Visual Inertial Odometry for Mobile Home Robots

Paulo André Pereira Nogueira

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisor

Professor José António da Cruz Pinto Gaspar

Examination Committee

Chairperson: Professor João Fernando Cardoso Silva Sequeira
Supervisor: Professor José António da Cruz Pinto Gaspar
Member: Professor Rodrigo Martins de Matos Ventura

December 2021

Declaration

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Agradecimentos

É com bastante satisfação que escrevo esta secção. Não só por esta simbolizar o término da longa jornada que tem sido esta dissertação, como o término desta etapa académica na minha vida. Não tem sido uma jornada fácil, mas não alteraria nada sobre este passado tão presente. Deixo agora um especial agradecimento a um conjunto de pessoas que me ofereceram o seu apoio e ajuda.

Em primeiro lugar gostava de agradecer ao meu orientador, Prof. José Gaspar, pela sua disponibilidade, motivação e sabedoria, e especialmente pela sua capacidade de tolerar a minha determinação disfarçada de teimosia. Sem qualquer dúvida foi uma influência tremenda neste final de época, particularmente pela sua dedicação e interesse na exploração do conhecimento.

Quero agradecer a todos os meus amigos e colegas que contribuíram em agilizar as dificuldades encontradas ao longo desta jornada, com uma menção especial ao meu melhor amigo Francisco pela sua amizade e carinho desde aquele dia que nos conhecemos como jovens piratas no meio de um convento de freiras. Seja em momentos de lazer ou até em grupos de estudo, sem estes períodos o caminho teria sido muito mais doloroso.

Deixo também uma palavra de agradecimento a todos os professores que me apoiaram neste caminho com sabedoria e paixão pelo conhecimento.

Quero agradecer a toda a minha família alargada, com especial enfase nos meus avós que todas as semanas me ligavam para me motivar a continuar e terminar este percurso árduo, lembrando-me que é necessário sacrifício para alcançar vitória.

Termino por expressar a minha sincera e profunda gratidão à minha família próxima, nomeadamente os meus pais, Ângela e Paulo, a minha irmã, Rute, e a minha namorada, Pris, pelo apoio, conselho e amor constantes que me permitiram voar em céus tão lamacentos.

A todos, um enorme *obrigado*!

Resumo

Robots conscientes dos seus ambientes de trabalho, e capazes de os percorrer livremente, serão uma realidade ubíqua nas casas do futuro. Odometria visual e inercial é hoje uma tecnologia possível em muitos projetos devido ao baixo custo de câmaras e unidades de medida inercial (IMU). Nesta dissertação analisamos como a inicialização do estado e propagação de dinâmicas com base em leituras da IMU, intrinsecamente sujeitas a enviesamentos (bias), devem corresponder aos movimentos capturados pela câmara. Analisamos as sinergias entre estes dois tipos de sensores no contexto de estimação de pose.

O aparato experimental considerado é um robot móvel com rodas, envolvendo um Raspberry Pi que controla o movimento dos motores, adquire dados de uma câmara, e adquire dados de uma IMU por intermédio de um Arduino. A calibração do sistema, nomeadamente a transformação rígida entre a câmara e IMU, é feita com uma ferramenta disponível publicamente. Os dados visuais e inerciais são processados usando um Unscented Kalman Filter com representação de estado baseada em grupos de Lie. Propomos uma inicialização de estado para este filtro que possibilita a combinação entre as leituras integradas da IMU e dos pontos do cenário acompanhados visualmente.

Validamos experimentalmente a exatidão das leituras da IMU e propomos uma métrica de erro que avalia a estimação de pose baseada nos dados adquiridos pelo nosso robot móvel. Foram obtidos resultados promissores. Estes mostram a importância de corresponder os pontos acompanhados nas primeiras imagens com as integrações das primeiras leituras da IMU, uma vez que erros significativos na estimação inicial de bias podem impedir a fusão de sensores e a convergência do filtro. Resultados mostram também que a operação do filtro de fusão permite sinergias entre os dois sensores. Enquanto a IMU fornece estimações instantâneas e seguras de velocidades de translação e rotação, o componente visual fornece correção à bias da IMU.

Palavras chave: Unidade de Medida Inercial (IMU), Bias, Estimação de Pose, Localização e Mapeamento Simultâneo (SLAM), Odometria Visual e Inertial (VIO), Unscented Kalman Filter (UKF), Grupos de Lie.

Abstract

Robots aware of their surrounding, and able to roam freely, are expected to become ubiquitous in future homes. Visual inertial odometry is an enabling technology which is possible nowadays due to novel low-cost cameras and inertial measurement units (IMUs). In this work we analyze how the system-state initialization and dynamics propagation of IMU readings, involving intrinsic bias, has to match the visual motion captured by the camera. Then we analyze the synergies between these two types of sensors in the context of pose estimation.

A small wheeled mobile robot is developed based on one Raspberry Pi, one camera and one IMU mounted on an Arduino. The calibration of the system, namely the rigid transformation between camera and IMU is performed with a free, public domain, tool. The visual and inertial data is processed by means of an Unscented Kalman Filter with Lie group embedding for state representation. We propose a state initialization for this filter that enables matching the integrated IMU readings with the tracked visual features.

We validate experimentally the accuracy of the IMU readings and propose an error metric to assess the pose estimation based on the datasets acquired with our mobile robot. Promising results were obtained. They show the importance of successfully matching the feature tracks in the first images with the starting integration of IMU readings, as significant errors in initial bias estimations may preclude sensors fusion and filter convergence. Results show also that the operation of the fusion filter allows synergies between the two sensors, while the IMU provides instantaneous and reliable estimations of translation and rotation speeds, the visual component provides IMU bias correction.

Keywords: Inertial Measurement Unit (IMU), Bias, Pose Estimation, Simultaneous Localization and Mapping (SLAM), Visual Inertial Odometry (VIO), Unscented Kalman Filter (UKF), Lie groups.

Contents

List of Figures

Chapter 1

Introduction

Mobile robots are seen, in general, as machines that can roam autonomously and perform many tasks. This probably originated in the countless media that portrayed even the most basic robots with these functions, and therefore molded the expectation of what a robot should be and do. Engineers understand that building a robot like this is not trivial and, more than that, it is not a low-price endeavor.

However, the vast improvements in computer processing speed and the availability of low-cost sensors such as Inertial Measurement Units (IMU) and cameras have opened the opportunity to create low-cost robots that have good performances in some of these prodded tasks. We should also note the rise of smartphones heavily influenced the price of these sensors, since most (if not all) smartphones nowadays have cameras and IMUs embedded. We believe it is possible to have robots in house that can navigate through them and perform tasks like surveillance, while only costing a small fraction of a typical salary, instead of, potentially, multiple salaries. This is the world we envision in this dissertation.

Visual inertial odometry, as compared with visual odometry, introduces inertial (IMU) information, which has the advantage of being metric motion information, a feature that a monocular camera cannot provide. The IMU specializes in measuring self-motion, being highly precise in local (small) time windows. It is often used in navigation applications like aerial vehicles. However, there are the recent low-cost IMU designs which enable the creation of low-cost navigation systems.

Combining cameras and IMUs in the context of pose estimation has multiple advantages, the main of which being the possibility of creating synergies between these sensors. In practice this means the visual information can help calibrate the IMU (bias), while the IMU provides motion scale to the visual information. With this in mind, we set out to develop a low-cost system that can perform visual inertial odometry (VIO), so we can critically analyze this synergy between sensors, and understand what are the key issues in using inexpensive equipment.

1.1 Visual Inertial Mobile Robot Navigation

Works as [Liu et al., 2014], that built navigation systems with low-cost IMUs, have shown it is critical handling properly the IMU bias. The work [Piras & Dabove, 2016] corroborates that critical aspect by showing that the difference in price is most heavily noticed in the presence of stronger bias. In general, research and development, bias is always considered an important aspect when working with IMUs, and there are multiple works like

[Scandaroli & Morin, 2011] that approach the issue of estimating and removing bias from the IMU readings, which are relevant to all IMUs independently of their price tag.

Simultaneous Location and Mapping (SLAM) is a set of problems that overlaps with the problem of pose (position and orientation) estimation. Works like [Potirakis et al., 2021] have compared the performance of visual SLAM (vSLAM) and visual inertial SLAM (viSLAM) concluded that typically viSLAM have better performances. Problems like the natural difficulty of vSLAM systems to capture motion scale and general robustness and accuracy are the main reasons.

Works as [Bo et al., 2020] have set out to implement already proven algorithms like [Qin et al., 2018] in low-cost ground robots. They show it may be necessary to compromise filter performance due to the lack of computational power of low-cost robots. In an opposite direction, of accepting computational complexity, [Qin et al., 2018] proposes a versatile monocular visual inertial state estimator by means of a nonlinear optimization-based method. This approach is based on the alignment of a visual structure obtained by means of a structure from motion strategy with the results of a IMU preintegration. This algorithm has shown to be as robust as the best available systems and it inspired some parts of our work.

These works show that it is worthwhile to pursue the idea of creating a low-cost visual inertial system dedicated to odometry. They also indicate that the process of IMU bias removal has a significant impact on the overall odometry performance. In this thesis we study inertial sensors combined with cameras, focusing on IMU bias estimation and removal, and vision / IMU synergies.

Visual inertial odometry, as a base component of mobile robots, has not only much research, but also real, interesting and relevant, datasets. The EuRoC MAV datasets [Burri et al., 2016] offer visual stereo inertial measurements with accurate motion and structure ground truth. These datasets are widely used in this area, whether to test new state of the art approaches or to establish benchmarks to compare algorithms. They will be used in this work along with our own datasets to test our solutions within a variety of scenarios.

1.2 Problem Formulation and Approach

Given a mobile robot the main objective consists of estimating its motion using onboard sensors. We consider using a camera and an IMU, and integrating the respective measurements, taking into account the lack of motion scale associated to a monocular camera and the bias of the IMU.

We set out to continue the work of [Cruz, 2015] as it developed a low-cost mobile robot based on a Raspberry Pi that is able to acquire visual information for pose estimation. As referred, we add one IMU. The IMU allows us to introduce linear acceleration and angular velocity measurements in the system, which capture motion scale. These measurements are offset by bias, which needs to be removed as to avoid erroneous estimations.

In summary, in this work we propose developing a low-cost mobile robot that can acquire visual and inertial data, and then process this data to obtain pose estimates using an Unscented Kalman Filter (UKF) based on Lie groups. While processing this data, we evaluate how the sensors cooperate in the estimation.

1.3 Thesis Structure

Chapter 1 introduces the problem to approach in the thesis, in particular presents a short discussion on low-cost visual inertial systems for pose estimation. Chapter 2 presents the engineering and mathematical background necessary to understand the concepts mentioned and/or proposed in this work. Chapter 3 describes the developed mobile robot and how it is able to acquire visual and inertial data. Chapter 4 exposes the pose estimator filter with all the proposed modifications that enables it to meet our goals. Chapter 5 provides an overview of the different experiments performed as well as the results attained and a critical analysis of these results. Lastly, Chapter 6 summarizes the work performed and highlights the main achievements in this work. Moreover, this chapter proposes further work to extend the activities described in this document.

Chapter 2

Background and State of the Art

In this chapter we cover several topics that serve as a theoretical basis for this thesis. We also mention works that inspired and served as basis of this thesis work. First, in Section 2.1, we introduce the mathematical models behind cameras. Section 2.2 focuses on Inertial Measurement Units (IMU): how they work, what they measure, and how can we model it. Then in Section 2.3, we highlight some sensor data filtering tools relevant to our goals. In Section 2.4 we introduce three sets of problems that imply localization in the context of a monocular visual system. Lastly, in Section 2.5 we highlight two state of the art visual inertial algorithms for pose estimation.

2.1 Camera Model

A camera model defines the relation between 3D points in the real world and 2D points in an image. A common model is the pinhole model (Figure 2.1) which is a perspective projection model. It builds the (2D) points projected in the image by drawing an optical ray from the (3D) world point to a small aperture in the camera (pinhole) with no lens. Extending this ray into the image plane will provide us with the projected point. This projection depends on the distance between the center of projection and the center of the image plane (principal point c), which is defined as the focal length f. These variables are part of the camera's intrinsic parameters.

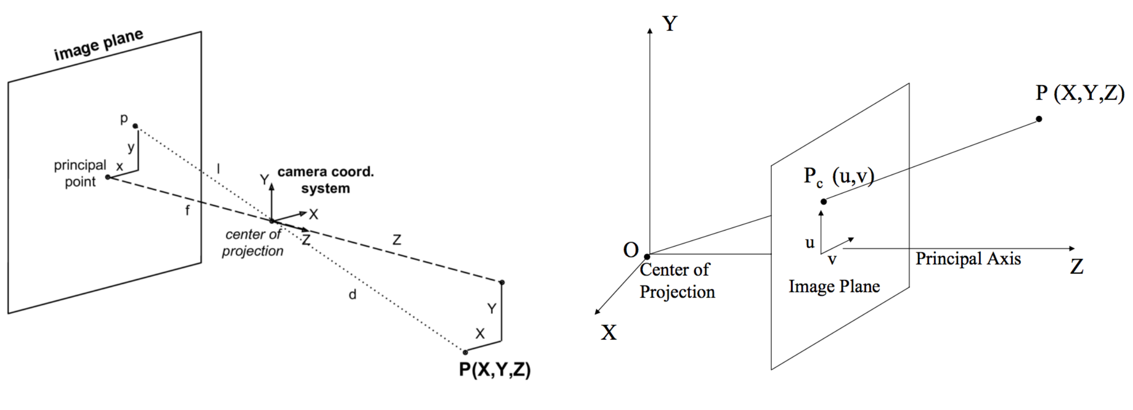


Figure 2.1: Pinhole camera model (left). Perspective projection geometry with frontal plane (right).

The model illustrated by Figure 2.1 (left) shows the image plane behind the center of projection. We use the computer vision common representation where the plane is in front of it (Figure 2.1 right). This frontal perspective projection maps the points from the real world to the image plane as

$$x = f\frac{X}{Z}, \quad y = f\frac{Y}{Z} \tag{2.1}$$

where X, Y and Z are the world coordinates, and x and y are their respective coordinates in the image.

Along with the mentioned focal length and principal point, the skew s is also an intrinsic parameters. It is non-zero if the optical axis and image plane are not orthogonal. All these variables can be grouped in one matrix. This intrinsic parameters matrix K is defined as

$$K = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.2}$$

Homogeneous coordinates can be used to model the projective matrix P, also known as the camera matrix, that maps the world points to the image plane as

$$\lambda[x\ y\ 1]^T = P[X\ Y\ Z\ 1]^T \quad \Leftrightarrow \quad \lambda\mathbf{x} = P\mathbf{X} \tag{2.3}$$

where λ is an arbitrary scale factor. P is constituted by two sets of parameters: intrinsic (2.2) and extrinsic. Extrinsic parameters are defined as the rigid transformation, with rotation R and translation t, that map the points from the world frame to the camera frame (since these frames might not be aligned). As such, we can define P as

$$P = K\begin{bmatrix} R & t \end{bmatrix}. \tag{2.4}$$

In Appendix E we detail our process to calibrate our camera.

2.2 Inertial Measurement Unit (IMU)

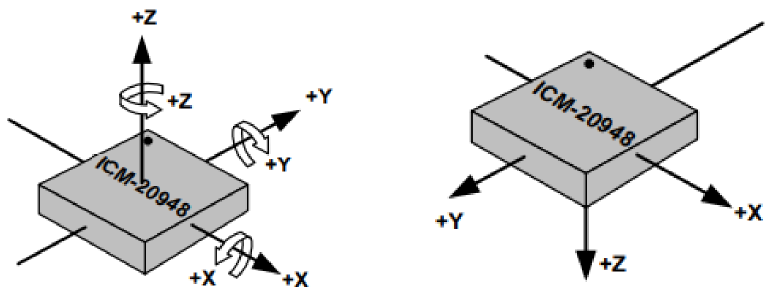
One of the goals of this work is to incorporate an Inertial Measurement Unit (IMU) into a preexisting system. To achieve this, it is important to have an understanding of how this sensor works and the measurements it provides.

Concept wise, an IMU is a combination of three sensors: an accelerometer, a gyroscope and a magnetometer. The magnetometer is not always present in the IMU, but in our case, the SparkFun 9DoF IMU Breakout ICM-20948, it is available to use. All these three sensors measure their respective properties in three spacial axes. The coordinate system of ICM-20948 for each sensor are shown in Figure 2.2. In the following we detail the properties each sensor measures and how they are represented when captured.

Accelerometer

Acceleration is perceived in relation to an observer, i.e., it is dependent on a coordinate system. For example, a stationary observer, seeing a free falling object says the object is accelerating at a rate of $9.81\,\mathrm{m/s^2}$.

An accelerometer measures proper acceleration, i.e., the acceleration felt by its body with regards to its body. In practice, it is the acceleration of a body relative to a free falling state. This means, an accelerometer in free fall



(a) Accelerometer and gyroscope (b) Magnetometer

Figure 2.2: SparkFun 9DoF IMU Breakout ICM-20948 axes orientation for the accelerometer and gyroscope (a) and for the magnetometer (b).

will measure zero acceleration for all axes. In contrast, a resting accelerometer on the surface of the Earth will measure, approximately, the gravity vector, $9.81\,\mathrm{m/s^2}$, upwards.

Instead of thinking about acceleration as the rate of change in velocity over time, it is more convenient to define acceleration relative to force and mass, $a = F/m$. This definition allows us to measure an acceleration in a moment to moment basis instead of being dependent on time.

There are several types of accelerometers. We describe here a MEMS accelerometer since it is the one present in our IMU. Figure 2.3 shows how a MEMS accelerometer works. The idea is that we have a mass connected to a spring system that allows it to move in one axis. This mass will have electrode branching out perpendicularly that will be located between two fixed electrodes. When the mass moves, the distance between the electrodes will change, thus changing the capacitance between them. By measuring the capacitance, estimates the force the mass is exerting on the springs, and consequently, estimates acceleration.

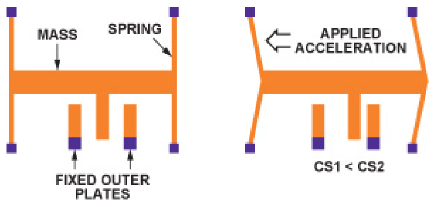


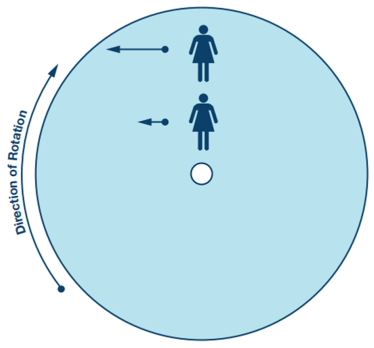
Figure 2.3: Inner working of a MEMS accelerometer.

Gyroscope

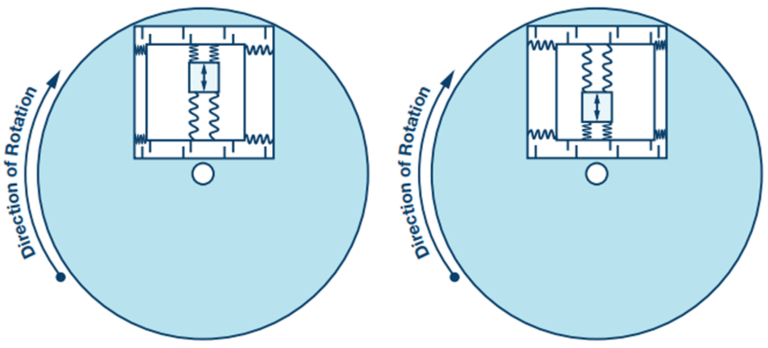
Depending on the type, a gyroscope can measure angular velocity and/or orientation. Similarly to the accelerometer, we consider a MEMS gyroscope, which measures angular velocity in degrees per second $^\circ/\mathrm{s}$ or radians per second $\mathrm{rad/s}$.

This type of gyroscopes captures angular velocity by measuring Coriolis acceleration. Let us consider a rotating platform with a standing person close to its center, as seen in Figure 2.4a. If the person leaves its current position towards the edge of the platform, its radial velocity will increase. This increase results in an increase in tangential velocity. This is visually represented by the size of the arrows on Figure 2.4a. The rate at which the person's tangential speed changes, caused by the change in radial velocity, is the Coriolis acceleration.

The MEMS gyroscope has a similar setup to that of this past scenario. As seen in Figure 2.4b, for one axis, it has a rotating platform with a resonating mass. As the mass resonates and the surface to which the gyroscope is fixed rotates, the inner block moves sideways, due to the Coriolis effect. Using electrodes in both blocks, we can derive a signal from the changes in capacitance. As the rate of rotation increases, so does the displacement of the mass, which in turn, changes capacitance, and therefore, the signal. This signal will be used to determine the angular velocity of the gyroscope.



(a) Example of Coriolis acceleration



(b) MEMS gyroscope's inner working

Figure 2.4: Visualizations of how MEMS gyroscopes capture angular velocity. From [Passaro et al., 2017].

Magnetometer

Magnetometers use the Hall Effect to measure magnetic fields. Let us consider a conductive plate with current flowing through it, as shown in Figure 2.5. If we place a magnetic field near the plate, the positive and negative charges will go to the edges of the plate perpendicularly to the wires. Assuming the field is not parallel to the current. This effect generates a potential difference, Hall voltage, that is proportional to the magnetic field. Thus, by evaluating this voltage and the current distortion, a magnetometer is able to measure a magnetic field's magnitude and orientation.

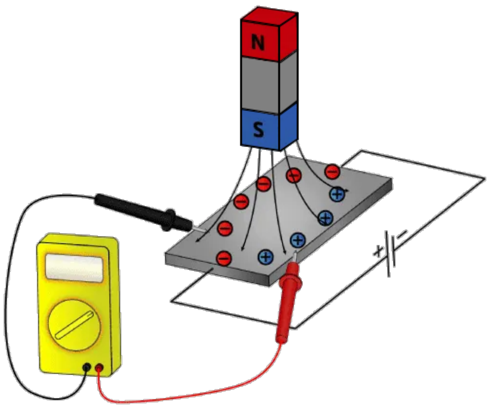
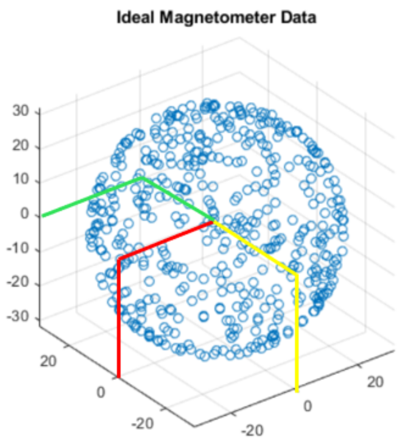
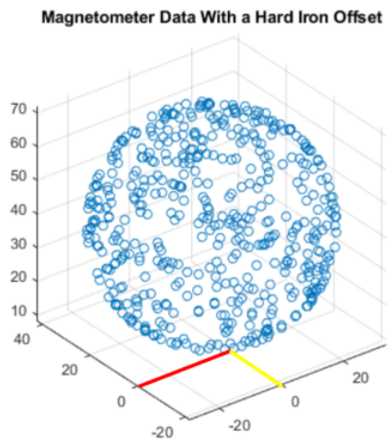


Figure 2.5: Example of Hall Effect.

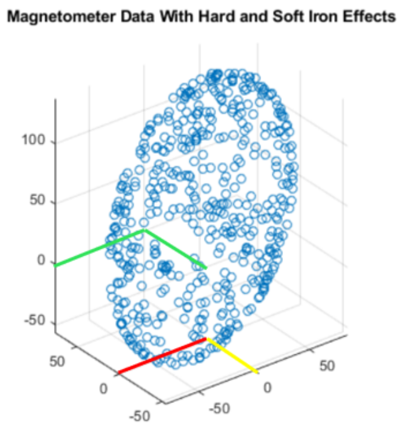
Typically magnetometers are used to measure Earth's magnetic field. However magnetometers are susceptible



(a) True values



(b) Hard iron effect



(c) Hard and soft iron effects

Figure 2.6: Representation of a magnetometer's data measurements when rotated in all possible orientations with: no disturbances (a); hard iron disturbances (b) and, hard and soft iron disturbances (c).

to disturbances. These can take the form of hard and soft iron effects. The first is caused by magnetic materials like magnets or electrified coils. The second is caused by ferromagnetic materials like iron, cobalt and nickel. If we rotated a magnetometer through all possible attitudes (orientation) and measured the true magnetometer data we should get vectorial measures contained within a sphere centered in the origin (magnetometer center) with each measure having the same magnitude, like shown in Figure 2.6a. Figure 2.6b shows the effect of hard iron sources. They will cause the measures to be offset (bias) from the origin, that is, the magnetometer. Soft iron sources will bend the measured magnetic field. Figure 2.6c shows both effects on magnetometer readings simultaneously.

Not much can be done to prevent these disturbances if they are external to the sensor's system or introduced occasionally. However, if the sources are part of the system and rotate with it, it is possible to calibrate them out. In practice this means finding parameters such that we can map Figure 2.6c into Figure 2.6a. These will be matrix A for soft iron distortion and vector b for hard iron distortion. We can then correct each measure by applying

$$m_{calib} = A(m - b). \tag{2.5}$$

Measurement Model

Having detailed how the sensors in an IMU work and their measurements, one may introduce the mathematical models of the measurements. We will first introduce bias and noise and how they effect estimations.

Noise and Bias Every sensor is projected to measure the true magnitude of a property. However, due to multiple reasons like temperature, calibration, disturbances, among others, measured signals can suffer unwanted modifications during the entire acquisition and processing stages. Different sensors are affected by different types of phenomenons. We will focus on the two specific types of phenomenons that often cause issues to IMU sensors: (white) noise and bias.

White noise, i.e., a random signal with equal intensity at different frequencies is considered additive to the sensor measurements. From hereon we will short white noise to just noise. Bias is an offset from the real value the signal should have. This property is more often associated with IMU sensors and usually is specified by the manufacturers as bias stability, which represents how stable the bias is over a certain period of time. This is because, after we process the gyroscope readings (usually by integrating), the bias will generate a drift from the mean value obtain, and therefore the bias stability will tell us how much drift there will be over time. Figure 2.7 shows a data signal (for example the gyroscope's angular rate) and what happens to it when affected by noise and bias.

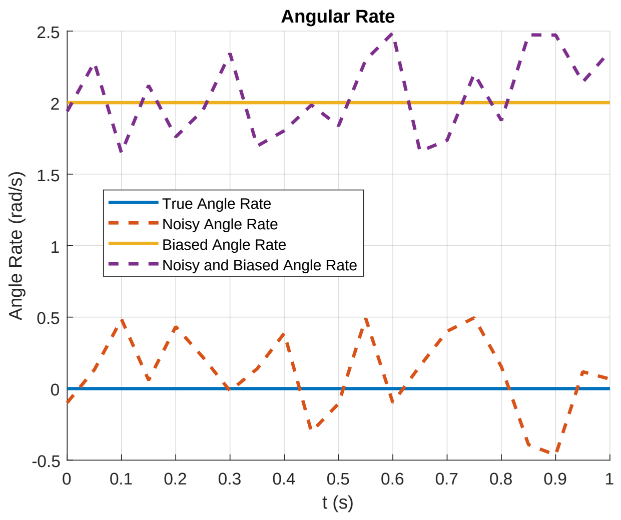


Figure 2.7: Graphical representation of noise and biased sensor data.

IMU Sensors Each one of the IMU sensors measures a vector with each value in it corresponding to the magnitude of that measurement for one 3D axis:

$$\begin{aligned} \omega &= [\omega_x \; \omega_y \; \omega_z]^T, \\ a &= [a_x \; a_y \; a_z]^T, \\ m &= [m_x \; m_y \; m_z]^T, \end{aligned} \tag{2.6}$$

with ω as the angular velocity, a as linear acceleration, m as the magnetic field.

Mathematically, both noise and bias can be modelled as two additive signals to the true signal. They will both be present for each sensor's output. For the gyroscope's output, these two disturbances, b_ω and n_ω, will be the only ones added to the real angular velocity. As for the accelerometer, we also need to consider the gravitational acceleration g along with the bias b_a and noise n_a. Finally for the magnetometer we need to model the noise n_m along with the hard and soft iron distortions A and b. The system's measurement model equations are

$$\begin{cases} \omega = \omega_{true} + b_\omega + n_\omega \\ a = a_{true} + g + b_a + n_a \\ m = A^{-1} m_{true} + b + n_m \end{cases} . \tag{2.7}$$

Data Integration In a system equipped with an IMU, the reported data tends to be integrated in order to estimate variables like orientation or velocity. Considering orientation estimation with IMU reading containing bias and noise (as shown in Figure 2.7), the integration of the measurements (as modeled in Equations (2.7)) will result in a drift over time of the real orientation value, as shown in Figure 2.8. The only way to combat this is to estimate the bias value at each time step and remove it from the measurements. In Section 4.2 we propose a method to estimate this value.

