Master Thesis

SPID – ROB 2018

# Position Estimation by Merging Low Cost IMU and Camera Data Using the Extended Kalman Filter


**Author:** Philipe Miranda de Moura
**Tutor:** Michel Legris
**Supervisors:** Stefan Wierda
Evert Schippers


August 27, 2018


This page intentionally left blank.

# Contents

# List of Figures

# Abstract

This work details the development of a positioning system that couples computer vision and inertial navigation in order to improve safety in a refinery. This combination of sensors was made since refineries typically are partially indoor – limiting the use of global-navigation-satellite-systems – and require a particular spark-free certified smartphone to be used. A client refinery has about 20,000 assets – among flanges and valves – that need to be opened, closed or maintained. Ensuring intervention is performed in the correct asset is currently done by checking QR codes attached to each one of them. However, the current solution is expensive to be maintained and does not allow navigation.

The navigation based on computer vision is provided by the QuickVision system developed at Fugro. It consists of a camera able to detect a known pattern and determine its pose with respect to the pattern coordinate frame.

The inertial navigation information was obtained using a MEMS IMU equipped with both a 3D accelerometer and gyroscope. In addition, the fact that the smartphone is hand-held by a human being walking through the plant allowed a Zero Velocity Update (ZUPT) to be implemented in the vertical component in order to reduce drift. The main contribution of this work was to investigate the accuracy over time of IMU-based position estimation when no visual patterns were in view.

The results obtained in the office under perfect conditions show that the gyroscope's noise causes random walk in the estimated orientation typically 1° off the true orientation, generating error in the estimated position in the order of a few meters after $10s$ without visual aiding.

By comparing the results, it can be inferred that MEMS IMU are not suitable for dead reckoning applications without holonomic constraints, for amounts of time greater than a few seconds. As consequence, visual information should be used during pattern absence as aid by tracking features present on images.

## Keywords:

Localization, Extended Kalman Filter, MEMS, ZUPT

# Résumé

Ce travail détaille le développement d'un système de positionnement qui associe la vision par ordinateur et la navigation inertielle afin d'améliorer la sécurité dans une raffinerie. Cette combinaison de capteurs a été adoptée dans la mesure où les raffineries sont partiellement couvertes- ce qui limite l'utilisation de systèmes de navigation par satellite- et nécessitent l'utilisation d'un smartphone certifié sans étincelles. Une raffinerie compte environ 20 000 éléments à contrôler – dont des les brides et des les vannes - qui doivent être ouverts, fermés ou entretenus. Une vérification à l'aide de QR codes est réalisée afin de s'assurer que l'intervention ait été effectuée sur le bon élément. Cependant, cette pratique est coûteuse et ne permet pas la navigation au sein de la raffinerie.

La navigation basée sur vision par ordinateur est assurée par le système QuickVision développé chez Fugro. Il consiste en une caméra capable de détecter un motif connu et de déterminer sa position et son orientation par rapport à son repère de coordonnées du motif. Les informations de navigation inertielle ont été obtenues à l'aide d'un MEMS IMU équipé d'un accéléromètre et d'un gyroscope 3D. En outre, le fait que le smartphone soit tenu en main par un ouvrier traversant le site permet l'utilisation de la mise à jour à vitesse nulle (ZUPT) du le composant vertical afin de réduire la dérive. La principale contribution de ce travail était de fournir une estimation de la position basée sur l'IMU alors qu'aucun motif visuel n'était visible.

Les résultats obtenus au bureau dans des conditions parfaites montrent que le bruit du gyroscope provoque une marche aléatoire dans l'orientation estimée de un degré par rapport à l'orientation réelle, générant une erreur dans la position estimée de l'ordre de quelques mètres au bout de dix secondes sans aide visuelle. Le système a été évalué en conditions réelles dans l'installation d'entraînement RDM et les résultats de la réalité augmentée sont présentés dans ce travail.

Les résultats obtenus sur le terrain ont une erreur de l'ordre d'une fraction de mètre pour une période d'indisponibilité d'aide visuelle de quatre secondes, ce qui satisfait les exigences du projet à court terme.

En comparant les résultats, on peut inférer que les MEMS IMU ne sont pas adaptés aux applications de navigation à l'estime sans contraintes holonomes, même pendant de courtes périodes. En conséquence, des informations visuelles doivent être utilisées en cas d'absence de motif afin de permettre le suivi des caractéristiques présentes sur les images.

## Mots-clés :

Localisation, Filtre de Kalman Étendu, MEMS, ZUPT

# Acknowledgments

For me, writing this section was as tough as the thesis itself. In my opinion, this section is supposed to enclose thanks to all people who directly or indirectly contributed to this work. I am pretty sure I will not be able to cover every single person who deserves, though.

First of all, I thank my grandmother in Portuguese, so she can read it by herself. Bichona, aqui registro os meus mais sinceros agradecimentos por tudo o que você fez por mim ao longo da minha vida. Acredite, você é a pessoa mais brilhante que eu já conheci, tendo me ensinado lições que não estão escritas em livro algum. Eu dedico esse trabalho especialmente a você.

From Fugro, I would like to especially thank my supervisor Evert Schippers – who guided me relentlessly, having helped me every time I struggled analyzing partial results I obtained. I am afraid I would not have managed to do this work without your assistance. I could not forget to mention Alexander Steele – for uncountable hours spent preparing datasets for me –, Antony Veness – for life lessons I will always carry with me –, Magdalena Drozdz, Wojciech Straszewski and Francesca Panzetta – for the several discussions on the Kalman filter algorithm –, Merlijn van Deen – who contributed with out-of-the-box ideas that often made me figure out the problems I faced –, Stefan Wierda – who was my line manager –, Diego Carvalho – for insights about QuickVision – and Yulia Melnikova, Miguel Labayen, Harm-Simon Hegge and Paul Werker – who helped me especially with Git, code debugging and encouraging words.

From ENSTA-Bretagne, I would like to thank Michel Legris – who always answered my emails within just a few hours – and Luc Jaulin – who coordinates the Robotics branch.

In addition, I would like to thank CAPES - Coordination of Improvement of Graduate Personnel (in free translation) – for funding my academic exchange –, as well as all teachers I had since I was a child – for your passion and effort you put in the beautiful mission of forming people –, especially Rosângela Nezi, Celuta Reissmann and Valny Boechat.

Finally, I would like to thank my relatives – uncle José Gonçalves, in particular – and friends from childhood, university and notably those I met in Brest and have been part of my life for the past two years, Jean Macedo, Rémy Tellier and Victor Pimenta deserving special mention. Last, but in no means least, I thank my girlfriend Sarah for good memories that will be in my mind for the rest of my life and for all the support you give me.

# Notation

1. Standard Lowercase Latin Alphabet

   $b$ Body frame
   $e$ Earth frame
   $f$ Transition function
   $h$ Output function
   $i$ Inertial frame
   $j$ j-ith instant
   $k$ k-ith instant
   $l$ Local frame
   $n$ Navigation frame
   $p$ Pattern frame
   $t$ Time
   $x$ $x$-component
   $y$ $y$-component
   $z$ $z$-component

2. Bold Lowercase Upright Latin Alphabet are vectors

   $\mathbf{a}$ Acceleration
   $\mathbf{b}$ Bias
   $\mathbf{f}$ Specific force
   $\mathbf{g}$ Gravity
   $\mathbf{q}$ Generic quaternion
   $\mathbf{r}$ Position
   $\mathbf{u}$ Control input
   $\mathbf{v}$ Velocity
   $\mathbf{w}$ Noise
   $\mathbf{x}$ Systems states
   $\mathbf{y}$ Output states *and* Residual
   $\mathbf{z}$ Observation states

3. Bold Lowercase Italic Latin Alphabet

   $\boldsymbol{u}$ Generic (3D) vector
   $\boldsymbol{v}$ Generic (3D) vector

4. Standard Uppercase Latin Alphabet

   $L$ Latitude

5. Bold Uppercase Upright Latin Alphabet are matrices

   $\mathbf{A}$ Continuous-time transition
   $\mathbf{B}$ Continuous-time input
   $\mathbf{C}$ Continuous-time output
   $\mathbf{D}$ Continuous-time feedforward
   $\mathbf{F}$ Continuous-time transition Jacobian
   $\mathbf{G}$ Noise transition
   $\mathbf{H}$ Continuous-time observation Jacobian
   $\mathbf{K}$ Kalman gain

**M** Observability
**P** States covariance
**Q** Prediction model covariance
**R** Observation covariance
**S** Innovation covariance

6. Bold Uppercase Italic Latin Alphabet are matrices

***C*** Rotation
***I*** Identity
***Q*** Matrix obtained from QR decomposition
***R*** Matrix obtained from QR decomposition

7. Standard Lowercase Greek Alphabet

$\mu$ Quaternion angle of rotation
$\omega$ Angular velocity

8. Bold Lowercase Greek Alphabet are vectors

$\boldsymbol{\lambda}$ Scale factor
$\boldsymbol{\psi}$ Orientation
$\boldsymbol{\omega}$ Angular velocities

9. Bold Uppercase Greek Alphabet

$\boldsymbol{\Gamma}$ Discrete-time input matrix
$\boldsymbol{\Lambda}$ $\mathrm{diag}(\boldsymbol{\lambda})$
$\boldsymbol{\Upsilon}$ $Q\left(q(\boldsymbol{\omega})\right)$
$\boldsymbol{\Phi}$ Discrete-time transition matrix
$\boldsymbol{\Omega}$ Skew-matrix of angular velocities

10. Acronyms

AV Allan Variance
DCM Direction Cosine Matrix
ECEF Earth-Centered Earth-Fixed
EKF Extended Kalman Filter
ENU East-North-Up
FOG Fiber Optic Gyroscope
GPS Global Positioning System
GNSS Global Navigation Satellite System
IMU Inertial Measurement Unit
INS Inertial Navigation System
MEMS Microelectromechanical systems
NED North-East-Down
ODE Ordinary Differential Equation
PSD Power Spectral Density
QR Quick Response Code *or* QR factorization
ZUPT Zero Velocity Update

# 1 Introduction

## 1.1 Background and Objectives

Localization is a crucial task accomplished by humans on a daily basis for moving between reference points (such as home and work) and for identifying assets, thus avoiding hazards (such as moving vehicles). In less friendly environments as power plants and refineries, localizing can be significantly tougher, given the large quantity of similar features (pipes, valves and flanges) in view. Also, the consequences of a mistaken localization can be catastrophic, potentially leading to explosions and deaths. In these scenarios, one may appreciate techniques for aiding the right decision to be taken.

The client refinery has approximately 20,000 flanges that need to undergo maintenance regularly. Hence, they are interested in having a low-cost reliable solution to identify them, improving safety while reducing time and therefore costs.

Nowadays a first step towards the automation of this process is already being used. It consists of having a QR code attached to each one of their flanges. The employee receives on a mobile phone application a list of instructions containing the ID numbers of the flanges that need some kind of intervention. Apart from flange replacement, another common procedure is to isolate two sides of a pipe, which is known as *blinding* the pipe. Once the employee reaches a flange on the list using the refinery blueprint, he scans the QR to verify he is looking to the right one. Then he intervenes on it and rescans to mark the specific flange as done. The main point is to ensure all interventions were performed at the right locations. Additionally, this method also allows real time monitoring and avoids typing errors by removing all typing requirements. The main drawback of the current solution is the requirement to install and maintain one QR code per flange.

As an alternative, Fugro is currently developing a solution that will integrate two of its existing products – QuickVision and Site-Spot – with inertial navigation. QuickVision allows pose estimation in camera images with preloaded patterns in view, while Site-Spot is a web service capable of interacting in a screen with 2D and 3D environments previously scanned by the company. Combining both solutions, one should be able to define the position of flanges in the 3D model and pinpoint them in camera images. This fusion alone would already represent an improvement to the QR code strategy by allowing the detection of multiple flanges with a single pattern in view.

There is a high operational cost 20,000 QR codes. One of the aims of this work is to facilitate this task by replacing all the QR codes by a small quantity of QuickVision patterns. This work proposes accomplishing this objective by using inertial sensors to determine mobile phone pose during periods in which patterns are not available.

Fiber Optic Gyroscopes (FOGs) are a type of high grade inertial sensors that have been used for the past few decades for localization purposes during long periods and distances. However, such an alternative is not suitable for this application due to its size ($\sim 20 \times 20 \times 20cm$), weight ($\sim 10kg$) and cost (starting from 50,000€). A lighter, smaller and cheaper type of inertial sensors are the Microelectromechanical systems (MEMS). Their weight can range from $0.5g$ to $350g$, measuring between $2\times2\times0.2cm$ and $7\times7\times7cm$, with the price ranging from 10€ to 6,000€. Its downside is the poor performance for navigation purposes. Nevertheless, there are some works in the subject, as in [1, 2, 3, 4, 5].

This thesis aims to develop an algorithm of pose estimation merging camera and inertial sensors data that achieves the requirements specified by the client. For data fusion, a Extended Kalman Filter algorithm will be implemented.

It is well known that MEMS devices can cause position drift to grow considerably fast compared to FOG devices. Because of it, this work focuses on investigating the filter performance during small aiding outage periods, i.e. up to 30 seconds. Since this work intents to obtain a Proof of Concept (PoC), we are using sensors that are larger than the ones typically found in smartphones because it is more suitable to integrate in our prototype. The gathered data is post-processed in an external computer.

## 1.2 Dissertation Outline

In the remaining part of this chapter, some basic concepts are presented. Also, the two Fugro's products (QuickVision and SITE-SPOT) that aided to calculate navigation data are explained.

In chapter 2, the reference frames used in this work are presented. Chapter 3 is dedicated to present different attitude representations, as well as its advantages and drawbacks. By the end of the chapter, a choice is made and explained. Chapter 4 covers the basics of deterministic systems, including the relation between different representations alternatives, the equivalence between continuous and discrete-time equations. The concept of observability is also introduced.

Chapter 5 details the main sources of error of an IMU and describes the model that will be used to represent them and in chapter 6, forward INS mechanization will be stated. Following, chapter 7 introduces the concept of stochastic processes that arises from taking noise into account in deterministic processes. The Kalman Filter algorithm is enclosed by chapter 8, while 9 explains the main numerical issues faced during the development of this thesis and presents the approaches used to overcome them.

Finally, chapter 10 wraps all concepts previously presented, allowing chapter 11 to summarize the major findings of this research and chapter 12 to draw conclusions and suggest further research.

## 1.3 Navigation

As can be inferred by comparing the definition of *navigation* given by [6] and [7], there is no absolute consensus of what it exactly is. Nevertheless, they both agree *navigation* is the capacity to determine position with respect to a known reference frame. Often this concept is extended to the ability of reach desired positions, in addition to determining them. This extension is out of the scope of this work, that focuses only in presenting a short-hand theory background on aided inertial navigation. There are different techniques for achieving required specifications and usually the adopted solution is a combination of them, respecting a compromise between cost and accuracy.

### 1.3.1 Position Fixing

Position Fixing consists on measuring parameters (e.g. position, orientation, time of travel, speed, etc) with respect to usually mapped objects in different reference frames. This category encloses GPS and radar technologies, for instance. The main disadvantage of this approach is the dependency on external environment for position determination.

### 1.3.2 Dead Reckoning

Dead Reckoning relies on measuring the velocity or acceleration of an object with respect to a known reference frame (from now on called navigation-frame) so that it can be integrated in order to determine the object's position. In most cases, the object's reference frame (body-frame) rotates with respect to the navigation-frame over time, making it necessary to take the rotation between them into account for proper position determination. In addition, this method requires correct position initialization for it to work. Apart from it, the requirement of precise velocity or acceleration measurement is considered to be its main drawback, especially when low-grade sensors are used.

## 1.4 SITE-SPOT

SITE-SPOT is a web-based service that allows visualization of 3D laser-scanned areas. Fugro is able to collect data from 360° laser scanners and cameras placed in several locations. Afterwards, all data is linked with respect to each other, creating a model of an entire site. Laser data is responsible to obtain 3D points and the camera improves the appearance for the user. Once the model exists, it is possible to extract coordinates of objects in the model, as well as overlay text or 3D models on the screen.

Figure 1 illustrates data obtained from the laser, while figure 2 displays the same scene after laser-camera data matching.

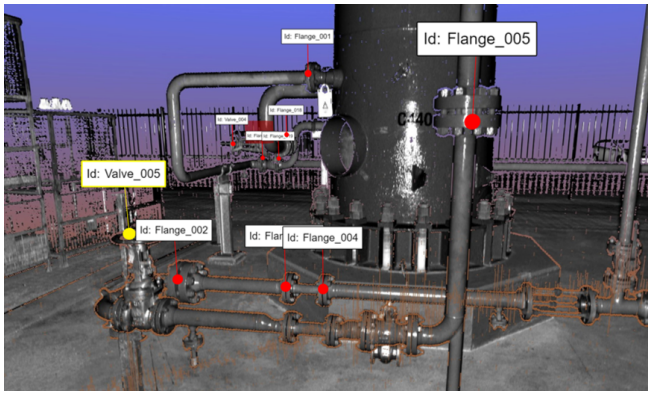



Figure 1: View of laser scan intensities on RDM training plant extracted from SITE-SPOT.

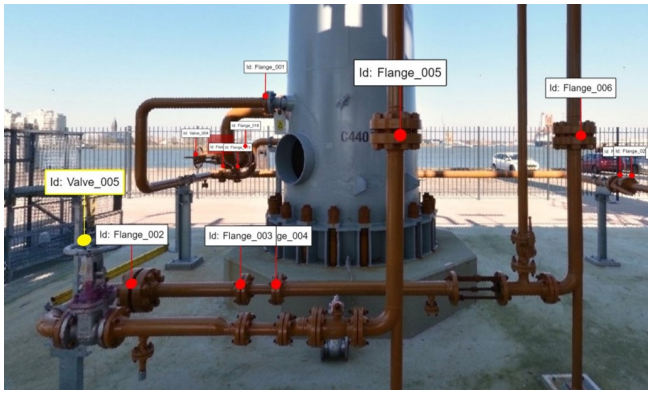



Figure 2: View of camera data on RDM training plant extracted from SITE-SPOT.

## 1.5 QuickVision

QuickVision is a product developed by Fugro to allow real-time tracking of a pattern position and orientation. By extension, it also allows to keep track of an object with a pattern attached to it if the latter is measured with respect to the former. A pattern consists on printing round dots on an uniform background. The computer running QuickVision algorithm knows beforehand which patterns it is trying to find in the image and is able to locate them even if they are tilted or partially occluded. QuickVision also enables augmented reality (AR): a 3D model can be shown on the computer screen based on patterns' poses with respect to the camera. An example of the use of the QuickVision for coupling two underwater structures is depicted on figure 3.

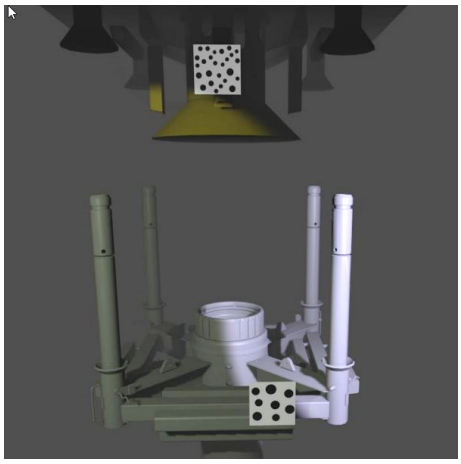


Figure 3: Render of an underwater application of QuickVision.

An example of the AR capabilities of QuickVision product is illustrated by figure 4. Note that the assets position's were obtained after extracting their coordinates from the 3D SITE-SPOT model. Also, it was needed to relate the reference frame used on SITE-SPOT with the one used by the patterns.

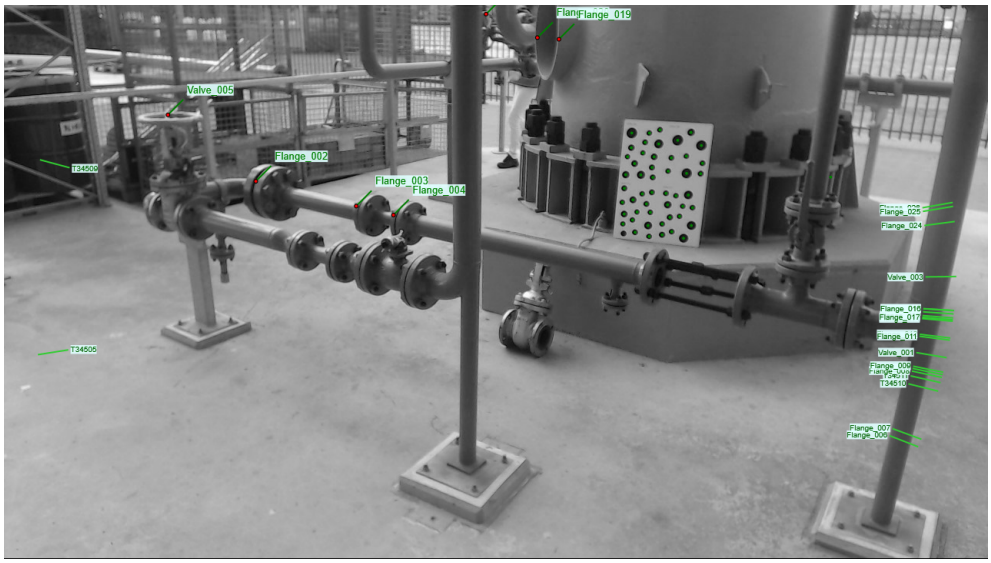



Figure 4: AR application of QuickVision in a refinery training plant. Assets that are not in view are represented in the image's border, pointing to their directions. From the image, it is straightforward to identify flanges 002, 003, 004 and valve 005.

