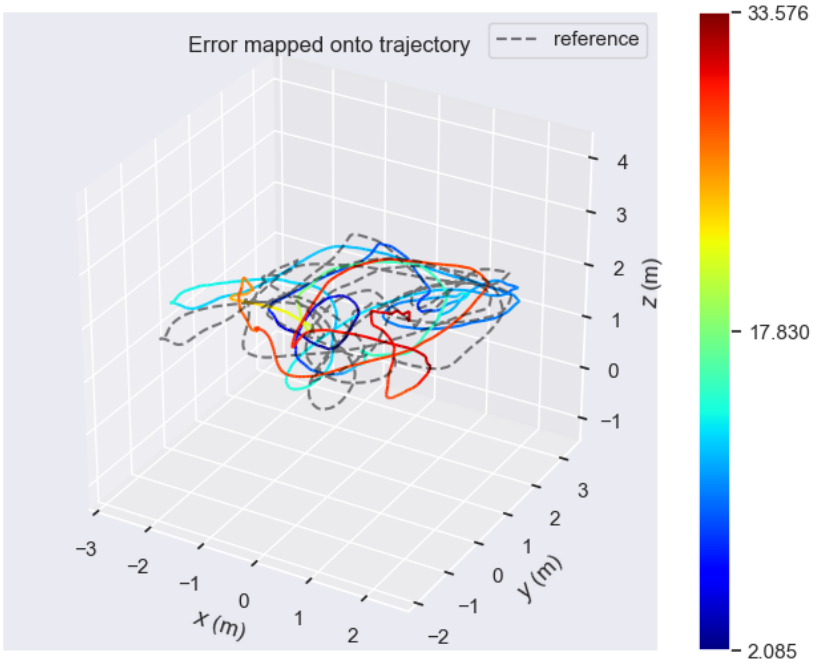


Figure 4.13: Rotation error mapped onto trajectory of `V1_02_medium`

From the color map in Fig. 4.13, the red circle part with high error is larger than the one in translation part. Fig. 4.14 gives the absolute pose error of rotation part. The average error is about 16.37 °. The minimum error is 2.08 ° and the maximum is 33.57 °. And the error keeps increasing after $t = 40$ s.

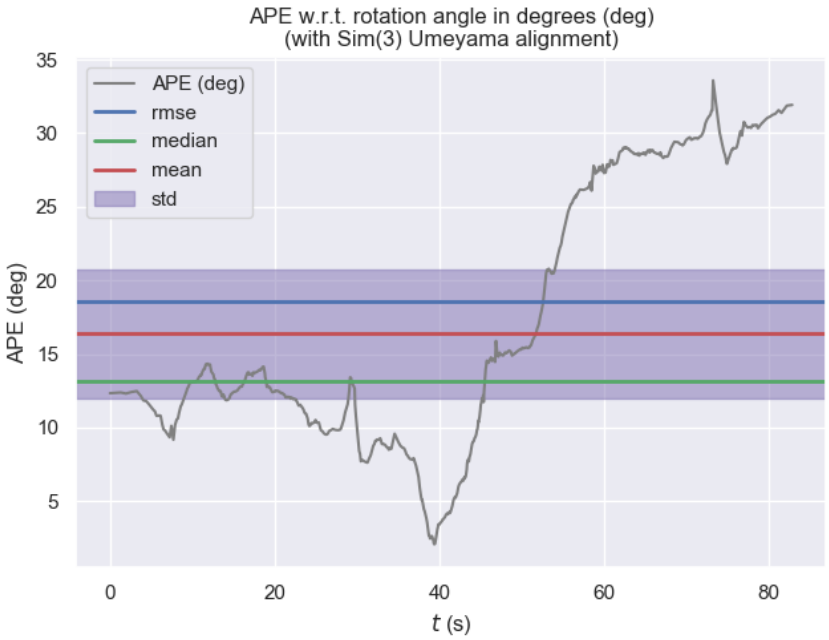


Figure 4.14: Absolute pose error with respect to rotation part of `V1_02_medium`

Fast motion and dark scene

`MH_05_difficult` is the most difficult out of these three datasets. The number of poses, length of path and duration time are shown in Table 4.3. The estimated path length is longer than the ground truth.

Table 4.3: Trajectory information of `MH_05_hard`

	Ground truth	Estimation
Number of poses	22212	944
Path length [m]	97.593	101.125
Duration [s]	111.055	110.850

Fig. 4.15 gives the ground truth and estimated trajectories of `MH_05_hard`. The estimated trajectory shifts a bit and is not closed, though it has similar shape with the ground truth.

In Fig. 4.16, the trajectory is represented respectively in x, y and z direction. Fig. 4.17 is the translation error map, and Fig 4.18 shows the absolute pose error with respect to translation. The average error is about 1.41 m, with standard deviation around 0.50 m. In the best case, the error can be as low as 0.30 m. While in the worst case the error reaches 2.62 m. It can be found that at the beginning and the end the error is extremely high.

For rotation, the comparison is shown in Fig. 4.19. The roll angle flips sometimes, and the other two look more or less matched with the ground truth.

