TÉCNICO LISBOA

Visual-Inertial Odometry using Square-Root UKF on Lie groups to estimate orientation of monocular camera

Gonçalo Dinis Ramos Costa Pereira

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisor(s): Prof./Dr. Alexandre José Malheiro Bernardino
Prof./Dr. José Alberto Rosado dos Santos Vítor

Examination Committee

Chairperson: Prof. Full Name
Supervisor: Prof. Full Name 1 (or 2)
Member of the Committee: Prof. Full Name 3

November 2021

Declaration

Acknowledgments

Resumo

O sistema oculomotor é responsável por orientar e controlar os movimentos oculares em humanos mas também noutros animais. Sacadas são movimentos extramamente rápidos e precisos e os 6 músculos extra-oculares são capazes te direcionar a linha de visão com apenas 2 graus de liberdade. Isto torna a modelação e design the um olho robótico inspirado inspirado na biologia humana, um problema bastante desafiante de se resolver. Um dos objectivos do projecto ORIENT é o desenvolvimento de um protótipo de um olho robótico que seja capaz de imitar o sistema humano. Estimar a orientação 3D do protótipo mecânico, é por isso, crucial para que seja possível implementar estratégias de controlo que possam ajudar a melhorar ainda mais o design actual. Em trabalhos anteriores, esta estimação foi feita com base num sensor inercial (IMU) e numa câmera separadamente. Contudo, só é viável usar a câmera para estimar orientações a baixa frequência devido ao desfoque da imagem, e o IMU tem uma relação sinal-ruído muito baixa para movimentos lentos. O foco principal deste trabalho é desenvolver um algoritmo que funde a informação destes dois sensores para estimar a orientação da câmera, utilizando uma abordagem bastante comum em odometria visual e inercial. Usamos a recentemente introduzida metodologia do Unscented Kalman Filter em grupos de Lie e comparamos com a ground-truth. Avaliamos a performance do filtro em ambiente de simulação e também com dados reais, adquiridos com o braço robótico, Kinova Gen3. Por último, é feita uma discussão de como esta abordagem pode ser usada para resolver este problema e sugerimos possíveis melhorias para trabalho futuro.

Palavras-chave: Sacadas, estimação de orientação, Odometria visual e inercial, Unscented Kalman Filter

Abstract

The oculomotor system is responsible for directing and controlling eye movements in humans but also in other animals. Eye saccades are extremely fast and precise movements and the 6 extra-ocular muscles are able to direct the eye requiring only two degrees of freedom. This makes modelling and design of a bio-inspired robotic eye a very challenging problem to solve. One of the goals of the ORIENT project is to develop a prototype robotic eye that can mimic the human system. Estimating the 3D orientation of the mechanical prototype is crucial to develop control strategies that could help us further improve the design. In previous works, this estimation was done using an IMU and a camera separately. However, the camera can only be used for low frequency estimation due to motion blur, and the IMU suffers from poor signal-noise ratio in slow movements. The main focus of this work is to develop an algorithm that fuses the information of these two sensors to estimate the orientation of the camera, using an approach very common in Visual-Inertial Odometry (VIO). We use the recently introduced Unscented Kalman Filter on Lie Groups methodology and compare to the ground-truth. We assess the performance of the filter in simulation and with a real world dataset, obtained using the Kinova Gen3 robotic arm. Finally, we discuss how this approach can be used to solve this problem and suggest possible improvements for future work.

Keywords: Eye saccades, orientation estimation, Visual-Inertial Odometry, Unscented Kalman Filter

Contents

List of Tables

List of Figures

Nomenclature

Groups, algebras and sets

$\mathbb{R}$	Set of real numbers.
$\mathbb{R}^N$	Real vector space of dimension N.
$SO(3)$	Special Orthogonal Group of dimension 3.
$\mathfrak{so}(3)$	Lie Algebra of $SO(3)$.
$SE(3)$	Special Euclidean Group of dimension 3.
$\mathfrak{se}(3)$	Lie Algebra of $SE(3)$.
$SE_{2+p}(3)$	Special Euclidean group of dimension 3, but in a different topology (with p landmarks).
$\mathfrak{se}_{2+p}(3)$	Lie Algebra of $SE_{2+p}(3)$.
$\mathcal{G}$	General Lie group.
$T_{\mathbf{x}}\mathcal{G}$	Tangent space of Lie Group $\mathcal{G}$ at $\mathbf{x} \in \mathcal{G}$.
$\mathcal{S}$	Lie group of the state. $\mathcal{S} := SE_{2+p}(3) \times \mathbb{R}^6$

Greek symbols

α	Scaling parameter for sigma point generation.
β	Scaling parameter for sigma point generation.
$\boldsymbol{\chi}$	General representation of the state in $\mathcal{S}$.
$\boldsymbol{\mu}$	General mean vector.
$\boldsymbol{\Sigma}$	General covariance matrix.
$\boldsymbol{\xi}$	General representation of uncertainty vector.
Δt	Time interval between two instants.
η	Normalization constant of probability density function.
γ	Scaling parameter for sigma point generation.

λ	Scaling parameter for sigma point generation.
σ	General standard deviation.
${}^{I}\boldsymbol{\omega}_{W,I}$	Angular velocity of I with respect to W, written in I.

Roman symbols

${}^{W}\mathbf{R}_{I}$	Rotation matrix from frame I to W.
$\mathbf{I}_N$	Identity matrix of size $N \times N$.
$\hat{\mathbf{x}}_k$	Mean of the state $x \in \mathcal{S}$ at time instant k.
$\hat{\boldsymbol{\chi}}_k$	Mean of the state component $\boldsymbol{\chi} \in SE_{2+p}(3)$ at time instant k.
$\hat{\mathbf{b}}_k$	Mean of the state component $b \in \mathbb{R}^6$ at time instant k.

Subscripts and Superscripts

i, j, k	Computational indexes.
n	Normal component.
W	World coordinate frame.
x, y, z	Cartesian components.
*	Adjoint.
ref	Reference condition.
T	Transpose.

Acronyms

API Application Programming Interface.

BFGS Broyden-Fletcher-Goldfarb-Shanno Gradient Projection algorithm.

DOF Degrees of Freedom.

EKF Extended Kalman Filter.

IMU Inertial Measurement Unit.

KLT Kanade–Lucas–Tomasi feature tracker.

MAP Maximum A Posteriori.

PDF Probability Density Function.

PF Particle Filter.

RMSE Root Mean Squared Error.

ROS Robot Operating System.

SLAM Simultaneous Localization and Mapping.

SLERP Spherical Linear Interpolation.

SR-UKF Square-Root Unscented Kalman Filter.

UKF Unscented Kalman Filter.

VIO Visual Inertial Odometry.

Chapter 1

Introduction

1.1 Context

The brain is, without a doubt, one of the most fundamental and complex organs not just in humans, but in other forms of life. There is still so much that we don't understand about the way it functions. In the case of the human brain, we use it everyday to perform seemingly simple tasks such as localising objects, identifying them and picking them up. We might not consciously think about these actions when we perform them, however, if we analyse them deeper, we realize that they are not so simple as we thought they were.

Our visual senses, namely our eyes, play a critical role when we wish to find an object and pick it up. Focusing solely on the eye movements, even from pure naive observation, it's noticeable that the brain is able to control the eyes so that they do incredibly fast and accurate movements (saccades). So we are forced to ask one of many questions: How is the brain able to control our eyes to reach the desired goal?

The human eye has 6 extra-ocular muscles, 3 agonist-antagonist pairs (Figure 1.1) and these muscles provide 3 degrees of freedom (DOF) for rotation. Yet, to point the eye in any given direction (gaze direction), only two degrees of freedom are required.

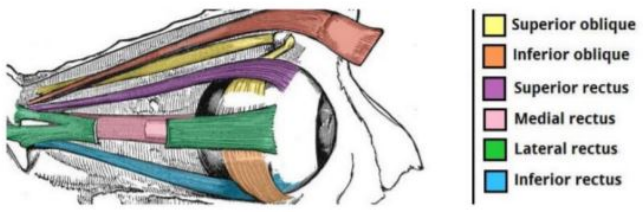


Figure 1.1: Side view of the 6 extraocular muscles. Taken from [1].

This poses a very challenging and interesting problem for neuroscientists but also in the robotics field because not only is the oculomotor system non-linear, but also, trying to develop a biommimetic model of the human eye with extra degrees of freedom brings additional constraints to the mathematical formulation of the problem [2]. So far, there is no robot that is able to reproduce saccades using the principles of the human system, with the actual complexity that comes from this problem. That's why there's currently research being held in Lisbon in the context of the ORIENT project in order to design and test a humanoid eye-head robotic system that follows the same principles as human psycho-physics [1].

There's already been quite alot of previous work done in this project, namely in the development of a (recent) biommimetic eye prototype (Figure 1.2) that consists of six independent motors controlling the 6 cables that approximate the extra-ocular muscle geometry of the human eye. The team has also developed models for these prototypes [2, 3] and subsequent control techniques [4, 5] as well as work in the computer vision area [6].

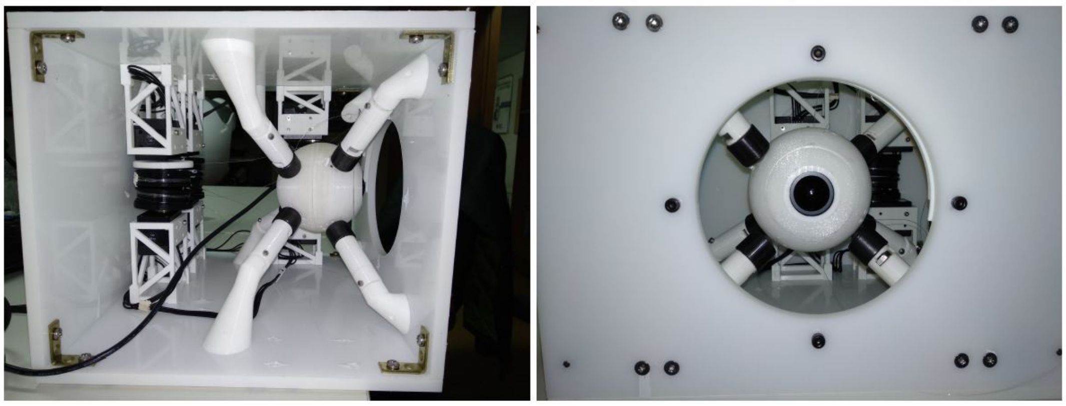


Figure 1.2: Current prototype of the human eye.