# 2 Reference Frames

This section gives a brief explanation about the reference frames adopted throughout the thesis. References [6, 7, 8, 9, 10, 11] dedicate dozens of pages to define each of the reference frames used in INSs. As done in [1] [2], this chapter is intended only to give the reader the basic knowledge for the following sections.

**The inertial frame** $i$ is the coordinate frame with origin at the center of the earth, with $z$ pointing north and aligned with the Earth's spinning axis, $x$ pointing to the vernal equinox and $y$ completing the right hand rule.

**The Earth frame** $e$ a.k.a. Earth-Centered Earth-Fixed (ECEF), is a coordinate frame with the same origin and $z$ axis as the inertial frame $i$, having $x$ pointing to the Greenwich meridian and $y$ completing the right hand rule.

**The navigation frame** $n$ is the coordinate frame in which the problem is desired to be resolved. Its origin is in the plane tangent to the Earth at the location the navigation is performed, with $x$ pointing north, $z$ pointing away from the center of the Earth and $y$ completing the right hand rule. The origin of the navigation frame $n$ is not fixed with respect to the origin of the ECEF frame, it moves along with the location at which the navigation is performed. However, this work considers it fixed, since the location studied varies in a range of less than 100m, representing a change in latitude and longitude at the order of $10^{-4\circ}$.

**The local frame** $l$ has the same origin as the navigation frame $n$, with $x$ pointing north, $z$ pointing towards the center of the Earth and $y$ completing the right hand rule.

**The pattern frame** $p$ is the right-handed coordinate frame with the same origin as the local frame $l$, same $z$ axis and a fixed $x$ and $y$ misalignment from the navigation frame $n$.

**The body frame** $b$ is the right-handed coordinate frame with the same origin as the navigation frame $n$ (and, consequently, as the local frame $l$ as well), fixed with respect to the IMU and with axis determined by the IMU manufacturer, at which the IMU outputs are valid.

**The camera frame** $cam$ is the coordinate frame with origin at the camera pinhole, with $y$ pointing forward, $z$ pointing up and $x$ completing the right hand rule.

Figure 5 [1] and 6 illustrate the frames presented. Note that the frames are denoted by superscripts.

[1] Figure kindly extract from [12]

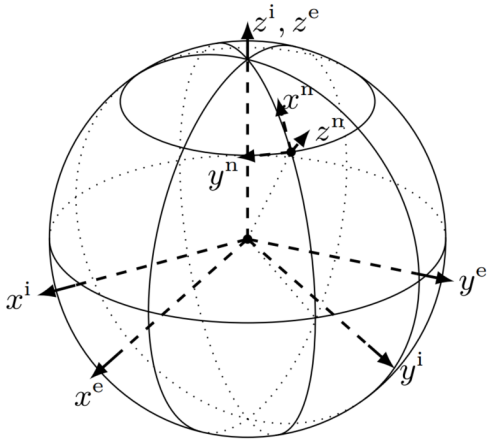



Figure 5: Representation of the axis of the inertial frame $i$, Earth frame $e$, navigation frame $n$.

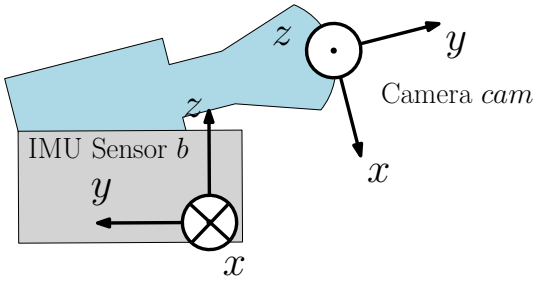



Figure 6: Representation of the axis of the body frame $b$ and camera frame $cam$.

# 3 Attitude Representation

## 3.1 Introduction

Many books and articles [6, 7, 8, 9, 13] dedicate entire sections on describing the mathematics behind different attitude representations. The goal of this section is to present the basics of attitude representations, helping the reader to understand the advantages and disadvantages of each one of them, as well as how to convert between them.

In this work, all rotations are around an orthogonal right-handed axis set and a positive rotation is the one in the clockwise direction along the axis from the origin to positive infinity, accordingly to what is used by most authors. Also, given that sometimes there are several different possible options for a frame to be defined (ENU - $x$-East, $y$-North, $z$-Up – NED - $x$-North, $y$-East, $z$-Down – and others), to avoid confusion, rotations around $x$, $y$ and $z$ as *roll*, *pitch* and *yaw* when possible. *Roll* is defined to be the rotation around the forward axis, while *yaw* is the one around the vertical axis and *pitch* is the one around the remaining axis, regardless axis naming.

Some authors explicitly distinguish *roll*, *pitch* and *yaw* from *bank*, *elevation* and *heading*, stating that the first three terms are the angular velocities associated with the angular positions expressed by last three terms, respectively. Since some other authors use all six words when referring to angular position, this convention will also be adopted in this thesis, except when stated otherwise. Angular velocities will be usually represented by the lowercase Greek letter omega ($\omega$) and rotation matrices will be denoted by $\boldsymbol{C}$ (and not $\mathbf{C}$).

There are at least four different representations of attitude:

- Euler Angles
- Direction Cosine Matrix
- Axis-Angle
- Quaternions

## 3.2 Euler Angles

Euler Angles probably are the most intuitive way of imagine and interpret rotations. It consists on three successive rotations around each one of the $x - y - z$ axis in a given sequence i.e. $\boldsymbol{C} = \boldsymbol{C}_1\boldsymbol{C}_2\boldsymbol{C}_3$, where $\boldsymbol{C}_i$ is the rotation around the axis $i$. Often the first rotation applied is roll (around the forward axis), then comes pitch (around the lateral axis) and finally comes yaw (around the vertical axis). It is important to remark that the order in which the rotations are applied matters, given that matrix multiplication is not a commutative operation. The greatest advantage of this particular order is that it keeps the heading constant regardless the roll and pitch applied, therefore explaining its wide use - especially in marine applications.

Using Euler Angles to represent orientation has a huge drawback, the limitation known as gimbal lock. Euler Angles correspond, by construction, to the rotation angles of each one of the axis of a set of three gimbals, as shown in the image 7, in which roll is the rotation around the blue axis, pitch is the rotation around the green axis and heading is the rotation around the pink axis. Defining the full rotation as so creates a singularity point when pitch equals $\pm 90°$. This means that the airplane in the figure 7b cannot rotate


(a) Gimbal Free (b) Gimbal Locked

Figure 7: Illustration of the gimbal lock limitation of Euler Angles representation of attitude.

around the axis perpendicular to both green and pink ones, given that the blue axis has became aligned with the pink one as direct consequence of pitching. This disadvantage is enough for avoiding this technique for representing orientation during computation. Nevertheless, Euler Angles will still be used for displaying results.

## 3.3 Direction Cosine Matrix

### 3.3.1 Overview

A Direction Cosine Matrix (DCM) is a $3 \times 3$ rotation matrix that relates vectors in two different frames, as follows:

$$\mathbf{r}^b = \boldsymbol{C}_a^b \mathbf{r}^a, \tag{1}$$

where $\boldsymbol{C}_a^b$ is the rotation matrix expressing rotation from frame $a$ to frame $b$. It is important to keep in mind that a rotation matrix must be orthonormal with determinant equals to positive 1, by definition. It needs to have determinant $+1$ in order not to scale the output vector with respect to the input vector. A matrix can only be orthogonal if it is a square matrix [14], which implies that a vector rotation always leads to an output vector with the same dimension of the input one. The definition of an orthonormal matrix is tied up by equation 2[2]

$$\boldsymbol{C}^{-1} = \boldsymbol{C}^T, \tag{2}$$

Given the high cost of inversion of a general matrix, this property of rotation matrices will prove to be very handy, since the matrix inversion is, in this case, synonym to matrix transposition. For better understanding, we are going to multiply $\left(\boldsymbol{C}_a^b\right)^{-1}$ on the left of both sides of equation 1, resulting in:

$$\left(\boldsymbol{C}_a^b\right)^{-1} \mathbf{r}^b = \left(\boldsymbol{C}_a^b\right)^{-1} \boldsymbol{C}_a^b \mathbf{r}^a \tag{3a}$$

$$\left(\boldsymbol{C}_a^b\right)^{-1} \mathbf{r}^b = \mathbf{r}^a \tag{3b}$$

$$\left(\boldsymbol{C}_a^b\right)^{T} \mathbf{r}^b = \mathbf{r}^a \tag{3c}$$

$$\boldsymbol{C}_b^a \mathbf{r}^b = \mathbf{r}^a \tag{3d}$$

[2]frames notations dropped for simplicity, without losing generality

### 3.3.2 Derivative of a Rotation

In order to be able to compute the orientation at every discrete instant $k$, one needs to solve the differential equation that establishes the relation between the orientation and its derivative.

$$\dot{\boldsymbol{C}}_b^a(t) = \lim_{\Delta t \to 0} \frac{\boldsymbol{C}_b^a(t + \Delta t) - \boldsymbol{C}_b^a(t)}{\Delta t} \tag{4}$$

Given that $\boldsymbol{C}_b^a(t + \Delta t)$ is also a rotation, then there is a matrix $\boldsymbol{S}(t, \Delta t)$ so that $\boldsymbol{C}_b^a(t + \Delta t) = \boldsymbol{S}(t, \Delta t)\boldsymbol{C}_b^a(t)$, leading to:

$$\dot{\boldsymbol{C}}_b^a(t) = \lim_{\Delta t \to 0} \frac{\boldsymbol{S}(t, \Delta t)\boldsymbol{C}_b^a(t) - \boldsymbol{C}_b^a(t)}{\Delta t} \tag{5a}$$

$$\dot{\boldsymbol{C}}_b^a(t) = \left[\lim_{\Delta t \to 0} \frac{\boldsymbol{S}(t, \Delta t) - \boldsymbol{I}}{\Delta t}\right] \boldsymbol{C}_b^a(t) \triangleq \boldsymbol{\Omega}_{ab}^a(t)\boldsymbol{C}_b^a(t), \tag{5b}$$

in which $\boldsymbol{\Omega}$ is called rotation **rate** matrix and shall not be confused with rotation matrix, meaning that rotation matrices properties may not apply.

According to [6], $\boldsymbol{\Omega}$ is a skew-symmetric matrix, i.e. $-\boldsymbol{\Omega} = \boldsymbol{\Omega}^T$. In other words, the vector $\begin{pmatrix}\omega_x & \omega_y & \omega_z\end{pmatrix}^T$ of angular velocities around $x$, $y$ and $z$ respectively leads to [11]:

$$\boldsymbol{\Omega}_{ab}^a = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} \tag{6}$$

that should be read as the angular velocity of the $b$-frame with respect to $a$-frame, expressed in $a$-frame. Note that $\boldsymbol{\omega}$ also has the same subscripts and superscripts, that were dropped for convenience.

Contrarily to the Euler Angles representation, Direction Cosine Matrices present no singularity and are, thus, most suitable for a general application. Its downside comes, however, from a practical point of view: $3 \times 3 = 9$ variables must be stored in memory. Also, reminding that floating point precision is a restriction, Direction Cosine Matrices can rapidly loose orthogonality, hence ceasing to be a rotation matrix (recall basic properties of rotation matrices in 3.3.1). An algorithm may be performed to re-obtain a rotation matrix to get around this issue.

## 3.4 Axis-Angle

Axis-Angle, also named *rotation vector*, is a orientation representation mainly used for introducing quaternions, and it will be the case here. Axis-angle describes a rotation with a (usually unitary) vector and a scalar, in which the vector defines the axis around which the rotation will occur, while the scalar is in charge of the amount of rotation. Figure 8 presents a visualization for it, being the dark green line the axis, $\begin{pmatrix}1 & 0 & 0\end{pmatrix}^T$ the vector before rotation, the black line the rotated vector and the blue arc the amount of rotation.

One might get confused with the fact that a rotation in a 3-dimensional space was defined using four parameters. To solve this, it is possible to incorporate the amount of rotation into the director vector, by simply scaling it up or down. In other words, three

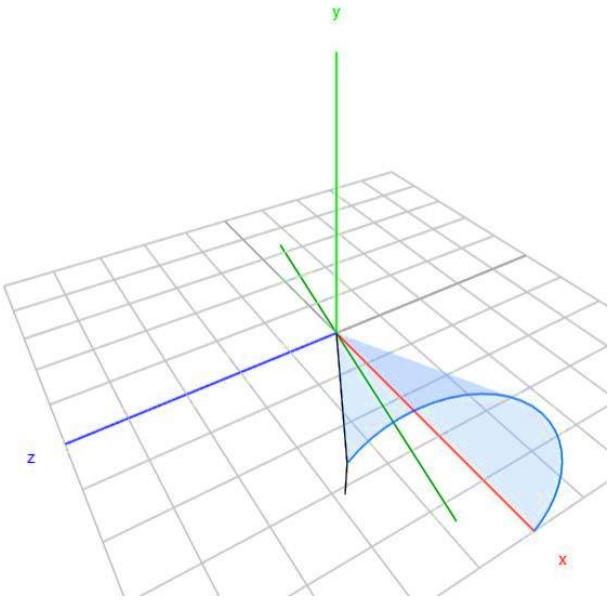



Figure 8: Illustration of Axis-Angle orientation representation

parameters are sufficient to define a 3-dimensional space and a fourth one was introduced for convenience.

## 3.5 Quaternions

Similarly to the Axis-Angle representation, a rotation quaternion $\mathbf{q}$ (henceforth simply called *quaternion*) is defined as follows: [3]

$$\mathbf{q} = \begin{pmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{pmatrix} = \begin{pmatrix} \cos(\mu/2) \\ (\mu_x/\mu)\sin(\mu/2) \\ (\mu_y/\mu)\sin(\mu/2) \\ (\mu_z/\mu)\sin(\mu/2) \end{pmatrix}, \tag{7}$$

where $\boldsymbol{\mu}$ is a rotation vector with magnitude $\mu$ and components $\begin{pmatrix} \mu_x & \mu_y & \mu_z \end{pmatrix}^T$. A quaternion is, therefore, a particular case of Axis-Angle representation. By inspection, $q_0$ is the term corresponding to the amount of angle, while the others define the axis of rotation. The main advantage of defining the rotation in such way is that its magnitude is always equals to 1, characteristic that will be used to get around numerical precision issues, as addressed in 9.3.

It may be familiar to the reader the close existing relation between complex numbers and rotations around one axis [15]. Quaternions can be seen as an extension of complex numbers, with three imaginary components, instead of only one, allowing rotations around any combinations of $x - y - z$ axis. Hence, quaternions operations derivate from complex numbers operations.

Quaternions do not present any singularities, contrarily to Euler Angles. Also, quaternions are represented by fewer parameters than DCMs (four compared to nine), which requires less memory to storage. Finally, since quaternions transformations are inherently

[3] Note that only rotation quaternions are defined as 7

orthonormal, one may only need to apply normalization, instead of matrix orthogonalization, being the latter more costly.

Further details can be found in [13, 16, 17].

## 3.6 The Choice of Attitude Representation

For the reasons discussed throughout the chapter, quaternion will be adopted as attitude representation. Nevertheless, the notation used is a mix between DCM and quaternion. As explained previously, quaternion operations are special and often it is preferable to convert a quaternion into a DCM. Performance of the two methods should not be an issue for modern computers. Numerical issues are discussed in 9.3.

# 4 Deterministic Processes

Up to now, equations were presented modeling each parameter's dynamics over time. This chapter will be dedicated to clarify the meaning of previous equations, as well as to present other alternative representations for them that might be more familiar or intuitive for the reader. This chapter will also present the equivalence between the continuous and the discrete-time models and other useful concepts, specially for understanding the following chapter about stochastic processes.

It is important to remark that many localization-related books do not address this topic in a dedicated section, what can lead to confusion. One exception is [10] – having inspired this chapter's structure – and other alternatives are mainly books on control theory, robotics or signal and systems, such as [18, 19, 20].

## 4.1 Continuous-Time Systems Models

### 4.1.1 Ordinary Differential Equations

According to [10], a single-input single-output linear dynamic system can be written in the form of the ordinary differential equation (ODE) 8

$$\begin{aligned} y^{(n)}(t) = \alpha_{n-1}(t)y^{(n-1)}(t) + \ldots + \alpha_1(t)\dot{y}(t) + \alpha_0(t)y(t) + \\ \beta_m(t)u^{(m)}(t) + \ldots + \beta_1(t)\dot{u}(t) + \beta_0(t)u(t) \end{aligned} \tag{8}$$

In this context, $y$ is the system output, $u$ is the system input and the superscripts are the $j$-th derivative. Note that order of an ODE is defined to be the highest derivative term present on it and that its solution depends on the initial conditions $y^{(0)}(t = t_0), y^{(1)}(t = t_0), \ldots, y^{(n-1)}(t = t_0)$, where $t_0$ denotes the initial time.

The equations developed in [20] for a frictionless cart attached to a vertical wall by a horizontal spring and damper moving along a horizontal straight rail, obtaining its position equation 9 after having applied Newton's laws to the free body diagram:

$$\ddot{y}(t) + \frac{B}{M}\dot{y}(t) + \frac{K}{M}y(t) = f(t) \Leftrightarrow \ddot{y} + \frac{B}{M}\dot{y} + \frac{K}{M}y = f \tag{9}$$

where $M$ is the mass of the cart, $K$ is the Hooke's constant of the spring, $B$ is the damping coefficient and $f$ is the horizontal force applied to the cart. Given that its highest derivative is the second one, it consists of a second order ODE.

Carrying the time-dependency notation $(t)$ is often cumbersome and, thus, is dropped. When it is the case, one needs to infer from the context which of the variables are constant or might vary in time. The equation 9 presented both notations and by only looking the latter it is difficult to state if $M$, $B$, $K$ and $f$ are constant or not, but from the context it is reasonable to assume that the mass, the spring and the damping coefficients are constant, while the applied force may vary.

Ordinary differential equations are intuitive and usually is the first representation undergraduates study. However it is not the most suitable one to be solved, either by hand or by computers.

### 4.1.2 Transfer Functions

Transfer functions arise from the application of the Laplace Transform on equation 8, which is a powerful method of solving ordinary differential equations. Even though its

relevance is within this domain, its use goes beyond the scope of this work, since the studied systems are time-variant for which the solutions presented are obtained using numerical methods.

Nevertheless, block diagrams will be used along the thesis and the goal of the current section is to ask the reader to extrapolate the concept of block diagrams to time-variant systems.

### 4.1.3 State Space

State space representation is an alternative to ordinary differential (and difference [4]) equations. One ODE of order $n$ can be represented by $n$ equations of order 1 [20].

Recall the cart's ODE:

$$\ddot{y}(t) = f(t) - \frac{B}{M}\dot{y}(t) - \frac{K}{M}y(t) \tag{10}$$

Let $x_1(t) = y(t)$ and $x_2(t) = \dot{y}(t)$, yielding to:

$$\begin{cases} \dot{x_1}(t) = \dot{y}(t) = x_2(t) \\ \dot{x_2}(t) = \ddot{y}(t) = -\frac{B}{M}x_2(t) - \frac{K}{M}x_1(t) + f(t) \end{cases} \tag{11}$$

The equation 11 can be rewritten in the matrix form 12, in which $y(t)$ is the output of the system, typically defined as the system's parameter that can be measured. In the example, it is the cart's position.

$$\begin{cases} \begin{pmatrix} \dot{x_1}(t) \\ \dot{x_2}(t) \end{pmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{K}{M} & -\frac{B}{M} \end{bmatrix} \begin{pmatrix} x_1(t) \\ x_2(t) \end{pmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} f(t) \\ \mathrm{y}(t) = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{pmatrix} x_1(t) \\ x_2(t) \end{pmatrix} + \begin{bmatrix} 0 \end{bmatrix} f(t) \end{cases} \tag{12}$$

Again, the time-notation can be dropped and equation 12 may be rewritten for a general case as 13.

$$\begin{cases} \dot{\mathbf{x}} = \mathbf{F}\mathbf{x} + \mathbf{B}\mathbf{u} \\ \mathbf{y} = \mathbf{H}\mathbf{x} + \mathbf{D}\mathbf{u} \end{cases} \tag{13}$$

where $\mathbf{x}$ is the state vector, $\mathbf{y}$ is the output vector, $\mathbf{u}$ is the input vector, $\mathbf{F}$ is the system transition matrix, $\mathbf{B}$ is the control matrix, $\mathbf{H}$ is the output matrix and $\mathbf{D}$ is the feedthrough matrix. The state vector ($\mathbf{x}$) is composed of state variables ($x_i$) and by definition has the smallest possible dimension able to represent the system.