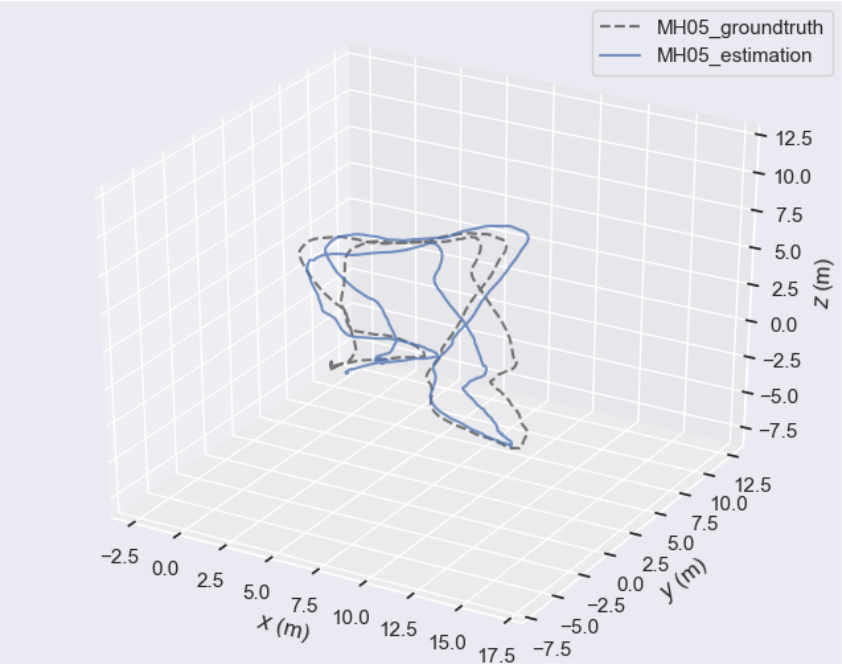


Figure 4.15: The ground truth and estimated trajectory of `MH_05_hard`

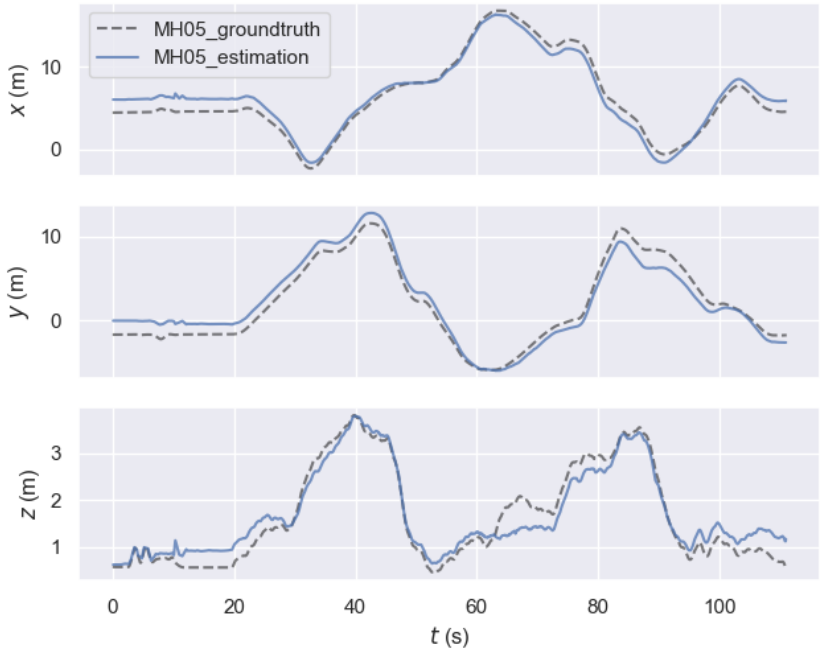


Figure 4.16: x, y, z translation of `MH_05_hard`

Fig. 4.20 and Fig. 4.21 contain more details about the absolute pose error of rotation part. The average error is about 6.35 °, which is smaller compared to the one in

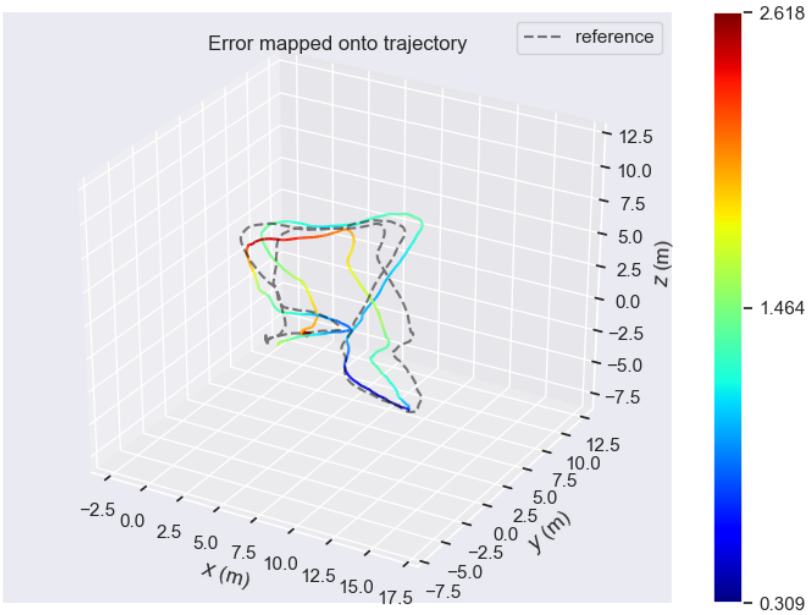


Figure 4.17: Translation error mapped onto trajectory of `MH_05_hard`

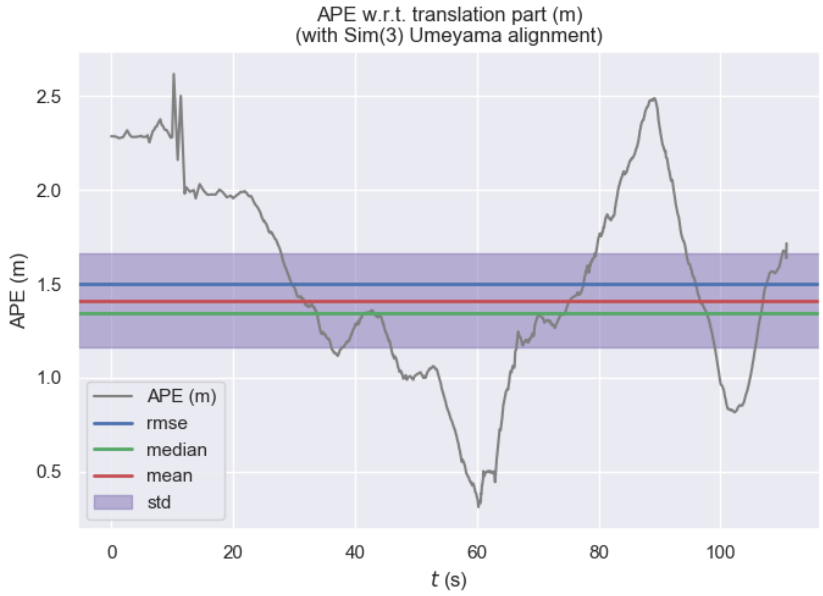


Figure 4.18: Absolute pose error with respect to translation part of `MH_05_hard`

`V1_02_medium`. The minimum error is 1.39 °, and the worst case is 13.82 ° occurring at the end.

In summary, the above results are not ideal. All three tests show that it is easily to drift over long time, from the fact they have high errors at the last tens of seconds. The second test has highest rotation error, possibly due to the fastest movement in