1.2 Motivation

The eye prototype uses a standard camera (similar to the uEye camera in B.1) and an Inertial Measurement Unit (IMU) to estimate the eye's orientation. In previous works [6], the orientation was estimated using these two sensors separately and from here the following conclusions were drawn:

- The camera can be used for low frequency motion estimation (10-20 Hz) but if the movements are too fast, there's motion blur problems. Eye saccades have peak velocities up to 700 deg/s [2] and so, only when the eye slows down are we able to capture images with decent quality.

- The IMU is faster (100-200 Hz) but has the problem of drifting over time due to accumulation of integration errors and so can only be used for short periods of time when estimating poses.

These two types sensors complement each other and have long been used to solve various problems in the field of robotics and navigation. To be more specific, the IMU and camera are frequently used to estimate the pose (orientation and pose) of an object, such as an aircraft [7]. This is the so called visual-inertial odometry (VIO) problem and its applications also include 3D reconstruction and augmented

[1]http://www.mbfys.ru.nl/~johnvo/OrientWeb/orient_1.html

reality. By taking advantage of the complementary nature of the camera and IMU, it's possible that we could get a more accurate and solid estimate of the orientation of the eye system.

1.3 Problem definition

Our eye system represented in Figure 1.2 is equipped with similar sensors to the ones used in this work, and to design a robotic eye model as close to human physiology as possible, we need to have high accuracy for the estimate of the camera's orientation during fast movements. The inertial sensor is able to give high rate measurements that enable the reconstruction of the trajectory transient (as we'll see later, eye saccades have step-like movements), but have a lot of motion drift. The camera is able to help during slow motions since the 2D images provide plenty of useful information. However, they have limited output rate, suffer from motion blur and aren't reliable in low texture scenes.

Hence, the challenge here is to identify the best mathematical framework/algorithm that can extract the best out of these two sensors to accurately estimate the orientation. The inertial sensor is rigidly attached to the camera as can be seen in Figure 1.3. In the context of eye saccades, the IMU will be used to track the trajectory of the movement and the camera will perform the correction of the trajectory with the use of images at beginning and end of each saccade.

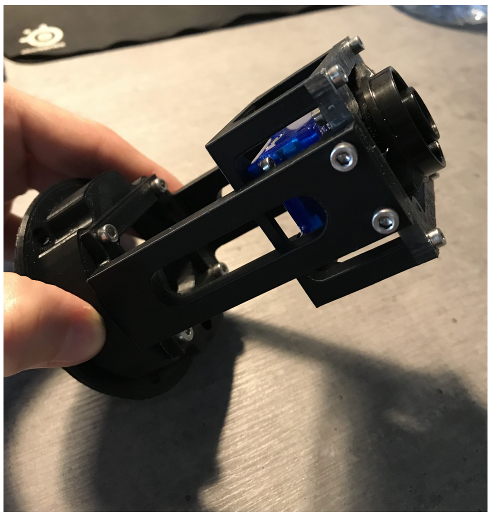


Figure 1.3: Eye system with camera and inertial sensor. This setup is just a replica of the actual eye prototype and was used for benchmarking purposes as will be shown later in section 4.3.1.

1.4 Objectives

As briefly mentioned in the previous section, the main objective of this thesis project is to develop an algorithm that estimates the orientation of camera in the specific context of saccadic eye movements. Human saccades are extremely fast and precise and therefore there's the concern of minimizing error during estimation. To tackle this, the approach will involve fusing information of both camera and inertial sensors. With this in mind, the following steps aim to help solve this problem:

- Analyse and study the different approaches/algorithms that have been used to perform sensor fusion;
- Perform real world benchmarking that allows the comparison of different algorithms.
- Test the algorithm with simulated data and real data. Assess the performance in terms of accuracy and error.

1.5 Thesis Outline

The thesis is structured as follows:

- In Chapter 2, the fundamentals necessary to understand the work such as mathematical foundations and notation are given.
- In Chapter 3 the detailed explanation of the chosen methods is provided. It includes the system kinematic model for our visual-inertial setup, the equations that describe evolution of the system that are fundamental for the VIO approach, and finally a description of the algorithm to be implemented.
- Chapter 4 describes the implementation procedure for the proposed approach. An overview of the architecture is given, as well as the description of how the system was implemented in simulation and in real world scenario.
- Chapter 5 presents the experiments performed both in simulation and reality and the gathered results. A discussion is made about the performance and accuracy of the implemented algorithms.
- Finally, Chapter 6 reviews the work as whole and provides an analysis of the contributions and limitations of the method. It's also provided some indications for future work.

1.6 State of the art

The use of inertial sensors in combination with cameras has been largely studied within the robotics community, mainly for the estimation of poses (position and orientation) of robots, aircrafts and another sensing platforms. These two types of sensors complement each other, are cheap, lightweight and easy to find anywhere. This process of estimating the state (position, velocity and orientation) of an agent using one or more cameras and IMUs is called Visual-Inertial Odometry (VIO) and is illustrated in Figure 1.5.

When it comes to existing literature, there are a lot of different ways to fuse the information from visual and inertial sensors. According to [7], the approaches to solve this problem can be categorized into *loosely-coupled* and *tightly-coupled* sensor fusion [8]. In the loosely-coupled framework, the visual and inertial measurements are processed separately, giving two independent motion estimates that are fused in the end (e.g. [9, 10]) In opposition, the tightly-coupled fusion computes the final estimate using the raw gyroscope, accelerometer and camera measurements (2D features) directly (e.g. [11, 12]). A conceptual diagram that compares these two methods is shown in Figure 1.4.

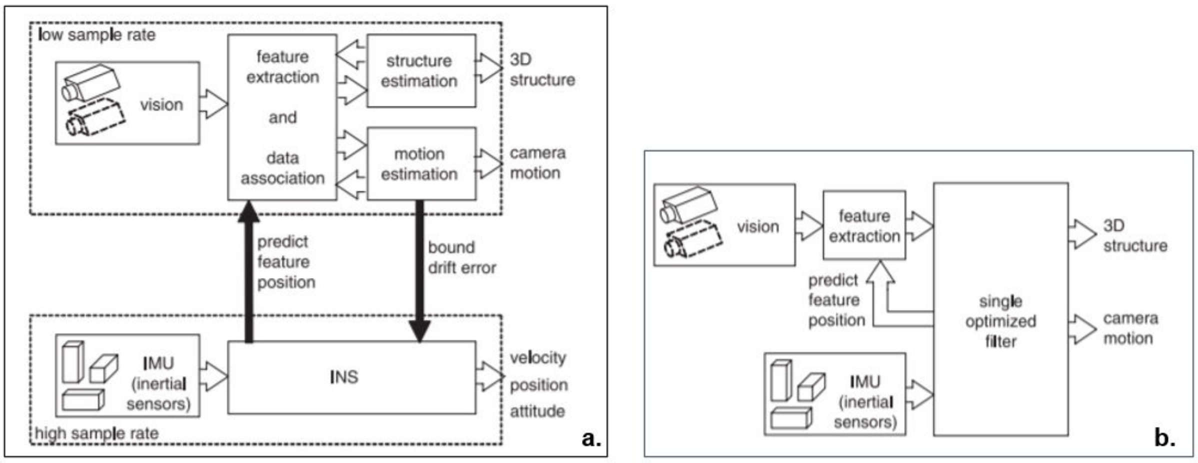


Figure 1.4: Comparison between loosely-coupled (a.) and tightly-couple sensor fusion(b.) [8].

Apart from these approaches, VIO methods can also be categorized in terms of the type of algorithm. There's three that stand-out and those are filtering, fixed-lag smoothing and full-smoothing [7]. Filtering algorithms only allow estimation the latest state of the system. Classic approaches include the Extented Kalman Filter (EKF) or the Unscented Kalman Filter (UKF) which use the covariance matrix to represent uncertainty. Fixed-lag smoothers estimate states included within a certain time window and rule out older states. This approach is usually more accurate than filtering since they are robust to outliers (by the use of robust cost functions for example). However, fixed-lag smoothers are similar to filters when it comes to inconsistency and linearization errors. Lastly, full smoothing approaches, estimate the entire history of states by solving a large non-linear optimization problem. This guarantees high accuracy since it can update based on the complete history, but has the drawback of being too computationally expensive due to the complex nature of the optimization problem. Therefore, full smoothers are not suited for real-time operations.

For visual-inertial sensor fusion, fixed-lag smoothing and full-smoothing become easily unfeasible due to their complexity and high computational cost. Filtering methods are usually the best option for state estimation when these two sensors are involved.