Note that in this context $y$ refers to the ODE's physical quantities, while y and $\mathbf{y}$ are the system output for the scalar and vectorial cases, respectively.

State-space representation usually makes clearer which of the physical quantities can be seen or measured. In other words, disregarding the feedthrough matrix $\mathbf{D}$ (that rarely exists), the output is the product between $\mathbf{H}$ and $\mathbf{x}$. In the previous example, only the cart's position can be sensed, given that $\mathbf{H} = \begin{bmatrix} 1 & 0 \end{bmatrix}$ (and that $\mathbf{x} = \begin{pmatrix} y & \dot{y} \end{pmatrix}^T$). Sometimes it is desired to infer *hidden* states ($\dot{y}$ in this case), a process known as state estimation.

[4] This term is properly defined in section 4.2

Chapter 4.3 explains the conditions in which this problem is solvable, while chapter 8 presents one particular state estimator: the Kalman filter.

Even so, a system can be defined using state-space representation in infinite different ways [18]. One of those infinite representations is usually more appropriate than others for solving a particular problem. An intuitive way of understanding it is presented by [14] when approaching change of basis. By inspection, the reader should be able to recognize the same pattern in both cases.

## 4.2 Discrete-Time Systems Equivalent Models

As most of applications nowadays, in this work a computer will be used to numerically solve the equations presented along the past chapters. However, a computer does not work in continuous time: it is based on cycles and, for that reason, is said to operate in discrete times, where the continuous-time equations just presented are not valid anymore. It is needed to derive the discrete-time equivalent equations, then. For a computer, solving $n$ 1st order ODE is much easier then solving 1 ODE of order $n$, hence the power of state-space representation. As a detailed development is made on [10], the following chapter will present a slightly different approach, aiming to give the reader the basics only.

### 4.2.1 Calculation of $\mathbf{\Phi}_k$ from $\mathbf{F}(t)$

The solution of a scalar ODE of form $\dot{x}(t) = ax(t)$ is well known from calculus courses, stated as:

$$x(t) = x_0 e^{-at} \tag{14}$$

Analogously, the solution of the matrix ODE $\dot{\mathbf{x}}(t) = \mathbf{F}\mathbf{x}(t)$ is:

$$\mathbf{x}(t) = \mathbf{x}_0 e^{-\mathbf{F}t} \tag{15}$$

One could get stuck when facing the exponential of a matrix, but it should not be a problem if we take the power series approach to compute it, from where equation 16:

$$e^{at} = \sum_{n=0}^{\infty} \frac{a^n}{n!} t^n = 1 + at + \frac{a^2}{2!}t^2 + \ldots + \frac{a^n}{n!}t^n + \ldots \tag{16}$$

For matrices, the expansion is straightforward, noting only that 1 is replaced by the identity matrix $\boldsymbol{I}$. If the solution interval is sufficient small, a first order approximation may be accurate enough, yielding to the following discrete-time equivalent model:

$$e^{\mathbf{F}\Delta t} = \sum_{n=0}^{\infty} \frac{(\mathbf{F}\Delta t)^n}{n!} = \boldsymbol{I} + \mathbf{F}\Delta t + \frac{(\mathbf{F}\Delta t)^2}{2!} + \frac{(\mathbf{F}\Delta t)^3}{3!} + \cdots \approx \boldsymbol{I} + \mathbf{F}\Delta t \triangleq \mathbf{\Phi} \tag{17}$$

In case $\mathbf{F}$ varies in time ($\mathbf{F}(t)$), it is usually possible to linearize it around the operation point using Taylor series expansion and assume $\mathbf{F}(t)$ is constant during tiny time intervals $\Delta t$. Note that non-linear systems may be seen as linear time-variant systems (as exemplified on equation 18) and, therefore, usually can also be considered constant during small time intervals.

$$\begin{pmatrix} x \\ \dot{x} \end{pmatrix}_{k+1} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ \dot{x} \end{pmatrix}_k + \begin{bmatrix} 0 \\ \Delta t \end{bmatrix} \begin{pmatrix} \ddot{x} \end{pmatrix}_k \tag{18a}$$

$$\begin{pmatrix} x \\ \dot{x} \end{pmatrix}_{k+1} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 + \Delta t \ddot{x}_k \dot{x}_k^{-1} \end{bmatrix} \begin{pmatrix} x \\ \dot{x} \end{pmatrix}_k \tag{18b}$$

$$\dot{\mathbf{x}} = \mathbf{F}\mathbf{x} \tag{19}$$

With the discrete form being:

$$\mathbf{x}_{k+1} = \mathbf{\Phi}_k \mathbf{x}_k \tag{20}$$

## 4.3 Observability

A system is called (completely) observable if it is possible to infer each of its internal states $\mathbf{x}$ from a finite time interval of its outputs $\mathbf{y}$. By definition, a non-(completely-)observable system cannot have all its internal states estimated. Therefore, before trying to estimate a system's internal states, it is important to verify if such a task is mathematically solvable.

From the system model, it is possible to compute the Observability Matrix $\mathbf{M}$. The number observable states is the rank of $\mathbf{M}$. As a refresher, the rank of a matrix is the number of dimensions of the space spanned by its columns (or rows). In other words, it is the number of linearly-independent columns (or rows). Indeed, if a column (or row) can be expressed as a linear combination of others, then this column (or row) belongs to the already covered hyper-space and thus is not able to point towards a new dimension.

Still, another way to get the intuition about the meaning of the rank of a matrix in the special case of a square matrix is to look at its eigenvalues and eigenvectors. Basically, if a matrix has one eigenvalue equals to zero, it means there is a pair of eigenvectors $(\vec{u}, \vec{v})$ whose cross-product $\vec{u} \times \vec{v}$ is also zero. Neglecting the trivial case in which $||\vec{u}|| = 0$ or $||\vec{v}|| = 0$, the cross-product is null if and only if $\vec{u}$ is parallel to $\vec{v}$, given that $\vec{u} \times \vec{v} = ||\vec{u}||||\vec{v}|| \sin(\theta)\vec{n}$, where $\theta$ is the angle between $\vec{u}$ and $\vec{v}$ and $\vec{n}$ is the director (unitary) vector, normal to both $\vec{u}$ and $\vec{v}$.

For rectangular matrices, one can benefit from approach just presented thanks to the QR decomposition. Every matrix $\mathbf{A}$ can be decomposed into the multiplication of two matrices $\mathbf{Q}$ and $\mathbf{R}$ in such a way that $\mathbf{A} = \mathbf{QR}$, where $\mathbf{R}$ is a upper triangular (square) matrix with the same rank as $\mathbf{A}$ [14].

The following subsections present the observability matrix only and the reader is invited to refer to [10] and [18] for details concerning the fundamentals behind them.

### 4.3.1 Continuous Linear Time-Invariant Case

The Continuous Linear Time-Invariant case is addressed in [18]. Consider the system below:

$$\begin{cases} \dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \\ \mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u} \end{cases} \tag{21}$$

The Observability Matrix $\mathbf{M}$ for the system 21 is defined as follows.

$$\mathbf{M} = \begin{bmatrix} \mathbf{C} \\ \mathbf{CA} \\ \vdots \\ \mathbf{CA}^{n-1} \end{bmatrix} \tag{22}$$

where $n$ is the number of internal states of $\mathbf{x}$

### 4.3.2 Discrete Linear Time-Variant Case

The Discrete Linear Time-Variant case is addressed in details in [10]. Consider the system below:

$$\begin{cases} \dot{\mathbf{x}}_k = \mathbf{\Phi}_k \mathbf{x}_{k-1} + \mathbf{\Gamma}_k \mathbf{u}_k \\ \mathbf{y}_k = \mathbf{H}_k \mathbf{x}_k + \mathbf{D}_k \mathbf{u}_k \end{cases} \tag{23}$$

The Observability Matrix $\mathbf{M}$ for the system 23 is defined as follows.

$$\mathbf{M} = \begin{bmatrix} \mathbf{H}_{k-N} \\ \mathbf{H}_{k-N+1}\mathbf{\Phi}_{k-N} \\ \vdots \\ \mathbf{H}_k \mathbf{\Phi}_{k-1} \dots \mathbf{\Phi}_{k-N} \end{bmatrix} \tag{24}$$

where $N$ is a sufficiently big finite integer. For linear time-invariant systems, $n$ (from the previous chapter) and $N$ are the same. However, for linear time-variant systems whose observability depends on the internal states, $N$ may be greater than $n$.

Furthermore, for a linear time-variant system, observability may be determined numerically from collected data, challenge addressed on [21].

# 5 Inertial Sensors

## 5.1 Overview

Inertial sensors are sensors capable of measuring a physical quantity with respect to a inertial reference frame, as the name suggests. They mainly consist of accelerometers and gyroscopes. Making no differentiation between *inertial sensors* and *inertial measurement units (IMUs)* can lead to misleading assumptions.

An IMU is a device that encapsulates multiple accelerometers and gyroscopes in order to deliver the information of the entire inertial movement. More precisely, IMUs commonly have three single-axis orthogonal gyroscopes aligned with three single-axis accelerometers.

Accelerometers are sensors that measure the sum of forces applied to them, meaning that in an ideal scenario their output will only be null during free fall periods. Similarly, gyroscopes are sensors that measure the angular rates they suffer and therefore will only have null output when their angular rates compensates the Earth's one.

There are many different types of inertial sensors, widely varying in price, size, weight, accuracy and precision. Here, only the ones used during experiments will be described, keeping in mind that the main goal is understand the big picture of their working principles so that we can develop models for their main sources of errors.

This chapter was conceived based on [1, 2, 3, 4, 6, 7, 8, 9, 11, 12, 22, 23]. They have some intersection areas, specially concerning inertial sensors' error characteristics and error models as well. Others are more dedicated in explaining various types of inertial sensors or in comparing results obtained by different error models.

## 5.2 Coning and Sculling

Most of modern IMUs are not mounted on gimbaled leveled platforms, due to the difficulty, cost, size and weight to produce them. They are instead rigidly fastened to the moving object, causing it to rotate with that object and thus making it necessary to compensate for misaligned frames between two different output timestamps.

In order to get proper results, this computation needs to be done at high frequencies typically around $1000Hz$. To avoid overloading the device receiving the sensor measurements, coning and sculling computations are frequently done in the IMU, which allows it to output at about $100 \sim 200Hz$.

The output is, then, a delta angle over delta time, instead of an angular rate, meaning that angular rate was internally integrated a few times before being output. However, one should notice that the orientation is changing when the angular rate is not null, thus leading the integration to be orientation-dependent. To sum it up, coning is the error that arises by computing this integral without properly taking different orientations into account.

Similarly, sculling is the error that arises by computing the acceleration average without properly taking different orientations into account.

## 5.3 Error Characteristics

Sensors in general are subjected to different sources of measurement errors, comprising bias, scale factor, cross-coupling and random noise. One of the characteristics that dis-

tinguishes a high grade sensor from a low grade one is the magnitude of such errors. Each of them will be briefly discussed below in the extent of inertial sensors.

### 5.3.1 Bias

Bias is a measurement error said to be constant[5] and independent of the environment in which the sensor is or the motion experienced by it. In other words, in the particular scenario in which a gyroscope is still, its bias in $x$, $y$ and $z$ are the averages of readings in each one of these axis, respectively. Considering bias as the only source of measurement imperfection, we would have[6]:

$$\mathbf{f}_{measured} = \mathbf{f}_{true} + \mathbf{b}_{\mathbf{f}} \tag{25a}$$

$$\boldsymbol{\omega}_{measured} = \boldsymbol{\omega}_{true} + \mathbf{b}_{\boldsymbol{\omega}} \tag{25b}$$

where $\mathbf{f}$ is the specific force and $\boldsymbol{\omega}$ is the angular rate and $\mathbf{b}$ is the bias[7].

### 5.3.2 Scale Factor and Cross-coupling

The error called *scale factor* is the one that linearly grows with the true motion values. For gaining the intuition of how it is possible, one could think about the distance measurement using a plastic tape measure: the longer the distance to be measured, the longer the tape stretches.

*Cross-coupling* is the error that arises from non-mutual-orthogonality among the axis. An acceleration along the forward axis of an accelerometer should only cause the forward-axis measurements to change, but, if its two other axis are not perfectly orthogonal to the forward one, then a small component of a forward acceleration will also be sensed by the non-forward axis. This source of error can be considered constant and suitable for calibration during production phase. Therefore, this work will assume a proper calibration by the manufacturer and will neglect any remaining associated error.

As done for the bias, consider the scale factor as being the only source of error in a measurement. Then, we obtain:

$$\begin{cases} \mathrm{f}_{x,measured} = \mathrm{f}_{x,true} + \lambda_{\mathrm{f}_x}\mathrm{f}_{x,true} = (1+\lambda_{\mathrm{f}_x})\mathrm{f}_{x,true} \\ \mathrm{f}_{y,measured} = \mathrm{f}_{y,true} + \lambda_{\mathrm{f}_y}\mathrm{f}_{y,true} = (1+\lambda_{\mathrm{f}_y})\mathrm{f}_{y,true} \\ \mathrm{f}_{z,measured} = \mathrm{f}_{z,true} + \lambda_{\mathrm{f}_z}\mathrm{f}_{z,true} = (1+\lambda_{\mathrm{f}_z})\mathrm{f}_{z,true} \end{cases} \tag{26a}$$

$$\begin{cases} \omega_{x,measured} = \omega_{x,true} + \lambda_{\omega_x}\omega_{x,true} = (1+\lambda_{\omega_x})\omega_{x,true} \\ \omega_{y,measured} = \omega_{y,true} + \lambda_{\omega_y}\omega_{y,true} = (1+\lambda_{\omega_y})\omega_{y,true} \\ \omega_{z,measured} = \omega_{z,true} + \lambda_{\omega_z}\omega_{z,true} = (1+\lambda_{\omega_z})\omega_{z,true} \end{cases} \tag{26b}$$

where $\boldsymbol{\lambda}_{\mathbf{f}}$ and $\boldsymbol{\lambda}_{\omega}$ are the scale factors (vectors) for the accelerometer and the gyroscope, respectively. Note that the operation $(1+\boldsymbol{\lambda}_{\mathbf{f}})\mathbf{f}$ (resp. $(1+\boldsymbol{\lambda}_{\boldsymbol{\omega}})\boldsymbol{\omega}$) is not defined,

[5] In reality, bias as well as other sources of error often depends, for instance, on the temperature, which is not an issue, given that it can be calibrated by the sensor's manufacturer.

[6] Subscripts and superscripts dropped for simplicity

[7] Remember that since $\mathbf{f}$ and $\boldsymbol{\omega}$ are both 3D vectors, $\mathbf{b}_{\mathbf{f}}$ and $\mathbf{b}_{\boldsymbol{\omega}}$ are too.

since $\boldsymbol{\lambda}_\mathbf{f}$ (resp. $\boldsymbol{\lambda}_{\boldsymbol{\omega}}$) is a vector. To simplify the notation, we define:

$$\boldsymbol{\Lambda}\vec{\boldsymbol{u}} \triangleq \text{diag}(\boldsymbol{\lambda})\vec{\boldsymbol{u}} = \begin{bmatrix} \lambda_x & 0 & 0 \\ 0 & \lambda_y & 0 \\ 0 & 0 & \lambda_z \end{bmatrix} \vec{\boldsymbol{u}} = \begin{pmatrix} \lambda_x u_x \\ \lambda_y u_y \\ \lambda_z u_z \end{pmatrix} \tag{27}$$

being $\vec{\boldsymbol{u}}$ a generic 3D vector.

Note that the off-diagonal elements of the matrix $\begin{bmatrix} \lambda_x & 0 & 0 \\ 0 & \lambda_y & 0 \\ 0 & 0 & \lambda_z \end{bmatrix}$ correspond to the cross-coupling, the element $a_{ij}$ ($\forall i \neq j$) being proportional to the non-orthogonality between axis $i$ and $j$.

### 5.3.3 Random Noise

As explained in [6], besides bias, scale factor and cross-coupling, inertial sensors output also have another kind of imperfection, called *random noise*, which is a random variable, meaning its values are not predictable and, thus, not possible to be modeled with respect to time. In this work the random noise will be represented by w (or by $\mathbf{w}$ for multidimensional variables). Random variables can be characterized by several different models, being the Gaussian distribution particularly useful and will be presented in 7. Further details about its characteristics and parametrization can be found in the references presented in the beginning of this chapter.

### 5.3.4 Full Error Characteristic

Up to now, this sub-chapter presented separately each one of the imperfections present in the inertial sensors output. Here, the full error characteristic will be presented. Note that the subscripts *true* and *measured* are replaced by the qualifiers $\tilde{}$ and $\hat{}$, respectively.

$$\tilde{\mathbf{f}} = (\boldsymbol{I}_3 + \boldsymbol{\Lambda}_\mathbf{f})\hat{\mathbf{f}} + \mathbf{b}_\mathbf{f} + \mathbf{w}_\mathbf{f} \tag{28a}$$

$$\tilde{\boldsymbol{\omega}} = (\boldsymbol{I}_3 + \boldsymbol{\Lambda}_{\boldsymbol{\omega}})\hat{\boldsymbol{\omega}} + \mathbf{b}_{\boldsymbol{\omega}} + \mathbf{w}_{\boldsymbol{\omega}} \tag{28b}$$

## 5.4 Error Model

As presented throughout the chapter, an inertial sensor output is the combination of the *true* value with other aspects, as bias, scale factor, cross-coupling and random noise, all of them also evolving over time. This chapter will briefly present a model often used to represent their dynamics. Several studies have been conducted during the last couple of decades. [22] compares the performance achieved using different models, [3] and [1] implement one of these models within the context of GPS/INS integration, while [23] and [4] explore in details Power Spectral Density (PSD) and Allan-Variance (AV), two important methods for modeling IMUs that will not be covered in this work, since Fugro had already performed them.

The first order Gauss-Markov process is stated by the references as being a relatively simple model that represents accurately the evolution of bias and scale factor, as follows:

$$\dot{\mathrm{x}} = -\frac{1}{T_c}\mathrm{x} + w \tag{29}$$

where $T_c$ is the correlation time of the process driven by the white noise $w$. $T_c$ can be obtained by Allan-Variance method and its value is usually big, meaning $x_k$ is approximately constant over small periods of time.

The discrete-time equivalence used in this work is:

$$\mathrm{x}_k = \left(1 - \frac{\Delta t}{T_c}\right) \mathrm{x}_{k-1} + w_k \tag{30}$$

# 6 Strapdown System Mechanization

While chapter 5 was dedicated to explain the errors involved in a inertial sensor output, the present one is focused on the manipulation needed for position estimation. An IMU measures two physical quantities: acceleration and angular rates. More precisely, it senses the *proper acceleration* (also called *specific force*) $\mathbf{f}^b$ at body frame $b$, meaning that a free fall towards the center of earth results in a null output. Concerning the gyroscope, it measures the angular rate of the body frame $b$ with respect to the inertial frame $i$. This quantity is denoted by $\boldsymbol{\omega}_{ib}^b$. As consequence, a gyroscope is, in theory, able to measure the Earth's angular rate.

Figure 9 schematizes the computation needed to determine position based on gyroscopes and accelerometers. First, initial values for the orientation, velocity and position must be given. Then, the attitude computer calculates the current orientation based on the gyroscopes. Next, the gravity compensator uses the orientation to subtract the gravity from the accelerometers readings, obtaining the acceleration in the navigation frame. Next, the velocity computer uses this value along with the initial velocity to output the current velocity, which, in turn, is used as input by the position computer, together with the initial position to determine the current one.