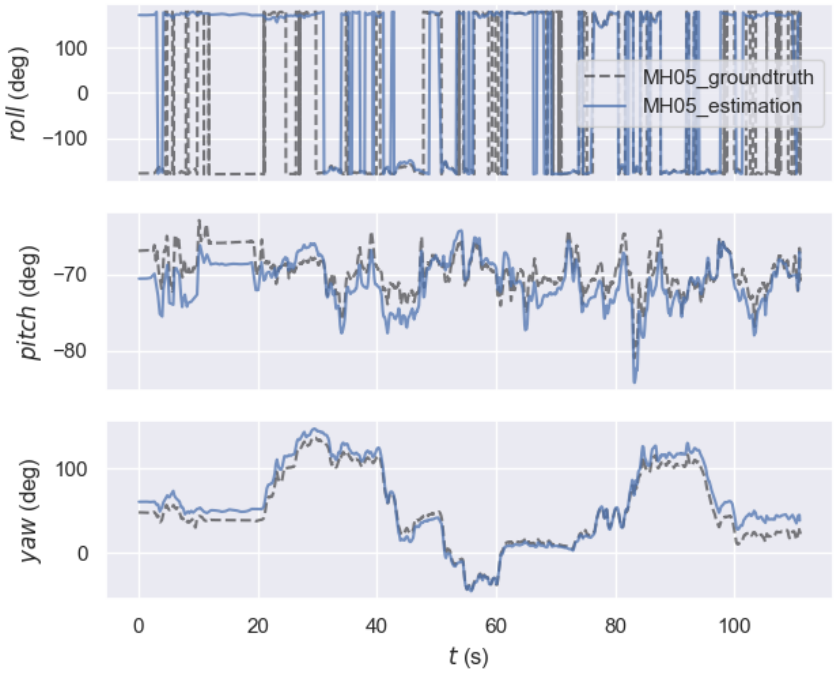


Figure 4.19: Roll, pitch and yaw angle of `MH_05_hard`

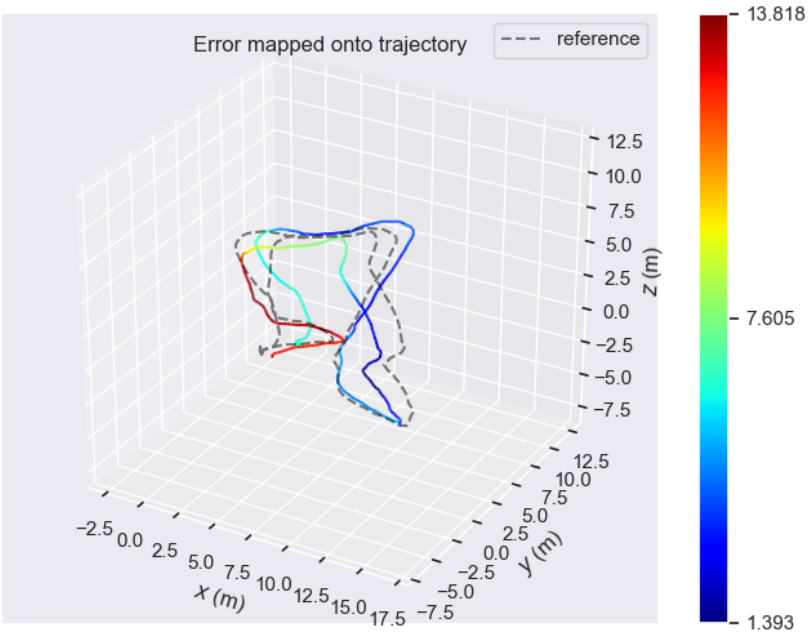


Figure 4.20: Rotation error mapped onto trajectory of `MH_05_hard`

this dataset. The third test has very bad translation result at initial state, possibly because the dark environment cannot provide enough visual information to the initialization procedure.

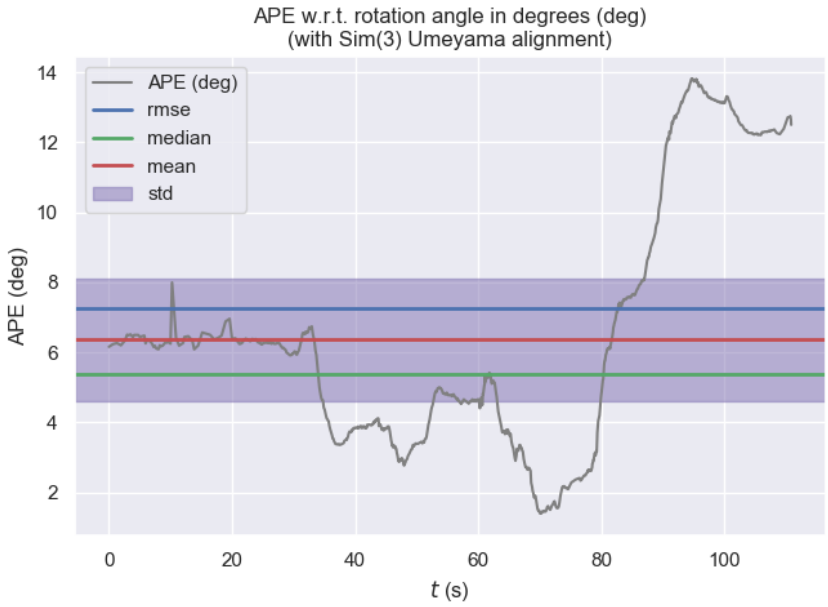


Figure 4.21: Absolute pose error with respect to rotation part of `MH_05_hard`

The proposed visual inertial algorithm still has considerable room to improve in terms of accuracy. Discussion on possible reasons that might cause the system deteriorating will be left to next chapter.

4.2 Computational performance

This section talks about the computation efficiency of the proposed method, in particular, the processing time per frame, and FPS. For real-time purpose, the processing is supposed to be around 30 FPS [36].

In the left of Fig. 4.22, the red line presents the processing time at specific frame, and the green line indicates the average value, which is around 43 ms and is equivalent to 23 FPS. The right of Fig. 4.22 is the histogram of processing time. Most of the processing time falls in the range 35 to 50 ms.

When testing the dataset `V1_01_easy`, the number of ORB feature points to detect is set as 1200. For more details about parameter configurations, refer to the Appendix A.1. This processing speed is acceptable in real application, and particularly it can be improved further using GPU to speed up feature detection.

From Fig. 4.23, the average processing time is around 38 ms, i.e., 26 FPS. And most of the processing time ranges from 30 to 45 ms.

The number of ORB feature points is set as 1000, and thus the processing speed is a little bit faster than the one mentioned above which has 1200 feature points

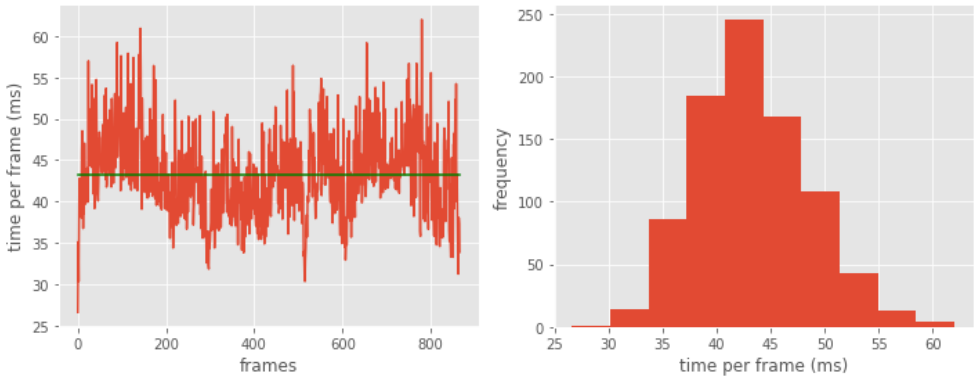


Figure 4.22: Processing time per frame and its frequency in `V1_01_easy`

to detect at each image frame. The selection of how many feature points to detect depends on the environment brightness, texture and other factors, so it is usually tuned by tries and errors. The difference indicates that the the amount of features is important parameter for processing speed.