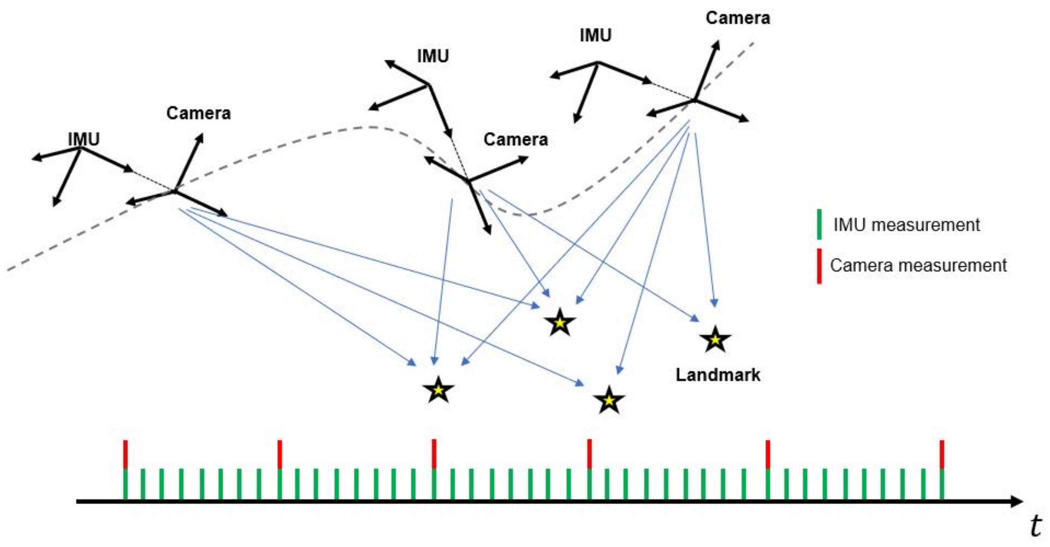


Figure 1.5: Illustration of the visual-inertial odometry problem showing the rigid attachement between camera and IMU frames.

Chapter 2

Background

2.1 Human eye

The human eye has six extra-ocular muscles that are responsible for pointing the eye in any given direction (gaze). These muscles are represented in Figure 1.1 and function as follows for the right eye: The Lateral Rectus (LR) rotates the eye rightward, the Superior Rectus (SR), rotates the eye up and anticlockwise, and the Superior Oblique (SO) rotates the eyes down and anticlockwise. For each of these muscles there is a corresponding muscle with which it forms a pair, respectively: the Medial Rectus (MR), in charge of rotating the eye leftward, the Inferior Rectus (IR) which rotates the eye down and clockwise and the Inferior Oblique (IO) that rotates it up and clockwise.

When it comes to its movement, the eye can't translate and therefore only has 3 degrees of freedom for rotational motion. However, Donder's law states that the torsional component of the eye orientation is a function of the vertical and horizontal components, meaning that the eye has not three but only two degrees of freedom. Listing's law can be seen as an extension of Donders' law, by quantitatively defining the amount of torsion. It states that, when the head is fixed and the optical axes are parallel (gazing at infinity), there is an eye orientation called primary position such that the eye assumes only the set of orientations that can be reached from the primary position by a single rotation about an axis in a plane (called Listing's plane) [2, 13, 14].

2.1.1 Saccadic eye movement

Saccades are extremely fast eye movements that humans make to direct the fovea to a region of interest. In order to quickly identify objects in the surroundings, humans and other animals have to direct the fovea as fast and as accurately as possible to the target. For instance, in monkeys, saccade peak velocities can reach 1300 deg/s and in humans around 700 deg/s [15, 16] (Figure 2.1. Each saccade captures a brief snapshot of a small piece high-acuity visual input. To have clear image of the whole environment around us, our visual perception ”glues” all the different snapshots captured [15].

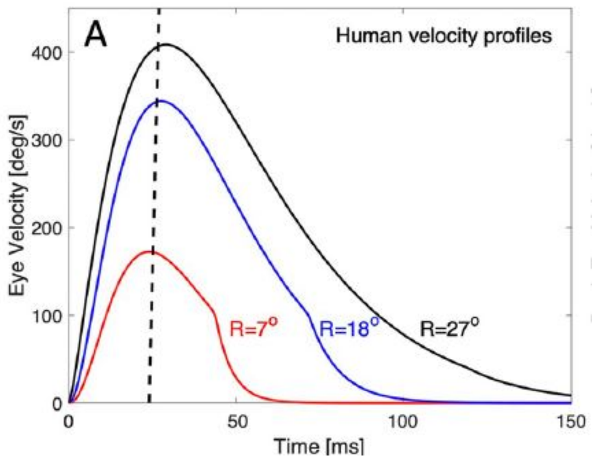


Figure 2.1: Example of velocity profiles of different amplitudes. From [2].

The perfect saccade is assumed to have a step-like shape that comes from a second-order low-pass system with an input that results from a combination of three different signals: a pulse, a step and an exponential decay (slide). This prediction of the input has also been observed in the firing of oculomotor neurons for a monkey saccade (shown in Figure 2.2).

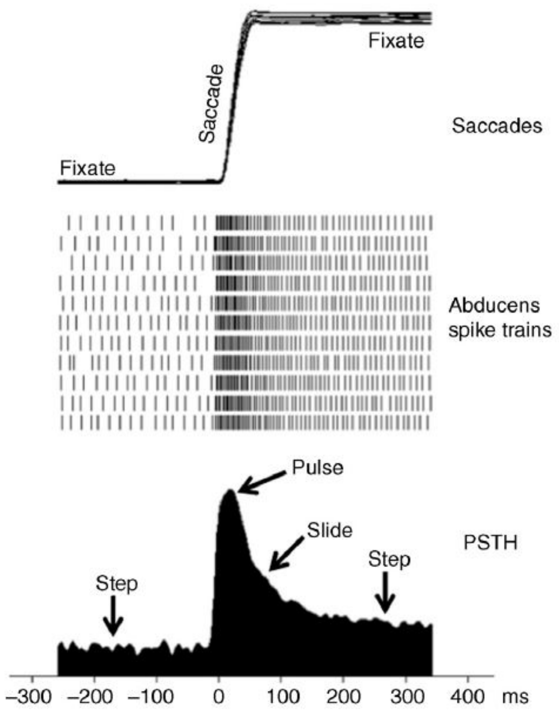


Figure 2.2: Recordings of oculomotor neuron activity in a monkey saccade [17].

2.2 Orientation parametrization

A rigid body's orientation can be represented in multiple ways. This section summarizes the most common used representations that were also relevant for this work. The use of each representation will depend on the specific applications which will be described in the next sections.

2.2.1 Rotation matrix

A rotation matrix, $\mathbf{R} \in \mathbb{R}^{3\times3}$, can be defined by the following properties:

$$\mathbf{R}^T\mathbf{R} = \mathbf{R}\mathbf{R}^T = \mathbf{I}_3, \quad \text{det}(\mathbf{R}) = 1 \tag{2.1}$$

As we'll see later in section 2.5.3, rotation matrices also belong to the *special orthogonal group*, $SO(3)$, which is a matrix Lie group [18, 19]. This group will prove to be very important for the purposes of this work. Considering a generic vector, $\mathbf{x} \in \mathbb{R}^3$, the rotated vector is given by:

$$\mathbf{x}' = \mathbf{R}\mathbf{x} \tag{2.2}$$

with $\mathbf{x}'$ also belonging to $\mathbb{R}^3$. It's also possible to perform multiple rotations of a given vector through matrix multiplication. The final rotated vector depends on the order in which the rotations are performed. Considering two different rotation matrices $\mathbf{R}_1, \mathbf{R}_2 \in SO(3)$:

$$\mathbf{R}_1\mathbf{R}_2 \neq \mathbf{R}_2\mathbf{R}_1 \tag{2.3}$$

2.2.2 Axis-angle

This representation describes an orientation with a rotation axis, $\hat{\mathbf{n}} = [n_x, n_y, n_z]$, and an angle θ. Considering a vector, $\mathbf{v} \in \mathbb{R}^3$, the rotated version, $\mathbf{u}$, is given by

$$\mathbf{u} = \mathbf{v}\cos(\theta) + (\hat{\mathbf{n}} \times \mathbf{v})\sin(\theta) + \hat{\mathbf{n}}(\hat{\mathbf{n}} \cdot v)(1 - \cos(\theta)) \tag{2.4}$$

It is also possible to rewrite the rotated vector in a matrix form [20],

$$\mathbf{u} = \Big(\mathbf{I}_3 + \sin\theta\hat{\mathbf{n}}^\wedge + (1 - \cos\theta)\left(\hat{\mathbf{n}}^\wedge\right)^2\Big)\mathbf{v} \tag{2.5}$$

where $\hat{\mathbf{n}}^\wedge$ is the skew-symmetric matrix operator to compute the cross product $\hat{\mathbf{n}} \times \mathbf{v} = \hat{\mathbf{n}}^\wedge\mathbf{v}$. This operator is also associated with the Lie algebra of $SO(3)$ which will be discussed further ahead in section 2.5. Equation 2.5 is known as *Rodriguez's formula* and from here we can extract the rotation matrix corresponding to a rotation around axis $\hat{\mathbf{n}}$ by an angle θ.

$$\mathbf{R}\left(\hat{\mathbf{n}}, \theta\right) = \mathbf{I}_3 + \sin\theta\hat{\mathbf{n}}^\wedge + (1 - \cos\theta)\left(\hat{\mathbf{n}}^\wedge\right)^2 \tag{2.6}$$

Finally, it's important to point out that this representation of the rotation matrix is also known as the *exponential map* and will be described in section 2.5.2 and 2.5.3.

2.2.3 Euler angles

Another very common representation is Euler angles. The rotations are defined using three angles, (ψ, θ, ϕ), where each angle represents a rotation around one of the three axis. The three rotations are sequential, and as with the case of rotation matrices, order is important. The most usual sequence is ZYX, which corresponds to rotating counter-clockwise around the Z-axis, then the Y-axis and finally the X-axis.

$$\mathbf{R}_z(\psi) = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathbf{R}_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$
$$\mathbf{R}_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix} \tag{2.7}$$

Combining these elemental rotations results in the following rotation matrix.

$$\begin{aligned} &\mathbf{R}_{zyx} = \mathbf{R}_x(\phi)\,\mathbf{R}_y(\theta)\,\mathbf{R}_z(\psi) = \\ &= \begin{bmatrix} \cos\theta\cos\psi & \cos\theta\sin\psi & -\sin\theta \\ \sin\phi\sin\theta\cos\psi - \cos\phi\sin\psi & \sin\phi\sin\theta\sin\psi + \cos\phi\cos\psi & \sin\phi\cos\theta \\ \cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi & \cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi & \cos\phi\cos\theta \end{bmatrix} \end{aligned} \tag{2.8}$$

This parametrization has the advantages over rotation matrices of being more intuitive and requiring only three variables in order to represent the rotation, whilst the latter requires nine parameters. However, Euler angles have the problem of not being a unique representation of a rotation, meaning that different sets of ψ, θ, ϕ can still refer to the same orientation. This common issue is called *gimbal lock*. More details about this can be found in [21].