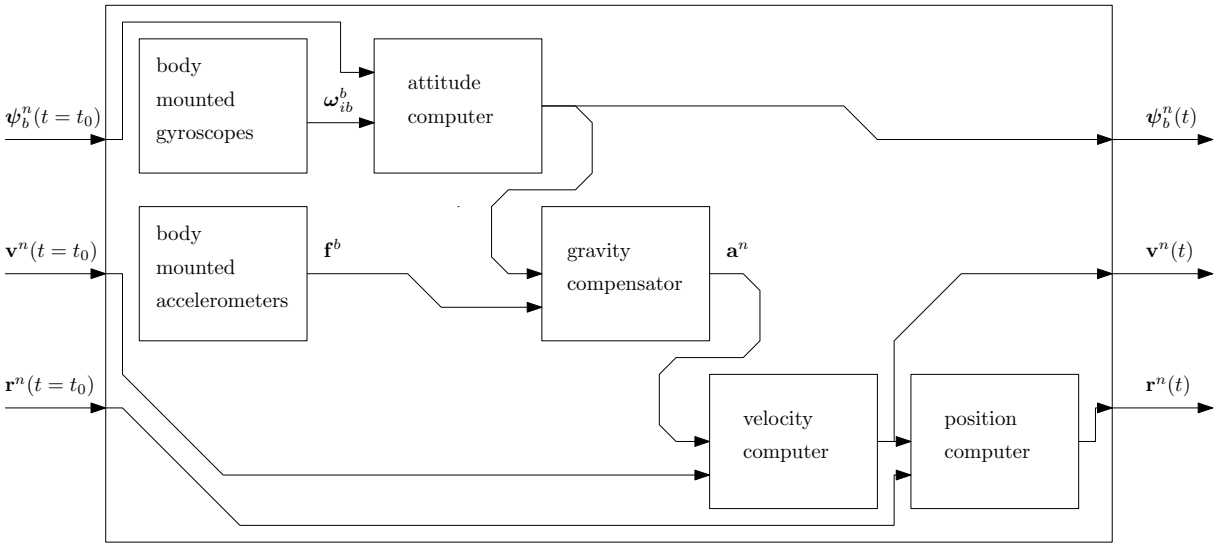



Figure 9: Block diagram illustrating the steps to estimate position in a strapdown system.

All equations are well detailed in the literature [3, 6, 7, 8, 9, 10, 11, 24]. Therefore, the equations will simply be stated and a brief explanation will be given concerning notations and simplifications adopted.

From [6], equation 31 is extracted.

$$\dot{\boldsymbol{C}}_b^n = \boldsymbol{C}_b^n \boldsymbol{\Omega}_{nb}^b = \boldsymbol{C}_b^n \boldsymbol{\Omega}_{ib}^b - (\boldsymbol{\Omega}_{ie}^n + \boldsymbol{\Omega}_{en}^n)\boldsymbol{C}_b^n \tag{31a}$$

$$\dot{\mathbf{v}}_{eb}^n = \boldsymbol{C}_b^n \mathbf{f}_{ib}^b + \mathbf{g}_b^n - (\boldsymbol{\Omega}_{en}^n + 2\boldsymbol{\Omega}_{ie}^n)\mathbf{v}_{eb}^n \tag{31b}$$

$$\dot{\mathbf{r}}_{eb}^n = \mathbf{v}_{eb}^n \tag{31c}$$

$$\boldsymbol{\Omega}_{ie}^{n} = \boldsymbol{\omega}_{ie} \begin{bmatrix} 0 & \sin(L) & 0 \\ -\sin(L) & 0 & -\cos(L) \\ 0 & \cos(L) & 0 \end{bmatrix}, \text{ where } \boldsymbol{\omega}_{ie} = 7.292115 \times 10^{-5} rad/s \tag{32a}$$

$$\mathbf{g}_{b}^{n} \triangleq \begin{pmatrix} 0 & 0 & g(L) \end{pmatrix}^{T} \tag{32b}$$

$$g(L) = 9.7803253359 \left( \frac{1 + 0.00193185265241 \sin(L)^2}{\sqrt{1 - 0.00669437999013 \sin(L)^2}} \right) ms^{-2} \tag{32c}$$

A brief explanation follows:

- $\boldsymbol{\Omega}_{ib}^{b}$ is the true angular rate matrix, i.e. the angular rate matrix extracted from the gyroscope output after correction for bias and scale factor.
- $\boldsymbol{\omega}_{ie}$ is the angular velocity between the $i$-frame and the $e$-frame $z$-axis.
- $\mathbf{f}_{ib}^{b}$ is the true specific force, i.e. the specific force output by the accelerometer and corrected for bias and scale factor.

The simplifications below are adopted, yielding to equation 33:

- Given that $e$-frame is considered static with respect to $n$-frame, $\boldsymbol{\Omega}_{en}^{n}$ is $\mathbf{0}_{3\times3}$
- Given that the device is going to be used by walking human beings, the velocity $\mathbf{v}_{eb}^{n}$ is assumed to be low (below 2m/s), and $2\boldsymbol{\Omega}_{ie}^{n}\mathbf{v}_{eb}^{n}$ is neglected

$$\dot{\boldsymbol{C}}_{b}^{n} = \boldsymbol{C}_{b}^{n}\boldsymbol{\Omega}_{ib}^{b} - \boldsymbol{\Omega}_{ie}^{n}\boldsymbol{C}_{b}^{n} \tag{33a}$$

$$\dot{\mathbf{v}}_{eb}^{n} = \boldsymbol{C}_{b}^{n}\mathbf{f}_{ib}^{b} + \mathbf{g}_{b}^{n} \tag{33b}$$

$$\dot{\mathbf{r}}_{eb}^{n} = \mathbf{v}_{eb}^{n} \tag{33c}$$

From equation 33 it is possible to reexplain the diagram on figure 9. Equation 33a is the attitude computer; equation 33b is both the gravity compensator and the velocity computer; equation 33c is the position computer.

# 7 Stochastic Processes

Last chapter was dedicated to present deterministic processes and different forms of representing them. The models presented were considered to be perfectly accurate. In other words, for the cart example, the input force $f(t)$ was supposed to be accurately known during all times. However, in a real-world application, some physical quantities are obtained through sensors, which are intrinsically noisy, meaning their output is not really true. This characteristic needs to be taken into account when estimating a system's states. This chapter will only serve as a bridge between the previous and the next one. Details on stochastic processes can be found in [10, 19, 25].

## 7.1 Gaussian Distributions

As presented in section 5, sensors (especially the ones used in this work) are inherently noisy. Due to different reasons, their output is usually off the true value. Once both scale factor and bias are removed, there is still the a factor left leading to uncertainties in the measurement. This value is random and was presented in 5.3.3.

Noise is often modeled as a Gaussian distribution for convenience and *accuracy*, given that such distribution is fully described by two values: a mean $\mu$ and a variance $\sigma^2$, being represented as $x \sim N(\mu, \sigma^2)$. In this context, *accuracy* was used in italic font because it is usually not possible to prove that a random variable perfectly behaves according to a Gaussian distribution, but at least it is usually a reasonable assumption, since as it is expected to happen in the real world, a Gaussian distribution states that the farther from the actual value an output is, the less likely it is to occur.

Its density function is described by:

$$p_x(X) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(X-\mu)^2}{2\sigma^2}\right) \tag{34}$$

## 7.2 Random Noise Unfolding

After random noise and Gaussian distributions have been briefly introduced, it is needed to include it in a deterministic process, taking into account the consequences of it into the states.

Imagine the simple scenario of a frictionless cart restricted to an unidimensional movement along a $x-$axis and equipped with a perfectly aligned accelerometer able to measure the cart's acceleration with respect to a fixed object without bias or scale-factor issues. One wants to estimate the cart's position based only on the accelerometer. The problem might be modeled as follows [8]:

$$\begin{pmatrix} x \\ \dot{x} \end{pmatrix}_{k+1} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 + \Delta t \ddot{x}_k \dot{x}_k^{-1} \end{bmatrix} \begin{pmatrix} x \\ \dot{x} \end{pmatrix}_k \tag{35}$$

$$\mathbf{x}_{k+1} = \mathbf{\Phi}_k \mathbf{x}_k \tag{36}$$

[8] Note that $\Delta t \ddot{x}_k \dot{x}_k^{-1} \dot{x}_k$ has no singularities.

And, taking into account the noise output by the sensor, we obtain:

$$\begin{pmatrix} x \\ \dot{x} \end{pmatrix}_{k+1} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 + \Delta t \ddot{x}_k / \dot{x}_k \end{bmatrix} \begin{pmatrix} x \\ \dot{x} \end{pmatrix}_k + \begin{bmatrix} 0 \\ \Delta t \end{bmatrix} \begin{pmatrix} w \end{pmatrix}_k \tag{37}$$

That can be rewritten as:

$$\mathbf{x}_{k+1} = \mathbf{\Phi}_k \mathbf{x}_k + \mathbf{G}_k \mathbf{w}_k \tag{38}$$

Of course a given uncertainty in the acceleration unfolds bigger uncertainty in velocity and an even bigger uncertainty in position. Here the *convenience* evoked in last section arises again: Gaussian distributions make easy to compute the uncertainty propagation. The equations will be presented along with the Kalman filter in chapter 8.

## 7.3 Synthesis of Notations Equivalences

Even though this work has a list of symbols, the present chapter appeared to be useful to make sure no confusion is caused by the notation adopted for dynamic systems in this work, which might differ from some reference books, notably from [18]. The reader should be able to spot the notation with which he/she is most familiar and, by inspection, find its equivalents adopted in this thesis.

### 7.3.1 General Notation

Professionals with background in signal processing or designing control systems are probably more comfortable with the following notation:

| System type | Continuous model | Discrete model |
|---|---|---|
| Linear time-invariant | $\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t)$<br>$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t)$ | $\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k$<br>$\mathbf{y}_k = \mathbf{C}\mathbf{x}_k + \mathbf{D}\mathbf{u}_k$ |
| Linear time-variant | $\dot{\mathbf{x}}(t) = \mathbf{A}(t)\mathbf{x}(t) + \mathbf{B}(t)\mathbf{u}(t)$<br>$\mathbf{y}(t) = \mathbf{C}(t)\mathbf{x}(t) + \mathbf{D}(t)\mathbf{u}(t)$ | $\mathbf{x}_{k+1} = \mathbf{A}_k\mathbf{x}_k + \mathbf{B}_k\mathbf{u}_k$<br>$\mathbf{y}_k = \mathbf{C}_k\mathbf{x}_k + \mathbf{D}_k\mathbf{u}_k$ |
| Non-linear time-invariant | $\dot{\mathbf{x}}(t) = a(\mathbf{x}(t), \mathbf{u}(t))$<br>$\mathbf{y}(t) = c(\mathbf{x}(t), \mathbf{u}(t))$ | $\mathbf{x}_{k+1} = a(\mathbf{x}_k, \mathbf{u}_k)$<br>$\mathbf{y}_k = c(\mathbf{x}_k, \mathbf{u}_k)$ |
| Non-linear time-variant | $\dot{\mathbf{x}}(t) = a(t, \mathbf{x}(t), \mathbf{u}(t))$<br>$\mathbf{y}(t) = c(t, \mathbf{x}(t), \mathbf{u}(t))$ | $\mathbf{x}_{k+1} = a_k(\mathbf{x}_k, \mathbf{u}_k)$<br>$\mathbf{y}_k = c_k(\mathbf{x}_k, \mathbf{u}_k)$ |

Note that even though both continuous-time and discrete-time equations are represented using the same letters, $\mathbf{A}(t) \neq \mathbf{A}_k$, as presented on chapter 4.2. This fact is also clearer in the notation adopted by this thesis.

Often time notation is dropped, leading to:

| System type | Continuous model | Discrete model |
|---|---|---|
| Linear | $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$<br>$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u}$ | $\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}$<br>$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u}$ |
| Non-linear | $\dot{\mathbf{x}} = a(\mathbf{x}, \mathbf{u})$<br>$\mathbf{y} = c(\mathbf{x}, \mathbf{u})$ | $\mathbf{x}_{k+1} = a(\mathbf{x}_k, \mathbf{u})$<br>$\mathbf{y} = c(\mathbf{x}, \mathbf{u})$ |

Once again it is important to note that in such a simplified notation, there is no difference between the representation of a time-invariant system, compared to a time-variant one. At first glance, this notation may appear to be misleading, but one should be able to clearly identify the type of the system from the context.

### 7.3.2 This Thesis' Notation

The notation used throughout this work is mainly inspired on [6, 10]. They both follow the same notation as other reference books within the Kalman filter[9] domain, as [26].

For simplicity, only the time-dropped notation will be presented.

| System type | Continuous model | Discrete model |
|---|---|---|
| Linear | $\dot{\mathbf{x}} = \mathbf{F}\mathbf{x} + \mathbf{G}\mathbf{w}$<br>$\mathbf{z} = \mathbf{H}\mathbf{x} + \boldsymbol{\nu}$ | $\mathbf{x}_{k+1} = \boldsymbol{\Phi}\mathbf{x}_k + \mathbf{G}\mathbf{w}$<br>$\mathbf{z} = \mathbf{H}\mathbf{x} + \boldsymbol{\nu}$ |
| Non-linear | $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}) + g(\mathbf{w})$<br>$\mathbf{z} = h(\mathbf{x}, \mathbf{u}) + \boldsymbol{\nu}$ | $\mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}) + g(\mathbf{w})$<br>$\mathbf{z} = h(\mathbf{x}, \mathbf{u}) + \boldsymbol{\nu}$ |

Please note that $\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$ (resp. $\mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k$) has been encapsulated by $\mathbf{F}\mathbf{x}$ (resp. $\boldsymbol{\Phi}\mathbf{x}_k$). Also, $\mathbf{G}\mathbf{w}$ is a new term, responsible for describing the system noise, as it is described on 7.

[9] The Kalman filter is presented on 8

# 8 The Kalman Filter

There are several books written about the Kalman Filter presenting the whole theory along with demonstrations, like [19]. Therefore, it is not worth redeveloping all from scratch. This chapter will rather just explain the filter from a practical point of view. Readers feeling the explanation throughout this chapter is not enough are kindly invited to refer to [25, 27].

## 8.1 An overview

As discussed in previous chapters, all sensors are noisy. Also, mathematical models for describing real phenomena are never perfectly accurate. The Kalman Filter was created for taking into account uncertainties related to both measurements and models. The filter is a recursive Bayesian filter for multivariate normal distributions and, by construction, is optimal when some assumptions are true [19].

The filter's goal is to compute the *most likely output*, based on the confidence we have on both model and measurements. For the Kalman Filter to work optimally, the three following hypothesis must be satisfied:

- The noise is white (a zero-mean uncorrelated Gaussian distribution);
- The process is a Markov's chain;
- The system is linear and time invariant.

A Markov chain may be roughly described as a memoryless stochastic process, which means the next step state vector depend only on the present one (and not on previous ones), which is usually true for a system properly represented using state-space representation.

The system's linearity and time invariance allows us to use the linear algebra described on Kalman's equations.

In case the studied system does not match these conditions, variations of the filter may still work satisfactorily. In particular, for systems that are locally linear and time invariant under zero-mean uncorrelated noise, the Extended Kalman Filter (EKF) is claimed to work fine [1, 6, 8].

## 8.2 The Standard Kalman Filter algorithm

The *most likely output* is the output that minimizes the quadratic weighted error between measurements and predicted states, as stated by equation 39. Note that $\hat{\mathbf{x}}$ represents an estimation on $\mathbf{x}$, $\tilde{\mathbf{y}}$ represent the approximated error between the measurement and the state, the subscript $_k$ represents the instant $k$ and $G_{a|b}$ is the Bayes' rule notation.

$$\hat{\mathbf{x}}_{k|k-1} = \mathbf{F}_k\hat{\mathbf{x}}_{k-1|k-1} + \mathbf{B}_k\mathbf{u}_k \tag{39a}$$

$$\mathbf{P}_{k|k-1} = \mathbf{F}_k\mathbf{P}_{k-1|k-1}\mathbf{F}_k^T + \mathbf{Q} \tag{39b}$$

$$\tilde{\mathbf{y}}_k = \mathbf{z}_k - \mathbf{H}_k\hat{\mathbf{x}}_{k|k-1} \tag{39c}$$

$$\mathbf{S}_k = \mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}_k^T + \mathbf{R} \tag{39d}$$

$$\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}_k^T\mathbf{S}_k^{-1} \tag{39e}$$

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k\tilde{\mathbf{y}}_k \tag{39f}$$

$$\mathbf{P}_{k|k} = (\boldsymbol{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_{k|k-1} \tag{39g}$$

Before diving in the next section with a simple example, it is important to understand the concept of this bunch of letters. Figure 10 is a graphic representation of the equation 39. The lower block (containing $\mathbf{A}$, $\mathbf{C}$ and $\mathbf{x}$) is the real system (the *process*) with the states to be estimated. It is a black box that outputs $\bar{\mathbf{y}}$ given the current states $\mathbf{x}$ and the input $\mathbf{u}$. The upper block is a mathematical model supposed to represent *accurately enough* the real process, so that the estimated states $\hat{\mathbf{x}}$ are equal (or almost) to the hidden internal true states $\mathbf{x}$. If both model and process were exactly the same (and had been initialized with the same values), they would output the same results. Of course, no mathematical model represents the reality perfectly, so the model tends to diverge from the process over time. To overcome it, the model's output is compared to the process' output, resulting in what is called *innovation* (or *"error"*, loosely speaking). This error is weighted by a gain $\mathbf{K}$ and added to the *a priori* (to the innovation) estimated states $\hat{\mathbf{x}}_{k|k-1}$, turning into the *a posteriori* (again, to the innovation) estimated states $\hat{\mathbf{x}}_{k|k}$ being fed to the model.

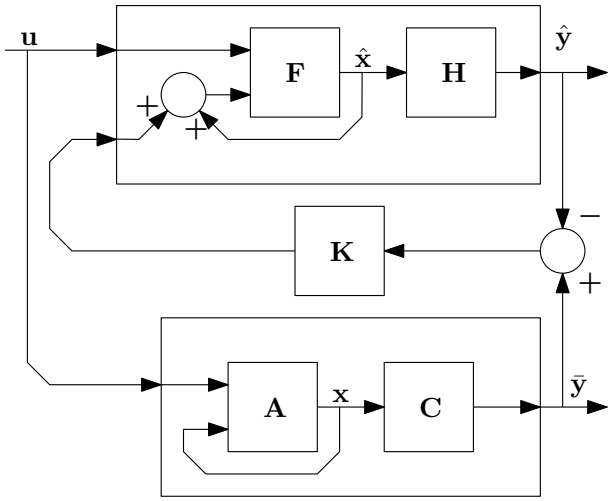



Figure 10: Block diagram of the Kalman filter state equations.

A control engineer should now be able to realize that in a problem of state estimation, the output is the vector of estimated states $\hat{\mathbf{x}}$, rather than the system output $\mathbf{y}$ itself. In addition, the set point is the process' output, the "measured quantity" is the predicted output $\hat{\mathbf{y}}$ and the "error" is the difference between them. Basically, a closed-loop control may be more easily seen on figure 11.

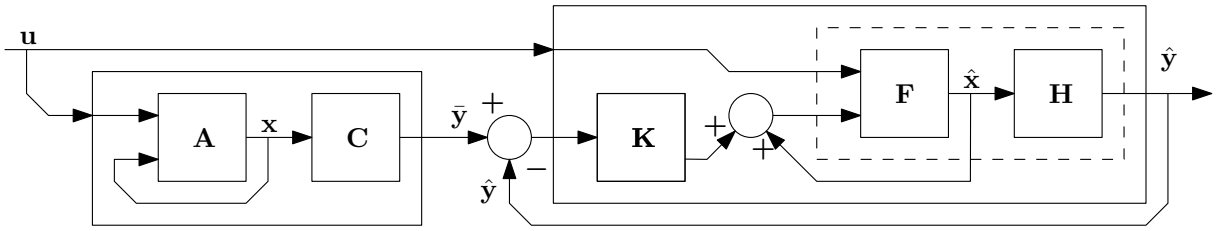



Figure 11: Rearrenged block diagram of the Kalman, making clearer the existence of a closed-loop.

To sum it up, the steps are:

1. Predict the estimated states $\hat{\mathbf{x}}$ at instant $k$, using the estimated states at the previous instant $k-1$ and the mathematical model. The result is the *a priori* state estimate denoted by $\hat{\mathbf{x}}_{k|k-1}$

2. Compute the innovation $\tilde{\mathbf{y}}_k$ at the instant $k$ by comparing the output $\hat{\mathbf{y}}_k$ (also at the instant $k$) given by the model with the output $\bar{\mathbf{y}}_k$ (again, at the same instant $k$) given by the real process.

3. Update the estimated states $\hat{\mathbf{x}}$ at instant $k$, given the innovation $\tilde{\mathbf{y}}_k$ through the equation 39f.

Now, one might wonder how the gain $\mathbf{K}$[10] is computed. The Kalman filter's estimates are, in fact, Gaussian distributions, in which the estimated states form a multidimensional mean vector and have an associated covariance matrix $\mathbf{P}$ that loosely speaking expresses the confidence for each of the states. The algorithm also uses covariance matrix $\mathbf{Q}$ that expresses the confidence we have at the mathematical model of prediction and another covariance matrix $\mathbf{R}$ that express the confidence we have at the output given by the process. As the example in the next section shows, the Kalman gain $\mathbf{K}$ is a sort of ratio between these three covariance matrices.