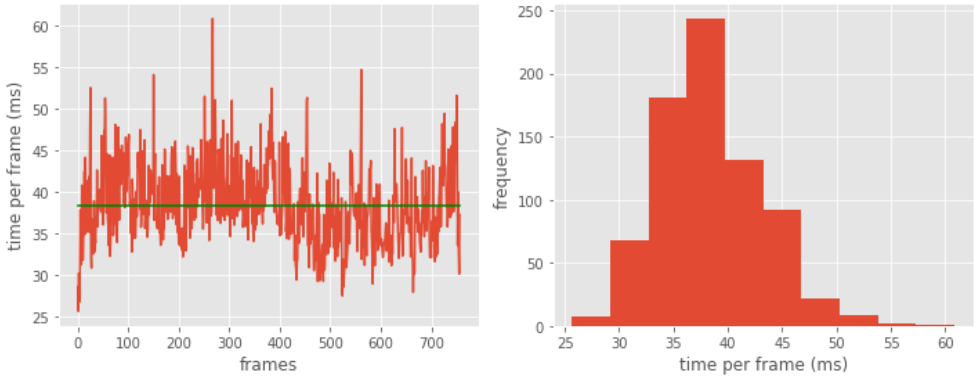


Figure 4.23: Processing time per frame and its frequency in `V1_02_medium`

From Fig. 4.24, the average processing time is about 52 ms, i.e., 19 FPS. And most of the processing time ranges from 45 to 65 ms.

Given that `MH_05_difficult` has fast motion and dark scene, the number of feature points is increased to 2000 to compensate for blur and low texture. Therefore, the processing time is higher than the previous two cases.

In summary, the processing speed ranges around 19 FPS to 26 FPS. The processing speed itself is affected by many factors, for example the number of feature points discussed before, and other aspects such as the strategy for optimization. In next chapter these will be further discussed.

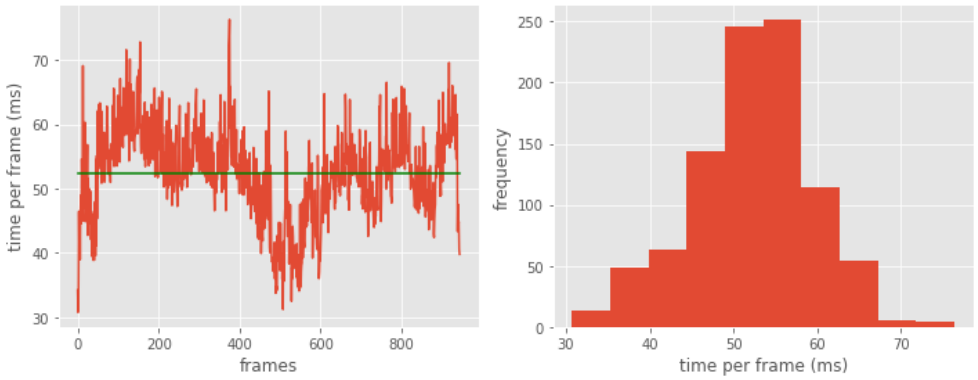


Figure 4.24: Processing time per frame and its frequency in `MH_05_difficult`

4.3 Results overview

Results discussed above are summarized into Table 4.4.

Table 4.4: Overview of the results

		V1_01	V1_02	MH_05
translation error [m]	min	0.12	0.11	0.30
	max	1.05	1.34	2.62
	mean	0.55	0.61	1.41
rotation error [°]	min	3.17	2.08	1.39
	max	22.55	33.57	6.35
	mean	9.11	16.37	13.82
processing speed [FPS]	min	20	22	15
	max	28	33	22
	mean	23	26	19

A comparison to existing top performing algorithms is also done to get a feeling of whether this work is good or not. Table 4.5 gathers the absolute translation errors, i.e., mean translation errors, of some well known algorithms.

Table 4.5: Absolute translation errors (RMSE) in meter of top performing algorithms [1]

	V1_01	V1_02	MH_05
MSCKF [37]	0.34	0.20	0.48
OKVIS [38]	0.09	0.20	0.47
ROVIO [39]	0.10	0.10	0.52
VINS-Mono [9]	0.07	0.10	0.35

From the above table, the mean translation errors of the top performing algorithms are smaller than the ones from this work, which are 0.55, 0.61 and 1.41 m for dataset `V1_01`, `V1_02` and `MH_05` in respective. In this sense, this work performs relatively bad compared to the well known algorithm. The next chapter will focus

on discussions about the current implementation and some possible directions to work on to make improvement.

5
Discussion

This chapter covers reflection on application of the proposed method, and discussion about some shortcomings and what might cause the degradation of the algorithm. It also talks about some strategies that are different with the implemented ones or haven't been applied in the implementation yet.

5.1 Application

The intention of this work is to find a way to better localize a vehicle without using GPS. The proposed method combines a stereo camera and IMU in a tightly coupled sensor fusion framework. Since the visual and inertial SLAM is one of the hottest topics in research, many works have been done and frameworks been proposed for different purposes such as autonomous driving, virtual reality and 3D reconstruction. This work is based on the some already existed research, and tries to adapt to the specific scenario of formula student driverless competition.

With ongoing development, the implementation is supposed to achieve a more robust and accurate performance, by providing more reliable initialization and state estimations. In specific, this work aims to serve as a localization and mapping module in the CFSD project. But in future, the work could be extended to a more general one and fit more scenarios for autonomous driving.

5.2 Direct vs. indirect methods

Methods that deal with image measurements and recover camera pose can be divided into two categories: direct methods and indirect methods.

The indirect methods are feature-based. It extracts a sparse set of salient features in each image and then finds matches in later images with invariant features. The relative camera motion can be recovered with epipolar geometry, and refined through bundle adjustment to minimize the reprojection error [40]. This is exactly what is exploited in this work. For the feature extraction and tracking, typical algorithms include optical flow, ORB, SIFT and SURF, which have already been briefly introduce in section 2.2.4.

On the other hand, the direct methods use all information in the image rather than just some features. It is based on the assumption that successive frames has almost the same photometric intensity at the same pixel location. Camera motion is recovered through photometric error minimization. It has been proved to outperform the feature-based methods in the cases with motions blur and little texture [41].

Fig. 5.1 is a simple comparison of the mentioned methods in terms of accuracy and computational cost [42]. It can be seen that the optical flow method has little difference with the ORB method in terms of accuracy, but costs less time. Therefore, one way to improve the current system could be changing the visual frontend to the optical flow one. On the other hand, the indirect methods highly depend on distinctive features in images, but there are plenty of scenes that no obvious corner points are available, where the feature detection algorithm extracts a bunch of useless features which will be filtered out later. To make the system more robust in situations where images are blurred and textureless, the direct method could also be considered in future development.