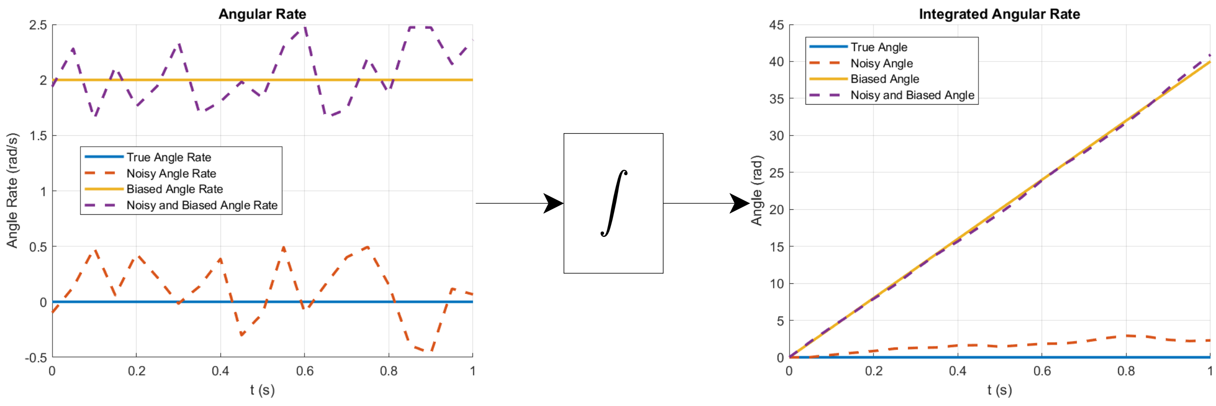


Figure 2.8: Graphical representation of integration of biased angular velocity measurements and the resulting drift.

2.3 Sensor Data Filtering Tools

As discussed in the previous sections, sensor signals usually suffer from unwanted noise and disturbances. To better handle them, and avoid their negative impact on estimations, many filtering tools have been developed in the last decades, such as the Kalman filter. The Kalman filter allows statistical sensor fusion with noise modeling, by estimating the optimal weighting and combination of the data sources.

Kalman Filters

Since it was introduced in [Kalman, 1960], the Kalman Filter has dominated all problems with linear systems due to being a statistically optimal filter. In a scenario of IMU sensor fusion where we can estimate orientation separately with an accelerometer and magnetometer, and with a gyroscope (see Appendix C), this would mean the filter calculates the perfect weight factor between the two orientation estimates every time. However, because the IMU system is not linear, it is not possible to use the traditional Kalman Filter. The next logical step is to propose linearizing the system model, i.e., using an Extended Kalman Filter (EKF).

The EKF, as shown for example in [Thrun et al., 2005], is an extension of the KF that can handle nonlinear systems. The main assumption behind the KF is that all models are linear, which leads to Gaussian probability distributions. The EKF solves this problem by locally linearizing the function with its first order derivative. Although this is a simple extension to the KF which allows its usage with nonlinear systems, it is still sub-optimal since it does not capture the full system model and the noise on the system.

Typically, the most crucial part of working with this family of filters is how we handle the process and measurement noises, which in our context corresponds to the gyroscope propagation and the accelerometer correction, respectively. The covariance of these parameters represents how much noise we consider to be present in our predictions and measurements. These parameters have an impact on the estimation of the weight that will balance the intermediate estimates, which means they can be interpreted as trust in a estimate/measurement. Often these parameters can be the difference between the filter converging or not.

Although EKF can be an adequate solution, an alternative that typically performs better is the Unscented Kalman Filter (UKF). In Appendix A we give a comprehensive explanation about all the Kalman Filters mentioned so far and detail the UKF.

Special Groups

In robotics, it has been long known the importance of Lie groups (see e.g. [Park et al., 1995]) like $SO(3)$ and $SE(3)$ in the context of orientation and rotations, and space of poses, respectively. More recently, probability distributions on $SE(3)$, and their role for control and estimation, have been well studied (see e.g. [Zefran et al., 1999]). We will now formally introduce these two mentioned Lie groups.

The Special Orthogonal Group $SO(3)$ This is the Lie group that includes all three-dimensional rotations. Each element of this groups is a 3×3 matrix. We can formally define the elements of $SO(3)$ as $SO(3) := \left\{\mathbf{R} \in \mathbb{R}^{3\times 3} \,:\, \mathbf{R}\mathbf{R}^T = \mathbf{I}_3, det(\mathbf{R}) = 1\right\}$, where $\mathbf{I}_3$ is the 3×3 identity matrix. The operator in this Lie group is the matrix multiplication. It allows us to combine two rotations into a single one. As this previous definition states, the identity element is the identity matrix $\mathbf{I}_3$, and an inverse element can be obtained by means of the transpose operator.

Each Lie group is associated with a Lie Algebra: a unique tangent vector space at the identity element the Lie group obtained by means of differentiation. A simple case can be used to show how one arrives at the Lie Algebra from this Lie group $SO(3)$. Starting with the orthogonal condition, $\dot{\left(\mathbf{R}\mathbf{R}^T\right)} = \dot{\left(\mathbf{I}_3\right)}$ and differentiating it over time we get $\dot{\mathbf{R}}\mathbf{R}^T + \mathbf{R}\dot{\mathbf{R}}^T = \mathbf{0}$. By applying the chain rule $B^T A^T = (AB)^T$ we get $\dot{\mathbf{R}}\mathbf{R}^T = -(\dot{\mathbf{R}}\mathbf{R}^T)^T$. This describes a skew-symmetric matrix which can be represented by the operator $[.]_\times$ over a 3D vector $\omega = [\omega_x\ \omega_y\ \omega_z]^T \in \mathbb{R}^3$ as

$$\omega^\wedge = [\omega]_\times = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} \in \mathfrak{so}(3) \tag{2.8}$$

This operation represents the transformation from $\mathbb{R}^3$ to the Lie Algebra $\mathfrak{so}(3)$ associated with the Lie group $SO(3)$. For a generic Lie algebra, the operator that represents the transformation from a vector space to the Lie algebra, is the same $(.)^\wedge$, but its implementation will be different.

If we consider a matrix $[\omega]_\times \in \mathfrak{so}(3)$, then the rotation matrix associated with this skew-symmetric matrix can be defined as $\exp([\omega]_\times) \in SO(3)$. The exponential of a matrix can be computed by definition. For a generic matrix A, the matrix exponential is an infinite series that can be written as $\exp(A) = \mathbf{I} + A + \frac{1}{2}A^2 + \frac{1}{3!}A^3 + \cdots = \sum_{n=0}^{\infty} \frac{1}{n!}A^n$. The Hamilton-Cayley theorem states the matrices satisfy their own characteristic equation. The infinite series is actually equal to a polynomial expression with dimension equal to the matrix's minus one. Going back to our specific Lie Algebra, we can finally define the exponential mapping from $\mathfrak{so}(3)$ to $SO(3)$, by means of the Rodrigues' Formula, as $\mathbf{R} = \exp([\omega]_\times) = \mathbf{I}_3 + \frac{\sin(||\omega||)}{||\omega||} + \frac{1-\cos(||\omega||)}{||\omega||^2}[\omega]_\times^2$.

If we instead want to map an element from $SO(3)$ to $\mathfrak{so}(3)$, we need to perform the logarithmic map. First we need to calculate the angle of rotation, θ, from the trace of the rotation matrix: $\theta = \arccos\left(\frac{\mathrm{Tr}(\mathbf{R})-1}{2}\right)$. With this value we can calculate the skew-symmetric matrix as $[\omega]_\times = \log R = \dfrac{\theta}{2\sin\theta}\left(\mathbf{R} - \mathbf{R}^\mathsf{T}\right)$.

Converting this matrix back to a 3D vector gives us a 3D vector ω. These values are the axis-angle representation of the original rotation matrix. This mean we can use this exponential and logarithmic mapping to convert between these rotation representations.

A final note about rotations is that, because the Lie algebra results of the differentiation of the Lie group, the transformation from the Lie algebra to the Lie group represents an integration. This means that if we transform an angular velocity vector $\omega \in \mathbb{R}^3$ to an element of the Lie group $SO(3)$, we are obtaining the rotation performed over the considered time frame.

The Special Euclidean Groups $SE(3)$ The elements of this group can be defined as $SE(3) := \left\{ \chi = \begin{bmatrix} \mathbf{R} & \mathbf{p} \\ \mathbf{0} & 1 \end{bmatrix} \in \mathbb{R}^{4\times4} \ : \ \mathbf{R} \in SO(3),\ \mathbf{p} \in \mathbb{R}^3 \right\}$ where $\mathbf{0}$ denotes a 1×3 zeros vector. These elements are named homogeneous transformation matrices, and they represent rigid transformations. These transformations are defined by a rotation $\mathbf{R}$ and a translation $\mathbf{p}$. The operator of this group is also the matrix multiplication. Combining elements with it allows us to stack rigid transformations.

These concepts are applied in robotics applications as an operator of displacement or a coordinate system. As for the Lie Algebra, which again is the result of differentiating the Lie group over time, we can consider its elements to be angular and linear velocities. Later in this chapter (Section 2.5.2), we will detail a filter based on Lie theory. Particularly, using an extension of this last group to represent part of the state space.

2.4 Visual Odometry, SfM and SLAM

There are several engineering problems that require localization. More specifically we will consider now a monocular visual setup. It is relevant to define each set of problems and how they overlap as to see how the solutions to one set of problems might be interesting to another. We will now look at three of the most important sets of problems within this context: Visual Odometry, Structure from Motion and Simultaneous Location and Mapping.

Visual Odometry

Odometry is the use of methods to estimate the position of an agent over time. Visual Odometry (VO) means the special case of performing this task using visual information. Typically this also implies estimating the orientation, which when combined with the position forms the pose, and both linear and angular velocities that facilitate these estimations. It is usually performed by tracking specific points in a camera frame (features). It is common to use Kalman Filters to engage this problem, modeling them as online problems, i.e., problems where the estimation and acquisition occur in parallel.

Structure from Motion

Another set of problems related to VO is Structure from Motion (SfM). These problems focus on reconstructing the 3D environment from 2D visual information. In practice, this reconstruction means estimating the rigid transformations that consistently map each camera frame in a considered world frame, as done in, for example, [Fitzgibbon & Zisserman, 1998]. SfM problems are typically formulated as offline problems, i.e., the data is captured first and the estimations are only performed afterwards, which prevents these problems of making estimations in real time.

Simultaneous Location and Mapping

Visual Odometry is an important concept in the context of Simultaneous Location and Mapping (SLAM), because, as the name implies, part of SLAM problems is to estimate position of an agent. Specifically visual SLAM (vSLAM [Karlsson et al., 2005]) when only using visual information to accomplish this. The main goal here is to estimate the rigid transformation (3D position and rotation) that maps each camera frame into the world frame, while building a map of 3D points.

As originally proposed in [Davison, 2003], this EKF based SLAM consists in iteratively estimating a set of variables of interest (state space), namely, the position, orientation, and linear and angular velocities, all regarding the world frame. It estimates these values by predicting the next state using a standard physical dynamical model and then making a correction based on measured features from the visual information. This is one of the most popular approaches to perform SLAM or VO.

However there are some issues to consider, most prominent of which, the fact all the estimated variables are not in scale with the real world. Meaning, if the filter estimated that the motion caused a displacement with a certain norm, we would have no way of knowing how this norm (distance) could be converted to real world units. This is because motion scale is unobservable for a monocular setup.

There are many possible approaches when faced with the prospect of estimating scale factor. In most monocular systems this variable needs special procedures in order to be estimated. For example, one can place an object with known size or give the algorithm a known distance to a captured object (for example [Lothe et al., 2010]). Another possibility is to make the mobile robot move a known, fixed distance and calibrate scale from that movement (as performed in [Cruz, 2015]). However, that is not the case for most Visual Inertial Odometry (VIO) methods. The inertial data can observe motion scale, and therefore, there is no need for specific procedures for its estimation. In some of these methods we can estimate the scale factor value explicitly, while in others the position estimates will be up to scale by the nature of the approach.

Algorithms such as the one developed in [Qin et al., 2018] solve the SLAM problem by performing a alignment between visual data and inertial data. The algorithm starts by solving vision-only SfM problem. This is the position and orientation for all frames, plus the detected features positions. All of this will be up to a scale factor that will be unknown at this point. Next, the algorithm performs a preintegration of the inertial information. This means, it estimates position, linear velocity and orientation differences between each image frame using the inertial data. Finally it aligns both estimations by means of optimization techniques, in order to estimate initializing values for gyroscope bias, linear velocity, gravity vector and metric scale.

An example of a filter where the scale factor is implicit in the estimated results is [Brossard et al., 2017]. The prediction step of this filter is just based on inertial information, which is metrically scaled, and the update step is based on the 3D positions of landmarks, which are also metrically scaled. The reason everything is already scaled to begin with is that the visual information is only being used to track features over the frames, and these features will only serve to evaluate the estimations of the already initialized landmarks.

2.5 State of the Art

Let us now highlight two state of the art works that perform visual inertial odometry (among other tasks): ORB-SLAM3 [Campos et al., 2020] and Fusion18 [Brossard et al., 2017]. The first is noticeable due to its robustness, versatility and accuracy while the latter proposes a promising filter structure based on Lie Groups.

2.5.1 Versatile Open-Source VIO

ORB-SLAM3 [Campos et al., 2020], which was released as a publicly available library during the development of this thesis, is state of the art in performing SLAM. This library can perform Visual, Visual Inertial and Multi Map SLAM. Multi Map SLAM consists in aligning multiple 3D maps estimated using different methods. The cameras this library supports are monocular, stereo and RGBD cameras, and it is also possible to choose between pinhole and fisheye lens models. Experimentally, ORB-SLAM3 is shown to be as robust as the best available systems and significantly more accurate. A visual overview of ORB-SLAM3 can be found in Figure 2.9.

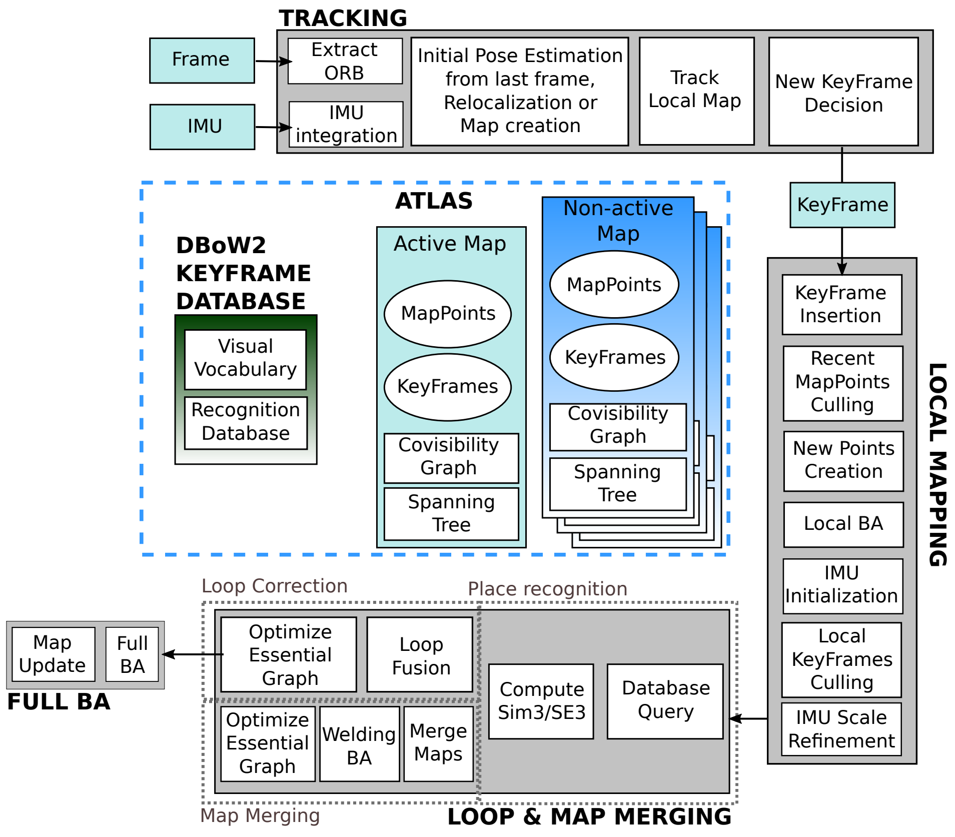


Figure 2.9: Main system components of ORB-SLAM3. From [Campos et al., 2020]

In terms of visual inertial SLAM, ORB-SLAM3 integrates the work of [Mur-Artal & Tardos, 2017b]. It introduces a system able to reuse a map with short, mid and long-term data association, using them in a local Bundle Adjustment (BA) based on IMU preintegration. However, it has a slow IMU initialization. As we will see later in this thesis, this can greatly harm robustness and accuracy. ORB-SLAM3 builds upon this system by proposing a fast initialization based on Maximum-a-Posteriori (MAP) estimation that takes visual and inertial sensor uncertainties into account.

2.5.2 UKF on Lie Groups for VIO

The filter proposed by [Brossard et al., 2017] is the basis for our work. First we will go into detail about the system models while also providing some Lie theory necessary for a complete comprehension of these models and the filter. Then we will focus on the UKF filter itself.

State Space This work uses the Lie group $\chi \in SE_{2+p}(3)$ which is an extension of $SE(3)$. It is used to represent the state space, which contains all the variables of interest that we want the filter to estimate. It containes the position $\mathbf{x} \in \mathbb{R}^3$, orientation $\mathbf{R} \in SO(3)$ and linear velocity $\mathbf{v} \in \mathbb{R}^3$ of the body, along with 3D positions of landmarks $\mathbf{p}_1, \ldots, \mathbf{p}_p \in \mathbb{R}^3$ detected in the scene. All theses variables are in the world frame. The state is formed by a square matrix χ with dimensions $(5+p) \times (5+p)$:

$$\chi = \begin{bmatrix} \mathbf{R} & \mathbf{v} & \mathbf{x} & \mathbf{p}_1 \cdots \mathbf{p}_p \\ \mathbf{0}_{(p+2)\times 3} & & \mathbf{I}_{(p+2)\times(p+2)} & \end{bmatrix}. \tag{2.9}$$

Works, such as [Park et al., 1995], noticed that using Lie groups to represent the state space had certain advantages. Using this for SLAM was properly introduced in [Barrau & Bonnabel, 2015]. This work showed that a Lie group based EKF (which was entitled Invariant EKF) reveals a greater numerical consistency when compared to a standard EKF. This representation also allows for a more compact and sturdy mathematical formulation that does not require variable conversions, which often leads to a decrease in accuracy and numerical consistency. This is particularly relevant for orientation as it has multiple forms of representation with different benefits, like quaternions, rotation matrices, angle-axis, and others.

Regarding the operation that defines this group, it represents a propagation. This means that if a system suffers a change χ in relation to the prior state χ_{t-1}, we can obtain the new system state as $\chi_t = \chi * \chi_{t-1}$. Because we are working with groups, we also have the possibility of reverting the state. This is because each element of this group must have an inverse, which means we can apply the inverse of a change in the system so we can revert the system state.

We need to consider the IMU bias that is also part of the state. We can define the bias vector $b \in \mathbb{R}^6$ as

$$b = \begin{bmatrix} b_\omega^T & b_a^T \end{bmatrix}^T \tag{2.10}$$

with the accelerometer bias b_a and gyroscope bias b_ω. Finally, the state of the filter is expressed as the tuple (χ, b).

Dynamic Model

The considered dynamic model assumes a flying body navigating on flat earth equipped with an IMU. We can model the system as

$$\text{body state} \begin{cases} \dot{\mathbf{R}} = \mathbf{R}(\omega - b_\omega + n_\omega)_\times \\ \dot{\mathbf{v}} = \mathbf{R}(a - b_a + n_a) - g \\ \dot{\mathbf{x}} = \mathbf{v} \end{cases}, \tag{2.11}$$

$$\text{IMU biases} \begin{cases} \dot{b_\omega} = n_{b_\omega} \\ \dot{b_a} = n_{b_a} \end{cases} , \tag{2.12}$$

$$\text{landmarks} \left\{ \dot{\mathbf{p}}_i = 0, i = 1, \ldots, p \right. , \tag{2.13}$$

where $(\omega)_\times$ portrays the skew-symmetric matrix related with the cross product with vector $\omega \in \mathbb{R}^3$ (as shown in Equation 2.8). Multiple noises are grouped as

$$n = \begin{bmatrix} n_\omega^T & n_a^T & n_{b_\omega}^T & n_{b_a}^T \end{bmatrix}^T \sim \mathcal{N}(0, Q). \tag{2.14}$$

Measurement Model

Along with the IMU, a visual inertial system also considers a calibrated monocular camera that provides visual information and observes and tracks p landmarks in the visual scene. The camera observes landmark $\mathbf{p}_i$ through the standard pinhole model and respective projection model (Section 2.1) as

$$\mathbf{y}_i = \begin{bmatrix} y_u^i \\ y_v^i \end{bmatrix} + n_{\mathbf{y}}^i, \tag{2.15}$$

where y_i is the result of projection:

$$\lambda \begin{bmatrix} y_u^i \\ y_v^i \\ 1 \end{bmatrix} = K[\mathbf{R}_{B\to C}^T\,(\mathbf{R}^T\,(\mathbf{p}_i - \mathbf{x}) - \mathbf{t}_{B\to C})], \tag{2.16}$$

with λ as the scale factor. This is a transformation from frame W to current image plane using the state's pose, $\mathbf{R}$ and $\mathbf{x}$ ($W \to B$, Body/IMU frame), the IMU-camera extrinsic parameters, $\mathbf{R}_{B\to C}$ and $\mathbf{t}_{B\to C}$ ($B \to C$, Camera frame), and the camera intrinsic parameters, K ($C \to$ image plane).

The filter will compare this projection to the expected position of the landmark in the image. This process will invalidate the 3D landmarks that are too distant to the respective expected 2D feature.

Camera-IMU Calibration

One of the objectives of this work is to achieve a system that can estimate position and orientation. For every source of data or estimated value there is a reference point, expressed over Cartesian coordinates.

Each sensor provides data relative to their own pose. For each image collected there is a camera pose. In order to estimate all camera poses for a sequence of images we must first establish a world referential for these poses. For example, is usually assumed the first image is aligned with the world frame. From there we need to estimate the position and rotation for each image relative to this referential frame.

Let us now consider the accelerometer and gyroscope measurements: linear acceleration and angular velocity, respectively, as depicted in Equation 2.6). Both of these measurements are related to an internal frame defined by the IMU (see Figure 2.2a). Similarly to the camera, we need to estimate a transformation that will map the IMU values to the world frame.

A frame transformation implies a rotation that aligns both frame's axis and a translation that aligns the centers. This displacement can be grouped into what is called a rigid transformation, and it can be represented as

$$T(x) = \mathbf{R}x + \mathbf{t}, \tag{2.17}$$

with $x \in \mathbb{R}^3$ as the point to transform, $\mathbf{R} \in SO(3)$ as a rotation matrix, $\mathbf{t} \in \mathbb{R}^3$ as a translation vector and $T(x) \in \mathbb{R}^3$ as the transformed point. As long as we have the $\mathbf{R}$ and $\mathbf{t}$ pair, we can transform measurements from one sensor's frame to another.

Unscented Kalman Filter

The Unscented Kalman Filter has been chosen by [Brossard et al., 2017] instead of an EKF in order to consider a nonlinear system. UKF's unscented transformation tends to perform better than the EKF's linearization. Fusion18 does not use the traditional UKF, it uses a variation: the Square-Root Unscented Kalman Filter. The major benefits in this choice are the increase in numerical properties and computational performance. The SR-UKF achieves this by propagating the square-root of the state's covariance, instead of the covariance itself. In Section A we present these filters with more detail.

We will focus on one of the two proposed Lie groups based filters in [Brossard et al., 2017]: the Right-UKF-LG (or as we and the authors have been calling it, Fusion18). Now that we know the system forms a Lie group, the next logical step is to establish how the respective Lie algebra is incorporated in this filter. This involves linearization and discretization of the system's dynamics. The discrete system evolves over time as

$$\chi_{n+1} = f(\chi_n, u_n, w_m) \tag{2.18}$$

where χ_n is the current state represented as a Lie group (2.9), with time instant n, the input variable $u_n = [w_n^T \, a_n^T]^T$ containing the gyroscope and accelerometer measurements, and the white Gaussian noise $w_n \sim \mathcal{N}(0, \mathbf{Q}_n)$. f represents the discretized dynamics model. This system also has a discrete measurement represented as

$$y_n = h(\chi_n, v_n) \tag{2.19}$$

with white Gaussian noise $v_n \sim \mathcal{N}(0, \mathbf{R}_n)$.

Time Discretization

Using the Euler method one may discretize Equations (2.11) to (2.13), not including rotation, for a small time step Δt:

$$\begin{aligned}
&\mathbf{R}_{t+\Delta t} = \mathbf{R}_t \mathbf{exp}\left[(\omega_t - b_{\omega,t})\Delta t + \mathbf{Cov}(n_\omega)^{1/2} g\sqrt{\Delta t}\right]_\times \\
&\mathbf{v}_{t+\Delta t} = \mathbf{v}_t + (\mathbf{R}_t(a_t - b_{a,t}) - g)\Delta t \\
&\mathbf{x}_{t+\Delta t} = \mathbf{x}_t + \mathbf{v}_t \Delta t \\
&b_{\omega,t+\Delta t} = b_{\omega,t} \\
&b_{a,t+\Delta t} = b_{a,t} \\
&\mathbf{p}_{i,t+\Delta t} = \mathbf{p}_{i,t}
\end{aligned} \quad . \tag{2.20}$$

Uncertainty on Lie Groups

Due to the specific state representation (2.9), we cannot characterize uncertainty with the typical additive noise approach. Uncertainty is defined as in [Barfoot & Furgale, 2014]. The probability distribution $\chi \sim \mathcal{N}_{\mathcal{R}}(\bar{\chi}, \mathbf{P})$ is characterized as

$$\chi = \mathbf{Exp}(\xi)\bar{\chi},\ \xi \sim \mathcal{N}(0, \mathbf{P}) \tag{2.21}$$

with

$$\mathbf{Exp}(\xi) = \mathbf{exp}_m(\xi^\wedge), \tag{2.22}$$

where $\mathbf{Exp}$ represents the complete transformation from a vector space to the Lie group, $\mathbf{exp}_m$ represent the exponential mapping from the Lie algebra to the Lie group and $(.)^\wedge$ represents the transformation from the vector space to the Lie algebra.

This last transformation was shown in Section 2.3 (more specifically Equation 2.8) for $SO(3)$. However, for this Lie group extension, although similar, the transformation requires a more complex definition (as provided by [Barfoot & Furgale, 2014] and [Barrau & Bonnabel, 2018]). The uncertainty ξ is mapped to our state by performing an exponential mapping. We can map uncertainty $\xi = [\xi_{\mathbf{R}}^T\ \xi_{\mathbf{v}}^T\ \xi_{\mathbf{x}}^T\ \xi_{\mathbf{p}_1}^T \cdots \xi_{\mathbf{p}_p}^T]^T$ to the Lie algebra through the transformation $\xi \rightarrow \xi^\wedge$:

$$\xi^\wedge = \begin{bmatrix} (\xi_{\mathbf{R}})_\times\ \xi_{\mathbf{v}}\ \xi_{\mathbf{x}}\ \xi_{\mathbf{p}_1}^T\ \cdots\ \xi_{\mathbf{p}_p}^T \\ \mathbf{0}_{(2+p)\times(5+p)} \end{bmatrix}. \tag{2.23}$$

Filter Architecture

Finally, the filter's architecture can be presented as

$$\text{state} \begin{cases} \chi = \mathbf{exp}(\xi)\bar{\chi} \\ b_n = \bar{b}_n + \tilde{b} \end{cases}, \begin{bmatrix} \xi \\ \tilde{b} \end{bmatrix} \sim \mathcal{N}(0, \mathbf{P}_n), \tag{2.24}$$

$$\text{dynamics} \Big\{ \chi_n, b_n = f(\chi_{n-1}, u_n - b_{n-1}, n_n) \quad , \tag{2.25}$$

$$\text{observations} \begin{cases} \mathbf{Y}_n = [\mathbf{y}_1^T\ \cdots\ \mathbf{y}_p^T] := \mathbf{Y}(\chi_n, w_n) \\ \mathbf{y}_i \text{ given in } (2.15), i = 1, \ldots, p \end{cases}, \tag{2.26}$$

with $(\bar{\chi}_n, \bar{b}_n) \in \mathbb{R}^{(15+3p)}$ as the mean estimate of the state at a time instance n, $\mathbf{P}_n \in \mathbb{R}^{(15+3p)\times(15+3p)}$ as the covariance matrix that represent the state's uncertainties $(\xi, \tilde{b})$, and vector $\mathbf{Y}_n$ that contains the observations of the p landmarks with associated Gaussian noise $w_n \sim \mathcal{N}(0, \mathbf{W})$.

Chapter 3

Mobile Robot and System Architecture

In this chapter we will build upon the hardware and software setup of [Cruz, 2015]. This previous work provided us with a mobile car capable of live streaming video feed from its camera, and a PC user interface capable of giving movement commands to the robot and acquiring images from the video feed. Along with maintaining (and upgrading) the previous system's features, our two main objectives are: integrate an IMU in the system and enable the acquisition of formatted visual-inertial datasets.

First we will see how our proposed global system is structured and what are its components (Section 3.1). Then we will go into detail about the components of this system, following the data flow. This means starting with the mobile robot (Section 3.2) and then going to the PC interface (Section 3.3). This chapter focuses more on capturing datasets, leaving the next one to talk about processing them. Finally we will explain how we estimated the ground truth of the acquired datasets (Section 3.4).

3.1 Global System

In terms of hardware, this system can be divided in two main parts that serve distinct purposes: the mobile car and the user's PC. The first has the job to acquire visual-inertial data and make it available, while the latter has to be able to receive the data, format it and store it locally. To achieve this we propose the system represented in Figure 3.1.