2.2.4 Unit quaternions

Unit quaternions are a 4-dimensional parametrization of orientation and are written with a scalar component and a vector (imaginary) component as

$$\mathbf{q} = q_w + q_x\mathbf{i} + q_y\mathbf{j} + q_z\mathbf{k}, \quad ||\mathbf{q}|| = 1 \tag{2.9}$$

where $\mathbf{i}, \mathbf{j}, \mathbf{k}$ are imaginary numbers that satisfy this condition:

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1 \tag{2.10}$$

Unit quaternions live on the unit sphere $||\mathbf{q}|| = \sqrt{q_w^2 + q_x^2 + q_y^2 + q_z^2} = 1$ where opposite sign quaternions $\mathbf{q}$ and $-\mathbf{q}$, represent the same rotation as can be seen in Figure 2.3.

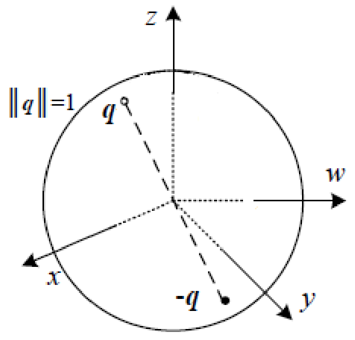


Figure 2.3: Unit sphere with opposite sign quaternions $-\mathbf{q}$ and $\mathbf{q}$ [20].

Furthermore, there's a way to write a quaternion formula through the axis/angle representation as

$$\mathbf{q} = \cos\left(\frac{\theta}{2}\right) + \hat{\mathbf{n}} \sin\left(\frac{\theta}{2}\right) \tag{2.11}$$

where $\hat{\mathbf{n}}$ and θ are the rotation axis and angle, respectively. Another useful expression is the rotation matrix associated with a quarternion, $\mathbf{R}(\mathbf{q})$ [21]

$$\mathbf{R}(\mathbf{q}) = \begin{bmatrix} q_w^2 + q_x^2 - q_y^2 - q_z^2 & 2q_xq_y + 2q_wq_z & 2q_xq_z - 2q_wq_y \\ 2q_xq_y - 2q_wq_z & q_w^2 - q_x^2 + q_y^2 - q_z^2 & 2q_yq_z + 2q_wq_x \\ 2q_xq_z + 2q_wq_y & 2q_yq_z - 2q_wq_x & q_w^2 - q_x^2 - q_y^2 + q_z^2 \end{bmatrix} \tag{2.12}$$

Even though parametrizing a rotation with quaternions is not very intuitive, there's a few advantages over the previous representations. Firstly, there's no gimbal lock as with the case of Euler angles and secondly, it's easier to perform interpolations. This last advantage plays an important role in the development of this work. A very common procedure used for quaternion interpolation is *SLERP* (Spherical Linear Interpolation) [22, 23]. Considering two quaternions $\mathbf{q}_0$ and $\mathbf{q}_1$, *SLERP* computes a new quaternion, $\mathbf{q}_2$, along the arc circle that connects $\mathbf{q}_0$ and $\mathbf{q}_1$.

$$\mathbf{q}_2 = \frac{\sin((1-\alpha)\theta)}{\sin(\theta)}\mathbf{q}_0 + \frac{\sin(\alpha\theta)}{\sin(\theta)}\mathbf{q}_1 \tag{2.13}$$

where $0 \leq \alpha \leq 1$ is the interpolation coefficient that determines how close $\mathbf{q}_2$ is from $\mathbf{q}_1$ or $\mathbf{q}_0$. The angle $\theta = \cos^{-1}(\mathbf{q}_0 \cdot \mathbf{q}_1)$ is half the distance between the two quaternions and the dot product is between the 4 dimensional quaternion vectors.

2.3 Coordinate frames

Throughout this work, a selected number of coordinates are necessary in order to represent measured quantities by the camera and Inertial Measurement Unit (IMU) and to describe the movements of the monocular visual-inertial system. The following coordinate frames were defined and their representation is depicted in Figure 2.4.

The IMU frame, $\{I\}$, belongs to the moving IMU. The origin is located at the center of the accelerometer and all the measurement vectors are represented in this frame.

The camera frame, $\{C\}$, is the coordinate frame of the uEye camera, centered at the pinhole and its orientation is according to the computer vision convention.

The world frame, $\{W\}$, is the frame in which the system will navigate. The pose of the IMU and the camera are determined with respect with this one.

The eye frame, $\{E\}$, is centered at the center of rotation of the eye. The coordinate system is according to the neuroscience convention. Here, the torsional component, x, is in the observation (gaze) direction, y is the horizontal component that points to the left and z is oriented vertically.

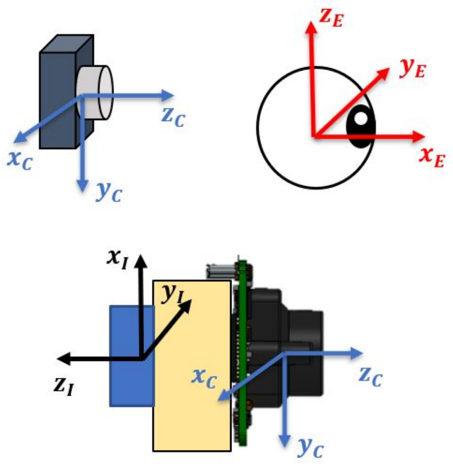


Figure 2.4: Computer vision (left) and neuroscience (right) conventions for the coordinate systems. Below, the coordinate frames of the system. The blue rectangle represents the IMU and the yellow the rigid attachment between the devices. The image of the camera was taken from the uEye datasheet (see appendix B.1).

Both the camera and IMU are rigidly attached to the eye that rotates around a fixed point in a spherical joint (in the Eye reference frame, mentioned above). So, their movements are constrained to move in a sphere, with coupled position and rotation [6].

2.4 Non-linear estimation

Most of the systems in the real-world are not linear and non-Gaussian, however, there are some approaches available nowadays that aim to solve this problem. There is a framework for solving these filtering problems called *Bayes filter* [18]. There are also several variations of this filter that have been used to tackle estimation: Extended Kalman Filter (EKF), Particle Filter (PF) and the sigma-point Kalman Filter, better know as *Unscented Kalman Filter* (UKF) [24].

2.4.1 ***Bayes Filter*** framework

Considering a general state vector $\mathbf{x}_k \in \mathbb{R}^N$, the objective is to come up with a PDF that represents the likelihood of this state using only measurements up the until the current instant. The system (eq.2.14) and observation (eq.2.15) are defined by the following models, respectively:

$$\mathbf{x}_k = f(\mathbf{x}_{k-1}, \mathbf{u}_k, \mathbf{w}_k), \quad k = 1...K \tag{2.14}$$

$$\mathbf{z}_k = h(\mathbf{x}_k, \mathbf{n}_k), \quad k = 0...K \tag{2.15}$$

where k is the discrete-time instant and K the duration. $\mathbf{u}_k \in \mathbb{R}^N$ is the input vector of the system, $\mathbf{w}_k \in \mathbb{R}^N$ is the process noise with an assumed Gaussian distribution $\mathcal{N}(\mathbf{0}, \mathbf{Q}_k)$, $\mathbf{z}_k \in \mathbb{R}^M$ is the observation vector and finally, $\mathbf{n}_k \in \mathbb{R}^M$ is the observation noise, also, with Gaussian distribution $\mathcal{N}(\mathbf{0}, \mathbf{R}_k)$. The PDF that we which to compute is called the *posterior belief* for $\mathbf{x}_k$:

$$p\left(\mathbf{x}_k | \check{\mathbf{x}}_0, \mathbf{u}_{1:k}, \mathbf{z}_{0:k}\right) \tag{2.16}$$

where $\check{\mathbf{x}}_0$ is the initial state estimate.
To solve the MAP (*Maximum A Posteriori*) problem, it's necessary to evaluate the PDF, $p(\mathbf{x}|\mathbf{u}, \mathbf{z})$, and find the best estimate for the system's state, $\hat{\mathbf{x}}$, for all time instants given the prior information and the measurements, $\mathbf{u}$ and $\mathbf{z}$, respectively. However, since we're dealing with a filtering problem, it is only necessary to estimate the state at time instant k. This yields the following PDF, factoring it into two parts:

$$p\left(\mathbf{x}_k | \mathbf{u}, \mathbf{z}\right) = \eta \, p\left(\mathbf{x}_k | \check{\mathbf{x}}_0, \mathbf{u}_{1:k}, \mathbf{z}_{0:k}\right) \; p\left(\mathbf{x}_k | \mathbf{u}_{k+1:K}, \mathbf{z}_{k+1:K}\right) \tag{2.17}$$

where η is a normalization constant that ensures that the axiom of total probability is preserved. The latter equation shows that to solve the filtering problem it's only necessary to focus on PDF given by 2.16. Taking into account the independence of all measurements and employing Bayes' rule to 2.16,

$$p\left(\mathbf{x}_k | \check{\mathbf{x}}_0, \mathbf{u}_{1:k}, \mathbf{z}_{0:k}\right) = \eta \, p\left(\mathbf{z}_k | \mathbf{x}_k\right) \; p\left(\mathbf{x}_k | \check{\mathbf{x}}_0, \mathbf{u}_{1:k}, \mathbf{z}_{0:k-1}\right) \tag{2.18}$$