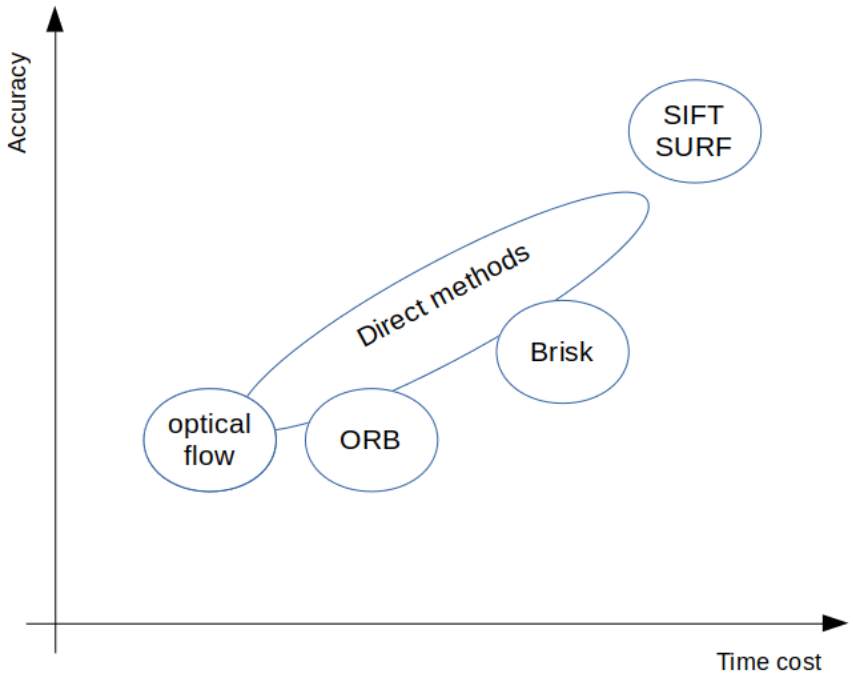


Figure 5.1: Comparison of direct and indirect methods in terms of accuracy and cost

5.3 System initialization

The proposed initialization procedure allows the system to start up with the estimated initial gravity, velocity and bias through fusing camera and IMU information in a loosely coupled way, and it is assumed that all sensors are static during initializing. This could be upgraded to deal with dynamic initialization or even adaptive

one where the initializing scheme is automatically selected.

Another downside of current system is that it lacks criteria on whether initialization is successful or the estimate is reliable enough. An failure example could be seen in the test of `MH_05_difficult`, where the translation error at beginning is extremely large. Wrong starting point would lead to considerable deviation in some cases.

5.4 Motion-only vs. motion-structure

The backend optimization scheme can be divided into two classes: motion-only and motion-structure optimization. The difference between them can be simply viewed as the difference between localization-only and localization-mapping problem. The former one only cares about the trajectory estimation, while the latter focuses on both the trajectory and the environment reconstruction.

In the implementation, the 3D landmark points are assumed to be fixed once they are obtained. The feature matching results are supposed to be good enough by using several outlier rejection schemes mentioned in section 3.2.3. Afterwards the features are triangulated to get 3D points that are relative to the camera coordinates and need to be transformed to the world coordinates. The transformation is done by applying the pose information of current frame, therefore, the derived 3D points would deviate from their true positions if current frame's pose is not optimal. And the error information will be propagated through further motion-only optimization and coordinates transformation. An example is shown in Fig. 3.9, some 3D points are not in their true position, rendering the reconstructed point clouds shifted to some extend, and make what should be a plane looks like a box. Therefore, although the assumption enables motion-only bundle adjustment and makes high-rate pose estimation output achievable without the need to optimize numbers of landmarks, it exposes the system to long term drifts.

To make improvement, the landmarks should be optimized as well, by adding the 3D points into the state vector, and finding the optimal values in the solution space. The reprojection residual is constructed as:

$$\mathbf{r} = \begin{bmatrix} \frac{x_j}{z_j} - u_j \\ \frac{y_j}{z_j} - v_j \end{bmatrix} \tag{5.1}$$

where $\mathbf{p}_j = [x_j y_j z_j]^T$ is the landmark position relative to frame j, and $[u_j v_j]^T$ is the pixel position in frame j. $\mathbf{p}_j$ is derived from the corresponding frame i:

$$\begin{bmatrix} x_j \\ y_j \\ z_j \\ 1 \end{bmatrix} = \mathbf{T}_{bc}^{-1} \mathbf{T}_{wb_j}^{-1} \mathbf{T}_{wb_i} \mathbf{T}_{bc} \begin{bmatrix} x_i \\ y_i \\ z_i \\ 1 \end{bmatrix} \tag{5.2}$$

where $\mathbf{T}_{bc}$ and $\mathbf{T}_{wb_i}$ represent the transformation from the camera frame to the body frame, and the body frame i to the world frame in respective. Every time minimizing

the error of Eq. 5.1, there are three variables x, y and z to optimize. Since there are much more landmarks than poses, this will add a lot of computational load unfortunately. A reasonable compromise is the inverse depth technique applied in VINS-Mono [9]:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \frac{1}{\lambda} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{5.3}$$

where $\lambda = \frac{1}{z}$ is defined as inverse depth. Substitute into Eq.5.2:

$$\begin{bmatrix} x_j \\ y_j \\ z_j \\ 1 \end{bmatrix} = \mathbf{T}_{bc}^{-1} \mathbf{T}_{wb_j}^{-1} \mathbf{T}_{wb_i} \mathbf{T}_{bc} \begin{bmatrix} \frac{1}{\lambda} u_i \\ \frac{1}{\lambda} v_i \\ z_i \\ 1 \end{bmatrix} \tag{5.4}$$

Therefore the residual is a function of λ, and the number of variables to optimize is now reduced from 3 to 1, alleviating the computational costs.

5.5 Marginalization

Since the trajectory grows gradually, the concept of sliding window is proposed to keep the number of poses to optimize constant and thus the processing time is acceptable. The strategy exploited in the implementation provides another explanation for the non-ideal testing results: when a new keyframe comes in, the oldest keyframe in the sliding window is dropped directly, which means the information carried by the oldest keyframe is discarded. The estimation is essentially a Bayesian optimization problem. Optimizing the states in the sliding window is a Maximum Likelihood (ML) estimate, if given a prior, then it could be upgraded to a Maximum a Posterior (MAP) estimate. By directly discarding the keyframe, information is lost and there will be no prior available. This could be solved by properly marginalizing out the unwanted variables.