This is a system that allows for multiple users to connect to the mobile robot at one, assuming a LAN layout. This means the robot can provide data for multiple users if necessary, and it can achieve this without jeopardizing the sample rates of acquisition due to reasons we will explain later in this chapter.

3.2 Mobile Robot

Our starting point was a mobile robot that had a Raspberry Pi Model B (R2) as the central processing component, a Pi Board Camera for image acquisition, a DRV8833 Dual Motor Driver Carrier with Multi-Chassis 4WD Kit (4 DC Motors) to enable movement and a USB battery box 4x18650 Coolook PB-2000 to power the system. With this setup, our first objective was to incorporate an IMU sensor. We used a SparkFun 9DoF (Degrees of Freedom) IMU Breakout ICM-20948 (Qwiic) SEN-15335 along with an Arduino Mega 2560 R3 that acted as middle man

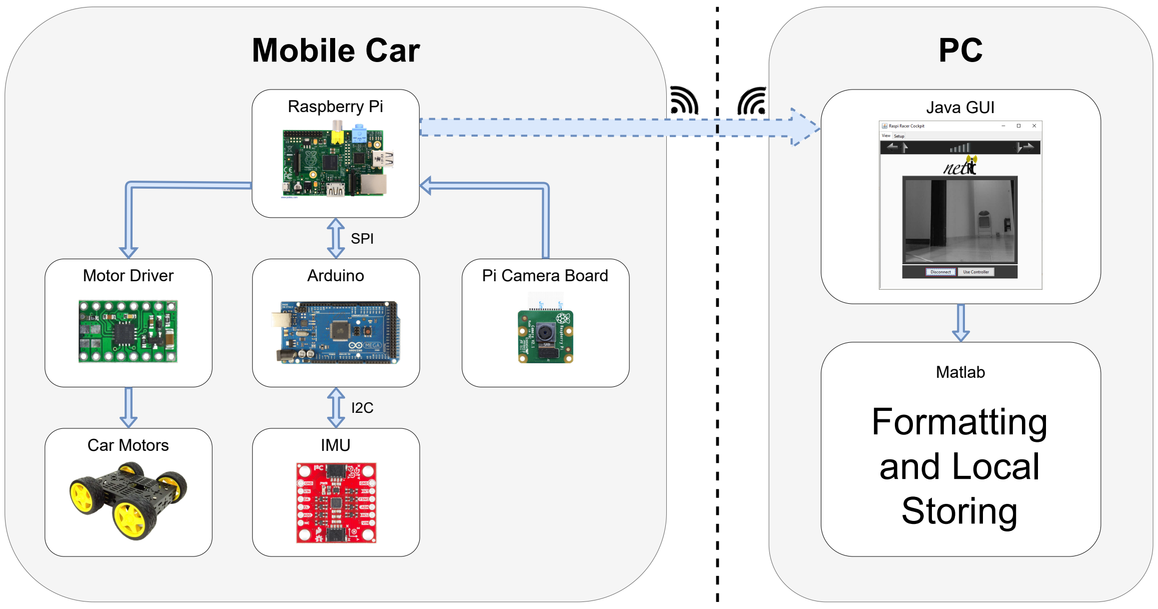


Figure 3.1: Acquisition system architecture.

between the Raspberry Pi and the IMU.

With this in mind, we propose the system depicted in Figure 3.2. All connections are straight forwards according to the communication protocols available (which we will mention bellow). Something worth mentioning is how each component is powered, since some power configuration might enable the system to turn on, despite it being unable to function properly. The 9 V power supply will power the Raspberry Pi and the DRV8833. The Raspberry Pi will in turn power the Arduino with 5 V, which in turn will power the IMU with 3.3 V. Another useful configuration is to power up the Arduino through its USB B slot while not using the 9 V power supply. This allows us to use the Arduino and IMU while letting the rest of the system stay idle (literally, since the Raspberry Pi will only have enough power to stay in an idle mode).

In Figure 3.3 we can see the achieved mobile robot along with each sensor's coordinate system. This car has a 3D printed platform on which is has the Raspberry Pi and motor driver. We had to incorporate a cardboard box since we had no where to place the Arduino. However we made sure the IMU stayed in the printed platform since it oscillation less than the cardboard box when the robot is moving.

Now let us go through the main components of the car: what they do and how they interact with other components or outside elements.

IMU and Arduino

First let us introduce our IMU. In this work we will consider a SparkFun 9DoF IMU Breakout SEN-15335 with a Digital Motion Processor (DMP) ICM-20948. This was the chosen IMU due to its price-quality ratio and its DMP. This IMU allows us to get raw values from its sensors - accelerometer, gyroscope and magnetometer - but with the help of the DMP we can obtain processed values, like calibrated angular rate and 9DoF orientation quaternions.

A disadvantage of this IMU is the fact it is fairly recent and access to its integrated DMP is not straight forward.

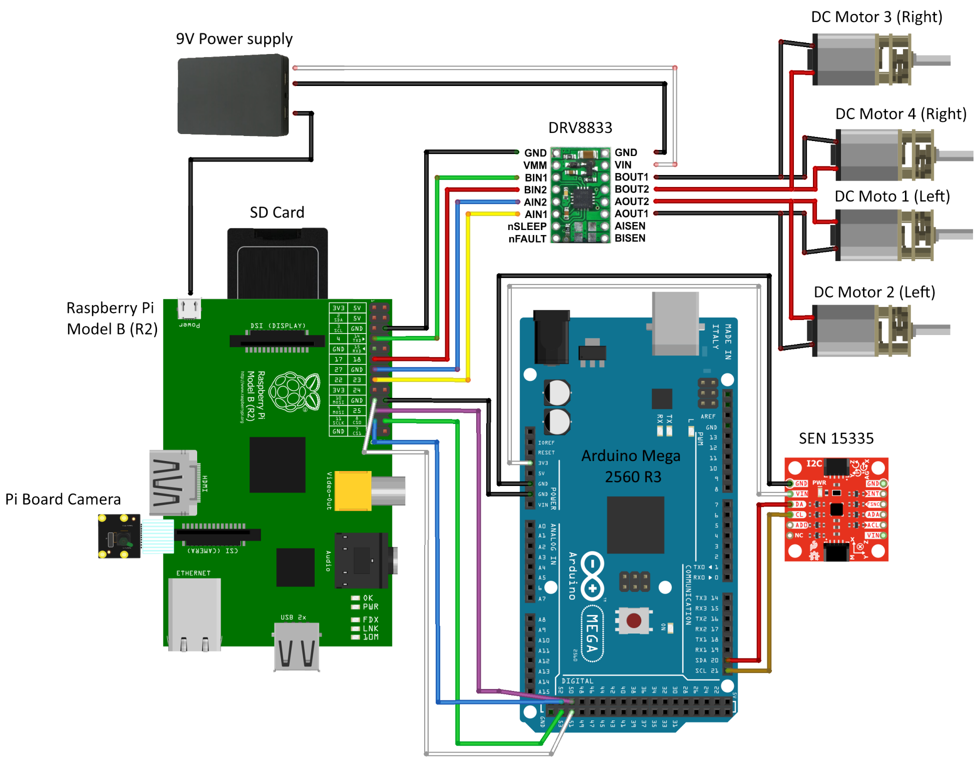


Figure 3.2: Hardware system connections.

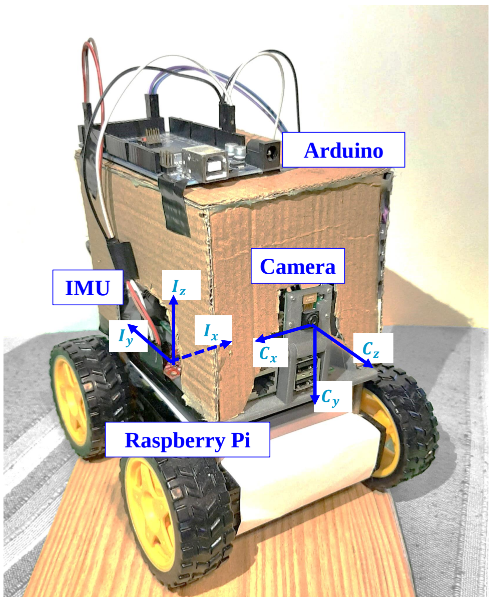
Sparkfun has provided an Arduino library for this purpose, but it is still a work in progress[1]. For example, this library did not have DMP PROGMEM support for older AVR platforms like the ATmega2560. Because this was the processor we were using for this work, we helped the library manager to include this support in release 1.2.7.

There is a Python library for this IMU provided by Sparkfun, but it does not have ICM-20948 support, which means we cannot access the DMP and therefore we only access raw sensor values. This is the reason why we needed an Arduino to act as a middle man, instead of just having the IMU communicate directly with the Raspberry Pi.

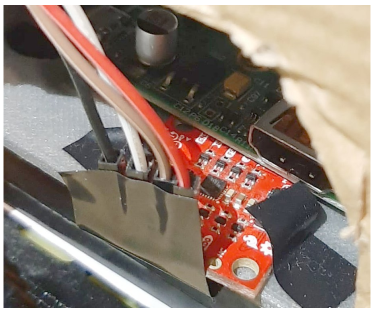
These communications between the IMU and the Arduino are done with the I2C protocol. These components have the fundamental goal of acquiring inertial data: accelerometer measurements, gyroscope measurements, magnetometer measurements and orientation quaternions.

As a final note, it is important to explain the roles of the magnetometer in this work. This sensor was only used to estimate ground truth, and not to perform odometry. While evaluating the estimated orientation (using all 9 DoF of the IMU) coming from the DMP, we could see the magnetometer did not have a significant negative impact. This can be justified by sensor fusion and calibration within the DMP. However, as a separated component, the readings of the magnetometer were not robust due to all the hard iron and soft iron disturbances present in the environment (Figure 2.6). As shown by the DMP, it is possible to handle these disturbances on a traditional house environment. However the same would not be true within an environment abundant in element that caused disturbances (e.g. factories). To avoid these environment dependent disturbances, we did not utilize the magnetometer to perform odometry.

[1]`https://github.com/sparkfun/SparkFun_ICM-20948_ArduinoLibrary`



(a) Complete system



(b) IMU detail

(c) Camera detail

Figure 3.3: Final hardware setup for the mobile robot (a), with close-ups of the IMU (b) and camera (c).

Raspberry Pi

This is the central component of the mobile robot. The Raspberry Pi makes all the sensor data - visual and inertial - available to a PC client. It also receives commands that control the motor wheels.

The Pi is connected to both sensors, the Pi board camera and the Arduino-IMU, and to the actuators, the dual motor driver. To communicate with a PC, the Pi has three servers: a visual server that provides the clients with video feed from the Pi board camera; an inertial multithreaded server that gets the inertial data from the Arduino and provides it to the clients, and a motor server that accepts commands for the DC motors (wheels). Using motion[2] (a program that monitors video signals) we were able to live stream the camera's feed to a specific IP address the PC client can access through a simple browser or using the Java GUI.

The Arduino and Raspberry Pi are communicating over the SPI protocol with an ACK based system of messaging. When the Pi receives a command to start a new dataset it asks the Arduino for a constant stream of data, which it samples and stores locally. Once it receives a message telling the dataset is complete, it communicates

[2]`https://github.com/dozencrows/motion/tree/mmal-test`

this to the Arduino and sends all the stored information to the client PC. This is why we can have multiple clients to this server without jeopardizing the sample rates. The only thing the Pi needs to do is to send the information to multiple users, but during acquisition the number of clients makes no difference.

3.3 PC Interface

The PC interface is mainly characterized by the Java GUI and by the MATLAB scripts that receive and format the visual-inertial data coming from the mobile robot. This first program allows the user to make a connection to the three mentioned servers as well as configure the ports and IP addresses of these servers. It is also through this GUI that we can send commands to the robot's motors. The robot only accepts 4 types of movement, which correspond to the 4 arrows in a keyboard: up arrow moves the car ahead, down arrow moves it backwards and the side arrows power the front wheel respective to their side. All these commands set the motor to work at a specific power, which means pressing an arrow makes the car move at a constant speed. It is possible to send commands changing the voltage provided to the motors, which changes the speed of a movement command.

Figure 3.4 displays the data flow throughout the system. At this point in an acquisition process, we have the Java program that, not only allows us to send commands, but provides us with the live camera feed from the robot and a sampled set of IMU measurements. With this in mind let us talk about the final components: the MATLAB scripts.

If the Raspberry Pi is the brains in the mobile robot side, then the MATLAB scripts are the brain on the user side. They give the order for a dataset to start being acquired (which means the Pi will start storing IMU measurements) and at the same time starts capturing sampled images from the live camera feed through the Java program. When the user gives the command to stop acquiring the dataset, the MATLAB scripts will gather all the data and format it so it can be used later. This means creating a file with the image names and timestamps, and also stack the IMU measurements in a convenient way to load later. Each dataset is composed by images, accelerometer, gyroscope and magnetometer measurements, orientation quaternions and the respective timestamps.

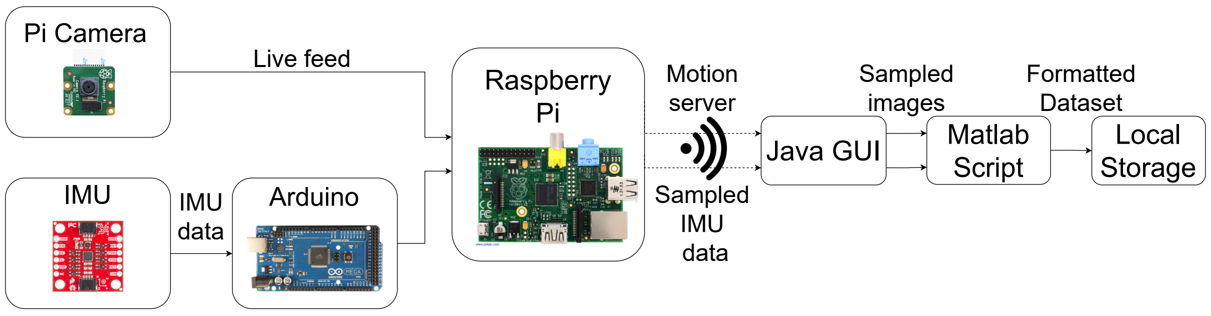


Figure 3.4: System data flow.

3.4 Trajectories Ground Truth

One of the most important parts of this process is estimating ground truth. These are the values that represent truth in a dataset. The reference values that will enable us to evaluate the results of pose estimation. Our general strategy was to record the car and capture key positions in the trajectory. From these we interpolated the rest of the trajectory.

We considered ground truth for three variables in our datasets: position, velocity and orientation. For orientation we used the estimated quaternions coming from the DMP, using all 9 DoF present in the IMU (from now on, entitled the 9 DoF quaternion). However, position and velocity need special attention depending on the movement that is being made. We will, however, assume the velocity is constant for all movements. This is, of course, not accurate, but it is adequate for our work, since we do not mean to estimate velocity deviation, but rather to see if the results are plausible.

In our datasets we only considered two types of movements: curves and straight lines. All movements are on the ground plane, so we will always set the z coordinate to 0. For straight lines, the interpolation is straightforward. We developed a script where we manually input the frames and positions (measured in real life in meters) on which the car started and stopped its movement: p_s and p_f respectively. This script will find the timestamp for each of these positions and interpolate the middle positions and the velocities from this. Let us consider t_s and t_f as the timestamps corresponding to p_s and p_f respectively, and d as the 3D vector representing distance between p_s and p_f. Let us also assume I as the number of samples (can be number of IMU or image samples, the method it equal for both). For a sample i in the middle of the movement and with μ as the percentage of time that passed since the beginning of the movement,

$$\mu = \frac{t_i - t_s}{t_f - t_s}, \tag{3.1}$$

we can interpolate position x_i as

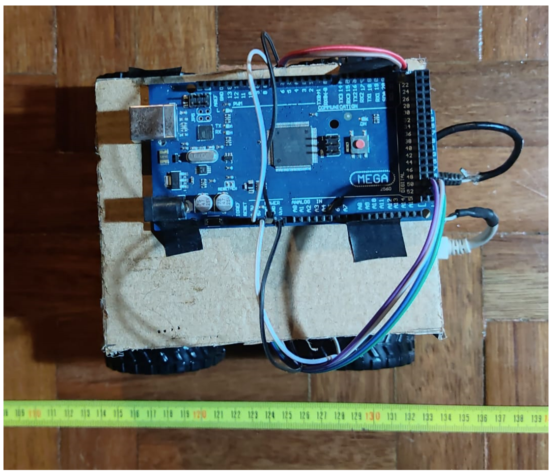
$$x_i = p_s + \mu d. \tag{3.2}$$

For curves the process is the same, with the only difference being in this last equation. We approximate the curve using a $\sin$ function multiplied by a scaling factor λ and a variable s that can be 1 or -1, depending if the curve goes inwards or outwards (can vary depending on the chosen referential). We will use μ as a argument of $\sin$ since μ is a percentage. This means the $\sin$ function will return 0 for the extreme values of μ and 1 for $50\,\%$. So for curves we can interpolate the trajectory as

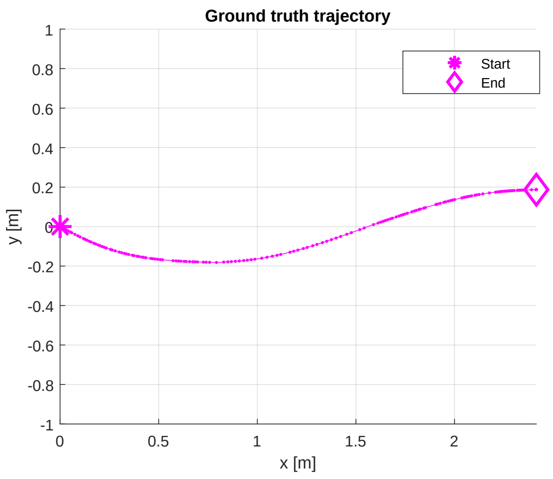
$$x_i = p_s + \mu d + s\lambda \sin(\mu). \tag{3.3}$$

An estimated ground truth for a movement with curves on a even floor ($z = 0$) can be seen in Figure 3.5, along with a photo depicting the car in a key point in the trajectory.

The trajectories and velocities do not take into account the ground truth orientation. That is because we do not want to bound these variables to the orientation. This allows for some leeway in filter initializations. But, of course, this also means we will need to take some measures to make sure we are correctly evaluating results, since the trajectories and the orientations will be in different coordinate systems. This will be discussed in Section 4.3.



(a) Photo of the robot in a key point in the trajectory



(b) Estimated ground truth

Figure 3.5: Example of estimated ground truth for a movement with curves (b) with a photo (a) that marks a key point in the trajectory.

Chapter 4

Visual Inertial Odometry with State Initialization

This chapter proposes a Visual Inertial Odometry system, based on Fusion18 [Brossard et al., 2017], adapted to our mobile robot IMU and camera sensors (Chapter 3). Section 4.1 summarizes Fusion18, providing the essential information to introduce the proposed navigation solution. Section 4.2 shows our Fusion18 adaptations and developments considering our hardware setup. Finally, Section 4.3 details the error metrics used in our work to assess results.

4.1 UKF based Robot Self-localization

Chapter 3 presented our mobile robot, equipped with one camera and one IMU. A number of differences exist between our robot and the original setup upon which was proposed and tested the VIO filter [Brossard et al., 2017] (detailed in Section 2.5.2). The first difference is found on the mounting (poses) of the camera and the IMU.

Camera-IMU Calibration

Because our camera and IMU do not have their coordinate systems aligned (as shown in Figure 3.3a), we considered 3 frames: the world frame W, the body frame B and the camera frame C. W is the fixed global frame on which we want to estimate our key variables on. B is a mobile frame that follows and is centered on the IMU. We named it the body frame instead of IMU frame since we will consider this frame to represent the entire car. Finally, C is the camera's mobile frame.

Although B and C are mobile, for our hardware setup (again, as seen in Figure 3.3), they are fixed in relation to each other. Using Kalibr framework [1], we can get the extrinsic parameters $(\mathbf{R}_{C\to B}, \mathbf{t}_{C\to B})$ and $(\mathbf{R}_{B\to C}, \mathbf{t}_{B\to C})$ that map C to B and B to C, respectively.

Now all we need is a transformation that can map B to W. That is the transformation our whole problem means to estimate: the body's pose. Let us now detail all the important models we considered to enable this pose estimation, namely, the state space, the dynamic model and the measurements model.

[1] `https://github.com/ethz-asl/kalibr`

State Space

The state contains all the variables of interest for the filter to estimate. This includes the body's pose which is formed by its position $\mathbf{x} \in \mathbb{R}^3$ and orientation $\mathbf{R} \in SO(3)$ (that is, the rotation that maps the body frame B to the world frame W), plus its velocity $\mathbf{v} \in \mathbb{R}^3$, the IMU biases $b_\omega \in \mathbb{R}^3$ and $b_a \in \mathbb{R}^3$, and the 3D position of p landmarks $\mathbf{p}_1, \ldots, \mathbf{p}_p \in \mathbb{R}^3$ in W. As proposed by [Brossard et al., 2017], the state will be represented by the pair (χ, b), with χ being defined as matrix

$$\chi = \begin{bmatrix} \mathbf{R} & \mathbf{v} & \mathbf{x} & \mathbf{p}_1 \cdots \mathbf{p}_p \\ \mathbf{0}_{(p+2)\times 3} & \multicolumn{3}{c}{\mathbf{I}_{(p+2)\times(p+2)}} \end{bmatrix} \tag{4.1}$$

with size $(3+2+p) \times (3+2+p)$ and with $\mathbf{0}_m$ and $\mathbf{I}_m$ being, respectively, a zero matrix and the identity matrix with sizes m. The authors defined the bias vector as $b = [b_\omega^T \; b_a^T]^T$ with the accelerometer bias b_a and gyroscope bias b_ω.

Dynamic Model

This filter integrates IMU information by means of the dynamic model of a flying body navigating on flat earth equipped with an IMU. This model can also be used with bodies navigating on the ground (like our robot presented in Chapter 3), since it allows for all scenarios present within the ground constrains (plain floors, inclinations, among others). The system dynamic model is fully presented in Section 2.5.2.

Measurement/Observation Model

The visual information will be used in the system through an observation model. It consists in a projection (Equation (2.16)) of the p landmarks (world frame) contained in the state (Equation 4.1) to the current image frame. The error of this projections will be used to update the state.

Filter Architecture

With the system models presented in this section and in Section 2.5.2, we can finally arrive at a filter as proposed by [Brossard et al., 2017] with architecture as shown in Figure 4.1. Within its context, x represents the state pair of (χ, b) for a time step n. $\bar{x}$ is the state estimate, $\bar{x}^-$ represents the state estimate after the time propagation, $\bar{x}^+$ is the state estimate after the measurement update, and finally, z represents the observations necessary for the update step.

4.2 Self Initialization and Feature Management

Now that we know the models used for this filter and it works, let us talk about what changes needed to be made for us to meet our objectives. The Fusion18 algorithm's high-level global strategy lies in Figure 4.2.

The two components that need intervention are: the initialization process and the feature management, with the latter being part of the filtering (as shown in Figure 4.1). In this work we propose an initialization module that allows the users to perform odometry without having to provide specific initialization values. An auxiliary

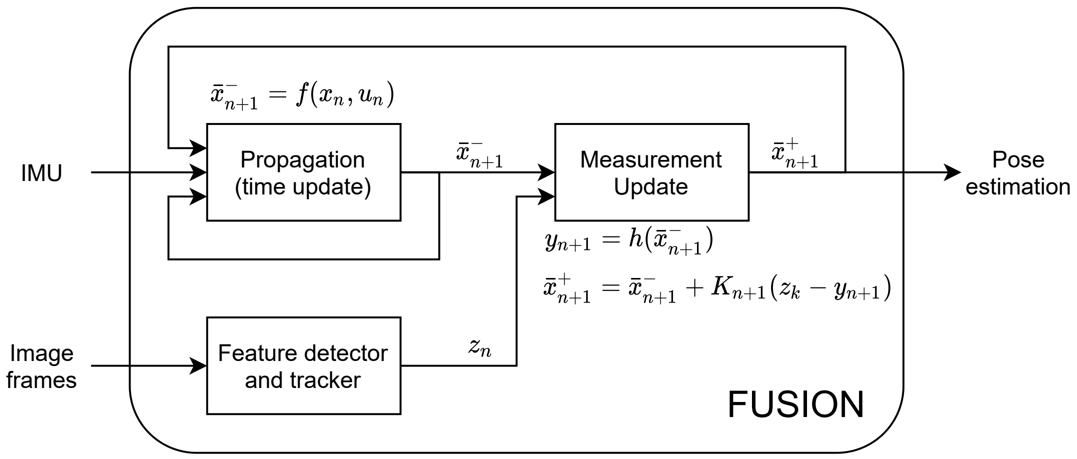

$\bar{x}^-_{n+1} = f(x_n, u_n)$
IMU
Propagation (time update)
$\bar{x}^-_{n+1}$
Measurement Update
$\bar{x}^+_{n+1}$
Pose estimation
$y_{n+1} = h(\bar{x}^-_{n+1})$
$\bar{x}^+_{n+1} = \bar{x}^-_{n+1} + K_{n+1}(z_k - y_{n+1})$
Image frames
Feature detector and tracker
z_n
FUSION

Figure 4.1: Visual Inertial UKF

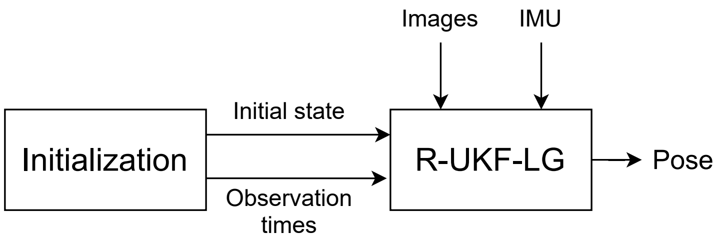


Figure 4.2: Global Fusion18 algorithm.

component we also tinkered with in order to fit better with our datasets was the result display. We will explore this topic in more detail in Section 4.3.

Although the proposed changes were developed to tackle specific issues with our datasets, they achieve a greater goal: enable Fusion18 to run on a variety of datasets. In order to run a dataset in the original version, a user needs to provide the sensor data, ground truth and initialization. Also, a dataset with varying sample rate would cause many errors. With our changes a user can provide a dataset that does not need to be of super high quality (we will explain what we mean by this as we go along) and the filter will be able estimate an initialization and run. Our approach was not to change or improve the datasets, but to enable the filter to accommodate any kind of dataset, as real life conditions can be quite harsh. We should also note all modifications to a dataset are done within the filter, so as not to permanently change any dataset.

Initialization

Although Fusion18 has a robust and accurate operation filter, it lacks on initialization. This filter assumes the user manually provides initialization values for the biases and 3D landmarks (Figure 4.3). For example, the Fusion18 default demonstration starts from the results of running ORB-SLAM2 [Mur-Artal & Tardos, 2017a] to initialize the filter to the specific dataset.

More in detail, the initialization is responsible for establishing observation times, the initial state and a bank of landmarks. Aside from the observation times, which is an auxiliary variable to the algorithm, in a noiseless

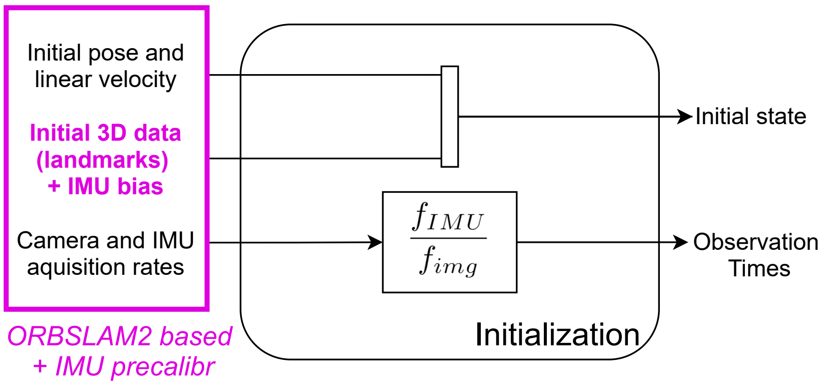

Initial pose and linear velocity
Initial 3D data (landmarks) + IMU bias
Camera and IMU aquisition rates
ORBSLAM2 based + IMU precalibr
$\frac{f_{IMU}}{f_{img}}$
Initialization
Initial state
Observation Times

Figure 4.3: Fusion18's default initialization relies on ORBSLAM2 data and an IMU precalibration.

case, the initialization has set the state and landmarks to their correct values in the world. This means one has to manually measure these values and feed them to the algorithm, which is not a practical approach. The alternative initialization methodology we propose is briefly presented in Figure 4.4.