Next, in the second factor of the previous equation, we look for the state, $\mathbf{x}_{k-1}$, and integrate over all possible values:

$$\begin{aligned} p\left(\mathbf{x}_k|\check{\mathbf{x}}_0, \mathbf{u}_{1:k}, \mathbf{z}_{0:k-1}\right) &= \int p\left(\mathbf{x}_k, \mathbf{x}_{k-1}|\check{\mathbf{x}}_0, \mathbf{u}_{1:k}, \mathbf{z}_{0:k-1}\right)\, d\mathbf{x}_{k-1} \\ &= \int p\left(\mathbf{x}_k|\mathbf{x}_{k-1}, \check{\mathbf{x}}_0, \mathbf{u}_{1:k}, \mathbf{z}_{0:k-1}\right)\; p\left(\mathbf{x}_{k-1}|\check{\mathbf{x}}_0, \mathbf{u}_{1:k}, \mathbf{z}_{0:k-1}\right)\, d\mathbf{x}_{k-1} \end{aligned} \tag{2.19}$$

Since the system has the Markov property, i.e., in order to compute the future state, $\mathbf{x}_k$, it's only necessary to have knowledge of the current state, $\mathbf{x}_{k-1}$, and not from previous states in the past, we have,

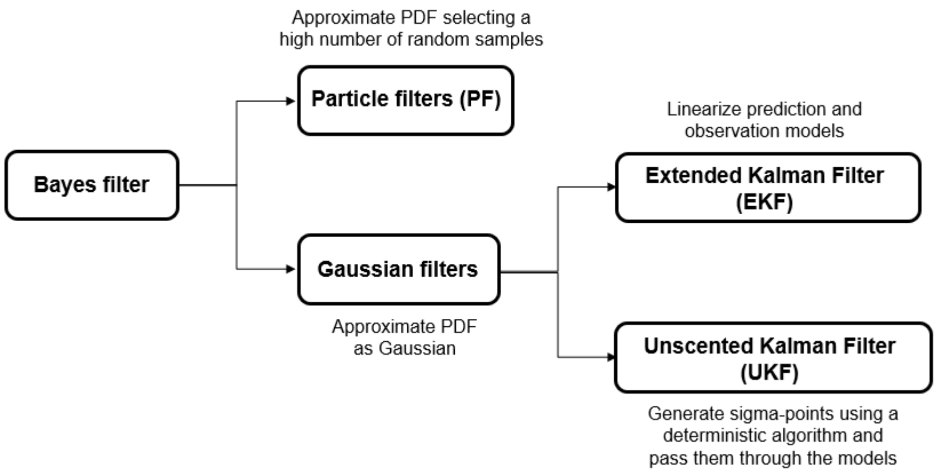
$$p\left(\mathbf{x}_k|\mathbf{x}_{k-1}, \check{\mathbf{x}}_0, \mathbf{u}_{1:k}, \mathbf{z}_{0:k-1}\right) = p\left(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{u}_k\right) \tag{2.20}$$

$$p\left(\mathbf{x}_{k-1}|\check{\mathbf{x}}_0, \mathbf{u}_{1:k}, \mathbf{z}_{0:k-1}\right) = p\left(\mathbf{x}_{k-1}|\check{\mathbf{x}}_0, \mathbf{u}_{1:k-1}, \mathbf{z}_{0:k-1}\right) \tag{2.21}$$

Finally, substituting 2.21, 2.20 and 2.19 into 2.18, yields the Bayes Filter:

$$p\left(\mathbf{x}_k|\check{\mathbf{x}}_0, \mathbf{u}_{1:k}, \mathbf{z}_{0:k}\right) = \eta\, p\left(\mathbf{z}_k|\mathbf{x}_k\right) \int p\left(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{u}_k\right)\; p\left(\mathbf{x}_{k-1}|\check{\mathbf{x}}_0, \mathbf{u}_{1:k-1}, \mathbf{z}_{0:k-1}\right)\, d\mathbf{x}_{k-1} \tag{2.22}$$

In theory, this filtering technique works, but in practice it cannot be really implemented, especially in nonlinear systems. Despite this, there are approximations that one can do in order to get accurate state estimations. These techniques are briefly mentioned in the beginning of this section and can be categorized as such:



Approximate PDF selecting a high number of random samples
Particle filters (PF)
Bayes filter
Linearize prediction and observation models
Extended Kalman Filter (EKF)
Gaussian filters
Approximate PDF as Gaussian
Unscented Kalman Filter (UKF)
Generate sigma-points using a deterministic algorithm and pass them through the models

Figure 2.5: Relationship between the estimation filters [18].

2.4.2 Unscented transformation and sigma points generation

In this work, the approach will be to use a version of the *Unscented Kalman Filter* or UKF. This estimation technique assumes that the PDFs can be approximated as Gaussians and then, instead of linearizing the prediction (system) and observation models as in the EKF, the PDFs go through the non linear models using the sigma point transformation [24, 25]. The filter is comprised of two main steps: prediction and update. Before starting, lets be reminded of all the variables in the prediction and observation models, 2.14 and 2.15 respectively.

$$\begin{aligned}
&\text{System state: } \mathbf{x}_k \in \mathbb{R}^N \\
&\text{Input: } \mathbf{u}_k \in \mathbb{R}^N \\
&\text{Process noise: } \mathbf{w}_k \in \mathbb{R}^N \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_k) \\
&\text{Measurement: } \mathbf{z}_k \in \mathbb{R}^M \\
&\text{Measurement noise: } \mathbf{n}_k \in \mathbb{R}^M \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_k)
\end{aligned} \tag{2.23}$$

where once again, k is a discrete time instant.

Prediction

Considering a prior estimate of the state, $\mathbf{x}_{k-1}$, and its Gaussian representation, $\mathcal{N}(\hat{\mathbf{x}}_{k-1}, \hat{\mathbf{P}}_{k-1})$, where $\hat{\mathbf{x}}_{k-1}$ is the state mean and $\hat{\mathbf{P}}_{k-1}$ the covariance matrix, the following actions are performed during the prediction step:

1. Firstly, we incorporate the noise in the system state by stacking the respective means and covariance as such:

$$\hat{\boldsymbol{\mu}}_x = \begin{bmatrix} \hat{\mathbf{x}}_{k-1} \\ \mathbf{0} \end{bmatrix}, \qquad \hat{\boldsymbol{\Sigma}}_x = \begin{bmatrix} \hat{\mathbf{P}}_{k-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{Q}_k \end{bmatrix} \tag{2.24}$$

   The resulting PDF representation will still be a Gaussian, $\mathcal{N}(\hat{\boldsymbol{\mu}}_x, \hat{\boldsymbol{\Sigma}}_x)$. The total size of this new vector will be $L = 2N$.

2. Perform the sigma point transformation. Compute a set of $2L + 1$ sigma vectors $\boldsymbol{\mathcal{X}}_i$ from the distribution $\mathcal{N}(\hat{\boldsymbol{\mu}}_x, \hat{\boldsymbol{\Sigma}}_x)$.

$$\begin{aligned}
&\boldsymbol{\mathcal{X}}_0 = \hat{\boldsymbol{\mu}}_x, \\
&\boldsymbol{\mathcal{X}}_i = \hat{\boldsymbol{\mu}}_x + \sqrt{L+\lambda}\,\text{col}_i\left(\sqrt{\hat{\boldsymbol{\Sigma}}_x}\right), \qquad \text{for } i = 1, ..., L \\
&\boldsymbol{\mathcal{X}}_{i+L} = \hat{\boldsymbol{\mu}}_x - \sqrt{L+\lambda}\,\text{col}_i\left(\sqrt{\hat{\boldsymbol{\Sigma}}_x}\right),
\end{aligned} \tag{2.25}$$

   where $\text{col}_i\left(\sqrt{\hat{\boldsymbol{\Sigma}}_x}\right)$ represents the i-th column vector of the matrix square root of $\hat{\boldsymbol{\Sigma}}_x$. This matrix square root can be calculated using the lower triangular Cholesky factorization. The scaling parameter λ is set to $\lambda = \alpha^2(L + \zeta) - L$, where α defines the spread of the sigma points around the mean $\hat{\boldsymbol{\mu}}_x$ and is usually a small positive number. ζ is another scaling parameter but it's normally set to 0 [25].

3. Propagate the sigma points through the system model in 2.14 with the latest input $\mathbf{u}_k$.

$$\begin{aligned} \boldsymbol{\mathcal{X}}_{k-1,i} &= \begin{bmatrix} \mathcal{X}^x_{k-1,i} \\ \mathcal{X}^w_{k,i} \end{bmatrix} \\ \check{\mathcal{X}}^x_{k,i} &= f(\mathcal{X}^x_{k-1,i}, \mathbf{u}_k, \mathcal{X}^w_{k,i}), \qquad i = 0, ..., 2L \end{aligned} \tag{2.26}$$

4. Compute the predicted belief $\mathcal{N}(\check{\mathbf{x}}_k, \check{\mathbf{P}}_k)$ using the weighted sample mean and covariance of the propagated sigma points.

$$\check{\mathbf{x}}_k = \sum_{i=0}^{2L} W_i^{(m)} \check{\mathcal{X}}^x_{k,i}, \tag{2.27}$$

$$\check{\mathbf{P}}_k = \sum_{i=0}^{2L} W_i^{(c)} \left(\check{\mathcal{X}}^x_{k,i} - \check{\mathbf{x}}_k\right) \left(\check{\mathcal{X}}^x_{k,i} - \check{\mathbf{x}}_k\right)^T, \tag{2.28}$$

where the weights W_i are given by

$$\begin{aligned} W_0^{(m)} &= \frac{\lambda}{L+\lambda} \\ W_0^{(c)} &= \frac{\lambda}{L+\lambda} + \left(1 - \alpha^2 + \beta\right) \\ W_i^{(m)} &= W_i^{(c)} = \frac{1}{2(L+\lambda)} \quad i = 1, \ldots, 2L \end{aligned} \tag{2.29}$$

the new parameter β inserts prior knowledge about the distribution of $\mathbf{x}_{k-1}$. Since the state is assumed to have a Gaussian distribution, the optimal value is $\beta = 2$.