Finally, given that the internal states $\mathbf{x}$ are never really known, sometimes a notation that drops the hat accent (ˆ) is preferred for simplicity.

## 8.3 A Simple Example

At first glance, it is not trivial to infer the role of each equation. Labbe [25] did a great job writing an interactive book with plenty of examples and intuitive thoughts about the Kalman filter, that served as inspiration for the following example. Imagine a scenario in which an almost-static one-dimensional system is estimated. That is one of the simplest possible cases, since $\mathbf{F}_k = 1$ (since $\hat{\mathrm{x}}_k = \hat{\mathrm{x}}_{k-1}$), $\mathbf{H}_k = 1$ (meaning the state is directly measured) and $\mathbf{B} = 0$ (meaning the is no inputs acting on the system). Remark that scalars are nothing but $1 \times 1$ matrices. As consequence, a matrix inversion turns into a division, while a transposition is the scalar itself. That said, equations become:

$$\hat{\mathrm{x}}_{k|k-1} = \hat{\mathrm{x}}_{k-1|k-1} \tag{40a}$$

$$\mathrm{P}_{k|k-1} = \mathrm{P}_{k-1|k-1} + \mathrm{Q} \tag{40b}$$

$$\tilde{\mathrm{y}}_k = \mathrm{z}_k - \hat{\mathrm{x}}_{k|k-1} \tag{40c}$$

$$\mathrm{S}_k = \mathrm{P}_{k|k-1} + \mathrm{R} \tag{40d}$$

$$\mathrm{K}_k = \mathrm{P}_{k|k-1}{\mathrm{S}_k}^{-1} \tag{40e}$$

$$\hat{\mathrm{x}}_{k|k} = \hat{\mathrm{x}}_{k|k-1} + \mathrm{K}_k\tilde{\mathrm{y}}_k \tag{40f}$$

$$\mathrm{P}_{k|k} = (1 - \mathrm{K}_k)\mathrm{P}_{k|k-1} \tag{40g}$$

The equations above could be true for tracking the position $\mathrm{x}_k$ of an object (let's say an offshore platform) that measures directly its position ($\mathrm{z}_k = \mathrm{x}_{k|k-1}$, for $\tilde{\mathrm{y}} = 0$) and is supposed static, given that $\mathrm{x}_{k|k-1} = \mathrm{x}_{k-1|k-1}$.

40a is the predicted state estimation

[10] Remark that the lowercase $k$ is used for denoting instants, while the uppercase K (or $\mathbf{K}$, its matrix version) is the Kalman gain.

40b is the predicted state covariance
40c is the *innovation* (i.e. the difference between measurement and prediction)
40d is the innovation covariance
40e is the optimal Kalman gain
40f is the updated state covariance
40g is the updated state covariance

To better understanding, let's rewrite 40e, as:

$$\mathrm{K}_k = \frac{\mathrm{P}_{k-1|k-1} + \mathrm{Q}}{\mathrm{P}_{k-1|k-1} + \mathrm{Q} + \mathrm{R}} \tag{41}$$

Analyzing equation 41, it is possible to verify that $0 < \mathrm{K}_k < 1$. It is near to 0 (resp 1) if $\mathrm{R} \gg Q$ (resp $\mathrm{R} \gg Q$). Imagine, now, the platform is on furious waters and it is equipped with a high precision sensor. Therefore, Q would be *large* (or *big*) and R would be *narrow* (or *small*). So, $\mathrm{K} \approx 1$, means the updated position $\hat{\mathrm{x}}_{k|k} \approx \hat{\mathrm{x}}_{k|k-1} + \tilde{\mathrm{y}}_k$. Its retained from 41 that modifying Q and R plays significantly on the filter's behavior, leading to fast convergence or catastrophic divergence.

## 8.4 The Extended Kalman Filter algorithm

The Extended Kalman Filter equations simply are the Standard Kalman Filter ones linearized around the operation point $\hat{\mathbf{x}}_k$, thanks to the Jacobian matrices 42b and 42e. Modifying $\mathbf{Q}$ and $\mathbf{R}$ over time is not only possible, as it is necessary for systems with variable integration time-step, given that the bigger the time-step is, the more uncertainty is added to the process.

$$\hat{\mathbf{x}}_{k|k-1} = f(\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_{k-1}) \tag{42a}$$

$$\mathbf{F}_{k-1} = \left.\frac{\partial f}{\partial \mathbf{x}}\right|_{\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_{k-1}} \tag{42b}$$

$$\mathbf{P}_{k|k-1} = \mathbf{F}_{k-1}\mathbf{P}_{k-1|k-1}\mathbf{F}_{k-1}^T + \mathbf{Q}_{k-1} \tag{42c}$$

$$\tilde{\mathbf{y}}_k = \mathbf{z}_k - h(\hat{\mathbf{x}}_{k|k-1}) \tag{42d}$$

$$\mathbf{H}_k = \left.\frac{\partial h}{\partial \mathbf{x}}\right|_{\hat{\mathbf{x}}_{k|k-1}} \tag{42e}$$

$$\mathbf{S}_k = \mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}_k^T + \mathbf{R}_k \tag{42f}$$

$$\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}_k^T\mathbf{S}_k^{-1} \tag{42g}$$

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k\tilde{\mathbf{y}}_k \tag{42h}$$

$$\mathbf{P}_{k|k} = (\boldsymbol{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_{k|k-1} \tag{42i}$$

The notation above, coming from the Bayes' rule, is quite verbose and will not be adopted in this work. Instead, the following equation will be used:

$$\hat{\mathbf{x}}_k = f(\mathbf{x}_{k-1}, \mathbf{u}_{k-1}) \tag{43a}$$

$$\mathbf{F}_{k-1} = \left.\frac{\partial f}{\partial \mathbf{x}}\right|_{\mathbf{x}_{k-1}, \mathbf{u}_{k-1}} \tag{43b}$$

$$\bar{\mathbf{P}}_k = \mathbf{F}_{k-1}\mathbf{P}_{k-1}\mathbf{F}_{k-1}^T + \mathbf{Q}_{k-1} \tag{43c}$$

$$\tilde{\mathbf{y}}_k = \mathbf{z}_k - h(\hat{\mathbf{x}}_k) \tag{43d}$$

$$\mathbf{H}_k = \frac{\partial h}{\partial \mathbf{x}}\Big|_{\hat{\mathbf{x}}_k} \tag{43e}$$

$$\mathbf{S}_k = \mathbf{H}_k \bar{\mathbf{P}}_k \mathbf{H}_k^T + \mathbf{R}_k \tag{43f}$$

$$\mathbf{K}_k = \bar{\mathbf{P}}_k \mathbf{H}_k^T \mathbf{S}_k^{-1} \tag{43g}$$

$$\mathbf{x}_k = \hat{\mathbf{x}}_k + \mathbf{K}_k \tilde{\mathbf{y}}_k \tag{43h}$$

$$\mathbf{P}_k = (\boldsymbol{I} - \mathbf{K}_k \mathbf{H}_k) \bar{\mathbf{P}}_k \tag{43i}$$

# 9 Numerical Issues

Due to numerical intrinsic truncation process in floating point computations, several rounding errors issues often raise. This chapter aims to present the strategies used to get around the main issues that deteriorated the numerical solution and led the simulation to diverge in some cases.

## 9.1 Covariance Matrix Symmetry

The covariance matrices $\mathbf{P}$ and $\mathbf{S}$ on the Kalman Filter Algorithm are, each of them, symmetric by definition. On the literature [8], some algorithms are proposed to get around the asymmetry caused by floating point finite precision. The alternatives present a trade-off between computational cost and accuracy to the analytical solution, in such a way that there is no consensus among experienced designers about which of them would be the most suitable algorithm regardless the application.

For simplicity, this thesis adopts:

$$\mathbf{A} = 0.5(\tilde{\mathbf{A}} + \tilde{\mathbf{A}}^T), \tag{44}$$

where $\tilde{\mathbf{A}}$ is a possibly asymmetrical covariance matrix and $\mathbf{A}$ is its corrected form.

## 9.2 Matrix Positiveness

Covariance matrices should be, by definition, positive semi-definite. That is, none of its eigenvalues should be strictly negative. Again, due to floating point precision, covariance matrices computations may result in non-positive semi-definite matrices. Tests were conducted and some of the eigenvalues were indeed slightly negative in some steps, typically reaching values up to $-10^{-10}$. It was chosen, then, to neglect this issue in the present work.

## 9.3 Quaternion Normalization

In this work, quaternions are used to represent rotations and therefore must have their norm equal to one. Details about operations involving quaternions are presented in [16] and the most usual computations are presented in [13].

The works of [24, 7, 8] present different approaches, leading to three slightly different numerical procedures with similar results.

Within this subsection, $\tilde{\mathbf{q}}$ will be used to denote the quaternion that possibly is not normalized, while $\mathbf{q}$ will stand for its corrected version, i.e. normalized.

We recall that in the quaternion context, $1 \triangleq \begin{pmatrix} 1 & 0 & 0 & 0 \end{pmatrix}^T$ is defined as being the identity quaternion.

The prior presents that the normalized quaternion is the computed one divided by its norm:

$$\mathbf{q} = \frac{\tilde{\mathbf{q}}}{||\tilde{\mathbf{q}}||} \tag{45}$$

An alternative procedure takes advantage of the property that the multiplication of a rotation quaternion by its conjugate should result in an identity quaternion ($\mathbf{q} \cdot \mathbf{q}^* = 1$)

to define an error quaternion $\Delta\mathbf{q}$ as being the difference to the identity to proceed the normalization. The equations from [7] are re-written below:

$$\Delta\mathbf{q} = 1 - \tilde{\mathbf{q}} \cdot \tilde{\mathbf{q}}^* \tag{46a}$$

$$\mathbf{q} = \frac{\tilde{\mathbf{q}}}{\sqrt{\tilde{\mathbf{q}} \cdot \tilde{\mathbf{q}}^*}} \tag{46b}$$

$$\mathbf{q} = (\tilde{\mathbf{q}} \cdot \tilde{\mathbf{q}}^*)^{-0.5} \cdot \tilde{\mathbf{q}} \tag{46c}$$

$$\mathbf{q} \approx (\tilde{\mathbf{q}} + 0.5\Delta\mathbf{q}) \cdot \tilde{\mathbf{q}} \tag{46d}$$

Finally, [8] states the following procedure:

$$\tilde{\mathbf{q}} = \mathbf{q} + \delta\mathbf{q} \tag{47a}$$

$$\mathbf{q} \cdot \mathbf{q}^* = 1 \tag{47b}$$

$$\tilde{\mathbf{q}} \cdot \tilde{\mathbf{q}}^* = \mathbf{q} \cdot \mathbf{q}^* + \mathbf{q} \cdot \delta\mathbf{q}^* + \delta\mathbf{q} \cdot \mathbf{q}^* + \delta\mathbf{q} \cdot \delta\mathbf{q}^* \approx \mathbf{q} \cdot \mathbf{q}^* + \mathbf{q} \cdot \delta\mathbf{q}^* + \delta\mathbf{q} \cdot \mathbf{q}^* \tag{47c}$$

$$\delta\mathbf{q} = \epsilon_{\mathbf{q}} \cdot \mathbf{q} \tag{47d}$$

$$\tilde{\mathbf{q}} \cdot \tilde{\mathbf{q}}^* = \mathbf{q} \cdot \mathbf{q}^* + \epsilon_{\mathbf{q}} \cdot \mathbf{q} \cdot \mathbf{q}^* = (1 + 2\epsilon_{\mathbf{q}}) \cdot \mathbf{q} \cdot \mathbf{q}^* = 1 + 2\epsilon_{\mathbf{q}} \tag{47e}$$

$$\epsilon_{\mathbf{q}} = 0.5(\tilde{\mathbf{q}} \cdot \tilde{\mathbf{q}}^* - 1) \tag{47f}$$

$$\mathbf{q} = (1 - \epsilon_{\mathbf{q}}) \cdot \tilde{\mathbf{q}} \tag{47g}$$

Tests were conducted in order to compare the normalized quaternion computed using each of these three methods and only difference at execution time level was observed, being the prior method the fastest and, therefore, the one adopted here.

## 9.4 Quaternion Ambiguity

As evoked in [16], a single orientation can be represented with quaternions using both $\mathbf{q}$ and $-\mathbf{q}$. This fact only becomes an issue when in the presence of orientation aiding in a Kalman filter environment, since it leads to a huge residual in the orientation components, what degrades the filter's performance.

Imagine the two following scenarios in which the prediction matches the observation, except from the ambiguity in the second one, i.e. in equation 48a the observed orientation is numerically the same as the predicted one, while in 48b although the observed orientation is the same as the predicted one, it was represented by its symmetric, causing the residual $\mathbf{y}$ to be considerably bigger than what it should be, leading to an undesired correction to the a posteriori state estimate.

$$\mathbf{y} = \mathbf{z} - h(\mathbf{x}) = \begin{pmatrix} 0.6 \\ 0 \\ 0.8 \\ 0 \end{pmatrix} - \begin{pmatrix} 0.6 \\ 0 \\ 0.8 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{48a}$$

$$\mathbf{y} = \mathbf{z} - h(\mathbf{x}) = \begin{pmatrix} -0.6 \\ 0 \\ -0.8 \\ 0 \end{pmatrix} - \begin{pmatrix} 0.6 \\ 0 \\ 0.8 \\ 0 \end{pmatrix} = \begin{pmatrix} 1.2 \\ 0 \\ 1.6 \\ 0 \end{pmatrix} \tag{48b}$$

Ensuring *smoothness*[11] in the observed orientation representation over time is enough to keep the residuals also smooth. Recalling a quaternion is, up to an extent, a generalization of a complex number, one might try to ensure *smoothness* by simply keeping positive the quaternion's real component. However, given that the predictions are *smooth*, the estimated real component can assume both positive and negative values. For this reason, the following algorithm is adopted:

$straight\_difference \leftarrow \mathbf{q}_{i-1} - \mathbf{q}_i$
$straight\_sum \leftarrow \Sigma straight\_difference$
$symmetric\_difference \leftarrow -\mathbf{q}_{i-1} - \mathbf{q}_i$
$symmetric\_sum \leftarrow \Sigma symmetric\_difference$
**if** $straight\_sum > symmetric\_sum$ **then**
  $\mathbf{q}_i \leftarrow -\mathbf{q}_i$
**else**
  $\mathbf{q}_i \leftarrow \mathbf{q}_i$
**end**

**Algorithm 1:** Ensures smoothness of observed orientation representation over time

[11] In this context, *smooth* quaternion data is a stream that does not swap with between the two possible attitude representations. In other words, if the stream was continuous (rather than discrete), it would mean that each of its components would be differentiable.

# 10 Application

This chapter gathers the pieces of information presented over the thesis that allow the tests and results on chapter 11. The discrete-time equivalences were computed using the method explained in chapter 4.2.1. New symbols are used and their meaning can be found on appendix A. Furthermore, part of the subscripts were dropped for improving readability.

## 10.1 The Prediction Model

As presented on chapter 8, the position estimation is made by merging the output of a mathematical model with observations made by aiding sources. Chapter 6 presented the mechanization equations, while chapter 5 modeled the inertial sensors that measure both 3D linear acceleration and 3D angular rates necessary to solve for position. Chapter 10.1.1 presents the simplified kinematic model in which the sensors errors are not taken into account and chapter 10.1.2 addresses the sensors.

### 10.1.1 The Simplified Kinematic Model

The states are defined on equation 49 and unfolded on equation 50. It comprises the 3D position, 3D velocity and orientation, expressed by a unitary quaternion as explained on chapter 3.

$$\hat{\mathbf{x}} = \begin{pmatrix} \hat{\mathbf{r}}^n & \hat{\mathbf{v}}^n & \hat{\boldsymbol{\psi}}_b^n \end{pmatrix} \tag{49}$$

$$\hat{\mathbf{x}} = \begin{pmatrix} x^n & y^n & z^n & \dot{x}^n & \dot{y}^n & \dot{z}^n & q_{0_b^n} & q_{1_b^n} & q_{2_b^n} & q_{3_b^n} \end{pmatrix}^T \tag{50}$$

In this context, $\mathbf{u}$ is not the usual *control input* that allows the process to be driven to a desired state, but rather a control input that (at least in theory) allows the estimated states to be driven to the desired values. It follows from the discussion dedicated to control engineers on chapter 8.2. After all, it is just an input to the system and it makes no difference calling it as *control input* or simply *input*. It consists of the IMU measurements and is presented on equation 51.

$$\tilde{\mathbf{u}} = \begin{pmatrix} \tilde{\mathbf{f}}^b & \tilde{\boldsymbol{\omega}}_{ib}^b \end{pmatrix}^T = \begin{pmatrix} \tilde{f}_x{}^b & \tilde{f}_y{}^b & \tilde{f}_z{}^b & \tilde{\omega}_{x_{ib}^b} & \tilde{\omega}_{y_{ib}^b} & \tilde{\omega}_{z_{ib}^b} \end{pmatrix}^T \tag{51}$$

The continuous-time equations are rewritten (equation 52) from equation 33 for convenience.

$$\begin{pmatrix} \dot{\mathbf{r}}^n \\ \dot{\mathbf{v}}^n \\ \dot{\boldsymbol{\psi}}_b^n \end{pmatrix} = f(t) = \begin{pmatrix} \mathbf{v}^n \\ \mathbf{g}^n + \boldsymbol{C}_b^n \mathbf{f}^b \\ 0.5 \left( Q\left(q(\boldsymbol{\omega}_{ib}^b)\right) + Q\left(q(\boldsymbol{\omega}_{ie}^n)\right) \right) \boldsymbol{\psi}_b^n \end{pmatrix} \tag{52}$$

$$\mathbf{F}(t) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \frac{\partial\left(\boldsymbol{C}_b^n \mathbf{f}^b\right)}{\partial \boldsymbol{\psi}_b^n} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{\partial\left(0.5\left(Q\left(q(\boldsymbol{\omega}_{ib}^b)\right)+Q\left(q(\boldsymbol{\omega}_{ie}^n)\right)\right)\boldsymbol{\psi}_b^n\right)}{\partial \boldsymbol{\psi}_b^n} \end{bmatrix} \tag{53}$$

### 10.1.2 The Full Kinematic Model

The full kinematic model is the assembly of the so-called *simplified* with the model that describes the sensors' bias and scale factors only, as equation 54 expresses.

$$\bar{\hat{\mathbf{x}}} \begin{pmatrix} \hat{\mathbf{x}} & \lambda_{\mathbf{a}} & b_{\mathbf{a}} & \lambda_{\boldsymbol{\omega}} & b_{\boldsymbol{\omega}} \end{pmatrix}^T \tag{54}$$

The augmented prediction function $f_{aug}$ is defined for convenience as stated on equation 55.

$$f_{aug}(\bar{\hat{\mathbf{x}}}, \mathbf{u}) = f(\hat{\mathbf{x}}, \mathbf{u}) + \bar{f}(\hat{\mathbf{x}}, \mathbf{u}) \tag{55}$$

$$\bar{f}(\hat{\mathbf{x}}, \mathbf{u}) = \begin{pmatrix} 0 \\ \boldsymbol{C}_b^n \Lambda_{\mathbf{a}} + \boldsymbol{C}_b^n {b_{\mathbf{a}}}^b \\ 0 \\ 0.5Q\left(q(\Lambda_{\boldsymbol{\omega}})\right)\boldsymbol{\psi} + 0.5Q\left(q(b_{\boldsymbol{\omega}})\right)\boldsymbol{\psi} \\ \lambda_{\mathbf{a}} \\ b_{\mathbf{a}} \\ \lambda_{\boldsymbol{\omega}} \\ b_{\boldsymbol{\omega}} \end{pmatrix} \tag{56}$$

## 10.2 Camera Aiding

### 10.2.1 Camera to IMU Calibration

As described on chapter 2 and depicted on figures 6 and 23, the camera and IMU are mounted together on an aluminum profile and the origins of their frames are not coincident. The translational offset is determined using a Vernier caliper, but, due to the difficult to determine the exact origin of the IMU and the impossibility of reaching the origin of the camera, the expected accuracy is at the order of millimeters.

For determining the attitude offset between both frames, Fugro has created a Levenberg–Marquardt algorithm implementation for it. The camera orientation over time is extracted from QuickVision, allowing rotation matrices $\boldsymbol{C}_{k-1}^k$ between every pair of two consecutive epochs to be computed. For an ideal gyroscope, the same rotation matrices can by computed by integration. The algorithm inputs camera and IMU data and estimates not only the attitude offset, but also the gyroscope bias and scale factor, as well as the time drift between the camera and the IMU clocks.