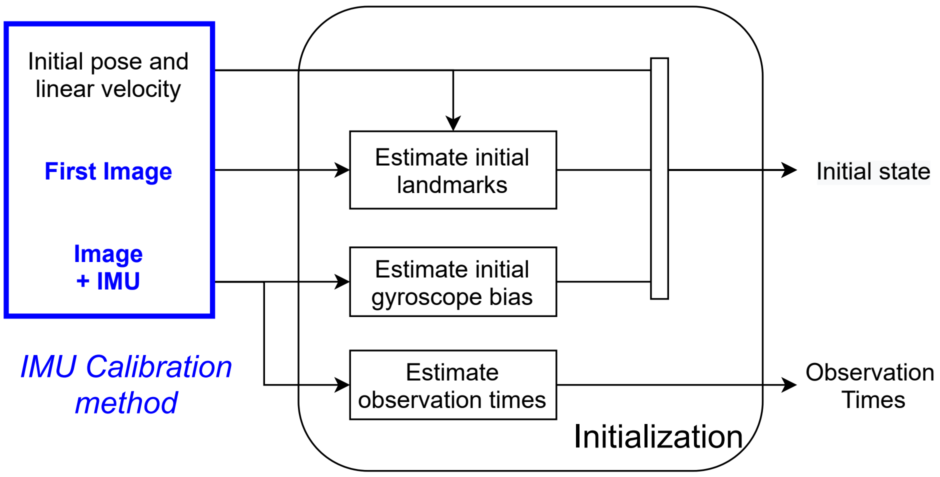

Initial pose and linear velocity
First Image
Image + IMU
IMU Calibration method
Estimate initial landmarks
Estimate initial gyroscope bias
Estimate observation times
Initialization
Initial state
Observation Times

Figure 4.4: Proposed initialization flowchart. The SLAM based initialization of Figure 4.3 is replaced by the IMU calibration, which is complemented by the proposed initial estimators of landmarks and gyroscope bias.

Similarly to the Fusion18 algorithm, we initialize pose and velocity from the ground truth, in order to allow comparing the estimated motion with the ground truth. This choice only implies that our starting pose is not aligned with the world frame W origin. In terms of how the filter works, all referential transformations are present where they need to be, which allows for any given initial pose. The filter will estimate pose starting on the initial one, regardless of its actual value.

Observation Times

The IMU and camera have different rates of acquisition. In a typical scenario, the IMU's rate is higher (as is with ours). This makes it so there usually are multiple IMU samples to be processed before each consecutive image. Considering Fusion18 processes an IMU sample at each iteration, one of the inputs of the algorithm is at which iterations there will be an images to process. This is what we define as observation times.

For datasets that have a rigorous sample rate for both sensor measurements, we can get these observation times by simply using the visual and inertial acquisition frequencies:

$$n = \frac{f_{IMU}}{f_{Cam}}. \tag{4.2}$$

With this we can conclude that we need to process an image after processing n IMU samples. However, if, for a given dataset, the time interval between samples changes, this equation cannot be used. The datasets we were able to acquire have this unfortunate property. This implies a special care to determine these observation instances.

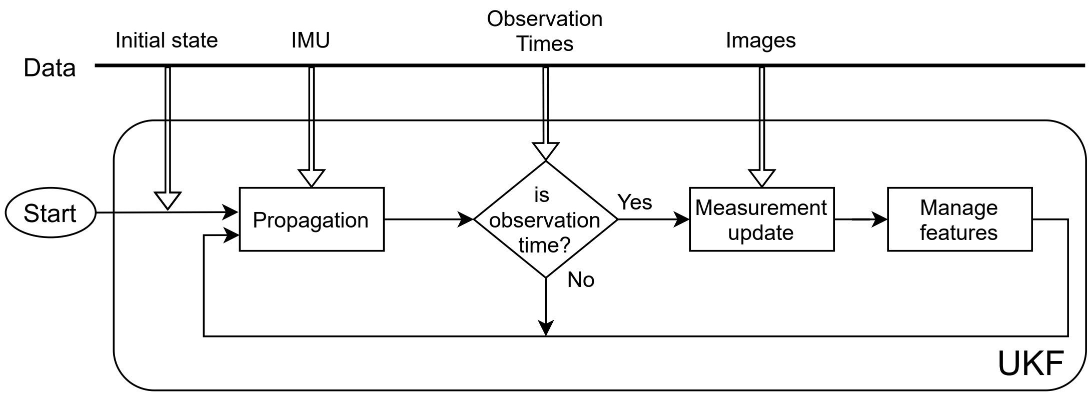


Figure 4.5: Flowchart for the general Fusion18 filter approach.

An important aspect we had to take into consideration was the fact Fusion18 is not prepared to process multiple images between two consecutive IMU samples (Figure 4.5). This filter propagates the state using IMU samples until it is observation time. Only in these instances does the filter perform an update step using images. Because we always have to propagate the state before making an update, we must guarantee there are not multiple images between two consecutive IMU samples. Our approach was to remove images that felt into this scenario. This fact also implies that we have to remove all images that, temporally, come before and after the first and last IMU samples, respectively. Algorithm 1 details our approach, assuming observation times, `obsTimes`, is represented as an array of boolean's with size equal to the number of IMU samples. We consider as inputs the visual and inertial arrays of samples and timestamps, `IMGS`, `t_IMGS`, `IMU` and `t_IMU`, respectively.

Landmark Initialization

There are two types of features we need to estimate in the initialization process. First we need to estimate the 3D position of landmarks for the initial state. Second, we need to get 2D features for latter replacement of the state's landmarks. Let us focus first on the latter since they are easier to get.

During the filter, it is probable that, at some point, the initial landmarks will stop being visible or will have an excessive projection error. Eventually they will need to be replaced. For this purpose, Fusion18 always keeps record of a good number of features in case any of the state's features need replacing. Not only does the filter save the image location of these features, but it also saves on which image frames they were detected. This allows for a reliable estimation of the 3D positions via triangulation of the past sightings. We can initialize these features

Algorithm 1 Function that calculates the observation times

```
Input: IMGS, t_IMGS, IMU, t_IMU
1: $n\_imgs$ = length(IMGS)
2: $n\_imu$ = length(IMU)
3: obsTimes = boolean($n\_imu$); obsTimes(All) = false
4: while t_IMGS(1) < t_IMU(1) do    ▷ Remove images with timestamps sooner than the first IMU timestamp
5:     Remove image
6: end while
7: $j = 1$    ▷ Remove multiple images between consecutive IMU samples and determine observation times
8: for $i = 1 : n\_imu$ and $j > n\_imgs$ do
9:     if t_IMU($i$) ≤ t_IMGS($j$) < t_IMU($i + 1$) then
10:        if $j < n\_imgs$ and t_IMU($i$) ≤ t_IMGS($j + 1$) < t_IMU($i + 1$) then
11:            while $j + 1 < n\_imgs$ and t_IMU($i$) ≤ t_IMGS($j + 1$) < t_IMU($i + 1$) do
12:                Remove image
13:            end while
14:        end if
15:        $j = j + 1.$
16:        obsTimes($i$) = true
17:    end if
18: end for
19: if $j < n\_imgs$ then    ▷ Remove images with timestamps latter than the last IMU timestamp
20:     Remove all images after $j$
21: end if
Output: IMGS, t_IMGS, IMU, t_IMU, obsTimes
```

and structures using a corner detection algorithm like minimum eigenvalue (which is Fusion18's preferred feature extraction algorithm).

The state's landmark initialization requires more subtlety. The 2D feature extraction is similar to the other features. To project these 2D features into the world frame W we need to perform multiple transformations. First we need to project this feature into the camera frame C using the camera's intrinsic parameters. We can convert the pixels, u and v, to metric coordinates, X and Y, as

$$X = \frac{u - c_x}{fs_x}, Y = \frac{v - c_y}{fs_y}. \tag{4.3}$$

However, with just one view, there is not enough information to estimate depth. Our solution was to set Z to the same value for all landmarks. This was shown to be an adequate solution, since we had an initial high variance to compensate for this prediction.

With this transformation we have estimated the 3D landmarks in the camera frame C. Now all that is left to do is to transform them to the world frame W. This can be done using the the camera-IMU extrinsic parameters, $\mathbf{R}_{C\to B}$ and $\mathbf{t}_{C\to B}$ ($C \to B$), and the initial state pose, $\mathbf{R}$ and $\mathbf{x}$ ($B \to W$).

During the execution of the filter, specifically before each update step, these landmarks will be projected into the current image plane (as in see in Section 2.5.2, specifically Equation (2.16)) and compared to their expected positions in the image. If the distance between the projections and the expected points surpasses 20 pixels, the 3D landmarks will be invalidated, i.e., discarded from any further usage. This will have a negative impact on the state estimation as the replacement of landmarks only happens after an update step.

An important aspect that we need to be aware in this landmark projection is the transformation from the world frame W to the body frame B. This uses the state's pose, which means that if the state is too deviated from reality it might jeopardize this observation and result in landmark invalidating, even if the landmark estimates were accurate. Some tests with our datasets proved these fears to be true.

Due to the disparity of acquisition rates, it is probable that the filter will process several IMU samples before handling the first image (not counting the initialization frame). The filter uses IMU samples to propagate the state, so each time it propagates the state, it is effectively integrating the IMU measurements, and therefore, the bias present in these measurements. As seen in Section 2.2, this would create an erroneous state pose that would inaccurately project the landmarks into the image plane, and would results in a collective landmark invalidation. But the worst thing about this is the fact the bias estimation is directly dependent on the landmark observation. A collective invalidation of landmarks means no IMU bias estimation, which in turn means the measurements will still contain bias and results in state deviation, which will cause worse landmark projections, and so on.

We call initialization period to the initial filter life until it obtains a state where the bias estimated based on visual information can reduce the measurement's bias in a way it results in accurate projections of landmarks, i.e., creating synergies between sensors. To assist the filter in this initialization period and avoid a cycle of sensor deterioration, we removed all, but one, IMU samples that came temporally before the first image, as to avoid multiple integration of bias. With this measure we were also trying to make sure the body's pose used in the first landmark observation was similar to the pose used in the landmark initialization, as to provide a successful (and invalidation-free) initial landmark observation. The success of this task results in an immediate bias estimation which helps in synergize the sensors.

Gyroscope Bias Estimation

As just mentioned, the performance of our filter is heavily dependent on the estimation of IMU bias. To tackle this issue, we propose a method to estimate the initial gyroscope bias. Our strategy to estimate this bias was inspired by [Qin et al., 2018]. To perform this estimation we are assuming an offline setup, i.e., we first need to acquire all data and only afterwards will we estimate the values. In Figure 4.6 we can see our approach for this estimation.

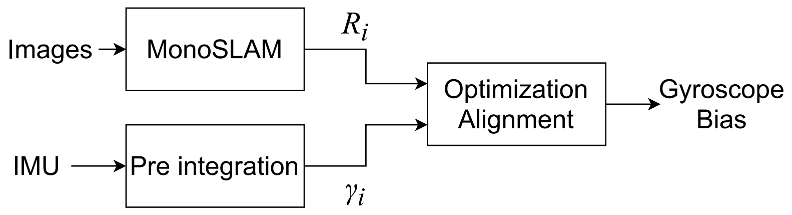


Figure 4.6: Proposed bias initialization strategy.

Let us consider I and J as the total number of images and IMU samples in the dataset, respectively. The first step is to estimate the orientation of all image frames, q_i (quaternions with $i = 1, \ldots, I$), using only the visual information. In Appendix B we introduce quaternions with detail. To achieve these orientation estimates we use the MonoSLAM algorithm of [Civera et al., 2008]. These orientations will be in the camera frame C.

Secondly (or in parallel), we preintegrate the inertial data in the camera frame C in order to estimate the rotation matrices between each image frame, γ_i, $i = 1, 2, \ldots, I$. We consider the starting bias to be null $[0\ 0\ 0]^T$. However, if the user already has an initial bias estimation, it can be inputted to this method.

For j as a discrete moment corresponding to a IMU sample within $[t_i, t_{i+1}]$, we can discretely integrate the gyroscope measurements as

$$\gamma_{j+1} = \gamma_j \otimes \begin{bmatrix} 1 \\ \frac{1}{2}(\omega_j - b_{\omega_j})\delta t \end{bmatrix}, \tag{4.4}$$

with δt as the time interval between IMU samples j and $j+1$. In order to get the rotation between image frames $i-1$ and i, we need to accumulate the rotations within that interval by doing the quaternion multiplication between consecutive rotations following the correct order.

We now have all the orientations at each image frame coming form the visual data and all rotations between each consecutive image frame coming from the inertial data. In a perfect world, for a arbitrary consecutive frames i and $i+1$, if we rotated the frame's orientation q_i over γ_{i+1}, we should get q_{i+1}. However, due to bias and noise, this will not happen. Because of this, we can build a cost function to estimate the bias:

$$\min_{\delta b_\omega} \sum_{i \in I} \left\| q_{i+1}^{-1} \otimes q_i \otimes \gamma_{i+1} \right\|^2 \tag{4.5}$$

with

$$\gamma_{i+1} \approx \gamma_{i+1} \otimes \begin{bmatrix} 1 \\ \frac{1}{2} J^{\gamma}_{b_\omega} \delta b_\omega \end{bmatrix}. \tag{4.6}$$

$J^{\gamma}_{b_\omega}$ is a part of the first-order Jacobian, J_{i+1}, of the covariance matrix, P_{i+1}, obtained in the IMU preintegration. In Appendix D you can see with detail how to estimate these matrices and how to perform the complete and general IMU preintegration process.

By initializing both starting orientations (MonoSLAM and preintegration) as identity quaternion (null rotation) we are guaranteeing both methods to estimate orientation in the same reference since we have both body B and camera C frames fixed in relation to each other. Because we are only interested in rotations, and the whole systems rotates as one, all estimations stay coherent.

It is important to note that this cost function represents the concept behind our approach. In practice we have a clear problem: rotation quaternions should be unitary, i.e., their norm should always be 1. This brings us to the problem of how do we, in one number, represent a 3D rotation in order to build a rotation cost function. We decided to convert the resulting quaternion to Euler angles and then estimate the squared norm of that array of angles. This has some disadvantages, but it proved itself to be adequate for our case. In Section 4.3 we will discuss a bit more about this topic and how the people behind Fusion18 tackled this problem to design an error metric.

With that done, we have now estimated the initial bias. It is possible to repropagate the rotations between frames as (4.6) with the estimated bias and repeat the process. However, excessive repetition of this method might lead to miss calculations since we cannot forget the gyroscope data also has the effect of noise present.

Feature Management

This process is part of the Fusion18 algorithm, as seen in Figure 4.5. This step is responsible for replacing non-visible landmarks in the state, as well as replace the bank of reserve features if necessary.

By default, the filter will only try to replace the state's landmarks after processing 120 IMU samples. This is a measure put in place to ensure the filter can robustly initialize the 3D position of new landmarks. Not only is this a limitation on the size of a dataset, it also means that, in this interval with the initial landmarks, it is imperative the state is not too far off from reality. If the state's pose causes a collective invalidation of landmarks in this initial period, that would cause the filter to run solely with the inertial data without any possibility of bias estimation until the 120-th iteration. This would result in filter divergence.

To combat this, we changed this iteration threshold to 30 and made all logic related changes to the original code necessary for this change to work properly. The most relevant logic change was that now, in early iterations, we might not be able to track features for the feature reserve bank on as many images as before. The default code tracked the 20 previous images, whereas now we might not have 20 images to track right away. In principal this negatively affects the initialization of a replacement landmark, but in our experiments this did not have a significant effect on the overall filter performance.

4.3 Estimated Pose Error Metrics

Evaluating the results of a given experiment might not be a straightforward process. It is important to establish what we will compare the results too and what exactly are we looking to study on those results. This section is dedicated to the error metrics used in this work and how they can provide objective evaluations of the results in Chapter 5. We will consider the Root-mean-square error (RMSE) metric that came default with the Fusion18 software and then we will propose a new Procrustes based error metric more suited to evaluate the results with our datasets.

The first metric is suited to study residuals between two instances. This implies the results must be in the same frame of reference as the ground truth. As presented in the software, this is a metric that focusses on the deviation between points. Because of this, we will be using it to evaluate all orientation related results. This is because orientation needs a point to point metric, instead of a global metric. Another reason is because rotation, either represented as quaternions, angle-axis or rotation matrix, need special handling that methods like the Procrustes problem will not provide. If we switch to an Euler angles representation, we will not need this special treatment, but we will walk into another horde of problems, like negative angles.

The metric based on the Procrustes problem consist in using it to align the estimated point cloud with the ground truth point cloud. This metric is useful when we have a result that is in a different coordinate system as the ground truth. It is also advantageous in scenarios where the results are not in the same scale as the ground truth. This is, of course, a metric that only allows us to see if a result is plausible, since it might make a result look close to reality, when in fact, it was the alignment that made it so.

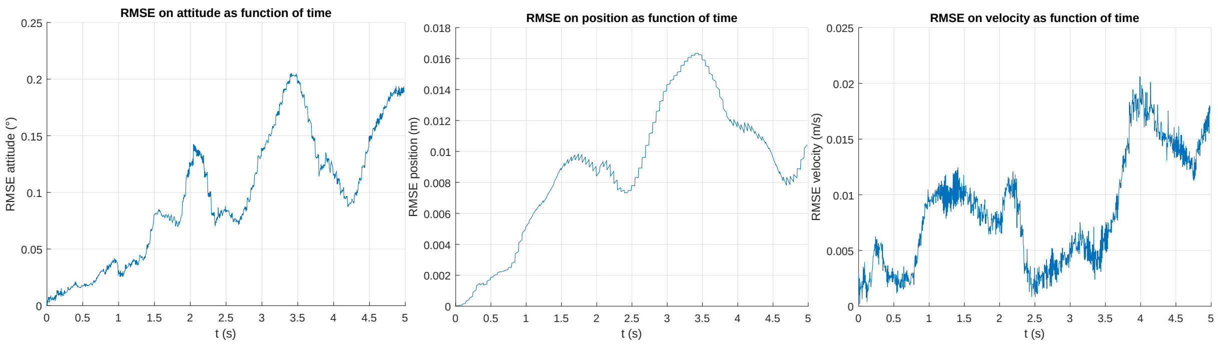
RMSE Metric

RMSE is a non-negative metric that focuses on the standard deviation of the residuals between pairs of single values. Of course, we can reduce points or even rotation to single values with methods such as norms and algebraic transformations. In the original Fusion18 software, this metric is used to evaluate the whole model, but, it is also used in a sample to sample basis. We will focus on the latter.

Position and Velocity Because both position and velocity are represented as 3D vectors, we just need to find the norm of the difference between the estimated values coming from the filter and the ground truth. After the norm we divide by 3 to perform the mean. The full RMSE expression used in Fusion18 for any 3D vector p is

$$p_{RMSE} = \frac{\sqrt{p_x^2 + p_y^2 + p_z^2}}{3}. \tag{4.7}$$

Orientation Because the orientation is represented as a rotation matrix, we first need to convert it into a 3D array form and then apply the RMSE calculation for a 3D array. For that we perform a logarithmic map to transform the rotation from the $SO(3)$ group (3D rotation group) to the Lie Algebra $\mathfrak{so}(3)$ and from that we can get an axis-angle representation from which we can extract a 3D array that represents the rotation. This mapping process is explained in detail in Section 2.3. After computing this 3D array, we apply (4.7) to the array and convert the result from radians to degrees. An example of Fusion18's RMSE functions for a position, velocity and orientation can be seen in Figure 4.7.



(a) Attitude (b) Position (c) Velocity

Figure 4.7: RMSE of the estimated attitude (a), position (b) and velocity (c) over time.

Procrustes Metric

Pose estimations (point clouds) resulting from different methods or runs of a pose estimation filter may be referred to different coordinate systems. One needs to align those clouds in order to compare them.

In this metric we use the Procrustes analysis [Kendall, 1989] to find the transformation that most closely maps the results to the ground truth and then we visually compare both trajectories. This alignment is characterized as a rigid transformation with scaling.

Procrustes Analysis Given two orientation matrices X and Y, the orthogonal Procrustes problem is a transformation where one finds the orthogonal matrix R that most closely maps X to Y:

$$\arg\min_R \|RX - Y\|_F \quad \text{subject to} \quad R^T R = I, \tag{4.8}$$

with $\|\cdot\|_F$ as the Frobenius norm, a special matrix norm. The solution for this problem was originally found by Peter Schönemann [Schoenemann, 1964]. It consists in solving the equivalent problem of finding the closest orthogonal matrix to a given matrix C. Considering $C = YX^T$ we get the problem $\min_R \|R - C\|_F \quad \text{subject to} \quad R^T R = I$. The solution to this problem can be found by performing singular value decomposition of the matrix C as $C = U\Sigma V^T$. With U and V^T, one can find the matrix R that solves the Procrustes problem as $R = UV^T$.

In Procrustes analysis [Kendall, 1989] one compares point clouds which occupy different volumes and have different mass centers. Two point clouds (objects) are compared after translating, rotating and uniformly scaling the objects, i.e. performing Procrustes superimposition.

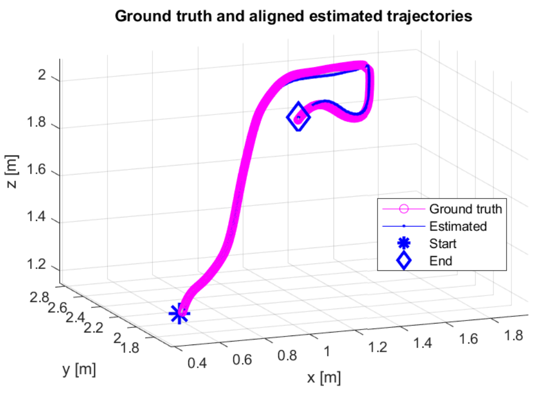


Figure 4.8: Procrustes error metric. Alignment of estimated trajectory to ground truth.

Trajectory Adjustments In our experiments two factors motivate the use of Procrustes analysis. The first is the irregular sampling of trajectories. In the ground truth, the irregular sampling is due to the way the ground truth is obtained (see Section 3.4). In the filter based trajectory estimations, the irregular sampling emerges from hardware limitations, namely the option of minimizing each IMU acquisition time being implying a non-constant sampling rate.

The second reason is the Fusion18 estimated trajectory for our datasets tends to not be proportional with the ground truth during the initialization period. We concluded it would be beneficial if we could mitigate the effects of these segments when performing the Procrustes alignment, as to better assess the results after this period.

Our approach to compute the metric involves three aspects: avoiding repeated values (samples) by choosing a single representative, re-sampling and repeating initial and final values. The first and last aspects seem contradictory but within the next paragraphs will be detailed how they are useful in conjunction.

About avoiding repeated values, we are considering repeated (consecutive) points that are equal or up to a distance of $1\,\mathrm{mm}$ from each other. This procedure compresses clusters of repeated points into single points. The points representing compressed clusters are estimated by doing a simple mean or by choosing the nearest neighbor to represent the entire cluster. As referred, this helps mitigating the initial overestimation of the trajectory. But will also help in other moments where there should be no movements, yet the estimations say otherwise, and even in simply better visualizing the results.

The main objective of re-sampling is to make sure all points are evenly distanced while maintaining (approximately) the trajectory's shape. Achieving this can be done with already established methods. This improves visualization and favors the Procrustes alignment, since mapping point clouds using a rigid transformation with scale works best when the point clouds have equidistant points.

Finally, repeating the initial and last points a number of times, can be seen as a compensation for the previous compressions. In addition, the repetition allows assigning larger weights to some stationary points during the Procrustes alignment, i.e. assuming long-duration (stationary) points are better handled by pose estimation than local (noisy) samples corresponding to motion.

Figure 4.8 illustrates the use of this error metric applied to a trajectory example, the default (demonstration) Fusion18 dataset. The figure shows, qualitatively, a good superimposition of point clouds estimated by the filter and the ground truth.

Chapter 5

Experiments and Results

This chapter presents all the experiments and results, from acquiring an IMU to running Fusion18 with our own datasets. In Section 5.1, we validate our IMU's measurements and estimations along with its calibration by means of visual experiments. In Section 5.2 we detail the datasets used and their characteristics . Finally we present in detail our experiments with the Fusion18 filter. These include seeing the impact one noisy IMU sample has on the overall results (Section 5.3), testing our own acquired datasets, and comparing our general initialization to an initialization using the results of running a VIO filter (Section 5.4).

5.1 Hardware Setup Testing and Calibration

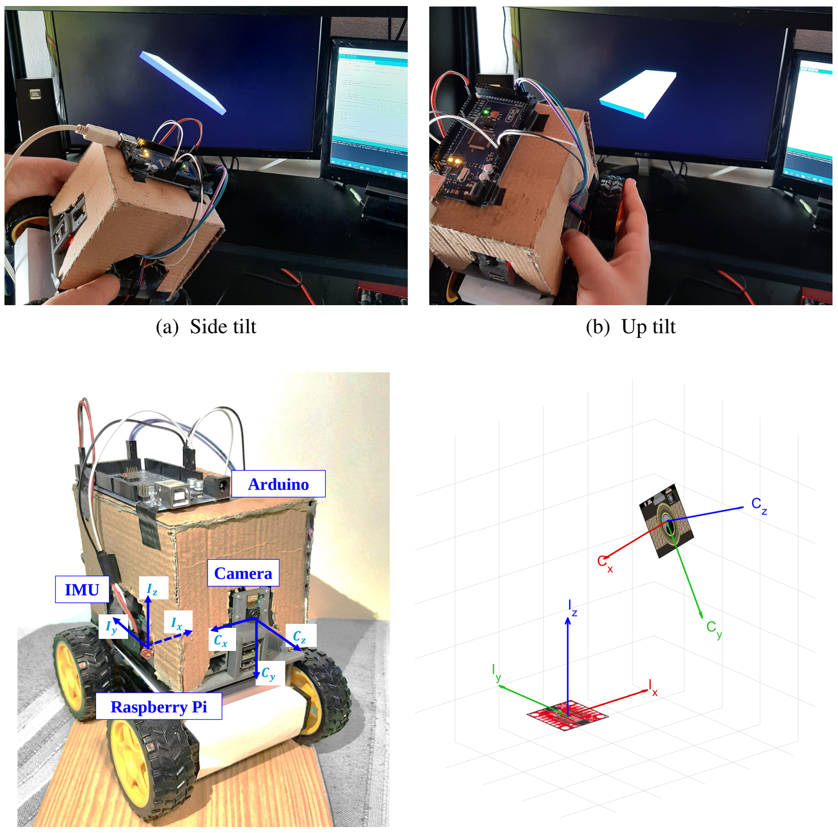
This section details the experiments that validate the performance of our IMU. Two main objectives: guarantee the IMU is putting out quality measures that can be used for sensor fusion, and make sure the IMU calibration is valid. First we visualize the orientation estimates in real time in order to validate the IMU's sensor measurements and orientation estimation. Then we also visualize the IMU-camera calibration and compare it to the real hardware setup.

Testing IMU Orientation Estimation

One of the strongest advantages of our IMU is its DMP. The DMP allows the processing of the IMU data in order to calibrate it, remove bias and estimate properties like orientation. We start by considering two types of orientation quaternions the DMP estimates: the 6 DoF quaternion and the 9 DoF quaternion. The first only takes into account the accelerometer and the gyroscope, while the latter also uses the magnetometer. In practice, the difference between the two orientations lie in the latter's alignment with Earth's magnetic field. In both cases, the IMU assumes the first position to be the center of the coordinate system and starts with a null rotation of $[1\ 0\ 0\ 0]^T$. Then, for the 9 DoF, there is a calibration period where the orientation is being calibrated with the magnetometer. This calibration period takes an average of $32\,\mathrm{s}$.

We used a toolbox to visualize the orientation in real time[1]. It receives quaternion information from a serial communication port and represents it visually using a rectangular cuboid. We used both 6 DoF and 9 DoF quaternions and aligned the screen with north for an easier validation of the results. Two images of the experiment can

[1] `https://github.com/ZaneL/quaternion_sensor_3d_nodejs`



(a) Side tilt

(b) Up tilt

(c) Setup

(d) Camera and IMU frames

Figure 5.1: Hardware setup validation experiments. Testing the orientation estimation of the DMP using a 3D visualization toolbox with our robot side tilted (a) and up tilted (b). Comparison between hardware setup (c) and estimated relation between the camera and IMU frames (d).

be seen in Figure 5.1.

The experiment has shown the cuboid reacts with an high level of accuracy and speed to each and every movement, regardless of its speed. The most relevant issue noticed was the horizontal drift. It was apparent rotation movements caused the most drift, although we could notice a slight drift even if not moving.

As explained in Appendix C, the accelerometer (and magnetometer if used) provide an absolute estimation of the orientation, while the gyroscope provides a relative estimation. If we compare the 6 DoF and 9 DoF orientations in terms of drift, we can conclude that, although both suffer from drift, the 9 DoF is more robust due to the addition of the magnetometer.

IMU-Camera Extrinsic Parameters

One of the key parts of the calibration work was to estimate the extrinsic parameters that performed the rigid transformation between the camera frame and the IMU frame, $(\mathbf{R}_{C\to B}, \mathbf{t}_{C\to B})$. Without these parameters we cannot have any form of synergy between the camera and IMU since they are representing data in different coordinate systems. To estimate these parameters we used the Kalibr toolbox[2]. In Appendix E we detail the process of estimating

[2]`https://github.com/ethz-asl/kalibr`

these parameters and explain the cautions one must have when performing this calibration.

To validate our results, we plotted the IMU in the origin of a referential. Then we applied the obtained rigid transformation into a second referential (camera frame) starting from the origin. In Figure 5.1d we can see the estimated transformation between the camera and IMU frames. We can evaluate these results by comparing them to Figure 5.1c. We should note the images in Figure 5.1d are just visual aides and, along with the rest of the plot, are not proportional to reality. We are looking to evaluate relative position and orientation.

Overall the results were found to be accurate. There are, however, some visual disparities we want to address. It is imperceptible in images (like Figure 3.3c), but in the real car, C_z is not parallel to the ground. This explains the camera's slight upwards rotation along the x axis. It is also not perceptible in Figure 3.3b, but the IMU is not parallel to the ground due to the wiring. This causes a slight rotation over I_y towards positive I_x. Because we plotted the IMU in the origin of the global coordinate system, this slight rotation will manifest itself on the camera, but reversed. And this is why the camera is a slightly rotated over I_y towards negative I_x.