Update

In this step, we take predicted belief and measurement model and use the regular Gaussian correction equations to perform the update of the state mean and covariance.

1. Similarly to the first step of the propagation, the predicted state, $\check{\mathbf{x}}_k$, and covariance, $\check{\mathbf{P}}_k$, are stacked with the observation noise:

$$\check{\boldsymbol{\mu}}_x = \begin{bmatrix} \check{\mathbf{x}}_k \\ \mathbf{0} \end{bmatrix}, \qquad \check{\boldsymbol{\Sigma}}_x = \begin{bmatrix} \check{\mathbf{P}}_k & \mathbf{0} \\ \mathbf{0} & \mathbf{R}_k \end{bmatrix} \tag{2.30}$$

Once again this is still a Gaussian PDF and we let $L = N + M$, which are the dimensions of the state vector and the measurement vector, respectively.

2. Perform the sigma point transformation. Compute a set of $2L+1$ sigma vectors $\boldsymbol{\mathcal{X}}_i$ from the

distribution $\mathcal{N}(\check{\boldsymbol{\mu}}_x, \check{\boldsymbol{\Sigma}}_x)$.

$$\begin{aligned}
\boldsymbol{\mathcal{X}}_0 &= \check{\boldsymbol{\mu}}_x, \\
\boldsymbol{\mathcal{X}}_i &= \check{\boldsymbol{\mu}}_x + \sqrt{L+\lambda}\,\text{col}_i\left(\sqrt{\check{\boldsymbol{\Sigma}}_x}\right), \qquad \text{for } i = 1, ..., L \\
\boldsymbol{\mathcal{X}}_{i+L} &= \check{\boldsymbol{\mu}}_x - \sqrt{L+\lambda}\,\text{col}_i\left(\sqrt{\check{\boldsymbol{\Sigma}}_x}\right),
\end{aligned} \tag{2.31}$$

3. Propagate the sigma points through the observation model in 2.15.

$$\begin{aligned}
\boldsymbol{\mathcal{X}}_{k,i} &= \begin{bmatrix} \mathcal{X}^x_{k,i} \\ \mathcal{X}^n_{k,i} \end{bmatrix} \\
\check{\mathcal{Z}}_{k,i} &= h(\mathcal{X}^x_{k,i}, \mathcal{X}^n_{k,i}), \qquad i = 0, ..., 2L
\end{aligned} \tag{2.32}$$

4. Compute the weighted sample mean of the predicted measurements, $\check{\mathbf{z}}_k$, and cross-covariance terms, $\boldsymbol{\Sigma}_{zz,k}$ and $\boldsymbol{\Sigma}_{xz,k}$, to compute the Kalman gain.

$$\check{\mathbf{z}}_k = \sum_{i=0}^{2L} W_i^{(m)} \check{\mathcal{Z}}_{k,i} \tag{2.33}$$

$$\boldsymbol{\Sigma}_{zz,k} = \sum_{i=0}^{2L} W_i^{(c)} \left(\check{\mathcal{Z}}_{k,i} - \check{\mathbf{z}}_k\right)\left(\check{\mathcal{Z}}_{k,i} - \check{\mathbf{z}}_k\right)^T \tag{2.34}$$

$$\boldsymbol{\Sigma}_{xz,k} = \sum_{i=0}^{2L} W_i^{(c)} \left(\check{\mathcal{X}}^x_{k,i} - \check{\mathbf{x}}_k\right)\left(\check{\mathcal{Z}}_{k,i} - \check{\mathbf{z}}_k\right)^T \tag{2.35}$$

the weights W_i are computed in the same way as in 2.29.

5. Finally, using the equations above, the current measurement, $\mathbf{z}_k$, and the predicted belief in 2.28, we compute the Kalman gain, $\boldsymbol{\mathcal{K}}_k$, and update the state mean, $\hat{\mathbf{x}}_k$, and covariance, $\hat{\mathbf{P}}_k$ (*posterior belief*).

$$\boldsymbol{\mathcal{K}}_k = \boldsymbol{\Sigma}_{xz,k}\boldsymbol{\Sigma}_{zz,k}^{-1} \tag{2.36}$$

$$\hat{\mathbf{x}}_k = \check{\mathbf{x}}_k + \boldsymbol{\mathcal{K}}_k\left(\mathbf{z}_k - \check{\mathbf{z}}_k\right) \tag{2.37}$$

$$\hat{\mathbf{P}}_k = \check{\mathbf{P}}_k - \boldsymbol{\mathcal{K}}_k\boldsymbol{\Sigma}_{xz,k}^T \tag{2.38}$$

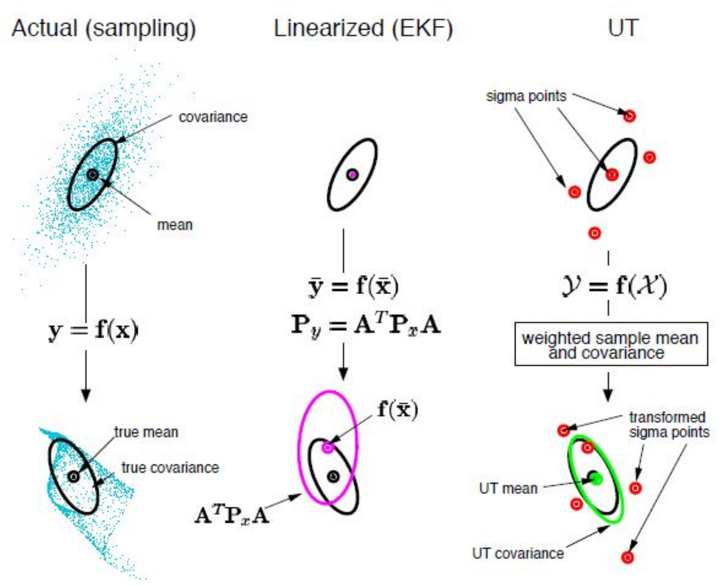

Actual (sampling)
Linearized (EKF)
UT
covariance
mean
$\mathbf{y} = \mathbf{f}(\mathbf{x})$
true mean
true covariance
$\bar{\mathbf{y}} = \mathbf{f}(\bar{\mathbf{x}})$
$\mathbf{P}_y = \mathbf{A}^T\mathbf{P}_x\mathbf{A}$
$\mathbf{f}(\bar{\mathbf{x}})$
$\mathbf{A}^T\mathbf{P}_x\mathbf{A}$
sigma points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

Figure 2.6: Example of the unscented transformation for a 2D system. The left plot shows the true mean and covariance using Monte-Carlo sampling. The middle plot shows the propagation using the EKF. The right plot shows the propagation of sigma points in the UKF [25].

2.5 Lie group theory

2.5.1 Definitions and properties

In mathematics, a group, $\mathcal{G}$, is a set of elements that together with an operation, $*$, respect the following properties or group axioms:

- **Closure**: For any $g_1, g_2 \in \mathcal{G}$, it holds that $g_3 = g_1 * g_2 \in \mathcal{G}$.
- **Associativity**: For any $g_1, g_2, g_3 \in \mathcal{G}$ it holds that $(g_1 * g_2) * g_3 = g_1 * (g_2 * g_3)$.
- **Identity**: There's an identity element, $e \in \mathcal{G}$ such that for any $g \in \mathcal{G}$ it's true that $g * e = e * g = g$.
- **Invertibility**: For every element $g \in \mathcal{G}$ there is an inverse counterpart $g^{-1} \in \mathcal{G}$ that verifies $g * g^{-1} = g^{-1} * g = e$.

A Lie group is a group that is also a differential manifold with operations that are smooth (the derivatives are continuous). Another property that is worth mentioning is commutativity in which the group operation of the two elements doesn't depend on the order in which they are written. As will be discussed later on, the matrix Lie groups, $SO(3)$ and $SE(3)$, do not share this property, meaning that they are non-commutative (or non-abelian) groups [18].

2.5.2 Exponential mapping and Lie algebras

Every Lie group is associated with a Lie algebra which is its tangent space at the identity element, completely capturing the local structure of the group. The exponential map relates a Lie group to its Lie algebra by locally mapping an element of the tangent space to the group. The inverse operation, the logarithmic map, transfers elements from the group back to its tangent space.

Considering an element, g, belonging to the Lie group, $\mathcal{G}$, the respective tangent space at g is given by $T_g\mathcal{G}$. If we consider a point $\mathfrak{g} \in T_g\mathcal{G}$, the exponential map results in a new element of the Lie group, g':

$$g' = g\,\mathrm{Exp}\left(\mathfrak{g}\right) \tag{2.39}$$

The inverse operation, the logarithmic map, transforms g' back to the tangent space, $T_g\mathcal{G}$,

$$\mathfrak{g} = \mathrm{Log}\left(g^{-1}g'\right) \tag{2.40}$$

It's important to note that in expressions 2.39 and 2.40, the operations Exp and Log differ from the regular definitions of the exponential and logarithmic maps. More details about these functions will be described for the specific cases in sections 2.5.3, 2.5.4 and 2.5.5. A visual representation of the transforms between Lie group and Lie algebra (tangent space) described is depicted in Figure 2.7.

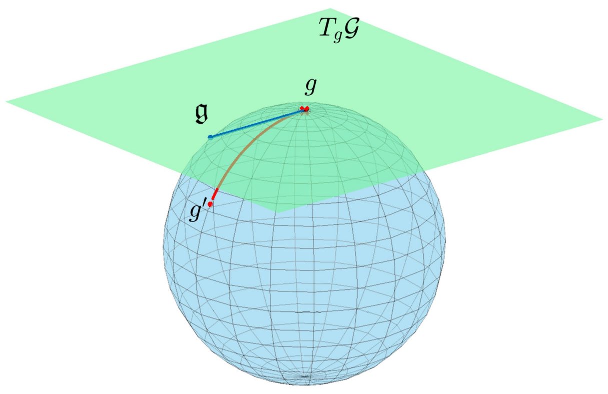


Figure 2.7: Representation of how the exponential mapping transforms points between the Lie group and the tangent space.