### 10.2.2 Observation Equations

Even after calibration, a small error in the orientation offset between camera and IMU might remain. As explained on chapter 8, the error of the aiding can be noisy, as long as it is zero-mean. As this condition cannot be satisfied, the camera aid is restricted to the position vector.

$$\mathbf{z} = \begin{pmatrix} \tilde{x}^n & \tilde{y}^n & \tilde{z}^n \end{pmatrix}^T \tag{57}$$

$$\mathbf{y} = \mathbf{z} - h(\hat{\mathbf{x}}) = \begin{pmatrix} \tilde{\mathbf{r}}^n \end{pmatrix} - \begin{pmatrix} \hat{\mathbf{r}}^n \end{pmatrix} \tag{58}$$

$$\mathbf{H} = \frac{\partial h}{\partial \hat{\mathbf{x}}} = \begin{bmatrix} \boldsymbol{I}_3 & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times4} \end{bmatrix} \tag{59}$$

## 10.3 Zero Velocity Update (ZUPT)

Zero Velocity Update is a kind of virtual aiding that one can implement to enforce an existing constraint. Cars in general are supposed not to move laterally or up and down. [1] takes advantage of this behavior by creating a virtual sensor that outputs measures of zero velocity for both lateral and vertical components.

Other works focus entirely on ZUPT: its detection, benefits, applications and limitations. Both [28] and [5] develop algorithms for detecting zero-velocity periods, so that ZUPT can be used. The topic is also addressed on [6] and [11] for explaining alignment and early usage of ZUPT. Algorithms to detect stationary periods that allow ZUPT to be performed may vary according to the application, available sensors and performance requirements.

For the system studied in the present work, in which the IMU is held by a human that is walking, it is reasonable to state that the vertical component of the mobile phone's velocity is - in average - zero and, from experiments, can hardly reach $7km/h$ (about $2m/s$). Vertical stationary periods are assumed at all times and vertical ZUPT is, then, always performed.

$$\mathbf{z}_{zupt} = \begin{pmatrix} 0 \end{pmatrix} \tag{60}$$

$$\mathbf{y}_{zupt} = \mathbf{z}_{zupt} - h_{zupt}(\hat{\mathbf{x}}) = \begin{pmatrix} 0 \end{pmatrix} - \begin{pmatrix} \hat{\mathbf{v}}_{z^n} \end{pmatrix} \tag{61}$$

$$\mathbf{H}_{zupt} = \frac{\partial h_{zupt}}{\partial \hat{\mathbf{x}}} = \begin{bmatrix} \mathbf{0}_{1\times5} & \boldsymbol{I}_1 & \mathbf{0}_{1\times4} \end{bmatrix} \tag{62}$$

## 10.4 Integration Architecture

For the Kalman Filter to work properly, it is crucial to use the adequate data in every single step. Figure 12 depicts a simplified scenario of data arrival time.

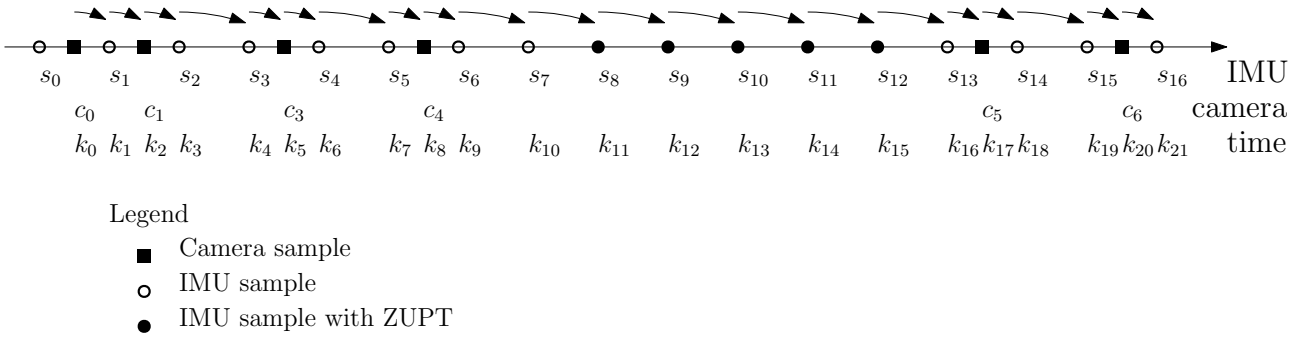



Figure 12: Filter steps in time according to sensor data arrival.

As explained in chapter 5.2, $s_j$ is IMU sample that is not the $\boldsymbol{\omega}^b_{ib,j}$ (i.e. the angular velocity at time $j$), but $\boldsymbol{\psi}^{b_j}_{b_{j-1}}/(t_j - t_{j-1})$ (i.e. the rotation from the instant $j-1$ to $j$ divided by the period between IMU samples). The straightforward consequence is that

$s_j$ is only valid between $s_{j-1}$ and $s_j$. For that reason, a camera sample is not used until another IMU sample arrives.

Lets use figure 12 to exemplify, starting from $s_1$. By $c_1$ arrival, the algorithm raises a flag for camera aiding availability, stores its values and time-stamp. With the arrival of $s_2$, the algorithm obtains values valid from $s_1$ to $s_2$, what comprises $c_1$. Updating to $s_2$ is then made in three steps:

1 - Prediction of the states at $c_1$ using the ones from $s_1$ and the IMU readings at $s_2$

2 - Update the states at $c_1$ using the camera measurements (which are also at $c_1$)

3 - Prediction of the states at $s_2$ using the ones from $c_1$ and the IMU readings at $s_2$

After $frequency_{camera}/frequency_{IMU}$ seconds without new camera samples injected into the algorithm (i.e. after a camera sample not arriving when it was supposed to), the algorithm enters in ZUPT mode, represented by bullet points in figure 12. In other words, a camera observation sample was supposed to arrive between $s_7$ and $s_8$, given that the IMU frequency is double the camera one for this example. ZUPT procedure is described below:

1 - Prediction of the states at $s_j$ using the ones from $s_{j-1}$ and the IMU readings at $s_j$

2 - Update the states at $s_j$ using the zero velocity assumption at $s_j$

## 10.5 Data Rejection

Due to lightening conditions, camera eventually may output one aberrant observation, off from the reality by more than three times the standard deviation[12]. Henceforth it will be called *outlier* and it corresponds to spikes when plotting observations over time. As this kind of disturbance is not a zero-mean Gaussian noise as the Kalman filter assumes, it drastically degrades the filter's performance. Of course there are no means to guarantee *perfect* data acquisition so it would allow to filter such anomalies out. Nevertheless, it is possible to develop an algorithm that will decide whether a given observation will be considered an outlier or not based on its neighbors.

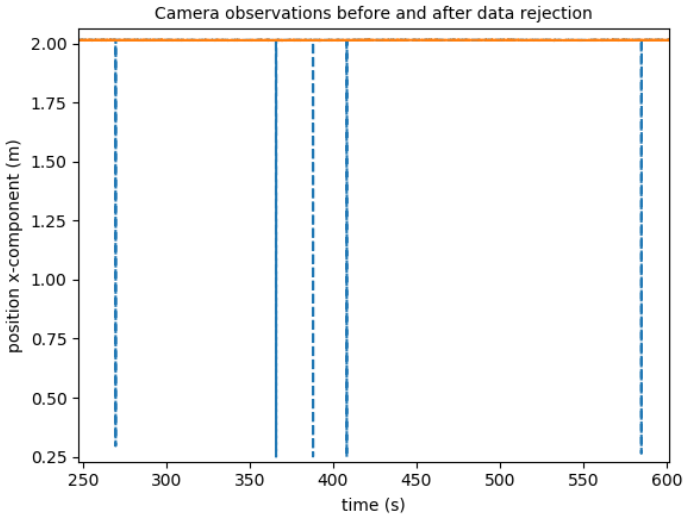
The procedure of approving or rejecting an observation is described in the algorithm 2, in which $th$ is a fixed threshold, the subscript $k$ is a fixed number defining how many previous observations will be taken into account and $s$ is the sum of the position's components observed by the camera.

$s_i = r_{x,i} + r_{y,i} + r_{z,i}$
**if** $s_i < median(s_i, s_{i-1}, \ldots, s_{i-k}) + th$ **and** $s_i > median(s_i, s_{i-1}, \ldots, s_{i-k}) - th$ **then**
  observation is used for the Kalman filter
**else**
  observation is rejected
**end**

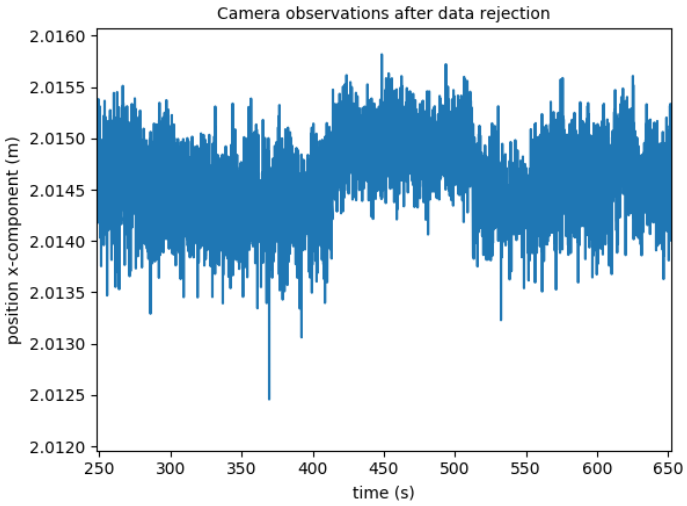
**Algorithm 2:** Filter that determines which measurements will be considered as outliers, based on $k$ previous measurements and a constant threshold $th$.

[12]Recall that for a Gaussian distribution, three times the standard deviation around the mean covers 99% of points belonging to the distribution

Figure 13 illustrates the result of applying data rejection algorithm to camera observations for an indoor static dataset.




(a) $x$-component of camera measured position before (blue dashed line) and after (orange solid line) applying the median filter.




(b) $x$-component of camera measured position after applying the median filter.

Figure 13: Result of data rejection algorithm application in an indoor static dataset.

A static dataset was chosen to have its results presented so that every major spike can be easily spotted by the reader as an outlier. Figure 13a displays the $x$-component of the camera position before the algorithm was applied, while figure 13b presents the remaining samples after the filter. It is important to remark that even for this indoor static dataset under roughly the same light conditions over the whole dataset, the spikes overcome $1.7m$. In addition, from figure 13b, a subcentimetric precision is observed.

## 10.6 Material

For the first and the second datasets (chapters 11.2 and 11.3 respectively), the equipment consisted of:

- One camera AlliedVision Manta G504-B [29]

- One MEMS IMU Silicon Sensing DMU-11 [30] [31]
- One computer
- 20 size A2 patterns

The material involved on the third dataset is detailed in the beginning of its section.

Contrarily to the DMU-11, this particular camera is able to time-stamp its output with respect to the time it was turned on. The IMU time-stamping needs, then, to be done by the computer. However, a computer is not designed to be good at time-stamping. Instead, it is designed to make the user have the impression that multiple functions are performed simultaneously. To do so, it spends a portion of time to a task and then swaps to another one. As consequence, IMU samples are times-stamped with a variable delay. To get around this issue, the following procedure is adopted using a sample counter output by the IMU:

1. Time-stamp the first logged sample
2. Time-stamp the last logged sample
3. Determine the frequency of output by dividing the value of the counter of the last logged sample by the difference between the time-stamps of the first and the last logged samples
4. Re-time-stamp the intermediate samples based on the counter value and the computed frequency

In the worst scenario, by the moment the first sample arrives to the computer, it is not busy and time-stamp it correctly, and by the moment the last sample arrives, the computer is busy and time-stamps the sample after 6ms. However, the camera to IMU calibration explained on chapter 10.2.1 also corrects for this time drift.

The software was developed using the distribution Anaconda of Python 3.6.4. Numpy, Scipy and Pyquaternion libraries were used and all computations were performed using double-precision floating-point format.

# 11 Tests and Results

## 11.1 Calibration Using Robot ABB IRB-120

As explained in chapter 5, sensors output are off the true value by cross-coupling, scale factor and bias. The main objective of this calibration is to determine such parameters. For it, we take advantage of the fact that, during static periods, only the specific force is sensed by the accelerometer. Hence, the magnitude of the measurements output by the 3D accelerometer should be constant regardless the pose, what can be achieved after estimating the parameters to their corrected values.

Imagine the following simple scenario in which only the bias plays. One places a 1D accelerometer vertically and outputs $9.85ms^{-2}$. Then, the same accelerometer is turned upside down and outputs $-9.75ms^{-2}$. It would be possible to infer a bias of $0.05ms^{-2}$. However, a noisy measurement would impair such conclusion, given that it would be no longer possible, for a single measurement, to distinguish between noise and bias. For a zero-mean noise, averaging the output removes noise and allows determining the bias under the additional circumstance of an *unbiased dataset*[13]. For instance, if data from the example were collected 10 minutes with the sensor upside up and only 1 minute with it upside down, the resulting average magnitude would be $9.84ms^{-2}$, which would be $0.04ms^{-2}$ off from an *unbiased dataset* collected the same amount of time both upside up and upside down.

For a 3D accelerometer also presenting scale factor and cross-coupling (apart from noise and bias), a similar stochastic approach and an unbiased dataset are also needed. For a 3D case, the unbiased dataset no longer consists of an static upside-down-upside-up setup, but of equally distributed poses around a sphere.

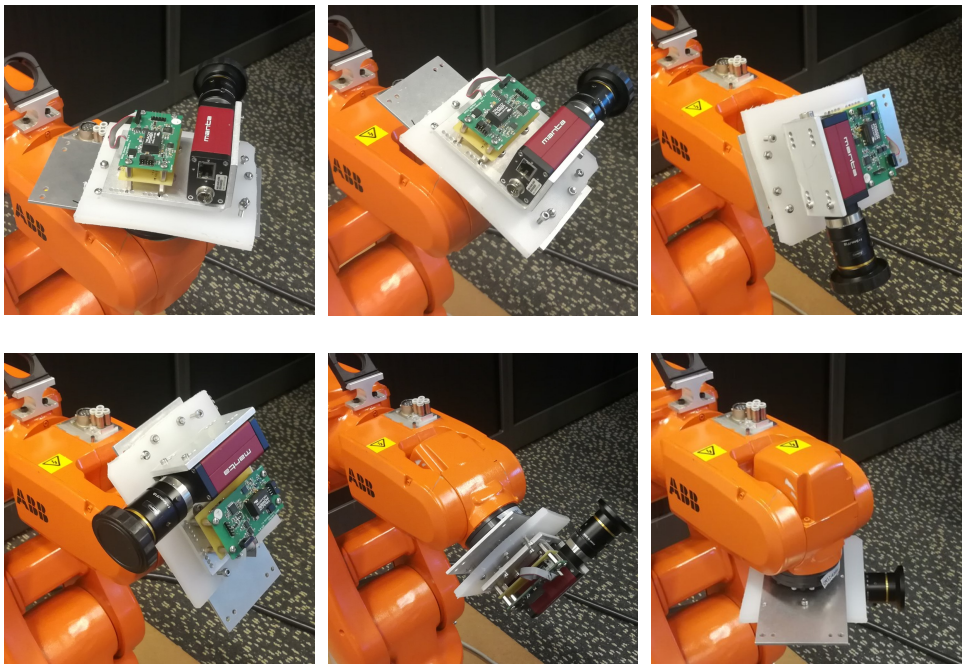
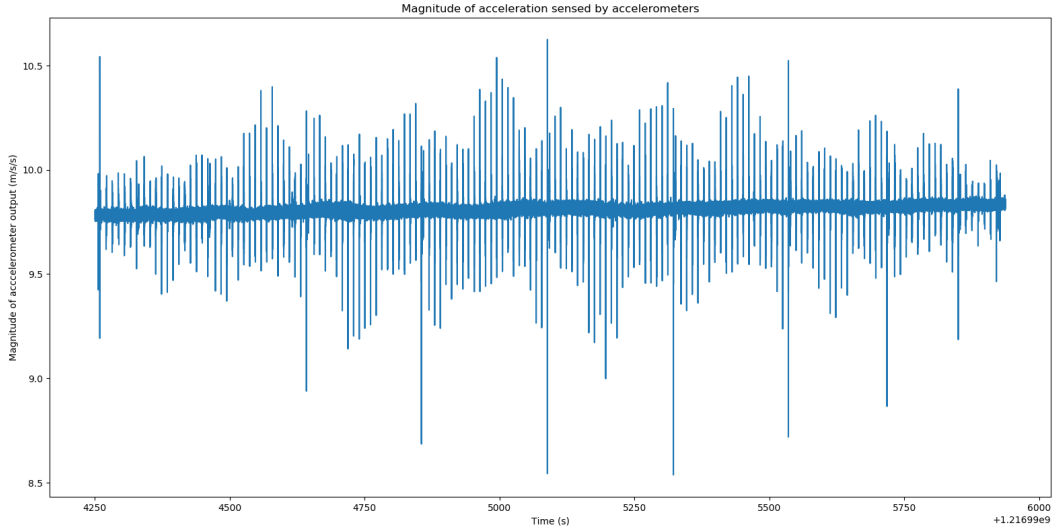


Figure 14: Some of the static poses performed using robot IRB-120.

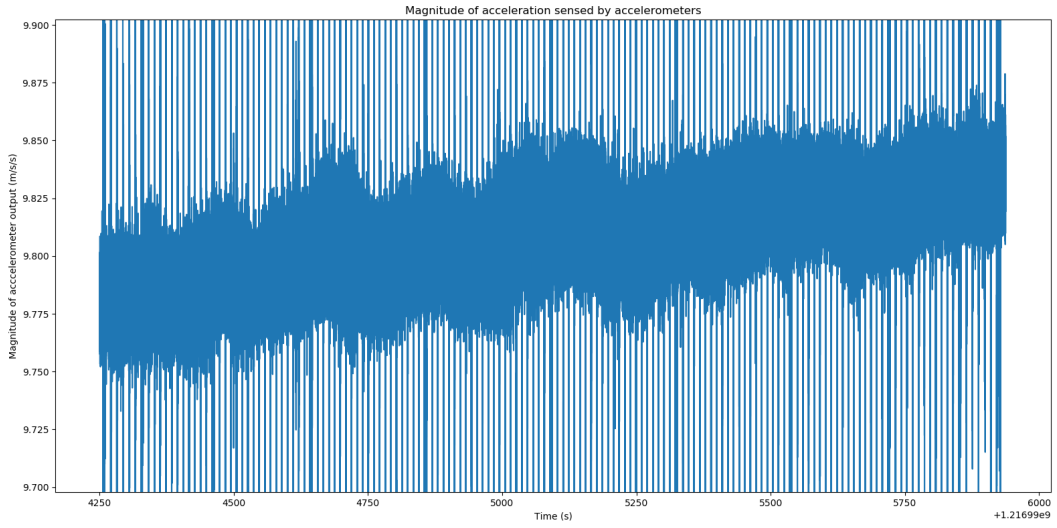
[13]Do not confuse *biased dataset* with *biased sensor*. A biased sensor outputs biased data, while a biased dataset is a dataset that leads to biased results. In other words, an unbiased dataset may contain biased data as long as unbiased results can be achieved.

The robot IRB-120, manufactured by ABB, was used for collect a few datasets with fixed amount of time static intervals in several different poses, equally distributed around the sphere created by turning its axis number 5 and 6, as defined on the robot's manual [32]. The robot was programmed to this task, increasing repeatability. Figure 14 illustrates some of the poses reached by the robot.

Datasets were collected under different temperature conditions, with different turn-on bias and with two different units of the IMU model DMU-11. For a given unit, the results obtained were the same regardless other varying conditions. Figure 15 shows the accelerations' magnitudes obtained from one of the datasets. Note that the spikes correspond to the moments at which the pose was not static, i.e. the periods during which the robot was moving from a pose to another.




(a) Magnitude of specific force output by the accelerometers before bias, scale factor and cross-coupling estimation.




(b) Previous plot zoomed in vertically, from where it is possible to verify orientation dependency in the output magnitude, characteristic typically presented in sensors with different bias and scale factors among axis.

Figure 15: Magnitude of the specific force output by the accelerometers in a calibration dataset collected using the robot.

## 11.2 The First Dataset - Static Indoor

### 11.2.1 Vertical and Heading Determination

Chapter 6 presents the importance of compensating gravity in $n$-frame from the specific force sensed by the accelerometers in $b$-frame. A challenge related to it is determining where vertical points to. A plumb-bob method is presented on [8] and is used for this particular dataset. A elastic string is hooked to the ceiling and a block of metal is attached to the other end and immersed in a bowl containing water, without touching its surface. The water in the bowl damps the string, allowing a static vertical scenario to be achieved relatively fast.