5.2 IMU and Video Datasets

Throughout this work we used datasets from two sources: EuRoC MAV [Burri et al., 2016] and our own mobile car. All the EuRoC MAV datasets are stereo visual inertial datasets. Each dataset also contains ground truth estimated using a Leica total station for position and a Vicon system for 6D pose. These datasets were and are used by many research works in this area duo to their robustness, versatility and easiness of use.

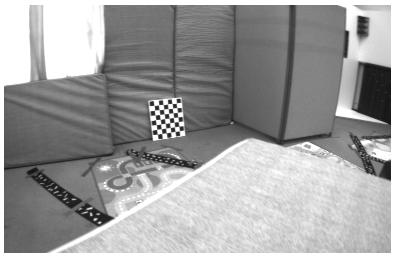
The main dataset used from [Burri et al., 2016] was the same dataset that came default with the Fusion18 filter: *V1_02_medium*. The Fusion18 software also came with initialization values for this dataset. This allowed us to test certain scenarios without having to worry much about the quality of the dataset.

The acquisition rates for this dataset are 20 Hz and 200 Hz for the cameras and IMU, respectively. In this dataset the MAV performs some simple aerial movement with plenty of horizontal rotations (Figure 5.2d) and always with visible targets for feature extraction (Figure 5.2). In order to shorten the execution time of the filter, we focused on an interval of 1000 IMU samples and 100 images. In Figure 5.2 you can see the first, a middle and the last images of this interval.

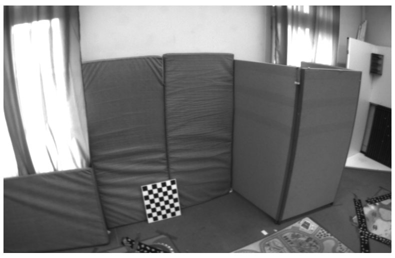
With the setup depicted in Chapter 3, we were able to capture visual inertial datasets. We were able to achieve stable frequencies for the visual data due to it being acquired in the PC over a live feed server. The same was not true for the inertial data, which was sampled on the Raspberry Pi. The inertial data has a varying frequency because of the Pi's incapacity to capture this data at a periodic rate, while also serving the other system's needs. Not only does the Pi have to support the server that streams the camera feed, it also has to depend on the latency of communications with the Arduino. These factors are too much at times and make it so the sample time varies.

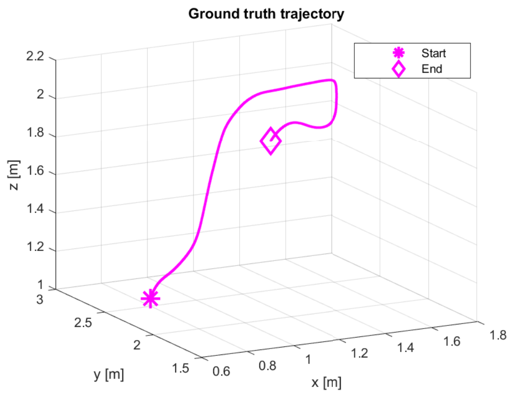
Throughout the acquisitions of dataset we always tried to make sure the camera was able to capture objects from which the filter would be able to extract features. We also started all datasets with a few seconds of no movement to help the filter have a short and robust initialization period. In addition we looked for maintaining a constant uniform lighting as well as a smooth constant movement. All ground truth values for the datasets have a null position values for the z axis since the car only travels on plain ground.

Movement Forward "*mf_04*" This first dataset's objective was having a simple trajectory with no rotations. The car makes a straight line trajectory at a constant speed (Figures 5.3). This dataset has acquisition frequencies


(a) First image

(b) Middle image


(c) Last image



(d) Ground truth trajectory

Figure 5.2: Images 109 (a), 159 (b) and 208 (c) of the used interval of the EuRoC MAV dataset *V1_02_medium* along with its ground truth trajectory.

of 13 Hz (average) and 5 Hz for the inertial and visual data, respectively.

Movement Curve "*mc_01*" This dataset was meant to introduce rotations. More specifically, this dataset introduced rotations over the IMU's z axis, as shown in Figure 5.4 (more specifically, it can be seen in the gyroscope measurements in Figure 5.4f). We were able to capture it at the frequency rates of 10 Hz (average) and 5 Hz for the inertial and visual data, respectively. Despite the curves, we were able to uphold our own rule of always maintaining visible objects (Figures 5.4a, 5.4b and 5.4c).

Due to the usage of random selection throughout the filter, every run is unique. This random selection is mostly found in the task of landmark replacement. We ran each experiment several times and displayed here results that represent best the several runs.

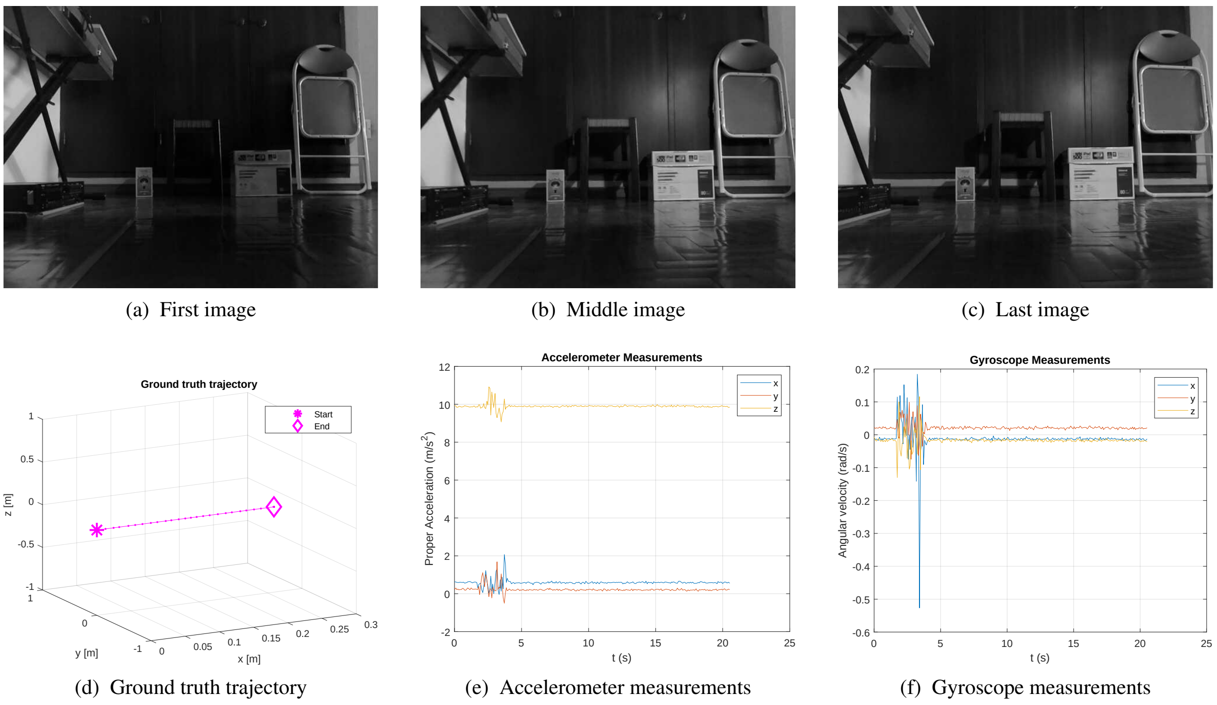


Figure 5.3: Wheeled mobile robot, dataset *mf_04*. Images 1 (a), 30 (b) and 103 (c). Ground truth trajectory (d). Accelerometer (d) and gyroscope (e) measurements.

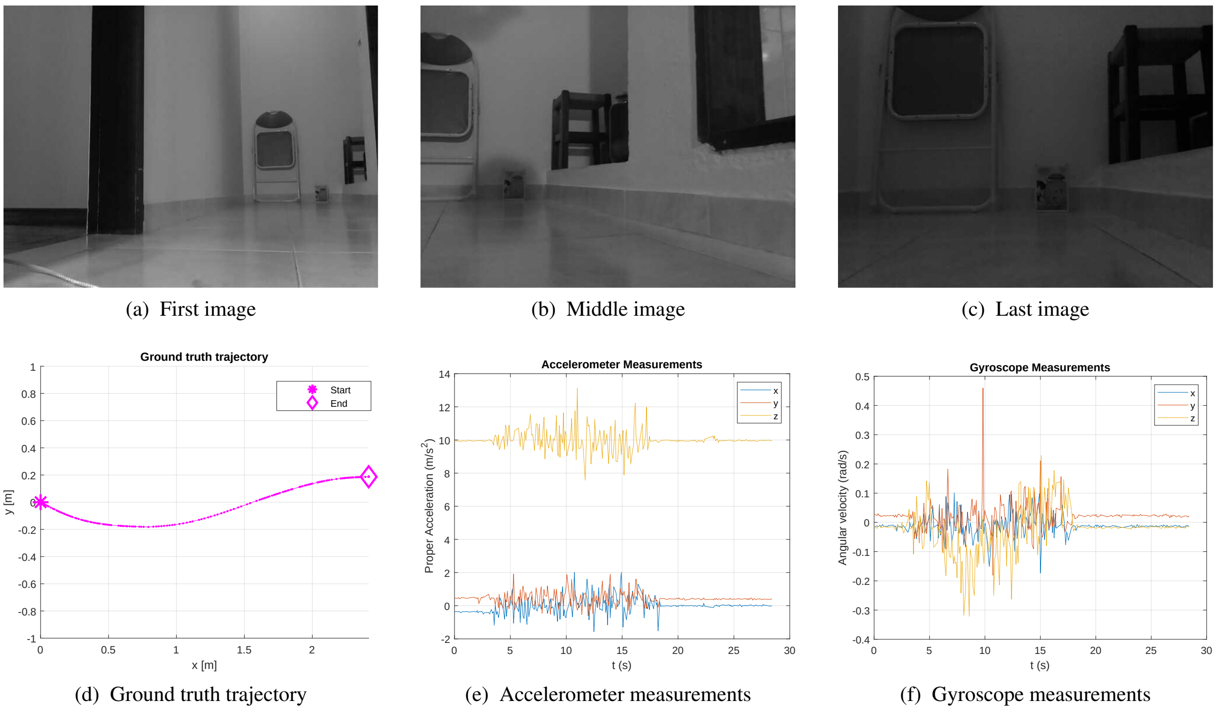


Figure 5.4: Wheeled mobile robot, dataset *mc_01*. Images 1 (a), 84 (b) and 145 (c). Ground truth trajectory (d). Accelerometer (d) and gyroscope (e) measurements.

5.3 Effect of One IMU Sample Error

In order to see how the filter handles wrong IMU data we thought of a simple test: insert one noisy IMU sample. The filter uses IMU samples to make small changes to the state (Equation (2.20)), while also propagating its uncertainty. When the filter has an image to process, it then makes an update to the state based on that uncertainty using both the previous state estimated with IMU data and the image data. This means the filter, even if temporarily, takes the IMU data as is, with the only safeguard being the state's uncertainty that can give less weight to the inertial estimates when doing an update. This experiment was meant to test how the filter would behave if faced with one particularly noisy IMU sample.

We used the default dataset (V1_02_medium) with a maximum of 1000 iterations, inserting the wrong data in iteration 502. Due to the dataset's visual frequency, the filter will process images on iterations 1, 11, 21, etc... We choose iteration 502 to insert wrong IMU data because it was right after the processing of an image. That means the filter would have to wait for iteration 511 until it got the next image to, possibly, correct the influence of the wrong IMU sample. This iteration is at $2.5\,\mathrm{s}$. With a_{502} and ω_{502} as the linear acceleration and angular velocity measurements for iteration 502, respectively, and with n as noise percentage, the noisy IMU data can be written as

$$a_{502,n} = a_{502} + a_{502}\frac{n}{100}, \tag{5.1}$$

$$\omega_{502,n} = \omega_{502} + \omega_{502}\frac{n}{100}. \tag{5.2}$$

Before talking about the results of this experiments it is important that we establish a baseline: a no noise run, as displayed in Figure 5.5.

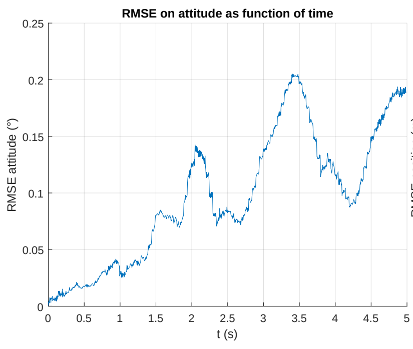
Let us consider the experiment with $400\,\%$ error in the sample (Figures 5.6a, 5.6b and 5.6c). In these results we can see an offset at $2.5\,\mathrm{s}$ in RMSE for each estimate. It is also very interesting how these RMSE's interact with each other. For example, we can see in Figure 5.6b that the position estimate recovers from the noisy sample almost instantly, but when compared with a no error run (Figure 5.5b) the peak RMSE is higher. That is because the velocity and orientation's RMSE (Figure 5.6c and Figure 5.6a, respectively) takes a while longer to recover from the noisy sample, which in term affects the position. In practice, the uncertainty of the IMU's information increases, which leads the filter to rely more on the visual information. However, as we can see in Figure 5.5b, from around $2.5\,\mathrm{s}$ to $3.5\,\mathrm{s}$, the RMSE is increasing due to the visual information. We can tell this because of the step like changes to the RMSE, which are the update steps that take the visual information into account. This ultimately leads to each update step having higher negative impact on the results from this interval.

Now let us focus on the experiment with $900\,\%$ error. Something interesting to look at when comparing Figures 5.5a and 5.6d is that, aside from the $2.5\,\mathrm{s}$ to $3.5\,\mathrm{s}$ interval that is different due to random selection, the RMSE's from $3.5\,\mathrm{s}$ to the end have the same shape and almost the same values. This is a scenario of an excellent recovery for the orientation estimates, as, contrary to the $400\,\%$ experiment, it was able to lowers the error between estimation and reality to the same values as the reference no noise run.

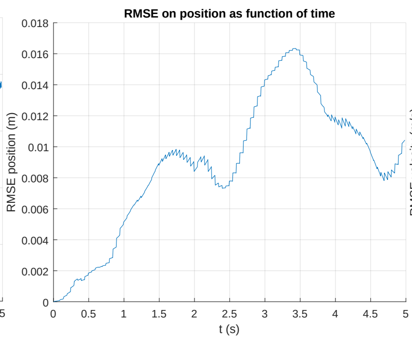
In the experiments with $1600\,\%$ noise, we can see the filter starting to fail under the impact of the noisy sample. The RMSE shape for each estimate is unrecognizable and although there is an effort for correction that lowers the RMSE, ultimately the filter cannot get a stable state and it falls in the loop of the sensors being hindrances to each other (see Section 4.2).

$3000\,\%$ noise is the final transition point for the filter. It is where is completely breaks without being able to recover. The offset in the estimates is so big that neither the inertial or the visual data can help to make a correction. As mentioned on Chapter 4, all features will be invalidated if the state's pose is too distant from the real pose. In the previous experiments with other noise percentages, although the sample caused an immediate invalidation of landmarks which negatively affects the state's estimation, the pose was still plausible which meant new features could facilitate a recovery. However, this last sample makes it so all transformations using the state's pose are implausible, and therefore cause the filter to be unable to converge.

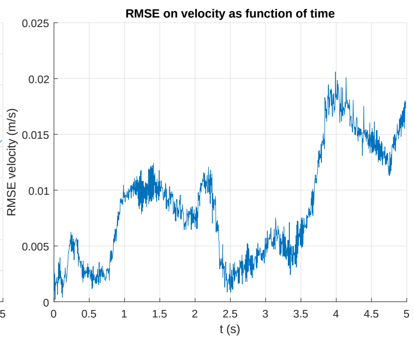
It is true that to converge, the filter must estimate a state that enables cooperation between sensors (as depicted in Chapter 4), but what this experiment showed is that once that state is reached, the filter becomes robust. It was necessary to introduce a sample with $1600\,\%$ noise in order to make the filter not converge.



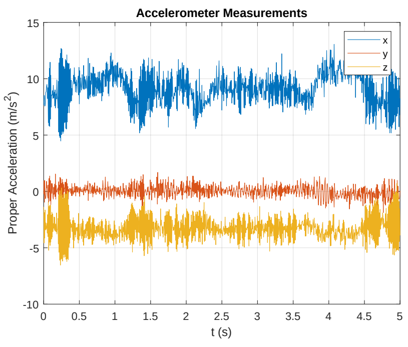
(a) Orientation RMSE



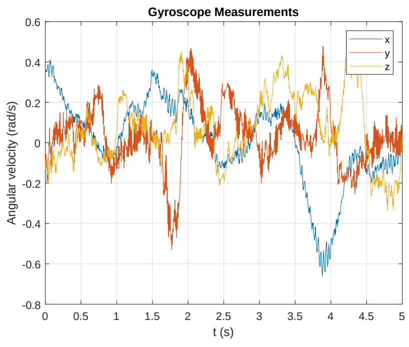
(b) Position RMSE



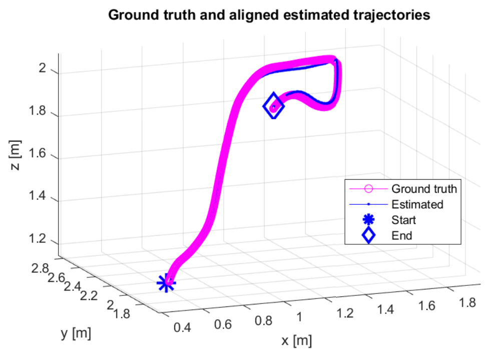
(c) Velocity RMSE



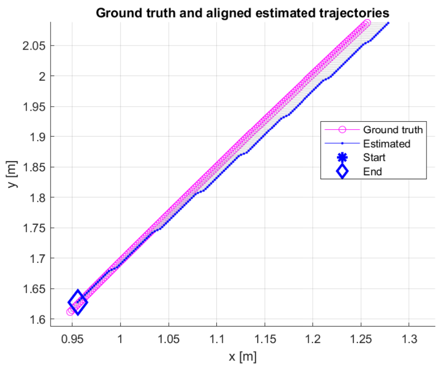
(d) Accelerometer measurements



(e) Gyroscope measurements

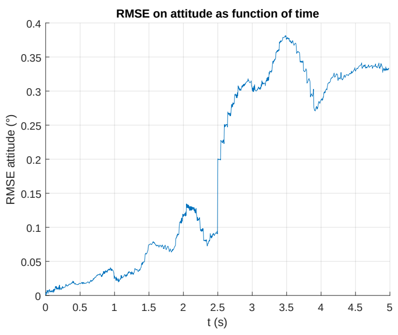


(f) Ground truth and estimates trajectories

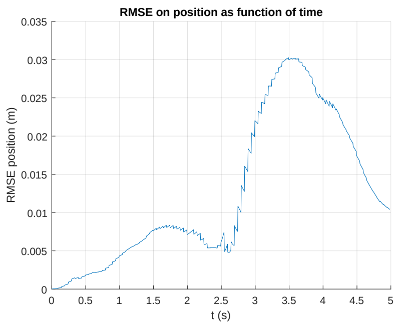


(g) Zoom on the trajectories' finish

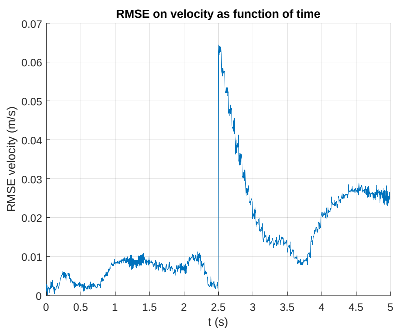
Figure 5.5: IMU samples of dataset *V1_02_medium* and results when running it on Fusion18.



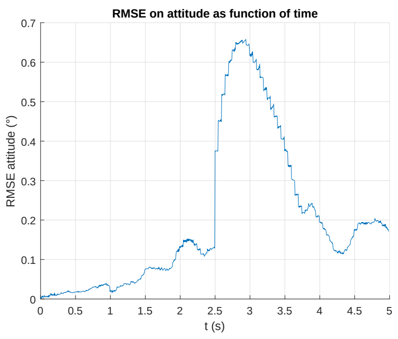
(a) Orientation RMSE 400 % error



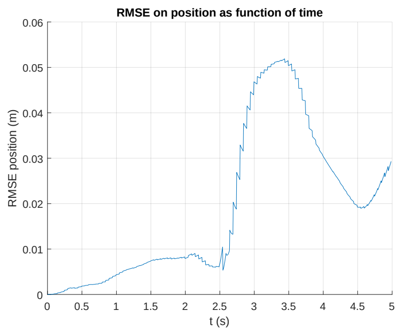
(b) Position RMSE 400 % error



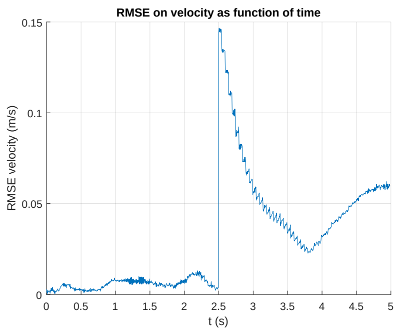
(c) Velocity RMSE 400 % error



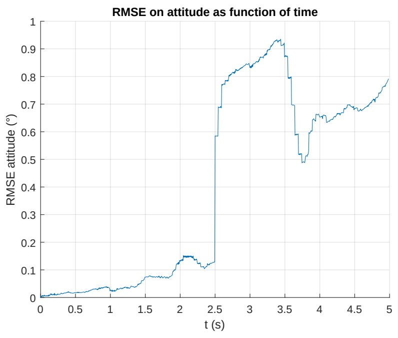
(d) Orientation RMSE 900 % error



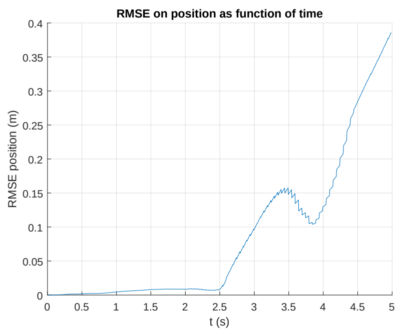
(e) Position RMSE 900 % error



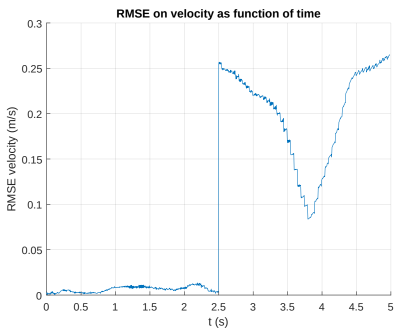
(f) Velocity RMSE 900 % error



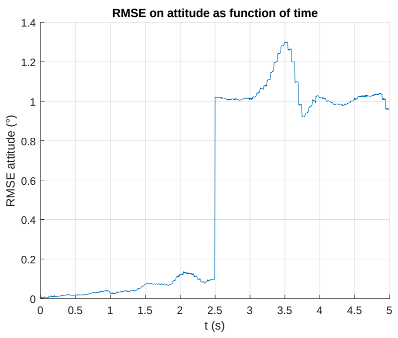
(g) Orientation RMSE 1600 % error



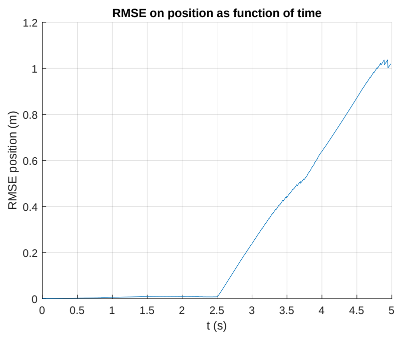
(h) Position RMSE 1600 % error



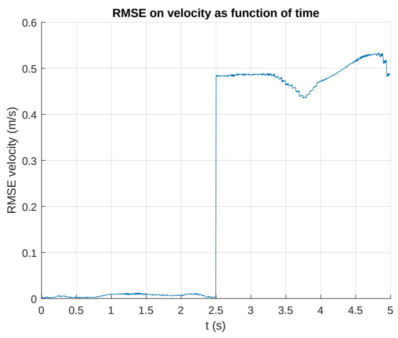
(i) Velocity RMSE 1600 % error



(j) Orientatio RMSE 3000 % error



(k) Position RMSE 3000 % error



(l) Velocity RMSE 3000 % error

Figure 5.6: Orientation, position and velocity RMSE for 400 %, 900 %, 1600 % and 3000 % error.

5.4 Mobile Robot Navigation

Before running Fusion18 with our datasets, let us go through some preliminary experiments associated to our modifications on Fusion18. First we compare the effects of drift when integration gyroscope measurements when subtracting the estimated bias from our proposed initialization (see Section 4.2). Then we study the filter's initial observation of landmarks and see how it behaves in different scenarios. Finally, we present the results when running Fusion18 with our datasets. Along this section we also touch upon multiple subjects that came up over the entire process that was enabling Fusion18 to run our own datasets with adequate results.

IMU Bias

To validate the gyroscope bias estimation we setup a small experiment using dataset *mf_04*. We estimate two values with the samples: angle and trajectory. In both we consider a case with the original raw values and another with the IMU samples after subtracting the estimated gyroscope bias. This allows us to see if removing the estimated bias prevents or reduces the estimates from drifting away from the correct values over time.

To estimate the angle we integrate the angular rate over time. As for the trajectories, we use the same propagation as the Fusion18 filter. Figure 5.7 shows the results for both cases. In Figure 5.7a, although slight, the effect of drift is noticeably smaller when removing the estimated gyroscope bias, since the line has a smaller slope.

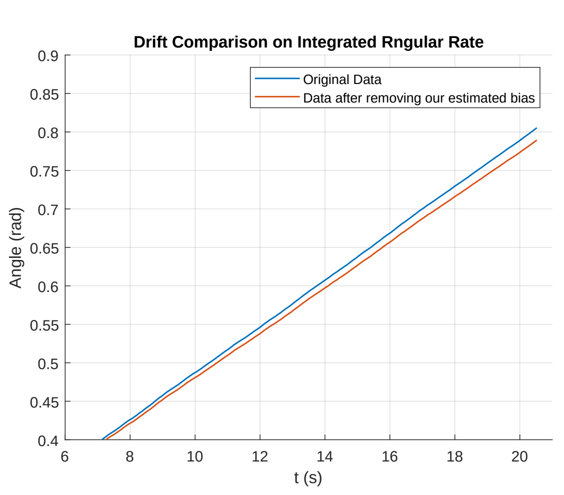
As for the trajectories, we should note they were aligned with our Procrustes metric and further processed with the methods present in Section 4.3. But this alignment was not equal for both, because if we aligned them both separately, we would not be able to see the drift effect. We aligned the trajectory with less bias and then used that transformation to align the biased trajectory. This allows us to see that, in fact, the subtraction of bias helps in reducing the effect of drift.

Although this experiment explicitly shows removing the estimated initial bias helps in the immediate estimations, it implies something more important. Estimating an initial bias that helps reduce the measurement's bias means we are facilitating the filters initialization process. Even if the bias is not showing a huge impact right away, what matters is that we have a better initial estimate than a null bias. This will help the filter get to a better bias value faster, and, as mentioned before, will help the IMU estimate a good pose, which will help in the observation of landmarks, which will help the bias estimation, and so on.

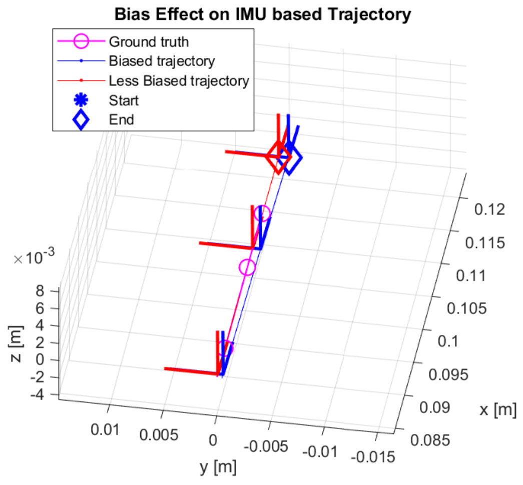
First Frames fusion with IMU Readings

One of the most broad problems we faced in this work, in terms of dependencies, was the handling of the initialization period. We approached this aspect in Section 4.2. We have detailed how the filter only uses landmarks that, when projected into the image frame (see Equation 2.16), have a small enough distance to the tracked features (20 pixel). Without features, the IMU biases will not be estimated, which causes a worse estimation of pose. And this same pose is used in the projection that can result in landmark invalidation. Also, as mentioned in Section 4.2, the filter has an initial period on which it will not replace the state's 3D landmarks, even if they all get invalidated. If we add of all this together we can see just how important this initialization period is. Most of the times it is the difference maker between the filter converging or not. So we have to make sure the first image frames can fuse with the IMU readings in order to create a positive work flow between these two sensors, where each sensor helps estimate properties for the other.