Another relevant property of Lie groups is the direct product, which allows the combination of two different groups, $\mathcal{G}$ and $\mathcal{H}$, into a new group, $\mathcal{G} \times \mathcal{H}$. If $\mathcal{G}$ has an operation $*$ and $\mathcal{H}$ an operation $\bullet$, the result of the direct product is defined component-wise as follows:

$$\left(g_1, h_1\right)\left(g_2, h_2\right) = \left(g_1 * g_2, h_1 \bullet h_2\right) \tag{2.41}$$

with $g_1, g_2 \in \mathcal{G}$ and $h_1, h_2 \in \mathcal{H}$. The resulting group $\mathcal{G} \times \mathcal{H}$ satisfies all the properties mentioned in 2.5.1. For the sake of completeness, the identity and invertibility are described below as they'll be important later on.

- **Identity**: The identity element of the direct product is $\left(e_{\mathcal{G}}, e_{\mathcal{H}}\right)$ such that $e_{\mathcal{G}} \in \mathcal{G}$ and $e_{\mathcal{H}} \in \mathcal{H}$.

- **Invertibility**: For every element $(g, h) \in \mathcal{G} \times \mathcal{H}$ there is an inverse $\left(g^{-1}, h^{-1}\right)$.

2.5.3 Special Orthogonal Group $SO(3)$

The special orthogonal group $SO(3)$ is composed by 3D rotation matrices and is defined as following,

$$SO(3) := \left\{ \mathbf{R} \in \mathbb{R}^{3\times 3} : \mathbf{R}^T\mathbf{R} = \mathbf{R}\mathbf{R}^T = \mathbf{I}_3, \text{det}(\mathbf{R}) = 1 \right\} \tag{2.42}$$

this set of matrices together with matrix multiplication as the group operation and the transpose as the inverse, becomes a matrix Lie Group, as will be the case with $SE(3)$ (2.5.4) and other topologies (2.5.5).

The tangent space to the group $SO(3)$ at the identity is denoted as $\mathfrak{so}(3)$ and is also called the Lie algebra. It's defined as the space of 3×3 skew symmetric matrices. Given a vector $\boldsymbol{\omega} \in \mathbb{R}^3$,

$$\boldsymbol{\omega}^\wedge = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}^\wedge = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} \in \mathfrak{so}(3) \tag{2.43}$$

where $(.)^\wedge$ is the *hat* operator as in [18, 26, 27]. Inversely, there's the *vee* operator, $(.)^\vee$ that maps a skew symmetric matrix into a vector in $\mathbb{R}^3$ as:

$$\boldsymbol{\Omega} = \boldsymbol{\omega}^\wedge \Rightarrow \boldsymbol{\Omega}^\vee = \boldsymbol{\omega} \tag{2.44}$$

The exponential map transforms an element of the Lie algebra $\mathfrak{so}(3)$ into a rotation matrix in $SO(3)$ and corresponds to the *Rodriguez's* formula in section 2.5. For convenience, the definition is re-written below:

$$\mathbf{R} = \exp\left(\boldsymbol{\omega}^\wedge\right) = \mathbf{I}_3 + \frac{\sin(\|\boldsymbol{\omega}\|)}{\|\boldsymbol{\omega}\|}\boldsymbol{\omega}^\wedge + \frac{1-\cos(\|\boldsymbol{\omega}\|)}{\|\boldsymbol{\omega}\|^2}\left(\boldsymbol{\omega}^\wedge\right)^2 \tag{2.45}$$

On the other hand, the logarithmic map converts a rotation matrix back into a skew symmetric matrix and is defined as:

$$\log(\mathbf{R}) = \frac{\theta\left(\mathbf{R}-\mathbf{R}^T\right)}{2\sin(\theta)} \text{ with } \theta = \cos^{-1}\left(\frac{\text{tr}(\mathbf{R})-1}{2}\right) \tag{2.46}$$

where θ is the rotation angle around a rotation axis, $\hat{\mathbf{n}}$. Letting $\boldsymbol{\omega} = \theta\hat{\mathbf{n}}$, then $\boldsymbol{\omega} = \log(\mathbf{R})^\vee$. By convention, θ is chosen such as $\|\boldsymbol{\omega}\| < \pi$. If this domain is not restricted, then the exponential map becomes surjective as every vector $\boldsymbol{\omega} = (\theta + 2\pi k)\,\hat{\mathbf{n}}$ with $k \in \mathbb{Z}$ would be a possible solution.

For the sake of simplicity of notation, the exponential and logarithmic maps are altered to operate in vectors instead of skew symmetric matrices:

$$\begin{aligned} &\text{Exp}: \ \mathbb{R}^3 \to \text{SO}(3) \ ; \ \boldsymbol{\omega} \mapsto \exp\left(\boldsymbol{\omega}^\wedge\right), \ \boldsymbol{\omega}^\wedge \in \mathfrak{so}(3) \\ &\text{Exp}\left(\boldsymbol{\omega}\right) = \exp\left(\boldsymbol{\omega}^\wedge\right) \end{aligned} \tag{2.47}$$

$$\begin{aligned} &\text{Log}: \ \text{SO}(3) \to \mathbb{R}^3 \ ; \ \mathbf{R} \mapsto \log\left(\mathbf{R}\right)^\vee \\ &\text{Log}(\mathbf{R}) = \log(\mathbf{R})^\vee \end{aligned} \tag{2.48}$$

2.5.4 Special Euclidean Group $SE(3)$

The special euclidean group $SE(3)$ allows for the representation of poses (orientation and position) in a matrix form and is defined by:

$$SE(3) := \left\{ \mathbf{T} \in \mathbb{R}^{4\times4} : \mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{o} \\ \mathbf{0}_{1\times3} & 1 \end{bmatrix}, \mathbf{R} \in SO(3); \mathbf{o} \in \mathbb{R}^3 \right\} \tag{2.49}$$

This group becomes a Lie group with matrix multiplication and the inverse operation given by:

$$\mathbf{T}^{-1} = \begin{bmatrix} \mathbf{R}^T & -\mathbf{R}^T\mathbf{o} \\ \mathbf{0} & 1 \end{bmatrix} \tag{2.50}$$

The lie algebra of $SE(3)$ is denoted as $\mathfrak{se}(3)$ and consists of the set of matrices of the form

$$\boldsymbol{\xi}^\wedge = \begin{bmatrix} \mathbf{v} \\ \boldsymbol{\omega} \end{bmatrix}^\wedge = \begin{bmatrix} \boldsymbol{\omega}^\wedge & \mathbf{v} \\ 0 & 0 \end{bmatrix} \in \mathfrak{se}(3) \tag{2.51}$$

with $\boldsymbol{\omega}^\wedge \in \mathfrak{so}(3)$ and $\mathbf{v} \in \mathbb{R}^3$. Once again, the *hat* operator is used to transform a vector in $\mathbb{R}^6$ into a matrix in $\mathbb{R}^{4\times4}$. Physically, $\boldsymbol{\omega}$ and $\mathbf{v}$ can be interpreted as rotational and translational velocities, respectively. This will be important in the process of propagation of the filter in chapter 3. Similarly as with case of $SO(3)$, the *vee* operator transforms the elements of $\mathfrak{se}(3)$ into $SE(3)$: $\boldsymbol{\Xi} = \boldsymbol{\xi}^\wedge \Rightarrow \boldsymbol{\Xi}^\vee = \boldsymbol{\xi}$.

Once again, the exponential map can relate elements of the Lie group to its Lie algebra using the following closed-form expression [18, 28]:

$$\begin{aligned} \mathbf{T} = \exp\left(\boldsymbol{\xi}^\wedge\right) &\equiv \sum_{n=0}^{\infty} \frac{1}{n!} \left(\boldsymbol{\xi}^\wedge\right)^n \\ &\equiv \mathbf{I}_4 + \boldsymbol{\xi}^\wedge + \left(\frac{1-\cos\phi}{\phi^2}\right)\left(\boldsymbol{\xi}^\wedge\right)^2 + \left(\frac{\phi - \sin\phi}{\phi^3}\right)\left(\boldsymbol{\xi}^\wedge\right)^3 \end{aligned} \tag{2.52}$$

with $\phi = \|\boldsymbol{\omega}\|$. We can also compute the inverse using

$$\boldsymbol{\xi} = \log(\mathbf{T})^\vee \tag{2.53}$$

2.5.5 Special Euclidean Group $SE(3)_{2+p}$

In recent works [29–31] it has been explored the Lie group structure that appears naturally in the SLAM problem. This "new" Lie group structure was named $SE(3)_{2+p}$, with p referring to the visual landmark measurements and includes the matrices written in the form

$$SE(3)_{2+p} := \left\{ \boldsymbol{\chi} \in \mathbb{R}^{(5+p)\times(5+p)} : \boldsymbol{\chi} = \begin{bmatrix} \mathbf{R} \;\; \mathbf{v} \;\; \mathbf{o} & \mathbf{p}_1 ... \mathbf{p}_p \\ \mathbf{0}_{(2+p)\times3} & \mathbf{I}_{(2+p)} \end{bmatrix}, \mathbf{R} \in SO(3); \mathbf{o}, \mathbf{v}, \mathbf{p}_1, ..., \mathbf{p}_p \in \mathbb{R}^3 \right\} \tag{2.54}$$

To compute the inverse of an element in $SE(3)_{2+p}$ we have

$$\boldsymbol{\chi}^{-1} = \begin{bmatrix} \mathbf{R}^T & -\mathbf{R}^T\mathbf{v} & -\mathbf{R}^T\mathbf{o} & -\mathbf{R}^T(\mathbf{p}_1...\mathbf{p}_p) \\ & \mathbf{0}_{(2+p)\times 3} & & \mathbf{I}_{(2+p)} \end{bmatrix} \tag{2.55}$$