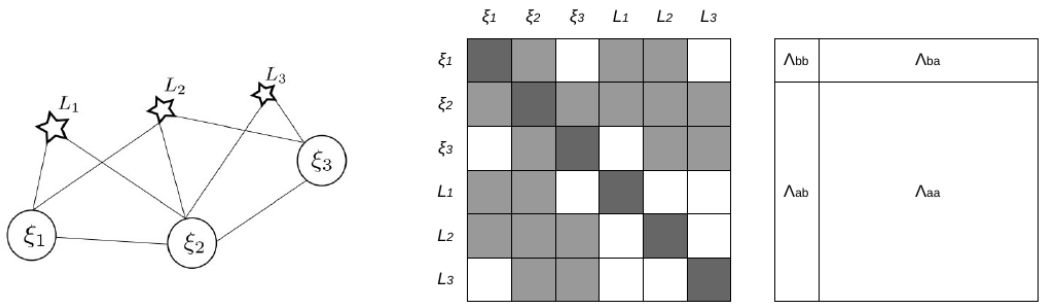


Figure 5.2: An example SLAM problem and its information matrix

Marginalization is the method that discards variables but retains the information. Take a simple example, in the left of Fig. 5.2, ξ and L represents pose and landmark in respective. In the middle shows the information matrix of this problem, where

non-white entry means non-zero value and there exists a constraint between the corresponding variables. To marginalize out ξ_1, the matrix is divided into 4 parts, with the up-left corner to be eliminated.

The Schur Complement is applied, with details shown in Fig. 5.3. In essence it is a Gaussian elimination problem, and ξ_1 is discarded but the constraint information is kept in Λ'_{aa}.

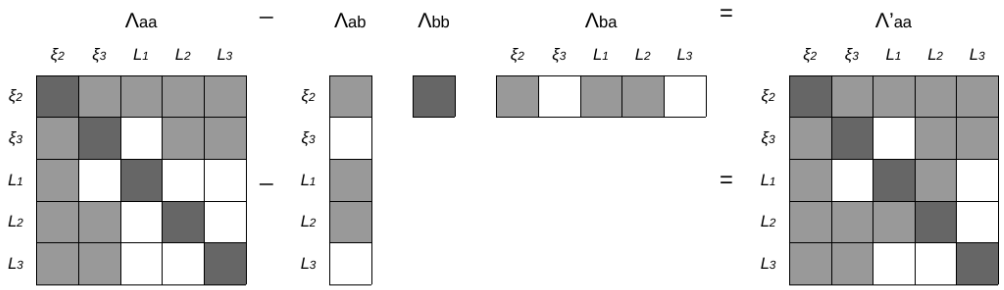


Figure 5.3: Marginalization with Schur Complement

5.6 User-defined parameters

There are considerable parameters need to be tuned based on experiments. In the Appendix A.1, some of the parameter values are given by the datasets such as sensor settings. Others are user-defined like feature detection settings, tracking settings, and optimization settings. Using different parameters would dramatically affect the results.

For instance, setting a high number of features can slow down the processing speed. As seen in Fig. 4.22 the average time per frame is around 43 ms with 1200 features, and in Fig. 4.24 the time increases to 52 ms with 2000 features. Generally speaking, one could set a low number of features to gain faster computation at the loss of accuracy, and vice versa. But how many features would be proper to use also depends on the quality of images, and typically more features are needed to compensate for the worse visual measurements. This number is just one of the most parameters that affect the overall performance.

The user-defined parameters also include various kinds of thresholds. For example, the minimum match distance decides how close should two feature descriptors be so that they are considered as identical. The max depth makes sure points that are too far away will be rejected since they are less accurate due to camera's measuring limits. The thresholds of keyframe rotation and translation determine how often should a frame be set to keyframe.

6
Conclusion and future work

This thesis is motivated by the CFSD19 project, aiming at delivering a self-driving formula race car. In this work a visual inertial odometry is implemented for the purpose of real-time state estimation. In the frontend the ORB detection method is applied to extract visual information, associate different image frames and keep track of the image sequence. The IMU measurements are processed based on the preintegration theory. A four-step initialization approach is proposed to derive initial estimations. In the backend, a least-square problem is constructed with information from the frontend and well-defined cost functions, and is solved through the Levenberg-Marquardt algorithm. The motion-only bundle adjustment is performed over a fixed-size sliding window. The implementation is open-sourced in the Github repository [43].

To answer the first research question in section 1.3 (i.e., how well does the proposed method estimate an egomotion state?), tests regarding to the estimation error of the proposed algorithm are performed with the EuRoC datasets. In general it can produce an estimated trajectory that has a similar shape as the ground truth, under situations varying from slow motion to fast motion and bright scene to dark scene. The minimum error could reach 0.1–0.3 m and 2–3 ° for translation and rotation errors respectively. However, overall it still suffers from drifts and some unexpected behaviors due to many factors. In respective, the mean errors of translation and rotation are 0.5–1.4 m and 9–16°.

Generally, the more complex the environment is, the less visual texture and thus higher estimation error, but possible reasons that might cause the non-ideal outcomes are also discussed. One of them is user-defined parameters such as the number of features to detect, maximum matching distance, max depth and so on. They should be tuned to fit different environment and motion settings. Therefore, it is concluded that extension research is still necessary to further enhance the system accuracy and robustness.

Unfortunately, when compared to the top performing algorithms, the current implementation does not achieve a descent result, which also indicates there is a big room for improvement.

To answer the second research question (i.e., how fast is the implemented algorithm?), processing speed of the algorithm is examined with EuRoC datasets. On average the processing speed can reach 19–26 FPS, which is reasonable in the CFSD

application. While compared to real-time purpose system which runs at around 30 FPS [36], the algorithm needs to be optimized further by, for example, involving graphic processing unit to extract feature points in images, or replacing the feature-based methods with the direct methods in the frontend.

6.1 Future work

As mentioned in previous sections, the next thing to do would be upgrading the submodules of current system, including the direct methods for visual frontend, dynamic initialization procedures, inverse depth optimization and marginalization as discussed in last chapter.

In detail, the future work would focus on improving the reliability of initialization results, upgrading the strategy of keyframes and map points management, realizing the functionality of optimizing motion and map simultaneously, enhancing the estimation accuracy and robustness by incorporating the marginalization information. Further, more advanced topics such as loop closure, global optimization, failure detection, and relocalization after losing track or bias corrupted, would be considered.

With a better foundation, the very initial goal will be aimed at: adapt the visual inertial SLAM to OpenDLV platform and serve for the CFSD project.

Bibliography

[1] Jeffrey Delmerico and Davide Scaramuzza. [ieee 2018 ieee international conference on robotics and automation (icra) - brisbane, australia (2018.5.21-2018.5.25)] 2018 ieee international conference on robotics and automation (icra) - a benchmark comparison of monocular visual-inertial odometry algo. In *IEEE International Conference on Robotics Automation*, 2018.

[2] Miranda A Schreurs and Sibyl D Steuwer. Autonomous driving-political, legal, social, and sustainability dimensions. In *Autonomes Fahren*, pages 151–173. Springer, 2015.

[3] Bernhard Friedrich. *The Effect of Autonomous Vehicles on Traffic*, pages 317–334. 05 2016.