One measure we put in place to fight this problem (along with others like initial gyroscope bias estimation) was



(a) Drift effect on angles estimation



(b) Drift effect on IMU based trajectory estimation

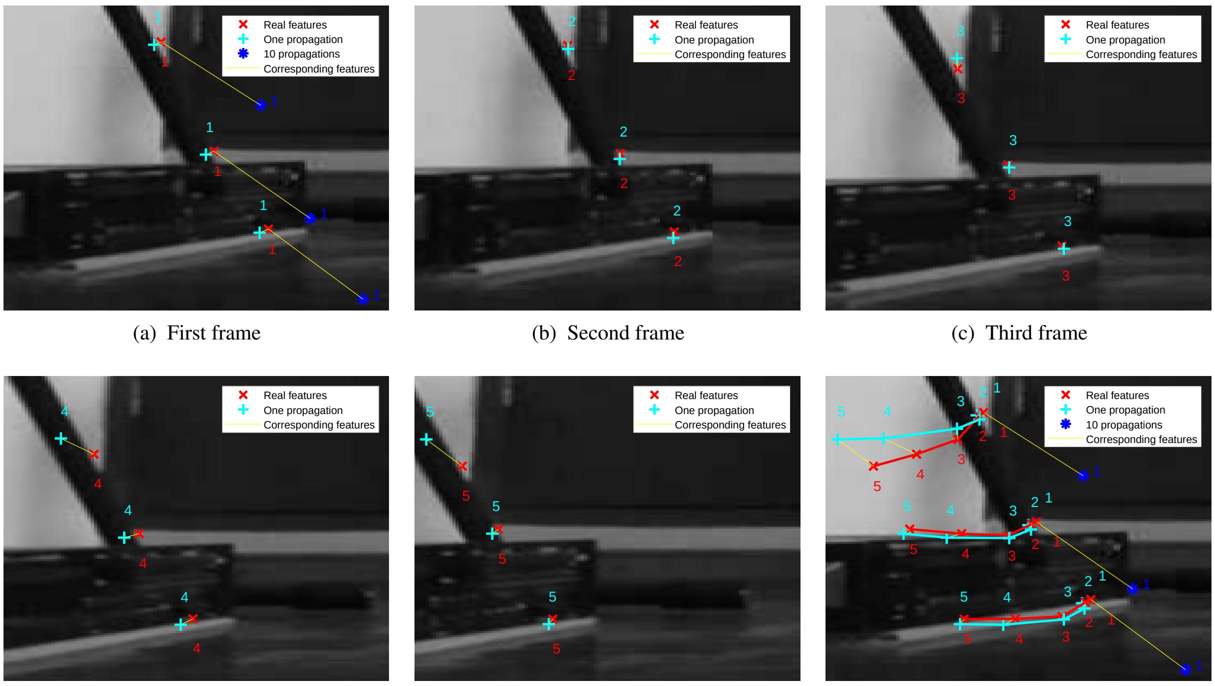
Figure 5.7: Comparing the effect of drift in angle (a) and trajectory (b) estimation for raw data and data after subtracting our estimated bias for dataset *mf_04*.

to remove all (but one) IMU samples that came temporally before the first image (not counting with the frame used for filter initialization), thus making sure there would only be one propagation before the first instance of landmark observation. With this we are trying to make sure the initial pose used to estimate the initial 3D landmarks is similar to the pose used in the first landmark observation. This way, most (or even all) initial landmarks are validated which results in an immediate start of bias estimation, thus beginning the cycle of cooperation between the two sensors.

To show the impact of this measure we plotted the real and estimated features (with and without the measure) in the first 5 frames that are used in the filter for observation. The results are in Figure 5.8, where red features are the real (tracked) features, cyan features are the estimated features after only one propagation (our measure), and dark blue features are the estimated features after 10 IMU propagations. For simplicity, we will refer to these scenarios by the color of their features. The reason why the blue features only appear in the first frame is because they were invalidated due to their error (with regards to the red features). The filter rendered these features as invalid so they would not deteriorate the state estimation. Hijacking the filter and demanding it to further estimate the state with these invalid landmarks resulted in reprojections outside the bounds of the image.

It is clear to see in Figure 5.8a that the number of propagations heavily influences the first projection (observation) of 3D landmarks. As said before, this is due to an erroneous body pose. But how exactly does the number of propagated IMU samples affect the pose, assuming everything is synchronized? Each time we propagate the state, we are effectively integrating the bias present in the measurements. As explained in Section 2.2, integrating bias results in a drift over time from the correct values, even if in a motionless scenario. This means the higher the number of times we propagate the state without a good bias estimate (to reduce the measurement bias), the higher the drift from reality.

Without this IMU sample removing strategy, and also the changes in feature management described in Section 4.2, all initial landmarks would get invalidated (as seen in Figure 5.8a) and then, because Fusion18 was set to only replace the state's landmarks after 120 IMU samples, the filter would run solely with the IMU. Due to bias,



(a) First frame (b) Second frame (c) Third frame

(d) Forth frame (e) Fifth frame (f) Feature tracking on first 5 frames

Figure 5.8: Feature tracking and estimation over the first 5 frames with and without IMU sample removal, dataset *mf_04*. In red we have the real (tracked) features. In cyan we have the estimated features with one initial propagation (our implemented measure). In dark blue we have the estimated features with 10 initial propagations.

by the time the filter replaced the landmarks, the pose would already be too far off from reality and the observation step would be unable to use any visual information, ultimately resulting in the filter diverging.

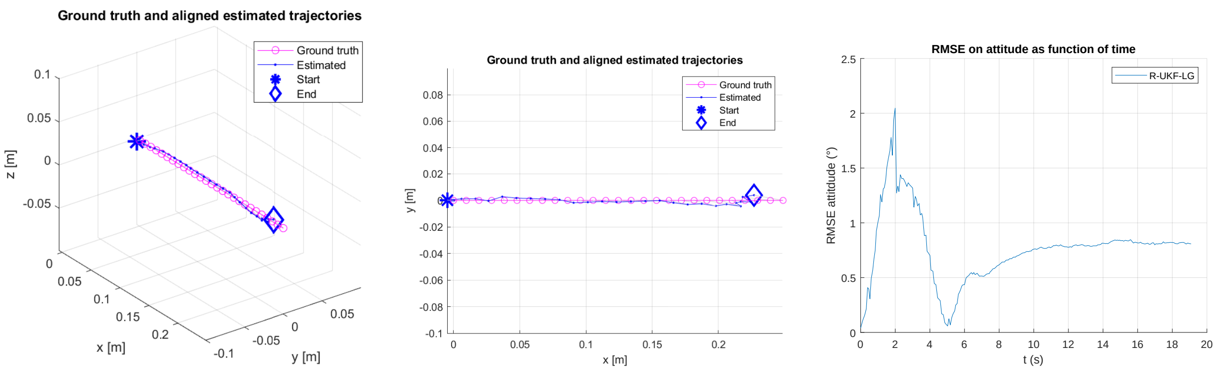
Our solution ensures the first landmark observation will not cause a general invalidation of features, which will provide us with an immediate start in IMU bias estimation. Although far from perfect, this provides us with adequate results that enable the filter to converge, as seen in Figure 5.8f.

Navigation Assessment

Let us finally detail the results of running Fusion18 with our own datasets. We consider the Procrustes error metric when running our own datasets, along with the point compression, re-sampling and repetition methods described in Section 4.3. Besides the benefits mentioned along with these methods, this allows us to evaluate the results in the same reference frame, since the ground truth is estimated while making assumptions different from the filter. The results for dataset *mf_04* are in Figure 5.9.

It is clear by looking at Figure 5.9c that the filter had an initialization period of 2 s before it had an estimation of the state that enabled sensor cooperation. This goes to show how important the first seconds of a dataset are. Which is why our datasets have an initial static period. It greatly helps the filter find a stable state that can create synergies.

The reason why the results with dataset *V1_01_medium* have such a seamless initialization period (Figure 5.5) is that the filter started with half of the state (pose and linear velocity) coming from ground truth values and



(a) Angled view of the trajectories (b) Down view of the trajectories (c) Orientation RMSE over time

Figure 5.9: Procrustes aligned estimated positions and ground truth positions point clouds from and angled view (a) and down view (b) along with orientation RMSE function (c) and when running the *mf_04* dataset.

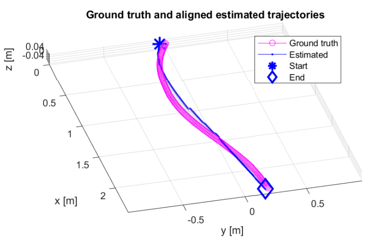
the other half (bias and 3D landmarks with corresponding features and backup features) coming from the result of running ORB-SLAM. This pretty much means the filter starts with values as close to reality as possible, and therefore, already has a state able to enable the synergy between both sensor's data. With the exception of the bias estimation, our initialization only requires initial measurements.

When running dataset *mc_01* we experimented with the process noise covariance, which is related to the IMU measurements, and by default resulted from a calibration explained in Appendix E. Navigation results on our dataset *mc_01* are shown in Figure 5.10. Trajectories are similar for both versions of the process noise. In Figure 5.10b we can notice the filter is not being able to arrange the synergy between sensors. To combat the early negative effects of the IMU bias, we increased the process noise covariance. As shown in Figure 5.10c, although not very robust, the synergy between sensors is present. However, around $t = 25\,\text{s}$ the RMSE has an almost vertical increase. This is due to the fact during the visual capture of dataset *mc_01* we were unable to have a stable luminosity (Figures 5.10d, 5.10e, 5.10f), which leads to detection and tracking of multiple feature in shadows. This will lead the vision portion of the filter to perceive an nonexistent movement and because we increased the process noise covariance, the filter will trust the vision more than the IMU. This is why this lighting problem does not appear to be present in Figure 5.10b.

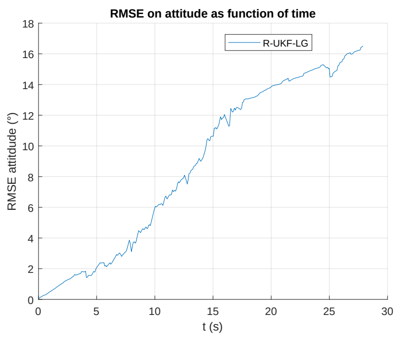
Overall, these results are well positive since they validate our initialization estimation and show it is possible to run our own acquired datasets with plausible results. But it left us with one final question: what if we started the filter with a longer initialization? What if we ran the filter up to a point and then started it again with the results of the first run? Would the second run be more robust due to its initialization? Would we notice a shorted initialization period? This is the final experiment we will present.

The steps for this experiment are: run Fusion18 on half of a dataset; save its final state along with all detected features and landmarks; run second half of the dataset but using this previous data to initialize it. The specific variables we are considering in this initialization are both sensor biases, the 3D landmark estimations and the reserve features along with their sighting in past images. We will still initialize the pose and linear velocity using ground truth.

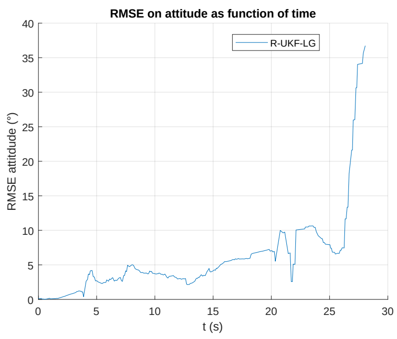
Let us consider the dataset *mfpf_02* specifically designed for the longer initialization experiment. The dataset *mfpf_02* was acquired in the same conditions as *mf_04*. Figures 5.3a, 5.3b and 5.3c show images of *mf_04* and illustrate the images of *mfpf_02* (not included as images are almost equal). In terms of movement, this dataset is



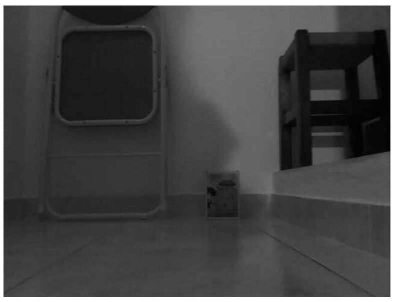
(a) Angled view of the trajectories

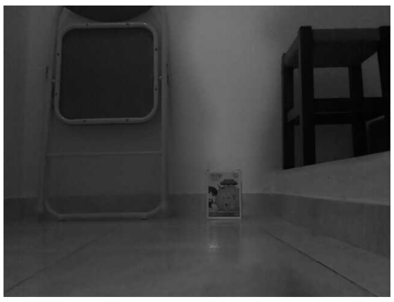


(b) Orientation RMSE over time



(c) Orientation RMSE over time for higher process noise


(d) Frame 108


(e) Frame 120

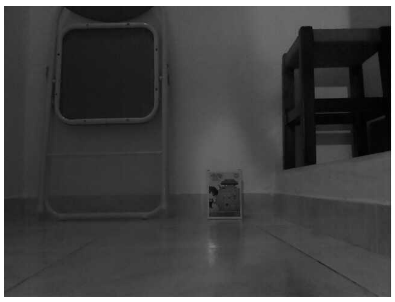

(f) Frame 128

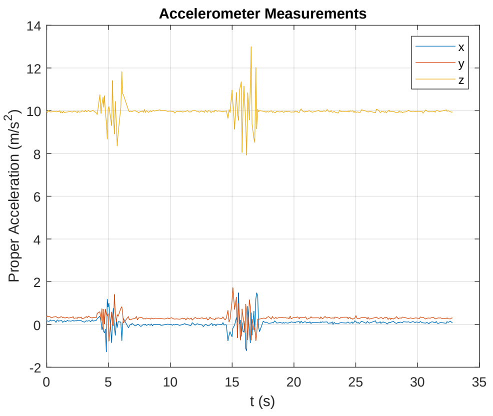
Figure 5.10: Procrustes aligned estimated positions and ground truth positions point clouds from and angled view (a) along with orientation RMSE function for the calibrated process noise (b) and for an increased process noise (c) when running the *mc_01* dataset. Image frames of *mc_01* that show uneven lighting.

also very similar to *mf_04*, except it is doubled. First the car has a motionless period to help the filter. Then it moves forwards and stops again. During this immobile period we will have our filter restart. After this the car moves forward again (in the same direction as before) and then stops, which (after a few seconds) marks the end of the acquisition. In sum we will run the filter on two similar trajectories that are separated by resting phase, on which we will make the filter reset. The IMU measurements for this dataset can be seen in Figure 5.11.

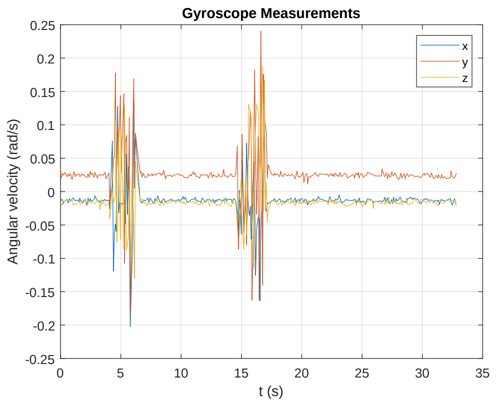
As expected, we will also perform this experiment using the initialization established in Section 4.2 to initialize the state after the reset. This will allow us to compare our generic and (mostly) online initialization to this calibration like approach. The estimated trajectories for both these scenarios are in Figure 5.12. In Figure 5.13 lie the corresponding orientation RMSE's.

Regarding trajectory, both initializations show good results. The trajectory with our initialization seems to indicate it better matched the movement of the car. However we have to take this comparison with care. This is because of the Procrustes based alignment performed to the trajectories. It only tells us they are plausible, but it does not allow us to directly compare two trajectories.

The orientation RMSE's show disparity in the results. The run with our initialization has a linearly increasing error which is the result of a inadequate initialization process. The run with the Fusion18 initialization answers our question: a stronger (longer) initialization results in a short, or almost non existing initialization period. It is similar to the results with Fusion18's default dataset (Figure 5.5), which possesses a strong initialization due to the fact it uses ground truth and estimated landmarks and bias from running ORB-SLAM [Mur-Artal et al., 2015].

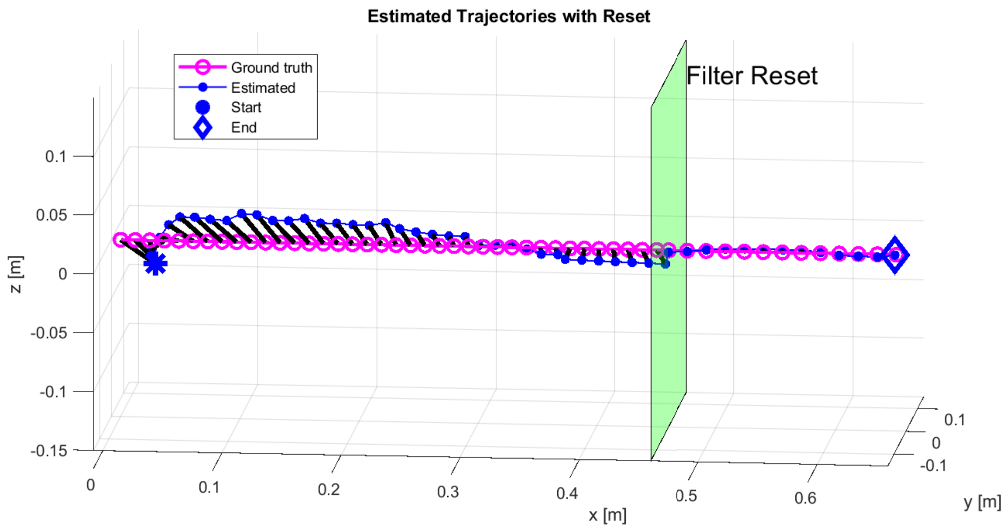


(a) Accelerometer measurements

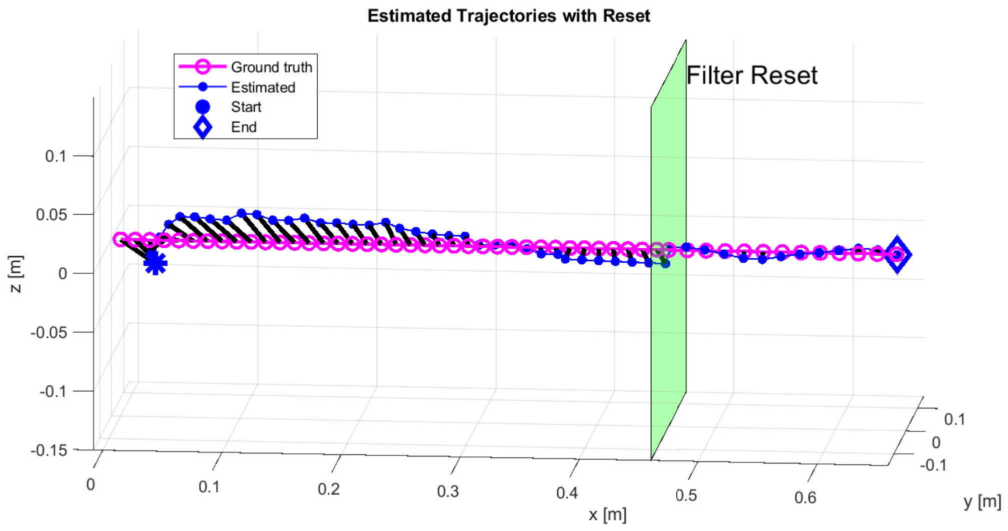


(b) Gyroscope measurements

Figure 5.11: *mfpf_02*'s accelerometer (a) and gyroscope (b) measurements.

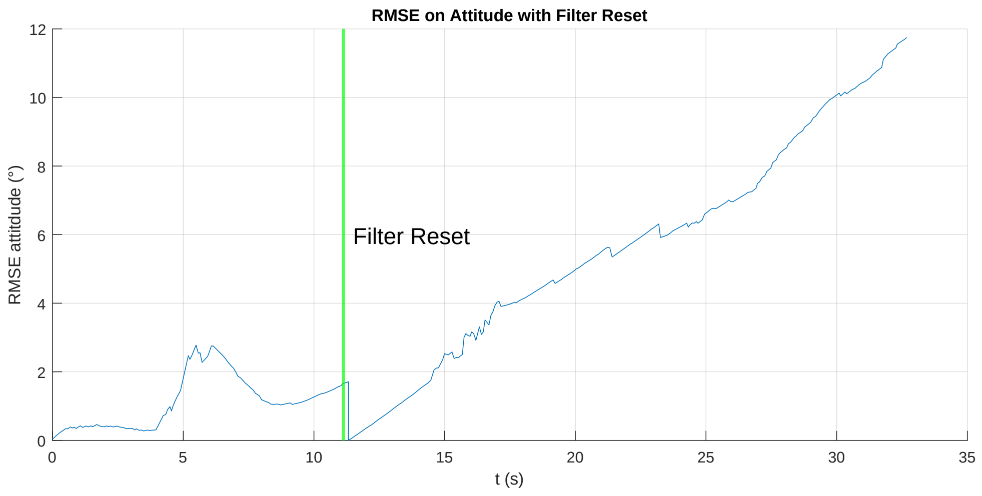


(a) Estimated trajectories with default initialization

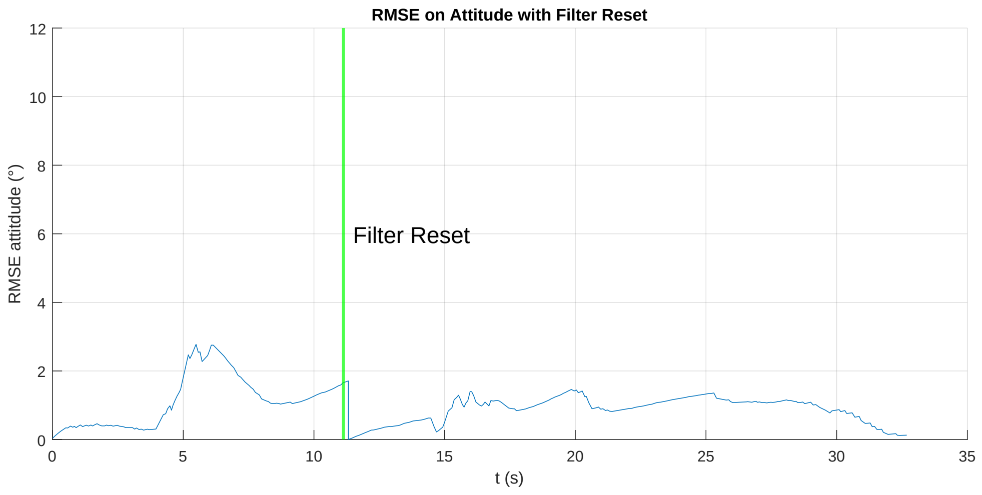


(b) Estimated trajectories with initialization from previous run

Figure 5.12: Ground truth and estimated trajectories when running the two halves of dataset *mfpf_02* with a reset in the middle. (a) uses our initialization after the reset. (b) uses as initialization the final state of the last half.



(a) Orientation RMSE with general initialization



(b) Orientation RMSE with Fusion18 estimated initialization

Figure 5.13: Orientation RMSE when running the two halves of dataset *mfpf_02* with a reset in the middle. (a) uses our initialization after the reset. (b) uses as initialization the final state of the last half.

Chapter 6

Conclusion and Future Work

This chapter summarizes the work developed, the proposed techniques, and the possible threads that can be pursued in future works. Section 6.1 lists these conclusions, and in Section 6.2 we will touch upon interesting topics for future works.

6.1 Conclusions

Our main point of interest with this work was to study the synergies between an IMU and a camera in the context of visual inertial odometry. It all started with the objective of acquiring a low-cost IMU and evaluate its viability in the context of pose estimation. There were some challenges dealing with the IMU, specifically due to its ongoing library development, but ultimately we were able to integrate it in our existing system and experimentally validate its measurements.

With this we were able to develop a system that was able to acquire visual inertial datasets. We developed a mobile robot that can acquire visual inertial datasets, based on a Raspberry Pi, a camera and an IMU. Although the acquisition was not perfect, it still managed to produce datasets that presented promising results in the context of pose estimation.

Our work proposes a state initialization for a Unscented Kalman Filter based on Lie groups. We were able to make an initial estimation of gyroscope bias that benefited the filter, and we also were able to estimate adequate initial 3D landmarks. We were able to use this entire filter to run over datasets with an irregular sample rate and still show adequate results.

The most crucial point of our work was how we experimentally saw the process that made the inertial and visual information synergize. We saw how important it was for the filter to quickly estimate the IMU bias so its measurements would not cause the state to drift from reality. Without this taking place we verified the landmark observations were too far from reality, which caused the filter to invalidate them. This would start a cycle of deterioration between both sensors. However, we were also able to experimentally get results that showed the opposite: a cycle of cooperation between sensors, where the bias estimation coming from the visual information would make the landmark projection more accurate, which in term would improve the bias estimation.

Ultimately we were able to advance towards the goal of having low-cost mobile home robots that can autonomously roam and perform tasks.

6.2 Future Work

There are plenty of opportunities for future works. Starting with the IMU, there is great potential in inexpensive IMUs. Traditionally, IMUs are used in systems to provide measurements, and any estimation is performed outside the IMU. However, we witnessed the quality of the onboard orientation estimation from our IMU. This opens plenty of new paths. Personally we think it would be interesting to develop a system where, instead of modeling the system around the IMU measurements, we could model it around the orientation coming directly from the IMU.

The topic of IMU sensor fusion for orientation estimation is something that, in our opinion, should be further explored. Filters like [Brossard et al., 2017] have a lot of merit for the results they display. However they opt for a sensor fusion approach of fusing the camera to the IMU. We believe this concept could be improved by first fusing each IMU sensor with a separated filter and only then fuse inertial and visual data. In a way we are suggesting a filter that lives inside another filter. This could also be seen as a similar suggestion to the last one, with the main difference being where the IMU sensor fusion would happen (onboard or externally). Overall, the current accessibility to IMUs should be capitalized on. Be it with a new filter, like the relatively recent Madgwick filter [Madgwick, 2010] that is currently one of the most used in filters for orientation estimation in digital motion processor, or with integration of existing technology.

It would be interesting to explore the aforementioned topics within the context of unreliable or unavailable visual information (dark rooms). To see if it is possible to commute to IMU filtering only, e.g. using Madgwick filtering and system dynamics, until it is possible to take again advantage of multi sensor synergies.

Regarding state initialization, we believe bias initialization methods should be pursued further. Again, IMUs with a DMP are able to make bias corrections onboard, which might be an interesting path to pursue. It would be interesting to study the performance of this real time calibration.

Appendix A

Kalman Filters

In this section we will go into detail about the general Kalman Filter and two of the most popular nonlinear variations of it. When talking about these filter we will assume a problem of sensor fusion where we can make predictions on the future state of the system based on its physical properties and, simultaneously, have a sensor measuring some physical property belonging or that can derive the state. The real question in a scenario like this is: what is the most optimal way of combining these two estimates? More intuitively, how do we know which estimate to trust more at each instant?

Kalman Filter

First introduced in [Kalman, 1960], the Kalman Filter is a mathematical method that proposes to solve problems like the one described above (although its full reach is much more sizable). The general idea is that it performs a prediction on the next state of the system based on its linear model and then it performs a correction to the prediction based on a measurement input. This correction is a weighted average between the prediction estimate and the measurement estimate. The focal point of this filter is to estimate this weight, which is usually named the Kalman gain K.

The system transition from time step $t-1$ to t can be generically modeled as

$$x_t = Fx_{t-1} + Bu_t + v_t, \tag{A.1}$$

where x is our target state space, F is the function that describes the evolution of the state without any control or noise, u is the control input with B that describes how the control changes the state, and v is the state process white Gaussian noise with covariance Q, $v \sim \mathcal{N}(0, Q)$.

The measurement (or observation) model can be expressed as

$$z_t = Hx_t + w_t, \tag{A.2}$$

where z is the measurement, H describes how to map the state to an observation, and w is the measurement white Gaussian noise with covariance R, $w \sim \mathcal{N}(0, R)$.

With these models we can formulate an optimal estimator with state covariance (uncertainty) P that provide a prediction as

$$\hat{x}_t = F\hat{x}_{t-1} + Bu_t \tag{A.3}$$

$$\hat{P}_t = FP_{t-1}F^T + Q \tag{A.4}$$

and a corrected with real measurements as

$$K_t = \hat{P}H^T(H\hat{P}H^T + R)^{-1} \tag{A.5}$$

$$x_t = \hat{x}_t + K_t(z_t - H\hat{x}_t) \tag{A.6}$$

$$P_t = (\mathbf{I} - K_tH)\hat{P}_t. \tag{A.7}$$

A final note is that, although this is an statistically optimal filter, it can only be used in linear systems. There are some nonlinear versions of this filter: the Extended Kalman Filter (EKF) and the Unscented Kalman Filter (UKF). In the next sections we will take a closer look into these alternatives.

Extended Kalman Filter

The approach the EKF takes into solving the nonlinearity problem is straightforward: let us locally linearize the functions using their first order derivative. As such the EKF nonlinear system models are

$$x_t = f(x_{t-1}, u_t, v_t), \tag{A.8}$$

$$z_t = h(x_t, w_t), \tag{A.9}$$

where f and h are nonlinear functions that model the state and measurements, respectively.

To perform this local linearization we must estimate the Jacobian matrices corresponding to the process and measurement models. These matrices will replace F and H as used in the Kalman Filter, and will be estimated at each iteration of the EKF. They are obtained as

$$F_t = \left.\frac{\partial f}{\partial x}\right|_{\hat{x}_{t-1|t-1}, u_t}, \tag{A.10}$$

$$H_t = \left.\frac{\partial h}{\partial x}\right|_{\hat{x}_{t|t-1}}. \tag{A.11}$$

Of course, because this is the most straightforward and simple solution to the nonlinear problem, the EKF will not have the best possible performance. That being said, it is still capable of providing quality estimates, and, due to its simple nature, is one of the most convenient solutions to our original problem o sensor fusion.