From a calibration dataset, the user indicates two locations of the string for a dozen of images taken from different poses. After it, an optimizer determines what is vertical and this axis is adopted for the pattern frame $p$, as described on chapter 2. The heading is indirectly determined from another previously calibrated pattern within 2° of accuracy.

### 11.2.2 Results

Knowing that the DMU-11 does not perform coning and sculling calculations, a static setup was created for this first dataset, by fixing the camera-IMU assembly to an untouched table. The goal is to get coning and sculling related errors out of the equations by ensuring the angular velocity is constant (zero in this case) between two outputs. Then, 15 minutes of data was acquired, the calibration procedures mentioned before were applied, camera data was despiked and the filter received as input all the camera data, except from the data comprised between 550s and 850s. Basically, a virtual gap (i.e. a fake aiding outage) was created in the camera data, so that it would be possible to compare the filter's performance with camera data.

Figure 16a shows roll and pitch angles computed by both camera and EKF, while figure 16b depicts the difference between them.

The errors achieved for roll and pitch are under 0.5° and they seem to be a Brownian motion arising from integrating random noised gyroscopes' outputs, since bias would produce clearer trends, like a ramp. The orientation was, then, injected into the mechanization equations [14] to allow velocity and position estimations. During this process, the average magnitude of the corrected accelerometers were scaled to the magnitude of the gravity computed from the theoretical model, so that a proper mechanization would result in no acceleration. Furthermore, vertical ZUPT was applied. Figure 16b presents the drift in position for both $x$ and $y$ components in meters.

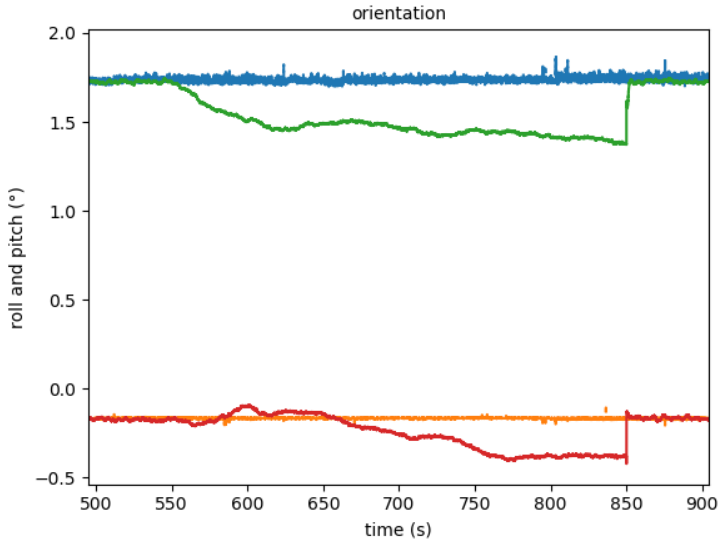
From figure 16b, it is possible to make three important conclusions:

1. The tiniest error in attitude yields to massive errors in position: after 20 seconds of outage, an error of $1.2m$ in the $x-$component was reached.

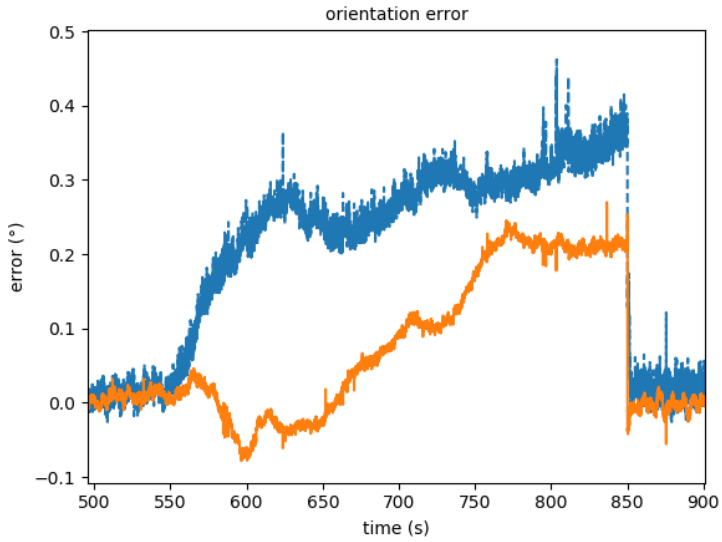
2. An attitude error that oscillates around zero can lead to a correct position estimation: during this particular aiding outage period, the $y-$component error is zero around 680s and 780s.

3. MEMS IMU random noises make unfeasible their use for dead reckoning, even during outage periods of about 10s.

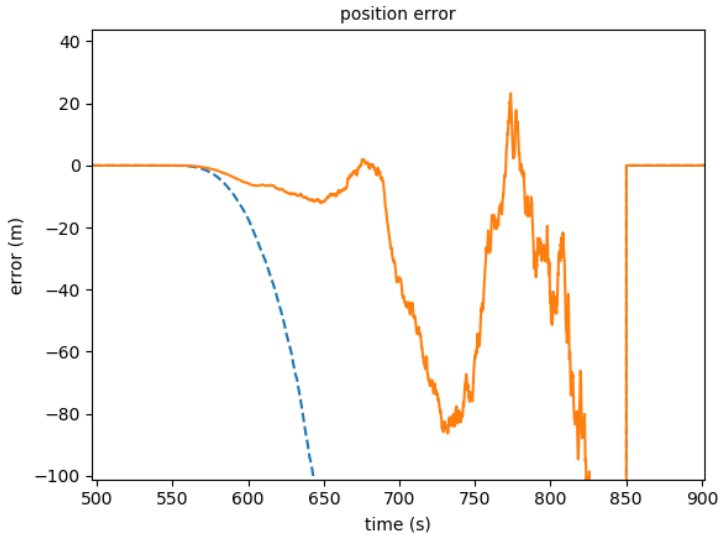
[14] recall diagram on figure 9




(a) Roll (upper lines) and pitch (lower lines) computed by both camera (continuous lines) and EKF (discontinuous lines) in degrees over time.




(b) Error in estimated roll (blue dashed line) and pitch (orange solid line) in degrees over time.




(c) Error in estimated position $x-$component (blue dashed line) and $y-$component (orange solid line) in meters over time.

Figure 16: Results obtained for orientation and position estimation during a 300-seconds period without feeding aiding from $t = 550s$ to $t = 850s$.

These conclusions are consistent with results obtained by [33]. Even so, other tests were conducted in order to:

1. Collect and analyze data from a training plant.
2. Investigate the possibility of using as IMU to bridge between two camera samples.

## 11.3 The Second Dataset - RDM Training Plant

### 11.3.1 Overview

RDM training plant is a site mainly dedicated to be used to prepare employees for going to the field in safety. It contains elements from a real chemical plant, as pipes with valves and flanges without any chemical products running through it and, thus, was chosen for collecting datasets. Figure 17 gives an overview of the plant, while figure 18 depicts the area at where the results will be presented.

Figure 17: Refinery training plant overview.

The datasets were collected using a camera Allied Vision G504-B [29] mounted on an IMU Silicon Sensing DMU-11 [30], both connected to a computer for logging, as well as to a portable energy supply. The assembly resembles the one previously depicted on picture 14 and was hand-held, as it is going to be once it is operational in the refinery. In addition, the results here presented are from a dataset in which a human acted as if he/she was looking for a specific flange. In other words, the human walked with a typical human-like velocity and moved the camera as the refinery's employee is expected to move.

### 11.3.2 Vertical and Heading Determination

As it can be seen from both figures 17 and 18, the plant consists of an outdoor area and, thus, the plumb-bob method described on chapter 11.2.1 could not be used. Instead, the

Figure 18: Refinery training plant flanges closeup.

vertical and heading provided by the SITE-SPOT model were used. From chapter 11.2.2 it is known that the least inaccuracy in tilt orientation is not enough for fast drifts in position estimates. Nevertheless, the results let us conclude that level in SITE-SPOT is accurate within 0.05°.

### 11.3.3 Results

Figure 19 presents the result the employee would see in the smartphone's screen for a frame[15] in which a pattern was detected and served as aiding to position estimation. This figure is obtained after reprojecting the asset's labels into the camera image using the pose estimated by the filter. From the image, it is possible to infer that the camera's auto-exposure software did not manage to reduce the exposure time enough to avoid flare. It is believed that the camera's iris was too open for an outdoor environment and that it might have degraded the performance of pattern detection of QuickVision algorithm, which is a reasonable explanation for the pattern not have been detected on image 21e.

Figure 20 presents the error in $x$ and $y$ components of the estimated positions over a 5-seconds period (for $156.7s < t < 162.5s$) during which no patterns were either in view or detected. Figure 19 is also the last camera image from which a pattern was detected before the sequence of figures presented on 21, that displays the results the user would see at the smartphone's screen over this aiding outage period cropped around the flange 013 for the convenience of the reader.

From the images, it can be seen that after $4s$ of outage, it is no longer possible to determine with certainty flange's 013 location.

[15]Please note that in this context, *frame* is equivalent to *photo*, rather than to *coordinate frame*.

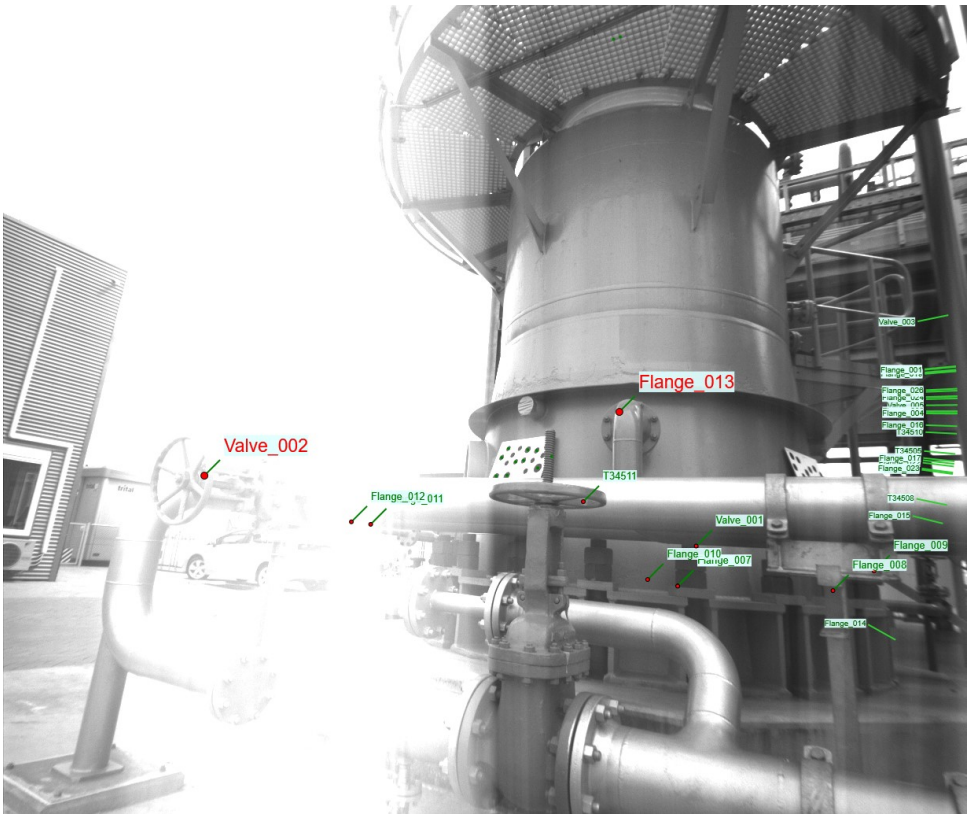



Figure 19: Last pose containing camera aiding ($t = 156.6s$).

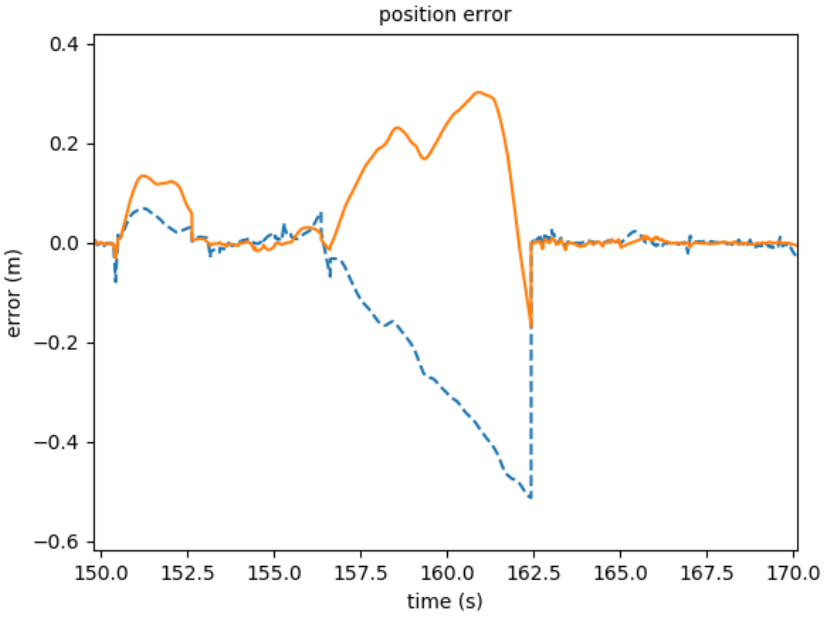
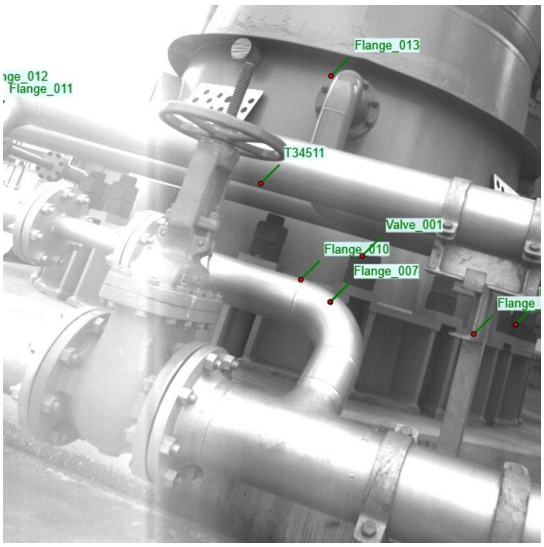
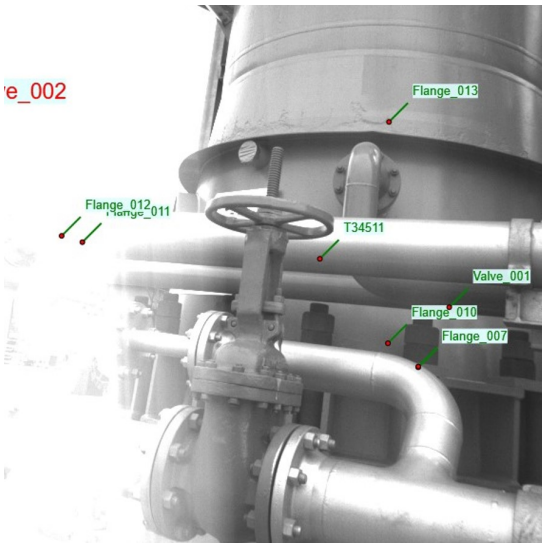


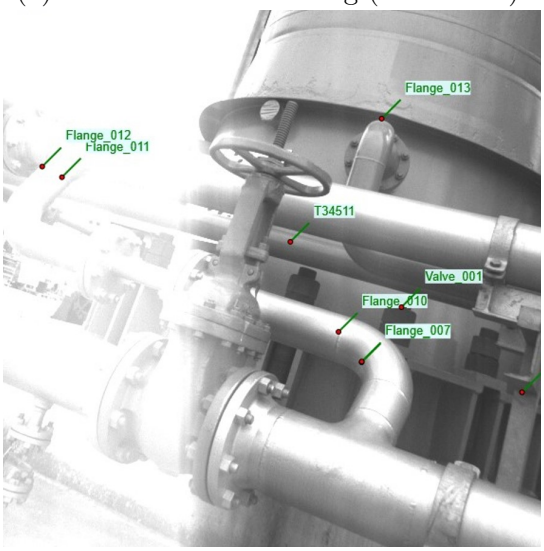

Figure 20: Error in position $x-$component (blue dashed line) and $y-$component (orange solid line) estimation in meters over time for the second dataset.



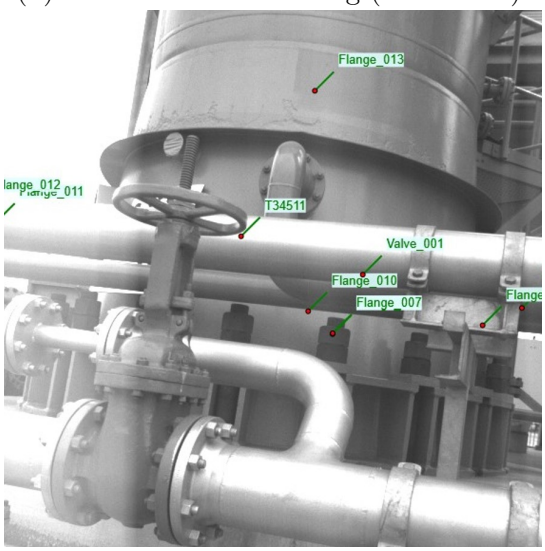

(a) After 1s without aiding ($t = 157.6s$).



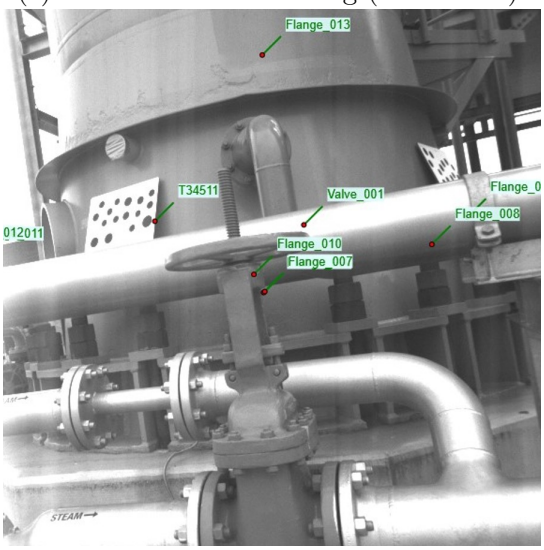

(b) After 2s without aiding ($t = 158.6s$).



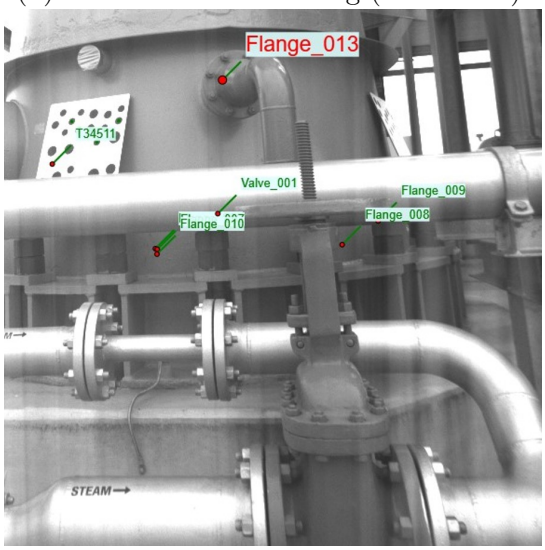

(c) After 3s without aiding ($t = 159.6s$).




(d) After 4s without aiding ($t = 160.6s$).




(e) After 5s without aiding ($t = 161.6s$).




(f) Aiding reestablished ($t = 162.6s$).

Figure 21: Sequence of estimated poses during a 5-seconds period without camera aiding followed by aiding reestablishment.

## 11.4 The Third Dataset - Outdoor

### 11.4.1 Overview and Setup

This dataset was taken to investigate the repeatability of the previous results under an *ideal controlled environment*. In other words, additional equipment and sensors were used in order to improve accuracy in time-stamping and vertical and heading determination. The equipment used for this dataset consists of:

- Camera Allied Vision G504-B, as before
- MEMS IMU Silicon Sensing DMU-11, as before
- FOG IMU iXblue Rovins, allowing to determine heading and level in another way compared to previous datasets
- GNSS Antenna Fugro AD-492, allowing to receive Pulse-per-second (PPS) signal from satellites for better time-stamping accuracy
- GNSS-Receiver Platform Fugro StarPack, allowing to interpret PPS signal from the antenna
- Synchronization Platform Fugro StarPort, allowing to synchronously time-stamp data from sensors
- Computer equipped with StarFix 2016, allowing to log data from StarPort
- Trolley with soft wheels, allowing to move the equipment

Figure 22 shows the assembly of the equipment, while figure 23 zooms in around the camera area. Note that vertical is determined by the FOG IMU and that the antenna is on top to improve GNSS signal. Also, camera, Rovins and DMU-11 are mounted to the same rigid body. Mounting this equipment on a trolley may not produce the same kind of movement a human would perform in the real scenario, but it was the only way found to include Rovins, GNSS-Antenna, StarPack and StarPort in the setup.