[4] Kriti Kumar, Ashley Varghese, Pavan K Reddy, N Narendra, Prashanth Swamy, M Girish Chandra, and P Balamuralidhar. An improved tracking using imu and vision fusion for mobile augmented reality applications. *arXiv preprint arXiv:1411.2335*, 2014.

[5] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[6] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Computer vision and image understanding*, 110(3):346–359, 2008.

[7] Mary B Alatise and Gerhard P Hancke. Pose estimation of a mobile robot based on fusion of imu data and vision data using an extended kalman filter. *Sensors*, 17(10):2164, 2017.

[8] Jeroen Diederik Hol. *Pose estimation and calibration algorithms for vision and inertial sensors*. PhD thesis, Institutionen för systemteknik, 2008.

[9] Tong Qin, Peiliang Li, and Shaojie Shen. Vins-mono: A robust and versatile monocular visual-inertial state estimator. *IEEE Transactions on Robotics*, 34(4):1004–1020, 2018.

[10] Christian Forster, Luca Carlone, Frank Dellaert, and Davide Scaramuzza. On-manifold preintegration for real-time visual–inertial odometry. *IEEE Transactions on Robotics*, 33(1):1–21, 2017.

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

[12] Christian Berger, Björnborg Nguyen, and Ola Benderius. Containerized development and microservices for self-driving vehicles: Experiences & best practices. In *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, pages 7–12. IEEE, 2017.

[13] Inc. The MathWorks. *Computer Vision Toolbox*. Natick, Massachusetts, United State, 2019.

[14] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *European conference on computer vision*, pages 404–417. Springer, 2006.

[15] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary R Bradski. Orb: An efficient alternative to sift or surf. In *ICCV*, volume 11, page 2. Citeseer, 2011.

[16] Oliver J Woodman. An introduction to inertial navigation. Technical report, University of Cambridge, Computer Laboratory, 2007.

[17] Todd Lupton and Salah Sukkarieh. Visual-inertial-aided navigation for high-dynamic motion in built environments without initial conditions. *IEEE Transactions on Robotics*, 28(1):61–76, 2011.

[18] Timothy Barfoot and Paul Furgale. Associating uncertainty with three-dimensional poses for use in estimation problems. *Robotics, IEEE Transactions on*, 30:679–693, 06 2014.

[19] John L Crassidis. Sigma-point kalman filtering for integrated gps and inertial navigation. *IEEE Transactions on Aerospace and Electronic Systems*, 42(2):750–756, 2006.

[20] Kaj Madsen, Hans Nielsen, and O Tingleff. Methods for non-linear least squares problems (2nd ed.). page 60, 01 2004.

[21] PE Frandsen, K Jonasson, HB Nielsen, and O Tingleff. Unconstrained optimization, informatics and mathematical modelling. *Technical University of Denmark, Copenhagen*, 2004.

[22] Kenneth Levenberg. A method for the solution of certain non-linear problems in least squares. *Quarterly of applied mathematics*, 2(2):164–168, 1944.

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

[24] Davide Scaramuzza and Friedrich Fraundorfer. Visual odometry [tutorial]. *IEEE robotics & automation magazine*, 18(4):80–92, 2011.

[25] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[26] Francesc Moreno-Noguer, Vincent Lepetit, and Pascal Fua. Accurate non-iterative o (n) solution to the pnp problem. In *2007 IEEE 11th International Conference on Computer Vision*, pages 1–8. IEEE, 2007.

[27] Steffen Gauglitz, Luca Foschini, Matthew Turk, and Tobias Höllerer. Efficiently selecting spatially distributed keypoints for visual tracking. In *2011 18th IEEE International Conference on Image Processing*, pages 1869–1872. IEEE, 2011.

[28] Igor Cvišic, Josip Cesic, Ivan Markovic, and Ivan Petrovic. Soft-slam: Computationally efficient stereo visual slam for autonomous uavs. *Journal of field robotics*, 2017.

[29] Wikipedia contributors. Extended kalman filter — Wikipedia, the free encyclopedia, 2019. [Online; accessed 14-October-2019].

[30] Hauke Strasdat, José MM Montiel, and Andrew J Davison. Visual slam: why filter? *Image and Vision Computing*, 30(2):65–77, 2012.

[31] Gabe Sibley, Larry Matthies, and Gaurav Sukhatme. Sliding window filter with application to planetary landing. *Journal of Field Robotics*, 27(5):587–608, 2010.

[32] Tue-Cuong Dong-Si and Anastasios I Mourikis. Motion tracking with fixed-lag smoothing: Algorithm and consistency analysis. In *2011 IEEE International Conference on Robotics and Automation*, pages 5655–5662. IEEE, 2011.

[33] Peter S Maybeck. *Stochastic models, estimation, and control*, volume 3. Academic press, 1982.

[34] Rainer Kümmerle, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige, and Wolfram Burgard. g 2 o: A general framework for graph optimization. In *2011 IEEE International Conference on Robotics and Automation*, pages 3607–3613. IEEE, 2011.

[35] Michael Grupp. evo: Python package for the evaluation of odometry and slam. `https://github.com/MichaelGrupp/evo`, 2017.

[36] Brian Patrick Williams, Georg Klein, and Ian D Reid. Real-time slam relocalisation. In *ICCV*, volume 7, pages 1–8, 2007.

[37] Anastasios /I Mourikis and Stergios /I Roumeliotis. A multi-state constraint kalman filter for vision-aided inertial navigation. In *IEEE International Conference on Robotics Automation*, 2007.

[38] Stefan Leutenegger, Simon Lynen, Michael Bosse, Roland Siegwart, and Paul Furgale. Keyframe-based visual–inertial odometry using nonlinear optimization. *International Journal of Robotics Research*, 34(3):314–334, 2014.

[39] Michael Bloesch, Sammy Omari, Marco Hutter, and Roland Siegwart. [ieee 2015 ieee/rsj international conference on intelligent robots and systems (iros) - hamburg, germany (2015.9.28-2015.10.2)] 2015 ieee/rsj international conference on intelligent robots and systems (iros) - robust visual inertial odometry using a direct ekf-based approach. In *IEEE/RSJ International Conference on Intelligent Robots Systems*, 2015.

[40] Michal Irani and P Anandan. About direct methods. In *International Workshop on Vision Algorithms*, pages 267–277. Springer, 1999.

[41] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. Svo: Fast semi-direct monocular visual odometry. In *2014 IEEE international conference on robotics and automation (ICRA)*, pages 15–22. IEEE, 2014.

[42] Xiang Gao, Tao Zhang, Yi Liu, and Qinrui Yan. *14 Lectures on Visual SLAM: From Theory to Practice*. Publishing House of Electronics Industry, 2017.

[43] `https://github.com/chalmersfsd/cfsd-perception-slam`.