Unscented Kalman Filter

The main point of the KF is to make probabilistic decisions based on Gaussian uncertainty. However, when the system is nonlinear, these distributions cease to be Gaussian, and therefore the KF can not longer be used. The EKF locally linearizes the model functions so the uncertainty distributions stay Gaussian. The UKF, first introduced in [Julier & Uhlmann, 1997], has a different approach based on the unscented transformation. The state's uncertainty will again be represented as a Gaussian distribution, but this time it will be represented by a chosen minimal set of weighted (sigma χ) points. These points fully capture the true mean and covariance of this distribution. Figure A.1 compares this new approach to EKF.

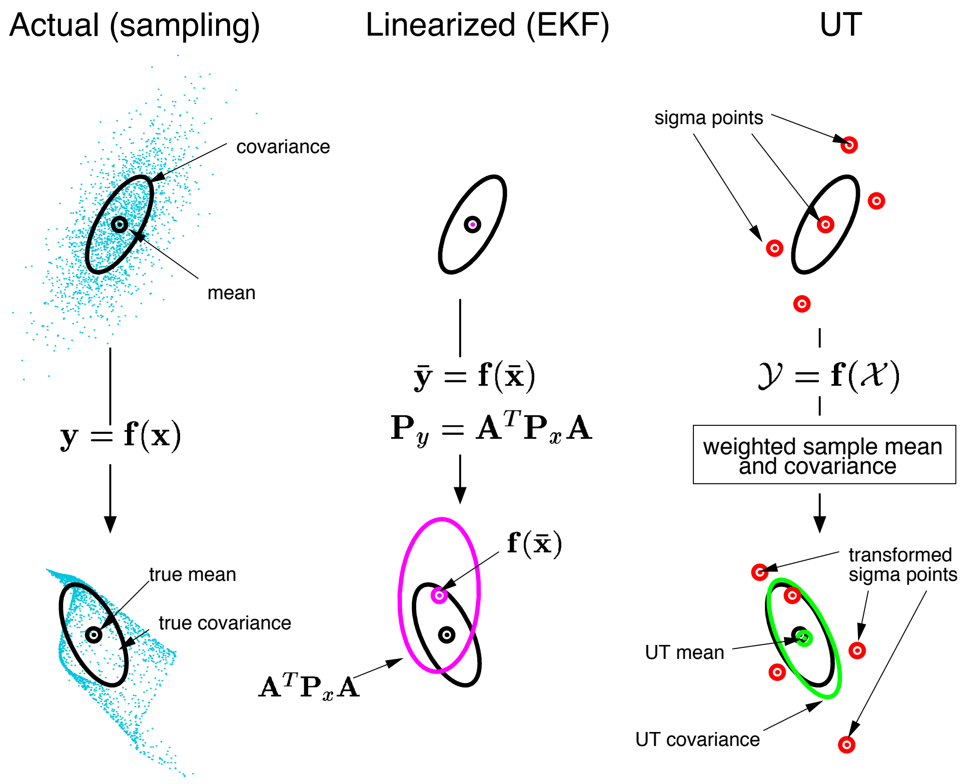

Actual (sampling)
Linearized (EKF)
UT
covariance
mean
$\mathbf{y} = \mathbf{f}(\mathbf{x})$
true mean
true covariance
$\bar{\mathbf{y}} = \mathbf{f}(\bar{\mathbf{x}})$
$\mathbf{P}_y = \mathbf{A}^T\mathbf{P}_x\mathbf{A}$
$\mathbf{f}(\bar{\mathbf{x}})$
$\mathbf{A}^T\mathbf{P}_x\mathbf{A}$
sigma points
$\mathcal{Y} = \mathbf{f}(\mathcal{X})$
weighted sample mean and covariance
transformed sigma points
UT mean
UT covariance

Figure A.1: Comparison between EKF and UKF in terms of uncertainty propagation. Left side shows the real mean and covariance. Middle presents the EKF first order linearization; Right showcases the unscented transformation used by UKF. From [Wan & Van Der Merwe, 2000].

For a state with dimension L, there will be $2L + 1$ sigma points χ that can be calculated as

$$\begin{aligned} \chi_0 &= \bar{x} \\ \chi_i &= \bar{x} + \left(\sqrt{(L+\lambda)P_x}\right)_i, \quad i = 1, \ldots, L \\ \chi_i &= \bar{x} - \left(\sqrt{(L+\lambda)P_x}\right)_{i-L}, \quad i = L+1, \ldots, 2L \end{aligned}, \tag{A.12}$$

with respective covariance and mean weights $W^{(c)}$ and $W^{(m)}$ as

$$\begin{aligned} W_0^{(m)} &= \frac{\lambda}{L+\lambda} \\ W_0^{(c)} &= \frac{\lambda}{L+\lambda} + (1 - \alpha^2 + \beta) \\ W_i^{(m)} &= W_i^{(c)} = \frac{1}{2(L+\lambda)}, \quad i = 1, \ldots, 2L \end{aligned}, \tag{A.13}$$

with $\lambda = \alpha^2(L + \kappa) - L$ is a scaling parameter, κ is a secondary scaling parameter (usually 0), α describes the spread of the points around its mean (usually small value like $1e-3$), and β is used to incorporate prior knowledge of the distribution ($\beta = 2$ is optimal for Gaussian distributions). $(.)_i$ is the ith row of a matrix.

We can propagate these sigma vectors through the nonlinear function as

$$y_i = f(\chi_i), \tag{A.14}$$

$$\bar{y} = \sum_{i=0}^{2L} W_i^{(m)} y_i, \tag{A.15}$$

$$P_y = \sum_{i=0}^{2L} W_i^{(c)} (y_i - \bar{y})(y_i - \bar{y})^T. \tag{A.16}$$

This approximation can capture the mean and covariance with accuracy to the third order (Taylor series expansion) for any nonlinearity. Overall, this filter presents a more accurate alternative to EKF, while maintaining the computational complexity in the same order. Equations A.17 to A.29 showcase the complete UKF algorithm.

Square-Root Unscented Kalman Filter (SR-UKF)

This final filter we will be addressing is the filter that got combined with the Lie group structure and results in Fusion18 [Brossard et al., 2017]. The SR-UKF, first introduced in [Van der Merwe & Wan, 2001], has mainly two advantages when compared with the traditional UKF: better computational performance and improved numerical properties. In UKF, the most computationally demanding step was the reoccurring estimation of sigma points at each update, because it implied calculating the matrix square-root of the state's covariance $P^{L\times L}$, which can be expressed as $P = SS^T$. An efficient implementation of UKF that uses Cholesky factorization has a complexity of $\mathcal{O}(L^3/6)$. The concept being SR-UKF is to propagate the square-root matrix S directly, which avoids refactorization at each time step. While the complexity remains similar to UKF (generally $\mathcal{O}(L^3)$, but can be $\mathcal{O}(L^2)$ for some specific problem formulations) this introduces the benefits mentioned above.

The three main linear algebra techniques used by this filter are:

- **QR Decomposition:** This is the factorization of a matrix $A \in \mathbb{R}^{L\times N}$, and can be expressed as $A^T = QR$, where $Q \in \mathbb{R}^{N\times N}$ is orthogonal, $R \in \mathbb{R}^{N\times L}$ is upper triangular, and $N \geq L$. $\tilde{R}$ (which is the upper triangular part of R) is the transpose of the Cholesky factor, that is,$\tilde{R} = S^T$. This implies $P = AA^T = \tilde{R}^T\tilde{R}$. The notation for the QR decomposition that returns this matrix $\tilde{R}$ is $\mathrm{qr}\{.\}$. The complexity of this operation is $\mathcal{O}(NL^2)$, while the Cholesky factorization results in the complexity of $\mathcal{O}(L^3)$ with added $\mathcal{O}(L^2)$ to form AA^T.

- **Cholesky Factor Updating:** If we considering S to be the Cholesky factor of $P = AA^T$, the Cholesky update with respects to $P = \pm\sqrt{v}uu^T$, and is symbolized as $S = \mathrm{cholupdate}\{S, u, \pm v\}$. This method is $\mathcal{O}(L^2)$ per update.

- **Efficient Least Squared:** The solutions of $(AA^T)x = A^Tb$ and $Ax = b$ are the same, and it can be found efficiently by using the QR decomposition with pivoting.

Equations A.30 to A.45 showcase the complete SR-UKF algorithm.

UKF Algorithm

Initialize with:

$$\bar{x}_0 = \mathbb{E}[x_0], \quad P_0 = \mathbb{E}\Big[(x_0 - \bar{x}_0)(x_0 - \bar{x}_0)^T\Big] \tag{A.17}$$

For $k \in \{1, \dots, \infty\}$
Calculate sigma points:

$$\chi_{k-1} = \Big[\bar{x}_{k-1} \quad \bar{x}_{k-1} + \rho\sqrt{P_{k-1}} \quad \bar{x}_{k-1} - \rho\sqrt{P_{k-1}}\Big] \tag{A.18}$$

Time update:

$$\chi_{k|k-1} = F[\chi_{k-1}, u_{k-1}] \tag{A.19}$$

$$\bar{x}_k^- = \sum_{i=0}^{2L} W_i^{(m)} \chi_{i,k|k-1} \tag{A.20}$$

$$P_k^- = \sum_{i=0}^{2L} W_i^{(c)} \Big(\chi_{i,k|k-1} - \bar{x}_k^-\Big)\Big(\chi_{i,k|k-1} - \bar{x}_k^-\Big)^T + R^v \tag{A.21}$$

$$\chi_{k|k-1} = \Big[\bar{x}_k^- \quad \bar{x}_k^- + \rho\sqrt{P_k^-} \quad \bar{x}_k^- - \rho\sqrt{P_k^-}\Big] \tag{A.22}$$

$$\mathcal{Y}_{k|k-1} = H[\chi_{k|k-1}] \tag{A.23}$$

$$\bar{y}_k^- = \sum_{i=0}^{2L} W_i^{(m)} \mathcal{Y}_{k|k-1} \tag{A.24}$$

Measurement update equations:

$$P_{y_k y_k} = \sum_{i=0}^{2L} W_i^{(c)} \Big(\mathcal{Y}_{i,k|k-1} - \bar{y}_k^-\Big)\Big(\mathcal{Y}_{i,k|k-1} - \bar{y}_k^-\Big)^T + R^n \tag{A.25}$$

$$P_{x_k y_k} = \sum_{i=0}^{2L} W_i^{(c)} \Big(\chi_{i,k|k-1} - \bar{x}_k^-\Big)\Big(\mathcal{Y}_{i,k|k-1} - \bar{y}_k^-\Big)^T \tag{A.26}$$

$$\mathcal{K}_k = P_{x_k y_k} P_{y_k y_k}^{-1} \tag{A.27}$$

$$\bar{x}_k = \bar{x}_k^- + \mathcal{K}_k(y_k - \bar{y}_k^-) \tag{A.28}$$

$$P_k = P_k^- - \mathcal{K}_k P_{y_k y_k} \mathcal{K}_k^T \tag{A.29}$$

with R^v as the process noise covariance and R^n as the measurement noise covariance.

SR-UKF Algorithm

Initialize with:

$$\bar{x}_0 = \mathbb{E}[x_0], \quad S_0 = \text{chol}\Big\{\mathbb{E}\Big[(x_0 - \bar{x}_0)(x_0 - \bar{x}_0)^T\Big]\Big\} \tag{A.30}$$

For $k \in \{1, \dots, \infty\}$

Calculate sigma points and time update:

$$\chi_{k-1} = \Big[\bar{x}_{k-1} \quad \bar{x}_{k-1} + \rho S_k \quad \bar{x}_{k-1} - \rho S_k\Big] \tag{A.31}$$

$$\chi_{k|k-1} = F[\chi_{k-1}, u_{k-1}] \tag{A.32}$$

$$\bar{x}_k^- = \sum_{i=0}^{2L} W_i^{(m)} \chi_{i,k|k-1} \tag{A.33}$$

$$S_k^- = \textbf{qr}\left[\left\{\sqrt{W_1^{(c)}}(\chi_{1:2L,k|k-1} - \bar{x}_k^-) \quad \sqrt{R^v}\right\}\right] \tag{A.34}$$

$$S_k^- = \textbf{cholupdate}\left\{S_k^-, \quad \chi_{0,k} - \bar{x}_k^-, \quad W_0^{(c)}\right\} \tag{A.35}$$

$$\chi_{k|k-1} = \Big[\bar{x}_k^- \quad \bar{x}_k^- + \rho S_k^- \quad \bar{x}_k^- - \rho S_k^-\Big] \tag{A.36}$$

$$\mathcal{Y}_{k|k-1} = H[\chi_{k|k-1}] \tag{A.37}$$

$$\bar{y}_k^- = \sum_{i=0}^{2L} W_i^{(m)} \mathcal{Y}_{k|k-1} \tag{A.38}$$

Measurement update equations:

$$S_{y_k} = \textbf{qr}\left[\left\{\sqrt{W_1^{(c)}}(\mathcal{Y}_{1:2L,k} - \bar{y}_k) \quad \sqrt{R_k^n}\right\}\right] \tag{A.39}$$

$$S_{y_k} = \textbf{cholupdate}\left\{S_{y_k}, \quad \mathcal{Y}_{0,k} - \bar{y}_k, \quad W_0^{(c)}\right\} \tag{A.40}$$

$$P_{x_k y_k} = \sum_{i=0}^{2L} W_i^{(c)} \Big(\chi_{i,k|k-1} - \bar{x}_k^-\Big)\Big(\mathcal{Y}_{i,k|k-1} - \bar{y}_k^-\Big)^T \tag{A.41}$$

$$\mathcal{K}_k = (P_{x_k y_k} / S_{y_k}^T) / S_{y_k} \tag{A.42}$$

$$\bar{x}_k = \bar{x}_k^- + \mathcal{K}_k (y_k - \bar{y}_k^-) \tag{A.43}$$

$$U = \mathcal{K}_k S_{y_k} \tag{A.44}$$

$$S_k = \textbf{cholupdate}\left\{S_k^-, \quad U, \quad -1\right\} \tag{A.45}$$

with R^v as the process noise covariance and R^n as the measurement noise covariance.

Appendix B

Quaternions

There are several ways to represent orientation. Probably the most intuitive way is to define the current rotation about a fixed coordinate system. One of the most computationally favorable ones is quaternions. Quaternions allows for a more compact and efficient representation of rotations while avoiding problems like gimbal lock. We will analyze what they are, what kinds of operations and how they allow and how they can be used in order to represent and handle spacial rotations.

We will be looking at what defines a quaternions, how they are used in the context of this problem and how to handle them in order to achieve the desired results. Depending on the context, quaternions can take different forms. We will explore how they are commonly used and represented in engineering, as in Book [Dixon, 2009].

B.1 Definition, Representation and Operations

There is a quaternion number system that extends complex number and defines a quaternion as

$$a + bi + cj + dk, \tag{B.1}$$

where a, b, c and d are real numbers and i, j and k are symbols that can be interpreted as unit-vectors. Each of this symbols points along one of the three spatial axes. This means a quaternions can be divided in two segments: scalar part and vector part.

As an extension of complex numbers, each complex symbol multiplied by it self results in -1:

$$i^2 = j^2 = k^2 = -1. \tag{B.2}$$

Multiplying symbols results in the following rules:

$$\begin{aligned} ij = k, &\quad ji = -k, \\ jk = i, &\quad kj = -i, \\ ki = j, &\quad ik = -j, \\ &ijk = -1. \end{aligned} \tag{B.3}$$

In engineering applications it is common for quaternions to be represented as a vector $Q = [q\ q_x\ q_y\ q_z]^T$. It is also common to work with unit quaternions which can be characterized by

$$|q|^2 = q_w^2 + q_x^2 + q_y^2 + q_z^2 = 1. \tag{B.4}$$

This allows for a 3 number representation since the fourth variable can be calculated knowing (B.4).

Addiction and Subtraction

Additions and subtractions of quaternions are straight forward: add or subtract each of the four component separately. For quaternions p and q we get:

$$\begin{aligned} p + q = (p_w + p_x i + p_y j + p_z k) + (q_w + q_x i + q_y j + q_z k) = \\ (p_w + q_w) + (p_x i + q_x i) + (p_y j + q_y j) + (p_z k + q_z k). \end{aligned} \tag{B.5}$$

This ensures the same properties of real number addiction and subtraction.

Multiplication

Considering two quaternions $p = p_w + p_x i + p_y j + p_z k$ and $q = q_w + q_x i + q_y j + q_z k$, their product, known as the Hamilton product, can be deducted using the distributive law by expanding the initial products and obtaining sums of products of basic elements:

$$\begin{aligned} pq = p \otimes q = & (p_w q_w - p_x q_x - p_y q_y - p_z q_z) \\ & +(p_w q_x + p_x q_w + p_y q_z - p_z q_y)i \\ & +(p_w q_y - p_x q_z + p_y q_w + p_z q_x)j \\ & +(p_w q_z + p_x q_y - p_y q_x + p_z q_w)k. \end{aligned} \tag{B.6}$$

This can be simplified as

$$\begin{aligned} (pq)_w &= p_w q_w - \vec{p} \cdot \vec{q}, \\ \vec{pq} &= \vec{p} \times \vec{q} + p_w \vec{p} + q_w \vec{q}, \end{aligned} \tag{B.7}$$

with $\vec{p}$, $\vec{q}$ and $\vec{pq}$ being the vector parts of p, q and pq, respectively.

Quaternions multiplication are associative and distributive, which respectively mean:

$$q_1 \otimes q_2 \otimes q_3 = (q_1 \otimes q_2) \otimes q_3 = q_1 \otimes (q_2 \otimes q_3) \tag{B.8}$$

and

$$q_1 \otimes (q_2 + q_3) = q_1 \otimes q_2 + q_1 \otimes q_3. \tag{B.9}$$

One of the most consequential properties of this multiplication is it being non-commutative. Looking at (B.6) and (B.3) it is intuitive why multiplication is non-commutative for quaternions.

Conjugate

Considering the quaternion $q = q_w + q_x i + q_y j + q_z k$, its conjugate is simply obtained by negating the vector / imaginary components:

$$q^{-1} = q_w - q_x i - q_y j - q_z k. \tag{B.10}$$

Some useful properties associated with this operation are:

$$(q^{-1})^{-1} = q, \tag{B.11}$$

$$(q + p)^{-1} = q^{-1} + p^{-1}, \tag{B.12}$$

$$(q \otimes p)^{-1} = q^{-1} \otimes p^{-1}. \tag{B.13}$$

B.2 Quaternions as 3D Rotations

According to Euler's rotation theorem, in a 3-dimensional space a displacement of a rigid body with one of its points staying fixed (3D rotation) is equivalent to a single rotation about an axis that goes through that fixed point. This means a 3D rotation can be represented as the combination of a unit vector $\vec{e} = (e_x, e_y, e_z) = e_x i + e_y j + e_z k$ that defines the rotation axis (Euler axis) and a scalar θ that defines the angle of rotation of the Euler axis (Figure B.1). This is known as the axis-angle representation.

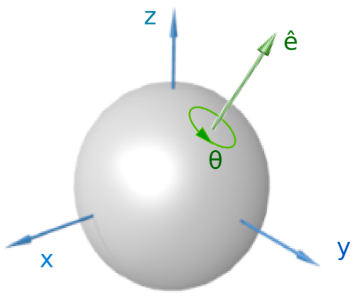


Figure B.1: Visualization of a rotation of a sphere in a three-dimensional space over $\vec{e}$ with an angle θ.

The fact a quaternion can be interpreted as the combination of a vector and a scalar perfectly matches the previous description of a 3D rotation. This change in representation can be done using a extension of Euler's formula:

$$e^{ix} = \cos x + i \sin x. \tag{B.14}$$

It results in the following expression for a quaternion q:

$$q = e^{\frac{\theta}{2}(u_x\mathbf{i} + u_y\mathbf{j} + u_z\mathbf{k})} = \cos\frac{\theta}{2} + (u_x\mathbf{i} + u_y\mathbf{j} + u_z\mathbf{k}) \sin\frac{\theta}{2} \tag{B.15}$$

Just like quaternion multiplication, a sequence of 3D rotations are not commutative as seen in Figure B.2. In order to obtain a single rotation resulting form the composition of a sequence of rotations, one only needs to multiply all rotation quaternions. For two rotation quaternions, q_1 and q_2, the resulting rotation q, which corresponds to the rotation q_1 followed by rotation q_2, can be attained as

$$q = q_2 q_1. \tag{B.16}$$

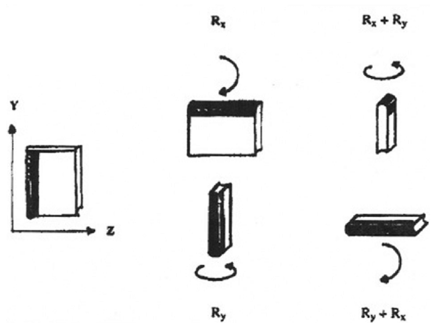


Figure B.2: Demonstration of why a sequence of rotations is not commutative.

If we consider a quaternion p with the real coordinate null, $p = p_x i + p_y j + p_z k$, it represents a simple 3D vector. If we consider it to be a position vector, then we can obtain the new position vector $p^{'}$ after a rotation q by performing

$$p^{'} = q \otimes p \otimes q^{-1}. \tag{B.17}$$

It is possible to manipulate this expression in order to obtain a more computationally friendly expression. Considering $\vec{q}$ as the vector part of q, q_w as the scalar part of q, $\vec{p}$ as the vector part of p, $\vec{p}^{'}$ as the vector part of $p^{'}$, and $\vec{h}$ vector as $\vec{h} = 2\vec{q} \times \vec{p}$, the resulting expression is

$$p^{'} = q \otimes p \otimes q^{-1} = (0, \vec{p}^{'}) = (0, \vec{p} + q_w \vec{h} + \vec{q} \times \vec{h}). \tag{B.18}$$

By converting a quaternion into a rotation matrix, (B.17) can be written as

$$p^{'} = Rp. \tag{B.19}$$

In the next chapter we will see what the matrix R looks like.

B.3 Conversions

There are multiple ways of representing rotations. We will see how rotation quaternions interact with those other conventions. All conversions assume the quaternion to be unitary.

Euler Angles

Considering the Euler angles to be ϕ, θ and ψ that represent rotations about the x-axis, y-axis and z-axis, respectively, the quaternion vector Q can be calculated as

$$Q = \begin{bmatrix} \cos(\phi/2)\cos(\theta/2)\cos(\psi/2) + \sin(\phi/2)\sin(\theta/2)\sin(\psi/2) \\ \sin(\phi/2)\cos(\theta/2)\cos(\psi/2) - \cos(\phi/2)\sin(\theta/2)\sin(\psi/2) \\ \cos(\phi/2)\sin(\theta/2)\cos(\psi/2) + \sin(\phi/2)\cos(\theta/2)\sin(\psi/2) \\ \cos(\phi/2)\cos(\theta/2)\sin(\psi/2) - \sin(\phi/2)\sin(\theta/2)\cos(\psi/2) \end{bmatrix} \tag{B.20}$$

A quaternion Q can be converted to the Euler angles following the relation

$$\begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} \text{arctan}\frac{2(q_0q_1+q_2q_3)}{1-2(q_1^2+q_2^2)} \\ \text{arcsin}(2(q_0q_2 - q_3q_1)) \\ \text{arctan}\frac{2(q_0q_3+q_1q_2)}{1-2(q_2^2+q_3^2)} \end{bmatrix} \tag{B.21}$$

Axis-Angle

For angle θ and axis $v = [x\ y\ z]^T$, we can recover the axis-angle representation from a quaternion as

$$\begin{aligned} \theta &= 2\arccos(q_w) \\ x &= \frac{q_x}{\sqrt{1-q_w^2}} \\ y &= \frac{q_y}{\sqrt{1-q_w^2}} \\ z &= \frac{q_z}{\sqrt{1-q_w^2}} \end{aligned} . \tag{B.22}$$

Rotation Matrices

The orthogonal matrix corresponding to a clockwise rotation is given by the homogeneous expression:

$$R = \begin{bmatrix} q_w^2 + q_x^2 - q_y^2 - q_z^2 & 2(q_xq_y - q_wq_z) & 2(q_wq_y + q_xq_z) \\ 2(q_xq_y + q_wq_z) & q_w^2 - q_x^2 + q_y^2 - q_z^2 & 2(q_yq_z - q_wq_x) \\ 2(q_xq_z - q_wq_y) & 2(q_wq_x + q_yq_z) & q_w^2 - q_x^2 - q_y^2 + q_z^2 \end{bmatrix} . \tag{B.23}$$

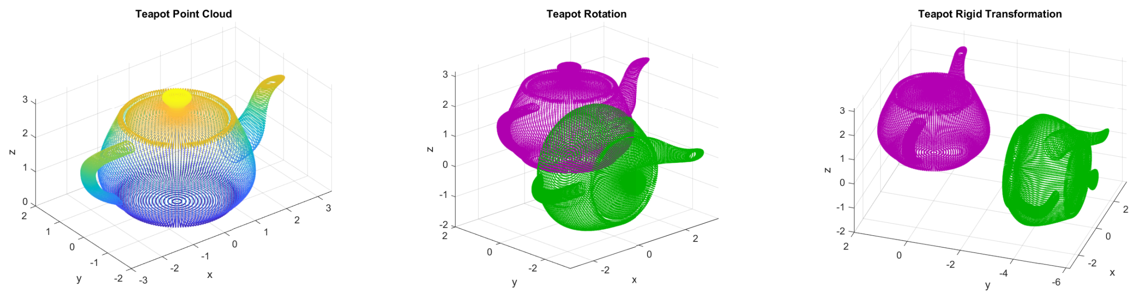
This is the matrix used in (B.19). If the quaternion is not a unit quaternion, that is, does not respect (B.4), then the matrix will be a scalar multiple of a rotation matrix.

B.4 Applications

In this section we will show some applications of quaternions. For that we will be using MATLAB's `teapot.ply` point cloud, shown in Figure B.3a.

Rotations

To perform a rotation over a point cloud with quaternions we can just apply (B.18) over every point.



(a) Teapot point cloud (b) Rotation (c) Rigid transformation

Figure B.3: Quaternion experiments with point clouds.

Rigid Transformations

There are two ways quaternions can represent rigid transformations: dual quaternions and quaternion-vector pair; Both this representations have 7 degrees of freedom, although the first stores 8 values.

By using dual quaternions, one is able to have a consistent representation of a rigid transformation. By reducing and simplifying the vector transformation equation using dual quaternions, we can arrive at (B.24), as shown in [Jia, 2013]. This means, as long as the rotation quaternion and translation vector are consistent in terms of coordinates, for this application, both approaches should be similar.

By using a quaternion-vector pair we can perform a rigid transformation over each point using

$$p^{'} = q \otimes p \otimes q^{-1} + t = (0, \vec{p}^{'}) + t = (0, \vec{p} + q_w \vec{h} + \vec{q} \times \vec{h} + t). \tag{B.24}$$

with t as the translation vector.

Appendix C

Sensor Fusion for Orientation Estimation

Sensor fusion consists of combining multiple data sources to generate a better understanding of the system. Examples include averaging two sensors for noise reduction, avoiding single point failure, estimating unmeasured states or increasing sensing coverage. Considering the 9 DoF IMU presented in Section 2.2 in this appendix we discuss sensor fusion to estimate orientation. We will start by highlighting the benefits of each sensor in this estimation. Then we will see how they can cooperate in order to mitigate each other's weaknesses.

Two different ways of estimating orientation are detailed here: (i) using both the accelerometer and the magnetometer, and (ii) using the gyroscope.

Accelerometer and Magnetometer

Before discussing how to estimate orientation with these two sensors, it is important that we understand what their readings look like in practice. For this, let us assume a static scenario. As said before, when static, the accelerometer only measures the gravitational acceleration vector pointing upward, since relative to a free fall, the accelerometer is accelerating upwards. The magnetometer measures the strength and direction of Earth's magnetic field. This means that it will measure a vector pointing to magnetic north. Due to the fact Earth's magnetic field lines extend from the inside of the planet to the outside, the measured vector will have a descending angle regarding the horizon, like shown in Figure C.1. This angle varies with Earth's location.

Our end goal with the orientation estimation for these sensors will be getting a rotation matrix. Let us (again) assume a static state first for simplicity. We can achieve this estimate with a Direction Cosine Matrix (DCM), which is equivalent to a rotation matrix (as shown in [Schaub & Junkins, 2009]). To estimate a DCM we need to find three axes: north, down and east. Each will be represented by a 3D vector. We can get down by using the accelerometer measurement, which in a static state should be $1\,\mathrm{g}$ pointing up. By performing the cross product of this down vector with the magnetometer's measurement we can get east. And finally north is the resulting cross product between east and down. These are the equations to find the three axes we need:

$$\begin{aligned} D &= -a, \\ E &= D \times m \\ N &= E \times D, \end{aligned} \tag{C.1}$$

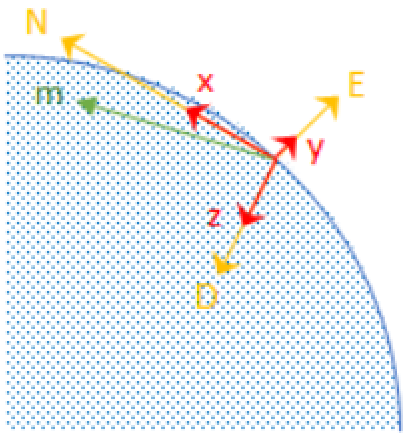


Figure C.1: Vectorial representation of vector measured by magnetometer (green).

with D, E and N being the down, east and north vectors, respectively. With the three axes we can now build a DCM as

$$R = DCM = [N\ E\ D]. \tag{C.2}$$

Figure C.2 depicts this estimation visually.