Pursuing *perfect* data, 4 crates were disposed in a square shape, with a container in its center. A size A2 pattern was placed against each of the vertical faces. This setup allows strong links between patterns, since patterns at opposite locations can be both seen in a single camera image. Figure 24 illustrates the arrangement. In addition, it is worth remarking that the FOG was turned on one hour before data was collected in order to initialize itself.

In one hand, from the datasets presented before, it was observed that drifts bigger than $0.25m$ in either $x$ or $y$ components of the estimated positions can result in misleading augmented reality reprojections on the user's smartphone. In the other hand, pattern detection is an expensive computational task and may not be executed at a high rate. This dataset investigate the filter's performance under a low aiding frequency of $0.3Hz$, compared to $15Hz$ from the previous ones. Note that camera images were logged at $1.8Hz$ and the filter was fed with only one every six frames. In this way, the ground truth is known every $0.555s$, instead of every $3.33s$.

### 11.4.2 Results

The collected dataset consists of an initial 500 seconds static period followed by 480 seconds of two and a quarter circular turns around the crates, as depicted on figure 25.

The results are presented in figure 26. Figure 26a displays the error between the ground truth (i.e. the camera) and the filter estimates, while figure 26b depicts the elapsed time between a camera sample and the previous one fed to the filter. From the image, it is clear that the camera time step is variable, what is expected since it depends

Figure 22: Equipment used for the third dataset. On top, the antenna. At bottom-left, the Rovins. Between the computer and the Rovins, the Camera mounted to the DMU-11. Under the computer, the StarPort and StarPack.

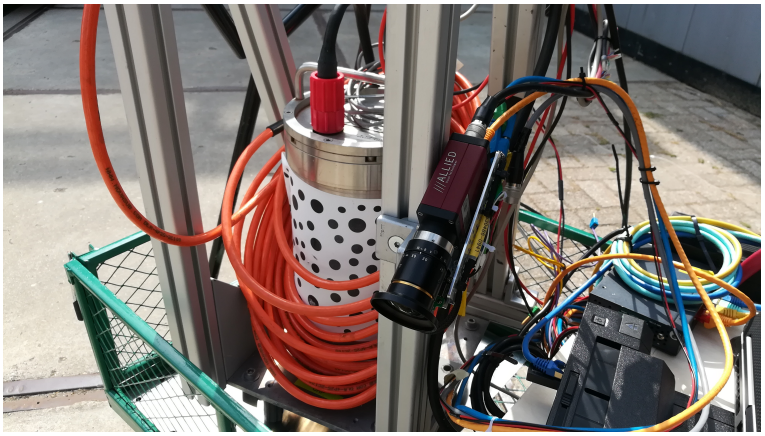



Figure 23: Equipment used for the third dataset, zoomed in around the camera area.

on the exposure time, that varies according to light conditions.

Analyzing figure 26 as a whole, it is possible to infer that, even though the estimated $y$-component was *trying to fly away*, the accuracy requirement for position was respected from $t = 680s$ to $t = 731.6s$, thanks to camera aiding. However, the camera sample expected to aid around $t = 735s$ did not exist and the errors of the estimates exceeded $0.25m$. Several different factors could explain the absence of a sample:

- The User Datagram Protocol (UDP) used for the communication between the sensors and the computer does not ensure data transmission.

Figure 24: Arrangement of patterns for third dataset, allowing a circular path of about 8 meters of diameter.

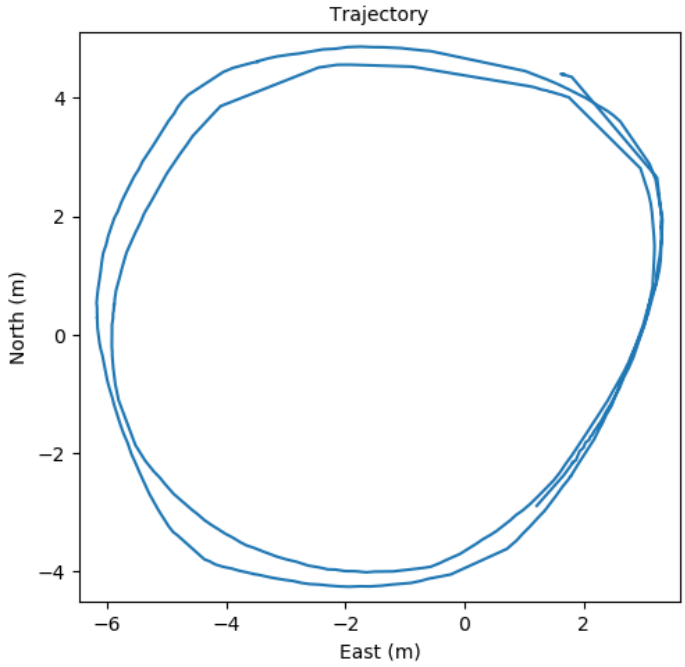
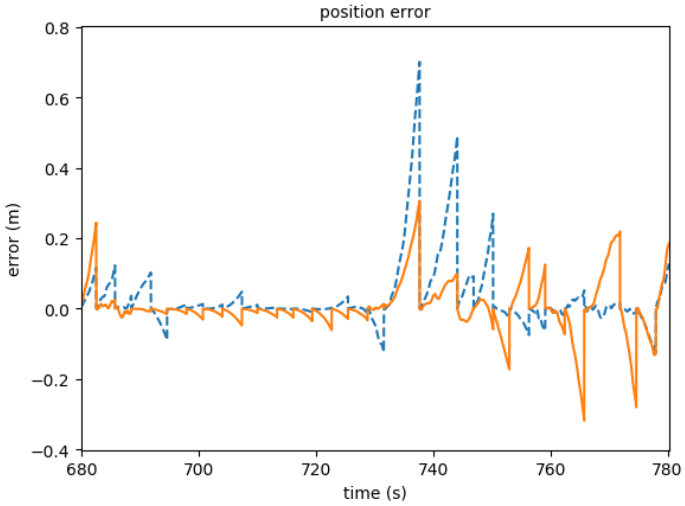


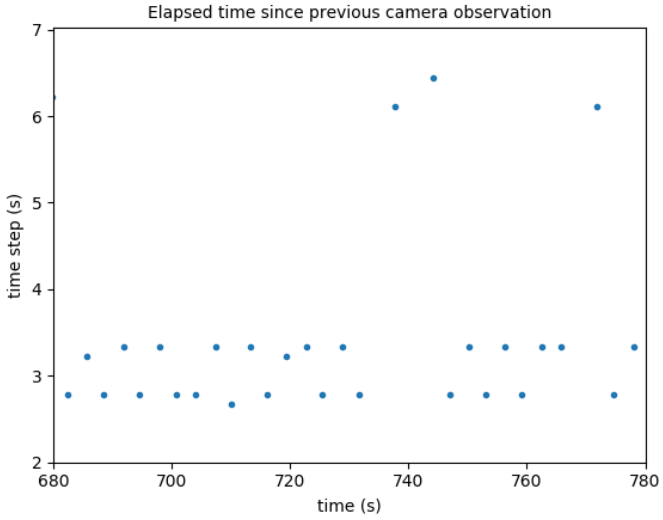

Figure 25: Trajectory traversed during third dataset.

- Overexposure might have occurred, as happened with the previous dataset.
- The sample might have been considered an outlier by the data rejection algorithm.
- The QuickVision algorithm might have failed to identify patterns from the image.

Still from figure 26, it is possible to notice that the error after the outage period is bigger than the error before the outage. From chapter 8 it is recalled that the filter needs time to converge, especially in the case the covariance matrices $\mathbf{Q}$ and $\mathbf{R}$ are not optimally tuned or the system is non-linear.




(a) Error in estimated position $x$-component (blue dashed line) and $y$-component (orange solid line) in meters over time.




(b) Time step between two consecutive camera observations.

Figure 26: Results obtained for the dataset using a FOG to determine vertical and heading.

# 12 Conclusions and Recommendations

## 12.1 Conclusions

The main objective of this dissertation is to investigate the performance of position estimation by merging data from a camera and a MEMS IMU with an Extended Kalman Filter algorithm. Visual aiding consists of the IMU's position in pattern frame $p$ and is provided by QuickVision software for every image a previously loaded pattern is detected.

The background knowledge required to accomplish this task is developed throughout the early chapters, as well as the simplifying assumptions adopted on this work.

Chapter 10 summarizes the theory previously presented and details other aspects of the implementation, as ZUPT aiding, the integration architecture and the material used for the datasets. Then, three different tests are presented in chapter 11, from which the following conclusions are inferred:

- Biases and scale factors cannot be distinguished for static data Therefore, the estimation vector is only fully observable for dynamic datasets.
- From a static dataset, in which the simplifying assumptions meet the non-simplified mathematical model, roll and pitch were observed to random walk around the ground truth due to random noise integration, reaching errors up to $0.5^\circ$ for roll and pitch after a 300-seconds period of visual aiding outage.
- Such errors in the orientation estimation caused the position estimation to drift $1.2m$ within the first $20s$ of outage for the same static data, which is not sufficient for the augmented reality application.
- Implementing vertical ZUPT allowed to stabilize the position vertical component, regardless discrepancies between the gravity model and the specific force output by the IMU. Minor improvements were also observed in the remaining components.
- From the dataset collected at a training plant where the position of some assets were known, it was possible to compute the augmented reality overlay by reprojecting assets into the camera images. Drifts around $30cm$ for one of the components can lead to an asset misdetection.
- Position estimation stays within the requirement boundaries even if only one visual aiding sample is available every 3 seconds. However, after a 6-seconds gap the position estimation might be jittered for about half a minute and not reliable within the desired accuracy.

## 12.2 Recommendations for Further Research

- For a real-time application running in a mobile phone, time-stamping might be an issue to be investigated. Initially, this work supposed there was no relative time drift between the camera and DMU-11, which proved to be a naive assumption after the calibration presented in chapter 11.1 was performed. In addition, an investigation on time-stamping accomplished by the mobile operating system adopted may be necessary, given that Windows clock can reach up to 3ms of error compared to GNSS-time, and that an IMU typically outputs every 5ms.

- Coning and sculling computations need to be carefully investigated. The MEMS IMU used in this work is of a higher grade than IMUs found on mobile phones and even so it does not compute coning and sculling internally. From a static dataset, it seems that coning is not the single source of error during orientation estimation, though.

- This thesis made a couple of simplifications reportedly said to degrade the filter's performance, under the assumption that the velocities are always low (especially compared to cars and airplanes), like that Coriolis and transport rate were negligible. Also, during development phase, the ODEs' first and second orders numerical approximations were briefly compared and the difference in results was insignificant, appearing to be hidden by other major sources of error.

- During the development of the present work, determination of the local gravity vector appeared to be one of the major sources of error in position estimation. [34] affirms to be able to observe it in the presence of a known calibration target.

- Performance limitation under non-linearities of Extended Kalman Filter (EKF) over Unscent Kalman Filter (UKF) is not a consensus among authors. While [6] and [10] cover EKF only, [1] covers both of them and states the latter achieves better results. Yet another approach is developed by integrating Extended Kalman Filter with neural networks [35].

- Allan Variance and Power Spectral Density provide statistics at sensor level only (rather than at process level). In addition, these techniques are not suitable for low grade IMUs [1]. Hence, these methods may lead to non-optimal covariance matrices and fine tuning might be performed. An automated algorithm is proposed by [36].

- Given all MEMS IMU limitations explained throughout the thesis, the orientation estimation achieved from them proved not to be enough for position estimation. An alternative to be studied is creating a camera-based aiding relying on image feature detection and tracking.

- Even though QuickVision software is able to determine the full pose of the camera (position and attitude), the smallest inaccuracy in the camera-IMU mounting angles calibration can create a systematic error in the orientation from the visual aiding. Therefore, only the position serves as aiding, rather then the full pose. Augmenting the estimation vector with the mounting angles between the camera and the IMU could be implemented to investigate the impact on position estimation.

# Bibliography


[1] Eun-Hwan Shin. Estimation techniques for low-cost inertial navigation. *UCGE report*, 20219, 2005.

[2] Eun-Hwan Shin. *Accuarcy improvement of low cost INS/GPS for land applications.* University of Calgary, 2001.

[3] Niklas Hjortsmarker. *Experimental system for validating GPS/INS integration algorithms.* 2005.

[4] Warren Flenniken. *Modeling inertial measurement units and anlyzing the effect of their errors in navigation applications.* PhD thesis, 2005.

[5] Khairi Abdulrahim, Terry Moore, Christopher Hide, and Chris Hill. Understanding the performance of zero velocity updates in mems-based pedestrian navigation. *International Journal of Advancements in Technology*, 5(2):53–60, 2014.

[6] P.D. Groves. *Principles of GNSS, Inertial, and Multisensor Integrated Navigation Systems.* GNSS technology and applications series. Artech House, 2008.

[7] D. Titterton, J.L. Weston, J. Weston, Institution of Electrical Engineers, American Institute of Aeronautics, and Astronautics. *Strapdown Inertial Navigation Technology.* Electromagnetics and Radar Series. Institution of Engineering and Technology, 2004.

[8] P.G. Savage. *Strapdown Analytics.* Number v. 1 in Strapdown analytics. Strapdown Associates, 2000.

[9] J.A. Farrell. *The Global Positioning System & Inertial Navigation.* McGraw-Hill Education, 1999.

[10] Jay Farrell. *Aided navigation: GPS with high rate sensors.* McGraw-Hill New York, 2008.

[11] C. Jekeli. *Inertial navigation systems with geodetic applications.* Walter de Gruyter, 2000.

[12] Manon Kok, Jeroen D Hol, and Thomas B Schön. Using inertial sensors for position and orientation estimation. *arXiv preprint arXiv:1704.06053*, 2017.

[13] James Diebel. Representing attitude: Euler angles, unit quaternions, and rotation vectors. *Matrix*, 58(15-16), 2006.

[14] G. Strang. *Linear Algebra and Its Applications.* Harcourt, Brace, Jovanovich, Publishers, 1988.

[15] J.W. Brown and R.V. Churchill. *Complex variables and applications*. Number v. 1 in Churchill-Brown series. McGraw-Hill, 1996.

[16] Jack B Kuipers et al. *Quaternions and rotation sequences*, volume 66. Princeton university press Princeton, 1999.

[17] Joan Sola. Quaternion kinematics for the error-state kalman filter. 2017.

[18] K. Ogata. *Modern Control Engineering*. Instrumentation and controls series. Prentice Hall, 2010.

[19] Sebastian Thrun. Probabilistic robotics. *Communications of the ACM*, 45(3), 2002.

[20] Afonso Celso del Nero Gomes. Sinais e sistemas - notas de aula.

[21] Chris Paige. Properties of numerical algorithms related to computing controllability. *IEEE Transactions on Automatic Control*, 26(1):130–138, 1981.

[22] Alex G Quinchia, Gianluca Falco, Emanuela Falletti, Fabio Dovis, and Carles Ferrer. A comparison between different error modeling of mems applied to gps/ins integrated systems. *Sensors*, 13(8):9549–9588, 2013.

[23] Haiying Hou. *Modeling inertial sensors errors using Allan variance*. University of Calgary, Department of Geomatics Engineering, 2004.

[24] B.L. Stevens, F.L. Lewis, and E.N. Johnson. *Aircraft Control and Simulation: Dynamics, Controls Design, and Autonomous Systems*. Wiley, 2015.

[25] RR Labbe. Kalman and bayesian filters in python, 2014.

[26] M.S. Grewal and A.P. Andrews. *Kalman filtering: theory and practice using MATLAB*. [Wiley InterScience Online Books, Electronic and Electrical Engineering Collection]. Wiley, 2001.

[27] Joan Sola. Simulataneous localization and mapping with the extended kalman filter. *Avery quick guide with MATLAB code*, 2013.

[28] Dorota A Grejner-Brzezinska, Yudan Yi, and Charles K Toth. Bridging gps gaps in urban canyons: The benefits of zupts. *Navigation*, 48(4):216–226, 2001.

[29] *Manta G-504B Technical Datasheet*.

[30] *DMU11 Technical Datasheet*.

[31] *DMU11 Evaluation Kit User Manual*.

[32] *Product Manual IRB 120*.

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

[34] J Kelly. On the observability and self-calibration of visual-inertial navigation systems. *University of Southern California, Los Angeles, USA, Tech. Rep. CRES-08-005*, 2008.

[35] Christopher L Goodall. Improving usability of low-cost ins/gps navigation systems using intelligent techniques. *Thesis*, 2009.

[36] Wojciech Straszewski, Magdalena Drozdz, and Hendrik Wouters. Automated tuning of kalman filter: Kalman filter tuning in the windows azure cloud environment. In *Inertial Sensors and Systems (INERTIAL), 2018 IEEE International Symposium on*, pages 1–4. IEEE, 2018.

# Appendices

# Appendix A

# Other Notations

$$\vec{q} = q = \begin{pmatrix} a & b & c & d \end{pmatrix}^T = \begin{pmatrix} -a & -b & -c & -d \end{pmatrix}^T \tag{A.1}$$

$$\boldsymbol{C} = C(q) = \begin{bmatrix} a^2+b^2-c^2-d^2 & 2bc-2ad & 2bd+2ac \\ 2bc+2ad & a^2-b^2+c^2-d^2 & 2cd-2ab \\ 2bd-2ac & 2cd+2ab & a^2-b^2-c^2+d^2 \end{bmatrix} \tag{A.2}$$

$$q_D^E = q_E^{D*} = \begin{pmatrix} a & b & c & d \end{pmatrix}^T, \text{ where } \quad q_D^{E*} = q_E^D = \begin{pmatrix} a & -b & -c & -d \end{pmatrix}^T \tag{A.3}$$

$$\boldsymbol{Q} = Q(q) = \begin{bmatrix} a & -b & -c & -d \\ b & a & d & -c \\ c & -d & a & b \\ d & c & -b & a \end{bmatrix} \tag{A.4}$$

$$\bar{\boldsymbol{Q}} = \bar{Q}(q) = \begin{bmatrix} a & -b & -c & -d \\ b & a & -d & c \\ c & d & a & -b \\ d & -c & b & a \end{bmatrix} \tag{A.5}$$

$$q_a \cdot q_b = Q(q_a)q_b = \bar{Q}(q_b)q_a \tag{A.6}$$

$$\frac{d\boldsymbol{C}}{dq} = \begin{bmatrix} \frac{d\boldsymbol{C}}{da} & \frac{d\boldsymbol{C}}{db} & \frac{d\boldsymbol{C}}{dc} & \frac{d\boldsymbol{C}}{dd} \end{bmatrix} \tag{A.7}$$

$$\frac{d\boldsymbol{C}}{da} = 2\begin{bmatrix} a & -d & c \\ d & a & -b \\ -c & b & a \end{bmatrix} \tag{A.8}$$

$$\frac{d\boldsymbol{C}}{db} = 2\begin{bmatrix} b & c & d \\ c & -b & -a \\ d & a & -b \end{bmatrix} \tag{A.9}$$

$$\frac{d\boldsymbol{C}}{dc} = 2\begin{bmatrix} -c & b & a \\ b & c & d \\ -a & d & -c \end{bmatrix} \tag{A.10}$$

$$\frac{d\boldsymbol{C}}{dd} = 2\begin{bmatrix} -d & -a & b \\ a & -d & c \\ b & c & d \end{bmatrix} \tag{A.11}$$

$$\frac{d}{dq}\left(\boldsymbol{C}\vec{u}_{x,y,z}\right) = \frac{d\boldsymbol{C}}{dq}\begin{bmatrix} \vec{u}_{x,y,z} & \mathbf{0}_{3\times1} & \mathbf{0}_{3\times1} & \mathbf{0}_{3\times1} \\ \mathbf{0}_{3\times1} & \vec{u}_{x,y,z} & \mathbf{0}_{3\times1} & \mathbf{0}_{3\times1} \\ \mathbf{0}_{3\times1} & \mathbf{0}_{3\times1} & \vec{u}_{x,y,z} & \mathbf{0}_{3\times1} \\ \mathbf{0}_{3\times1} & \mathbf{0}_{3\times1} & \mathbf{0}_{3\times1} & \vec{u}_{x,y,z} \end{bmatrix} \tag{A.12}$$

$$\frac{d}{dq_a}\left(Q(q_a)q_b\right) = \bar{Q}(q_b) \tag{A.13}$$

$$q\left(\vec{u}_{x,y,z}\right) \triangleq \begin{pmatrix} 0 & u_x & u_y & u_z \end{pmatrix}^T \tag{A.14}$$