A

Appendix

A.1 Configuration

```
#########################################################
##### Sensor settings #####
# Camera
imageWidth: 752
imageHeight: 480
cameraFrequency: 20
# Standard deviation of pixel-level measurement
stdX: 0.5
stdY: 0.5

# Left camera intrinsics
camLeft: !!opencv-matrix
    rows: 3
    cols: 3
    dt: d
    data: [ 458.654, 0, 367.215,
            0, 457.296, 248.375,
            0, 0, 1 ]
distLeft: !!opencv-matrix
    rows: 5
    cols: 1
    dt: d
    data: [ -0.28340811, 0.07395907,
            0.00019359, 1.76187114e-05, 0 ]

# Right camera intrinsics
camRight: !!opencv-matrix
    rows: 3
    cols: 3
    dt: d
    data: [ 457.587, 0, 379.999,
            0, 456.134, 255.238,
            0, 0, 1 ]
distRight: !!opencv-matrix
```

```
    rows: 5
    cols: 1
    dt: d
    data: [ -0.28368365,  0.07451284,
            -0.00010473, -3.55590700e-05, 0 ]

# Camera extrinsics
rotationLeftToRight: !!opencv-matrix
    rows: 3
    cols: 3
    dt: d
    data: [ 0.999997, 0.00231207, 0.000376008,
            -0.00231714, 0.999898, 0.0140898,
            -0.000343393, -0.0140907, 0.999901 ]
translationLeftToRight: !!opencv-matrix
    rows: 3
    cols: 1
    dt: d
    data: [ -0.110074, 0.000399122, -0.000853703 ]

# IMU paramteres
## IMU coordinate system    camera coordinate system
##        x |  / z                    / z
##          | /                      /
##           ------ y               ------ x
##                                  |
##                                  | y
samplingRate: 200 # [Hz]
# (euroc) inertial sensor noise model parameters (static)
gyroscope_noise_density: 1.6968e-04
    # [ rad / s / sqrt(Hz) ] ( gyro "white noise" )
gyroscope_random_walk: 1.9393e-05
    # [ rad / s^2 / sqrt(Hz) ] ( gyro bias diffusion )
accelerometer_noise_density: 2.0000e-3
    # [ m / s^2 / sqrt(Hz) ]   ( accel "white noise" )
accelerometer_random_walk: 3.0000e-3
    # [ m / s^3 / sqrt(Hz) ]  ( accel bias diffusion )

# Need to estimate extrinsics or not:
# 0 - the given extrinsics below is good enough,
#     will be kept fixed
# 1 - the given extrinsics below is just an initial
#     guess, need to be optimized
estimateExtrinsics: 0
# Camera (left) and imu extrinsics
rotationImuToCamera: !!opencv-matrix
```

```
    rows: 3
    cols: 3
    dt: d
    data: [ 0.0148655, 0.999557, -0.0257744,
            -0.999881, 0.0149672, 0.00375619,
            0.0041403, 0.0257155, 0.999661 ]
translationImuToCamera: !!opencv-matrix
    rows: 3
    cols: 1
    dt: d
    data: [ 0.0652229, -0.0207064, -0.0080546 ]

#--------------------------------------------------------------
##### Feature detection settings #####
delay: 10

# Use ORB of OpenCV or ORB_SLAM2
cvORB: 0 # default: not use OpenCV ORB

# ORB feature detector parameters
numberOfFeatures: 1200
scaleFactor: 1.2
levelPyramid: 8

# (OpenCV) ORB Detector
edgeThreshold: 31
scoreType: 1    # 0 is ORB::HARRIS_SCORE,
                # 1 is ORB::FAST_SCORE
patchSize: 31
fastThreshold: 20
# For OpenCV ORB, detect in several grids to
# make the distribution of keypoints more evenly
gridRow: 1
gridCol: 1

# (ORB_SLAM2) ORB Extractor: Fast threshold
# Image is divided in a grid. At each cell FAST
# are extracted imposing a minimum response.
# Firstly impose iniThFAST. If no corners are
# detected we impose a lower value minThFAST
# Lower these values if the images have low contrast
iniThFAST: 20
minThFAST: 7

# Feature matching parameters
matchRatio: 2               # the distance of a good
```

```
                                # match should be smaller
                                # than minDist*matchRatio
minMatchDist: 30.0              # another number for selecting
                                # good matches, from experience
maxVerticalPixelDist: 0.1       # selecting good matching by
                                # the vertical coordinates as
                                # images have been rectified
maxFeatureAge: 8                # Max age of a feature
maxDepth: 10                    # Max depth or distance
                                #  w.r.t camera

#########################################################
##### Tracking settings #####
# Keyframe selection
keyframeRotation: 0.2
keyframeTranslation: 0.1
maxImuTime: 4

# Reinitialization if bias is corrupted
maxGyrBias: 0.1
maxAccBias: 0.6

# Initialize with SfM
initializeWithSfm: 1

# SfM frame selection
sfmRotation: 0
sfmTranslation: 0

# Method for solving PnP problem:
# 0-SOLVEPNP_ITERATIVE (default)
# 1-SOLVEPNP_EPNP (Efficient Perspective-n-Point
#   Camera Pose Estimation)
# 2-SOLVEPNP_P3P (Complete Solution Classification
#   for the Perspective-Three-Point Problem)
# 3-SOLVEPNP_DLS (A Direct Least-Squares (DLS)
#   Method for PnP)
# 4-SOLVEPNP_UPNP (Exhaustive Linearization for
#   Robust Camera Pose and Focal Length Estimation)
# 5-SOLVEPNP_AP3P (An Efficient Algebraic Solution
#   to the Perspective-Three-Point Problem)
#   [not available in OpenCV 3.2.0]
solvePnP: 0

#########################################################
##### Optimization settings #####
```

```
minimizer_progress_to_stdout: 0
max_num_iterations: 20 # default: 50
max_solver_time_in_seconds: 10 # default: 1e6
num_threads: 2 # default: 1
check_gradients: 0

# Gravity magnitude for initial accelerometer correction
gravity: 9.81734     # [m/s^2] gravitational acceleration
                     # (might need slightly modification
                     # in case of different location)

# Prior covariance factor
priorWeight: 1e-5
# Loop Closure
loopClosure: 0
# Global optimization
globalOptimize: 0

#########################################################
##### Loop closure settings #####
vocabulary: ../config/vocabulary/ORBvoc.bin
minFrameInterval: 40
minScore: 0.01

#########################################################
##### Viewer settings #####
viewScale: 1
pointSize: 4
landmarkSize: 2
cameraSize: 0.4
cameraLineWidth: 3
lineWidth: 2
viewpointX: 10
viewpointY: 10
viewpointZ: -30
viewpointF: 2000
background: 0 # 0-black, 1-white
## world frame
##    x |  / z
##      | /
##       ------ y
# set the up direction as AxisX:
#  0-AxisNone, 1-AxisNegX, 2-AxisX,
#  3-AxisNegY, 4-AxisY, 5-AxisNegZ, 6-AxisZ
axisDirection: 2
```