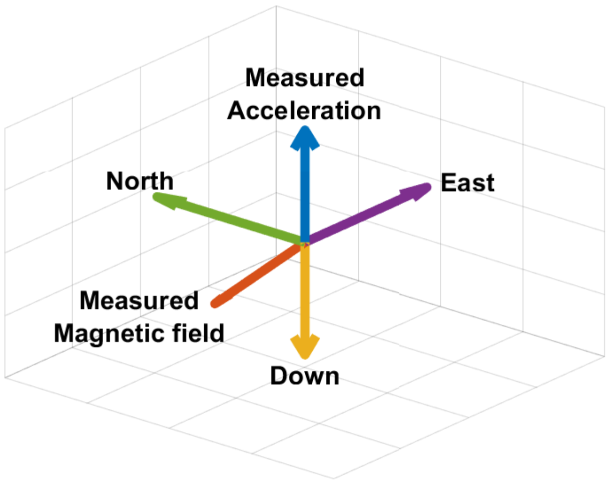


Figure C.2: Vectorial representation of how to estimate orientation with accelerometer and magnetometer readings.

The biggest advantage of this method for estimating orientation is the fact that it is an absolute estimation. There is no need for an initial estimation. However this method is easily prone to disturbances. In Section 2.2 we detail how hard and soft iron sources can disturb the magnetometer's readings. This can jeopardize the estimation of the east vector (and therefore north as well). In the same section we also talk about how we can reduce these disturbances, however, if they are external to the system, i.e., do not follow it, then it is not possible to avoid them.

One of the biggest drawbacks from this orientation estimation approach is how it expects the accelerometer to only measure the gravity vector. The accelerometer will measure all linear acceleration and even acceleration based on rotations not centered in the accelerometer. Every translation or rotation that generates linear acceleration will throw off the estimation of where the down axis is. This can be combated through some methods. For example, we can make a prediction of the linear acceleration caused by actuators and remove it from the measurements to try and single out the gravity vector. However this case assumes we have actuator and that we can use their commands to make a prediction about the linear acceleration. We can also discard measures that vary beyond a threshold of the $1\,\mathrm{g}$ measurement, but we would stop the orientation estimation in those cases.

Gyroscope

The gyroscope measures angular velocity. By integrating the gyroscope's measures in a certain time interval we can obtain the rotation performed over that amount of time. By doing this in small steps and applying the estimate to the previous orientation we can estimate the current orientation. This process is called dead reckoning. One of its flaws is the fact we need an initial estimate of orientation, due to this being a relative estimation. Depending on the system and the purpose behind this estimation, this might, or might not be a problem. For example, a system that is known to start with a period of no movement while parallel to the ground can just assume a null orientation (which is an identity matrix). However, if no such kinds of assumptions can be made, then it is necessary that this estimation is provided with an initial orientation. This can be done by using ground truth or making an initial estimation with an absolute method, like the one described with the accelerometer and gyroscope.

It is important to consider the effect of integrating bias (as shown in Figure 2.8). If not removed from the sensor readings, then the estimations will drift from reality. It is possible to use the accelerometer and magnetometer to correct for this drift in gyroscope based estimations.

Sensor Fusion Filters

Now that we saw two different ways of estimating orientation, their advantages and their disadvantages, let us discuss how we can combine estimates so they can compensate for each other's flaws. There are several filters that can do this job: Complementary, Kalman, or even dedicated filters like Madgwick or Mahony. From a conceptual standpoint they all typically do a similar job: they estimate orientation at each time step with the gyroscope measurements and use the accelerometer (and magnetometer if available) for the initial estimation and bias correction. Let us now take a closer look into one of these filters: Madgwick's.

Madgwick

This filter, first introduced in [Madgwick, 2010], has a straightforward strategy: estimate orientation using gyroscope, estimate orientation using accelerometer (and magnetometer if available), and finally combine the two estimates as a weighted mean. The estimation using the gyroscope is the same as described before:

$$\dot{q}_{\omega,t} = \frac{1}{2} q_{est,t-1} \otimes \begin{bmatrix} 1 \\ \omega_t \end{bmatrix}, \tag{C.3}$$

$$q_{\omega,t} = q_{est,t-1} + \dot{q}_{\omega,t} \Delta t, \tag{C.4}$$

where q_ω is a quaternions estimated using the gyroscope measurements, q_{est} is the previous orientation estimate, t is the current time step, ω is the gyroscope readings, and $\otimes$ is the quaternion multiplication (in Appendix B we explain what quaternions are in depth).

However, the same is not true for the accelerometer. The way this filter uses accelerometer measurements for the attitude estimation if by modeling this estimation as an optimization problem instead of directly computing the angle. This problem can be defined as

$$\min_{q_a \in \mathbb{R}^{4\times 1}} f(q_a, g, a) \tag{C.5}$$

with the cost function being

$$f(q_a, g, a) = q_a^{-1} \otimes \begin{bmatrix} 0 \\ g \end{bmatrix} \otimes q_a - \begin{bmatrix} 0 \\ a \end{bmatrix}, \tag{C.6}$$

where q_a is the orientation estimated using the accelerometer, g is the gravity vector and a is the accelerometer readings. As [Madgwick, 2010] proposes, this optimization problem can be solved with a gradient descent algorithm.

Finally, with (C.4) and (C.5), the orientation for a time step t can be estimated with weight γ as

$$q_{est,t} = \gamma q_{a,t} + (1-\gamma) q_{\omega,t}, \quad 0 \le \gamma \le 1. \tag{C.7}$$

Because this filter has a generic and straightforwards approach, it cannot explicitly track bias and model system dynamics. However, due to its accuracy and it being computational inexpensive, this is one of the most used filters for IMU orientation estimation in microprocessors.

Appendix D

Complete IMU Preintegration

This section details how to perform a complete IMU integration. Let us assume we have $i = 1, \dots, I$ images and $j = 1, \dots, J$ IMU samples in a given dataset, and also the world W and camera C reference frames. The goal of preintegration is to estimate the position α_i and velocity β_i, rotation γ_i differences between each consecutive pair of image frames. The first step is to define the dynamical model.

Given two time instants corresponding to two consecutive image frames i and $i+1$, we can propagate position p, velocity v and orientation q (or R) during the time interval $[t_i, t_{i+1}]$ (with duration Δt) in the world frame as

$$\begin{aligned} p_{i+1}^W &= p_i^W + v_i^W \Delta t + \int\int_{t\in[t_i,t_{i+1}]} \big(R_t^W(a_t - b_{a_t} - n_a) - g^W\big) dt^2 \\ v_{i+1}^W &= v_i^W + \int_{t\in[t_i,t_{i+1}]} \big(R_t^W(a_t - b_{a_t} - n_a) - g^W\big) dt \\ q_{i+1}^W &= q_i^W \otimes \int_{t\in[t_i,t_{i+1}]} \frac{1}{2}\Omega(\omega_t - b_{\omega_t} - n_\omega) q_t^C dt \end{aligned} \tag{D.1}$$

where

$$\Omega(\omega) = \begin{bmatrix} -[\omega]_\times & \omega \\ -\omega^T & 0 \end{bmatrix}. \tag{D.2}$$

and $[\omega]_\times$ is the skew-symmetric matrix formed with vector ω.

We will focus on a preintegrationg strategy because propagation strategy can be computationally demanding for optimization-based algorithms. With the rotation from W to C, $R_{W\to C}$, we can preintegrate the IMU measurements for a generic system as

$$\begin{aligned} R_{W\to C} * p_{i+1}^W &= R_{W\to C}\big(p_i^W + v_i^W - \frac{1}{2} g^W \Delta t^2\big) + \alpha_{i+1}^C \\ R_{W\to C} * v_{i+1}^W &= R_{W\to C}\big(v_i^W - g^W \Delta t\big) + \beta_{i+1}^C \\ q_{W\to C} \otimes q_{i+1}^W &= \gamma_{i+1}^C \end{aligned} , \tag{D.3}$$

where

$$\begin{aligned}\alpha^C_{i+1} &= \int\int_{t\in[t_i,t_{i+1}]} R^C_t(a_t - b_{a_t} - n_a)dt^2 \\ \beta^C_{i+1} &= \int_{t\in[t_i,t_{i+1}]} R^C_t(a_t - b_{a_t} - n_a)dt \\ \gamma^C_{i+1} &= \int_{t\in[t_i,t_{i+1}]} \frac{1}{2}\Omega(\omega_t - b_{\omega_t} - n_\omega)\gamma^C_t dt\end{aligned} \quad . \tag{D.4}$$

By using the Euler method for discretization we can propagate α, β and γ at each (IMU) step j so that $t_j \in [t_i, t_{i+1}]$ as

$$\begin{aligned}\alpha^C_{j+1} &= \alpha^C_j + \beta^C_j\delta t + \frac{1}{2}\mathbf{R}(\gamma^C_j)(a_j - b_{a_j})\delta t^2 \\ \beta^C_{j+1} &= \beta^C_j + \mathbf{R}(\gamma^C_j)(a_j - b_{a_j})\delta t \\ \gamma^C_{j+1} &= \gamma_j \otimes \begin{bmatrix} 1 \\ \frac{1}{2}(\omega_i - b_{\omega_j})\delta t\end{bmatrix}\end{aligned} \quad , \tag{D.5}$$

with their initial values being 0 and the quaternion identity, and $\mathbf{R}$ being the conversion from quaternion to rotation matrix.

Now let us focus on the covariance propagation. The rotation quaternion γ^C_t is overparameterized (four-dimensional). We will define its error term as a perturbation around its mean

$$\gamma^C_t \approx \gamma^C_t \otimes \begin{bmatrix} 1 \\ \frac{1}{2}\delta\theta^C_t \end{bmatrix}, \tag{D.6}$$

where $\delta\theta^C_t$ is a small 3D perturbation. The error terms of (D.4) can be derived as

$$\begin{bmatrix} \delta\dot{\alpha}^C_t \\ \delta\dot{\beta}^C_t \\ \delta\dot{\theta}^C_t \\ \delta\dot{b}^C_{a_t} \\ \delta\dot{b}^C_{\omega_t} \end{bmatrix} = \begin{bmatrix} 0 & \mathbf{I} & 0 & 0 & 0 \\ 0 & 0 & -R^C_t[a_t - b_{a_t}]_\times & -R^C_t & 0 \\ 0 & 0 & -[\omega_t - b_{\omega_t}]_\times & 0 & -\mathbf{I} \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \delta\alpha^C_t \\ \delta\beta^C_t \\ \delta\theta^C_t \\ \delta b^C_{a_t} \\ \delta b^C_{\omega_t} \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ -R^C_t & 0 & 0 & 0 \\ 0 & -\mathbf{I} & 0 & 0 \\ 0 & 0 & \mathbf{I} & 0 \\ 0 & 0 & 0 & \mathbf{I} \end{bmatrix} \begin{bmatrix} n_a \\ n_\omega \\ n_{b_a} \\ n_{b_\omega} \end{bmatrix} = F_t\delta z^C_t + G_t n_t. \tag{D.7}$$

With the chosen discretization, F_t is constant over the integration period, which means, for a time step δt, $F_d \approx \mathbf{I} + F_t\delta t$. For continuous time noise covariance matrix $Q_t = diag(\sigma^2_a, \sigma^2_\omega, \sigma^2_{b_a}, \sigma^2_{b_\omega})$, we can estimate the discrete time noise covariance Q_d as

$$\begin{aligned} Q_d &= \int_0^{\delta t} F_d(\tau)G_tQ_tG_t^TF_d(\tau)^T \\ &= \delta t F_d G_t Q_t G_t^T F_d^T \\ &\approx \delta t G_t Q_t G_t^T. \end{aligned} \tag{D.8}$$

Finally we can propagate the covariance P^C_{i+1} (with the first covariance being a zeros matrix) as

$$P^C_{t+\delta t} = (\mathbf{I} + F_t \delta t) P^C_t (\mathbf{I} + F_t \delta t)^T + deltat G_t Q_t G^T_t, \quad t \in [t_i, t_{i+1}]. \tag{D.9}$$

The first order Jacobian matrix, which is the matrix we need to make the aligned for our gyroscope bias estimation, can be propagated (with initial $J = \mathbf{I}$) as

$$J_{t+\delta t} = (\mathbf{I} + F_t \delta t) J_t, \quad t \in [t_i, t_{i+1}]. \tag{D.10}$$

After estimating the bias we can avoid repeating this entire process by making a first order approximation of α^C_i, β^C_i, and γ^C_i:

$$\begin{aligned} \alpha^C_{i+1} &\approx \hat{\alpha}^C_{i+1} + J^\alpha_{b_a} \delta b_a + J^\alpha_{b_\omega} \delta b_\omega \\ \beta^C_{i+1} &\approx \hat{\beta}^C_{i+1} + J^\beta_{b_a} \delta b_a + J^\beta_{b_\omega} \delta b_\omega \\ \gamma^C_{i+1} &\approx \hat{\gamma}_{i+1} \otimes \begin{bmatrix} 1 \\ \frac{1}{2} J^\gamma_{b_\omega} \delta b_\omega \end{bmatrix} \end{aligned}, \tag{D.11}$$

where $J^\alpha_{b_a}$ is a 3×3 matrix contained in matrix J_{i+1} such as, for a general J,

$$J = \begin{bmatrix} J^\alpha & J^\alpha_\beta & J^\alpha_\gamma & J^\alpha_{b_a} & J^\alpha_{b_\omega} \\ J^\beta_\alpha & J^\beta & J^\beta_\gamma & J^\beta_{b_a} & J^\beta_{b_\omega} \\ J^\gamma_\alpha & J^\gamma_\beta & J^\gamma & J^\gamma_{b_a} & J^\gamma_{b_\omega} \\ J^{b_a}_\alpha & J^{b_a}_\beta & J^{b_a}_\gamma & J^{b_a} & J^{b_a}_{b_\omega} \\ J^{b_\omega}_\alpha & J^{b_\omega}_\beta & J^{b_\omega}_\gamma & J^{b_\omega}_{b_a} & J^{b_\omega} \end{bmatrix}, \tag{D.12}$$

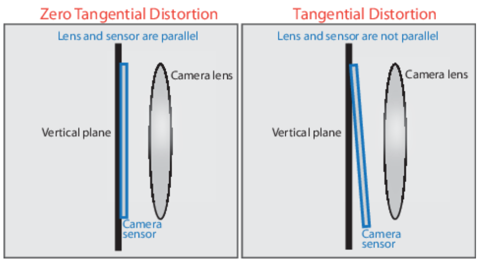
assuming the order in (D.7).

Appendix E

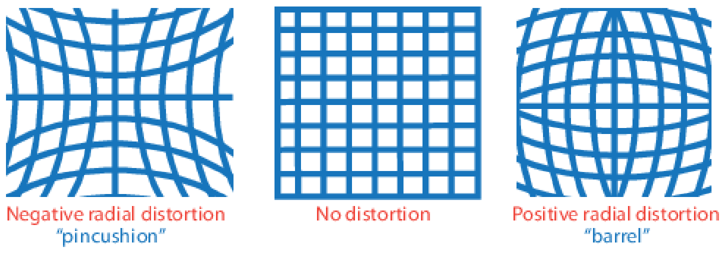
Sensor Calibration

E.1 Camera Calibration

All camera captured images are the result of a transformation between certain 3D points in the real world and 2D points in the image plane. This transformation can be affected by the camera's physical properties in the form of distortions. These can be modelled as tangential and radial. The first is the result of the lens and image plane not being parallel (Figure E.1a) while the second occurs when light rays bend more near the edges of a lens than they do at its optical center (Figure E.1b). The goal of camera calibration is to estimate this transformation (intrinsic and extrinsic parameters) along with these distortion coefficients.



(a) Tangential distortion



(b) Radial distortion

Figure E.1: Visual examples of tangential (a) and radial (b) distortion in a camera.

Because the extrinsic parameters are external to the camera, i.e., vary depending on the problem we are considering, we can assume the world frame is centered in the image frame, which in practice means we are ignoring them for the calibration. We can estimate the camera's intrinsic parameters and distortion coefficients using the MATLAB single camera calibrator app [1]. This app uses a simple odd chess pattern (Figure E.2) to find the intrinsic parameters along with the distortion coefficients (according to a given camera model).

The first step in the calibration is to acquiring images. The images should capture the pattern at different orientations (relative to the camera) and at the distances objects of interest will be in real applications. There should be images that excite all segments of the image frame, because lens distortion increases radially from the

[1]https://www.mathworks.com/help/vision/ug/using-the-single-camera-calibrator-app.html

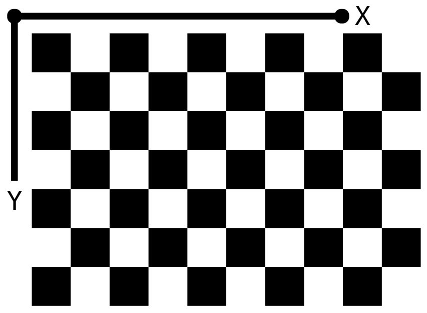


Figure E.2: Checkerboard pattern used in the camera calibration.

center of the image. So it is important that the pattern is captured near the edges of images so the algorithm can obtain good estimates of the distortion coefficients.

E.2 IMU to Camera Parameters Estimation

In order to run our own datasets we need to estimate the extrinsic parameters between IMU and camera. We used Kalibr toolbox[2] for this purpose. This ROS toolbox allows us to perform a spacial and temporal calibration of a visual inertial system (camera-IMU). It is important to point out that to perform this calibration we need a body that incorporated these sensors and that they remain fixed. If the sensor placement changes, a new calibration will be required.

The requirements for this calibration were to provide the toolbox with the IMU and camera's intrinsic parameters. For the IMU, this consists in the noise density and random walk for both the accelerometer and gyroscope. In a perfect world, these would come in the datasheet provided by the manufacturer. However, this is often not true. And even if it is, there are external factors that might end up floating these parameters, like temperature or usage.

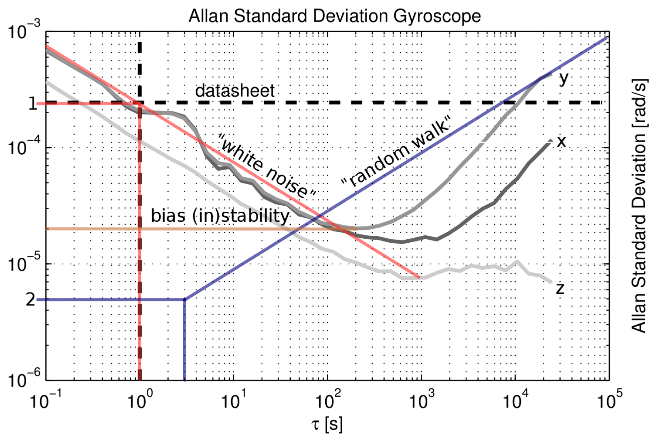


Figure E.3: Allan standard deviation of a gyroscope with the resulting lines that feature the noise and random walk, from the Kalibr wiki.

[2]https://github.com/ethz-asl/kalibr

These required parameters can be estimated using kalibr_allan toolbox [3]. This is an Matlab toolbox designed to be used in tandem with the Kalibr toolbox, although it can be used for other purposes. It uses the method introduced by [IEEE, 1998]. This method is performed over an acquired IMU dataset (accelerometer and gyroscope) with a long duration (at least 4 hours) while standing still. It consists in creating a plot with the log-log Allan deviation (Figure E.3) and fitting two lines into it. One with with $-1/2$ slope for the white noise and one with $1/2$ slope for the random walk.

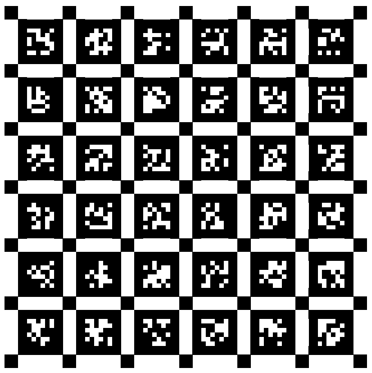


Figure E.4: Aprilgrid target used in the IMU-camera calibration.

With all the required parameters, we can now obtain the visual-inertial data for calibration. This dataset needs to visually capture an Aprilgrid target, like then one shown in Figure E.4. This specific target has two advantages when compared to a checkerboard or circlegrid: a partially visible boards works and the target's pose is completely characterized, meaning, it is invulnerable to flipping problems like an upside down view of the target. For an optimal calibration, there are several movements that should be performed. All movements should be slow and smooth and should be performed at least three times each. This setup assumes the target to be the central point of our coordinate system. This implies the rotation movements are done around the target and not around the body's center.

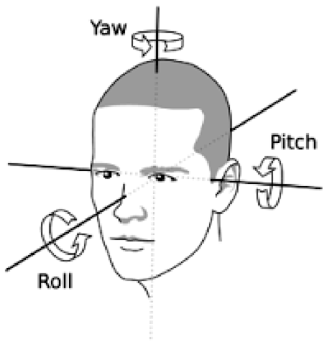


Figure E.5: The yaw, pitch and roll angles, with their respective axes, in a human head.

To acquire this dataset, we used a wired connection to the IMU in order to get the highest acquisition rate possible. With this setup we were able to achieve a rate of 56 Hz. The same wasn't necessary for the camera since,

[3]`https://github.com/rpng/kalibr_allan`

with the setup of Chapter 3, we were already able to, comfortably, achieve a frequency of 20 Hz.

During the capture, first, we rotate the body over all axes of the body must be excited (Figure E.5) while keeping the full target in view. This allows a proper gyroscope and camera calibration. Then, for the accelerometer, we move the body along those same three axes: up and down, side to side and front and back. For these movements, it is not super important to capture the full view of the target. Finally, we should include some arbitrary movements with the target in full view for added accuracy.

Bibliography

[Barfoot & Furgale, 2014] Barfoot, T. D. & Furgale, P. T. (2014). Associating Uncertainty With Three-Dimensional Poses for Use in Estimation Problems. *IEEE Transactions on Robotics*, 30(3), 679–693.

[Barrau & Bonnabel, 2015] Barrau, A. & Bonnabel, S. (2015). An EKF-SLAM algorithm with consistency properties. *Computing Research Repository*, abs/1510.06263.

[Barrau & Bonnabel, 2018] Barrau, A. & Bonnabel, S. (2018). Invariant Kalman Filtering. *Annual Review of Control, Robotics, and Autonomous Systems*, 1.

[Bo et al., 2020] Bo, L., Li, L., & Liu, H. (2020). SoC Implementation of Visual-inertial Odometry for Low-cost Ground Robots. *Journal of Physics: Conference Series*, 1453, 012091.

[Brossard et al., 2017] Brossard, M., Bonnabel, S., & Barrau, A. (2017). Unscented Kalman Filtering on Lie Groups for Fusion of IMU and Monocular Vision. *International Conference on Robotics and Automation (ICRA)*.

[Burri et al., 2016] Burri, M., Nikolic, J., Gohl, P., Schneider, T., Rehder, J., Omari, S., Achtelik, M. W., & Siegwart, R. (2016). The EuRoC micro aerial vehicle datasets. *The International Journal of Robotics Research*, 35.

[Campos et al., 2020] Campos, C., Elvira, R., Gómez, J., Montiel, J., & Tardós, J. D. (2020). ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual-Inertial and Multi-Map SLAM. *arXiv preprint arXiv:2007.11898*.

[Civera et al., 2008] Civera, J., Davison, A. J., & Montiel, J. M. M. (2008). Inverse Depth Parametrization for Monocular SLAM. *IEEE Transactions on Robotics*, 24(5), 932–945.

[Cruz, 2015] Cruz, J. (2015). Wireless Mobile Camera. Master's thesis, Instituto Superior Técnico.

[Davison, 2003] Davison, A. J. (2003). Real-time simultaneous localisation and mapping with a single camera. In *Proceedings Ninth IEEE International Conference on Computer Vision* (pp. 1403–1410 vol.2).

[Dixon, 2009] Dixon, J. C. (2009). Appendix D: Quaternions for Engineers. In *Suspension Geometry and Computation* (pp. 381–392). John Wiley & Sons.

[Fitzgibbon & Zisserman, 1998] Fitzgibbon, A. & Zisserman, A. (1998). Automatic Camera Recovery for Closed or Open Image Sequences. *European Conference on Computer Vision*.

[IEEE, 1998] IEEE (1998). IEEE Standard Specification Format Guide and Test Procedure for Single-Axis Interferometric Fiber Optic Gyros. *IEEE Std 952-1997*, (pp. 1–84).

[Jia, 2013] Jia, Y. (2013). Applications of Dual Quaternions in Three Dimensional Transformation and Interpolation. *Iowa State University: Ames, IA, USA*.

[Julier & Uhlmann, 1997] Julier, S. J. & Uhlmann, J. K. (1997). New extension of the Kalman filter to nonlinear systems. In I. Kadar (Ed.), *Signal Processing, Sensor Fusion, and Target Recognition VI*, volume 3068 (pp. 182 – 193).: International Society for Optics and Photonics SPIE.

[Kalman, 1960] Kalman, R. E. (1960). A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D), 35–45.

[Karlsson et al., 2005] Karlsson, N., di Bernardo, E., Ostrowski, J., Goncalves, L., Pirjanian, P., & Munich, M. (2005). The vSLAM Algorithm for Robust Localization and Mapping. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation* (pp. 24–29).

[Kendall, 1989] Kendall, D. G. (1989). A survey of the statistical theory of shape. *Statistical Science*, 4(2), 87–99.

[Liu et al., 2014] Liu, Y., Noguchi, N., & Ishii, K. (2014). Development of a Low-cost IMU by Using Sensor Fusion for Attitude Angle Estimation. *IFAC Proceedings Volumes*, 47(3), 4435–4440.

[Lothe et al., 2010] Lothe, P., Bourgeois, S., Royer, E., Dhome, M., & Naudet, S. (2010). Real-Time Vehicle Global Localisation with a Single Camera in Dense Urban Areas: Exploitation of Coarse 3D City Models. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (pp. 863–870).

[Madgwick, 2010] Madgwick, S. (2010). An efficient orientation filter for inertial and inertial/magnetic sensor arrays. *Report x-io and University of Bristol (UK)*, 25, 113–118.

[Mur-Artal et al., 2015] Mur-Artal, R., Montiel, J., & Tardós, J. D. (2015). ORB-SLAM: a Versatile and Accurate Monocular SLAM System. *IEEE Transactions on Robotics*, 31(5), 1147–1163.

[Mur-Artal & Tardos, 2017a] Mur-Artal, R. & Tardos, J. D. (2017a). ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras. *IEEE Transactions on Robotics*, 33(5), 1255–1262.

[Mur-Artal & Tardos, 2017b] Mur-Artal, R. & Tardos, J. D. (2017b). Visual-Inertial Monocular SLAM With Map Reuse. *IEEE Robotics and Automation Letters*, 2(2), 796—803.

[Park et al., 1995] Park, F., Bobrow, J., & Ploen, S. (1995). A Lie Group Formulation of Robot Dynamics. *The International Journal of Robotics Research*, 14(6), 609–618.

[Passaro et al., 2017] Passaro, V., Cuccovillo, A., Vaiani, L., De Carlo, M., & Campanella, C. E. (2017). Gyroscope Technology and Applications: A Review in the Industrial Perspective. *Sensors*, 17, 2284.

[Piras & Dabove, 2016] Piras, M. & Dabove, P. (2016). Comparison of two different mass-market IMU generations: Bias analyses and real time applications. In *2016 IEEE/ION Position, Location and Navigation Symposium (PLANS)* (pp. 34–41).

[Potirakis et al., 2021] Potirakis, S. M., Servières, M., Renaudin, V., Dupuis, A., & Antigny, N. (2021). Visual and Visual-Inertial SLAM: State of the Art, Classification, and Experimental Benchmarking. *Journal of Sensors*.

[Qin et al., 2018] Qin, T., Li, P., & Shen, S. (2018). VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator. *IEEE Transactions on Robotics*, 34(4), 1004–1020.

[Scandaroli & Morin, 2011] Scandaroli, G. G. & Morin, P. (2011). Nonlinear filter design for pose and IMU bias estimation. In *2011 IEEE International Conference on Robotics and Automation* (pp. 4524–4530).

[Schaub & Junkins, 2009] Schaub, H. & Junkins, J. L. (2009). *Analytical Mechanics of Space Systems*. AIAA education series. American Institute of Aeronautics and Astronautics, 2 edition.

[Schoenemann, 1964] Schoenemann, P. H. (1964). *A solution of the orthogonal Procrustes problem with applications to orthogonal and oblique rotation*. University of Illinois at Urbana-Champaign.

[Thrun et al., 2005] Thrun, S., Burgard, W., & Fox, D. (2005). *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press.

[Van der Merwe & Wan, 2001] Van der Merwe, R. & Wan, E. (2001). The square-root unscented Kalman filter for state and parameter-estimation. In *2001 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings*, volume 6 (pp. 3461–3464 vol.6).

[Wan & Van Der Merwe, 2000] Wan, E. A. & Van Der Merwe, R. (2000). The unscented Kalman filter for nonlinear estimation. In *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium* (pp. 153–158).: IEEE.

[Zefran et al., 1999] Zefran, M., Kumar, V., & Croke, C. (1999). Metrics and Connections for Rigid-Body Kinematics. *I. J. Robotic Res.*, 18, 242.