And since $\mathbf{R}^T \in SO(3)$ and $-\mathbf{R}^T\mathbf{v}, -\mathbf{R}^T\mathbf{o}, -\mathbf{R}^T(\mathbf{p}_1...\mathbf{p}_p) \in \mathbb{R}^3$ the inverse element $\boldsymbol{\chi}^{-1}$ is also a member of $SE(3)_{2+p}$. In a similar manner as with $SE(3)$, the Lie algebra of $SE(3)_{2+p}$ is defined as

$$\boldsymbol{\xi}^{\wedge} = \begin{bmatrix} \boldsymbol{\xi}_{\mathbf{R}}^{\wedge} \ \boldsymbol{\xi}_{\mathbf{v}} \ \boldsymbol{\xi}_{\mathbf{o}} \ \boldsymbol{\xi}_{\mathbf{p}_1} \cdots \boldsymbol{\xi}_{\mathbf{p}_p} \\ \mathbf{0}_{(2+p)\times(5+p)} \end{bmatrix} \in \mathfrak{se}(3)_{2+p} \tag{2.56}$$

with $\boldsymbol{\xi}_{\mathbf{R}}^{\wedge} \in \mathfrak{so}(3)$. Finally, the exponential map relates the elements of the Lie algebra to the matrix Lie group with the closed-form expression:

$$\boldsymbol{\chi} = \mathrm{Exp}(\boldsymbol{\xi}) = \exp(\boldsymbol{\xi}^{\wedge}) = \mathbf{I}_{5+p} + \boldsymbol{\xi}^{\wedge} + \left(\frac{1-\cos(\|\boldsymbol{\xi}_{\mathbf{R}}\|)}{\|\boldsymbol{\xi}_{\mathrm{R}}\|}\right)(\boldsymbol{\xi}^{\wedge})^2 + \left(\frac{\|\boldsymbol{\xi}_{\mathbf{R}}\| - \sin(\|\boldsymbol{\xi}_{\mathrm{R}}\|)}{\|\boldsymbol{\xi}_{\mathrm{R}}\|^3}\right)(\boldsymbol{\xi}^{\wedge})^3 \tag{2.57}$$

2.5.6 Gaussian distributions and uncertainty description

When working with Gaussian random variables living in a vectorspace, i.e, $\mathbf{x} \in \mathbb{R}^N$, they usually take the form

$$\mathbf{x} \sim \mathcal{N}(\hat{\mathbf{x}}, \boldsymbol{\Sigma}) \quad \text{or} \quad \mathbf{x} = \hat{\mathbf{x}} + \boldsymbol{\epsilon},\ \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}) \tag{2.58}$$

where $\hat{\mathbf{x}}$ is a noise-free component also known as the mean and $\boldsymbol{\epsilon}$ is a small noise component with zero-mean and covariance, $\boldsymbol{\Sigma}$. This representation, however, does not work for matrix Lie groups because the closure property mentioned in 2.5.1 and 2.5.3 would not hold ($\mathbf{R}_1 + \mathbf{R}_2 \notin SO(3)$). So, the way we define random variables for matrix Lie groups is using the exponential map [18, 27]. For example, in $SO(3)$ the uncertainty definition is given by:

$$\mathbf{R} = \hat{\mathbf{R}}\,\mathrm{Exp}(\boldsymbol{\epsilon}), \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}) \tag{2.59}$$

with $\hat{\mathbf{R}}$ being the noise-free rotation matrix (or mean) and $\boldsymbol{\epsilon}$ the small perturbation with zero-mean and covariance, $\boldsymbol{\Sigma}$.

2.6 Camera model and image formation

To obtain an image from a camera it's necessary to transform 3D points from the world to 2D points in the camera plane. When capturing a frame, the light rays that reflect from the objects enter the tiny aperture on the camera (pinhole) and form the image on the sensor. The camera pinhole is also called center of projection, principal point, or focal point. Considering an observation, $\mathbf{O}$, of a point, $\mathbf{P}$, in 3D space projected using an ideal camera, the resulting image is flipped because the image plane is behind the pinhole. To avoid this, the frontal projection model (depicted in Figure 2.8) assumes that the image plane is in front of the center of projection [18].

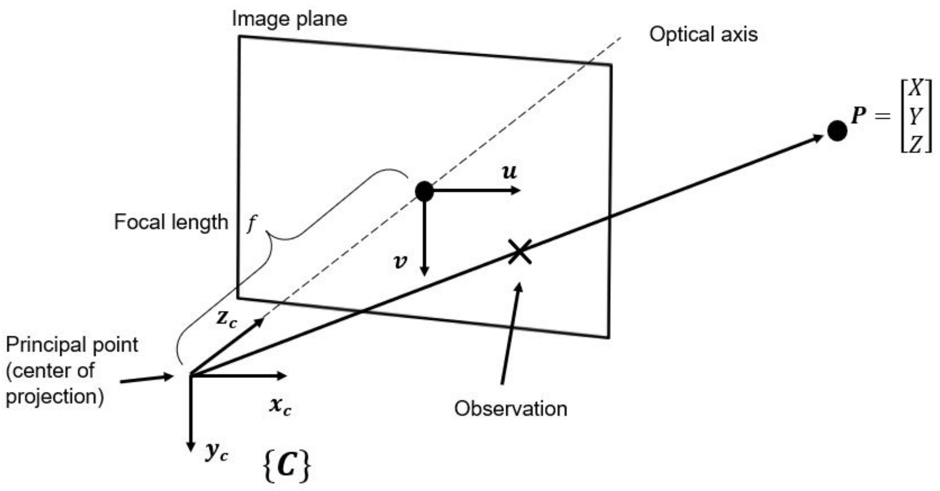


Image plane
Optical axis
$P = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$
u
v
Focal length f
z_c
Principal point (center of projection)
x_c
Observation
y_c
$\{C\}$

Figure 2.8: Camera projection model. In this model, the image plane is in front of the pinhole which avoids flipping the image.

To transform a point in the real world, $\mathbf{P} = [X\ Y\ Z]^T$, into a point in the image plane, $\mathbf{p} = [x\ y]^T$, we use the following expression,

$$x = f\frac{X}{Z},\ y = f\frac{Y}{Z} \tag{2.60}$$

where f is the focal length (distance between the image plane and the pinhole) in meters. From this equation, it's noticeable that is not possible to recover the Z coordinate (depth) from the image after projection.

2.6.1 Camera intrinsics

Once we have projected a 3D point through an ideal pinhole using a projection model, we still have to transform the resulting coordinates according to the pixel sensor spacing and the relative position of the sensor plane to the origin. To convert the points of the image plane into the actual digital image that we see through the camera, we have to resort to its intrinsic parameters, given by matrix $\boldsymbol{K}$ in equation 2.61.

$$\boldsymbol{K} = \begin{bmatrix} fs_x & s & c_x \\ 0 & fs_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.61}$$

where f is once again the focal length, (s_x, s_y) are scaling parameters in $pixel/m$ that convert the points into pixels, (c_x, c_y) are the offsets to the optical centre in pixels, and finally, a skew s (from non-orthogonality between optical axis and image plane).

To have a complete camera model, it's also assumed that the camera frame is not always aligned with the world reference frame. Therefore, to convert points from the world frame to the camera frame, we a need rotation (represented my a matrix $\mathbf{R}$) and a translation, $\mathbf{t}$. These are usually called the extrinsic parameters of the camera. It's also common in computer vision to represent this model in homogeneous coordinates as in equation 2.62

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \boldsymbol{K} \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.62}$$

with $\lambda = Z$ being the unknown scaling parameter. The expression above is also commonly represented as

$$\lambda \mathbf{x} = \mathbf{P}\mathbf{X} \tag{2.63}$$

with $\mathbf{P}$ being the 3×4 *camera matrix*.

2.6.2 Lens distortion

The camera model discussed above assumes that cameras obey a linear projection model where straight lines in the world result in straight lines in the image. Unfortunately, in practice, lenses have considerable non-linearities that result, for example, in radial distortion, which manifests itself as a visible curvature in the projection of straight lines. There's also tangential distortion and comes from the camera sensor mis-alignment during the manufacturing process. It occurs when the camera sensor is not in parallel with the lens [20]. These effects are shown in Figures 2.9 and 2.10. For this work, we considered the combined radial-tangential distortion polynomial model expressed in the equation below. The radial parameters are given by k_1, k_2 and the tangential parameters by p_1, p_2.

$$\begin{aligned} x_d' &= x\left(1 + k_1 r^2 + k_2 r^4\right) + \left(2p_1 xy + p_2\left(r^2 + 2x^2\right)\right) \\ y_d' &= y\left(1 + k_1 r^2 + k_2 r^4\right) + \left(p_1\left(r^2 + 2y^2\right) + 2p_2 xy\right) \end{aligned} \tag{2.64}$$

where $r^2 = x^2 + y^2$. x, y are the undistorted image coordinates from equation 2.60 and x_d', y_d' are corresponding distorted coordinates.

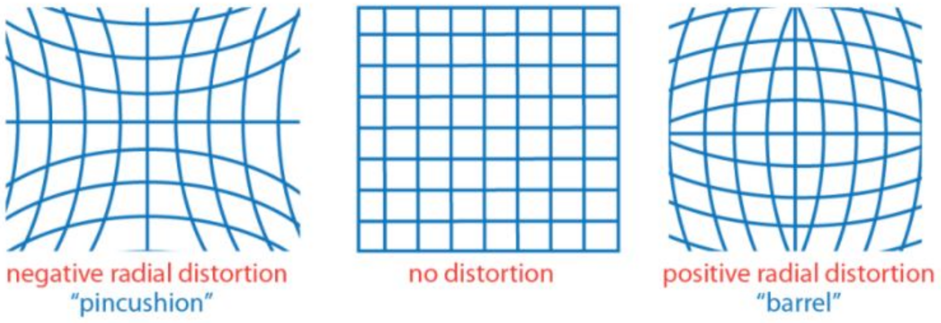


Figure 2.9: Example of radial distortion effect.

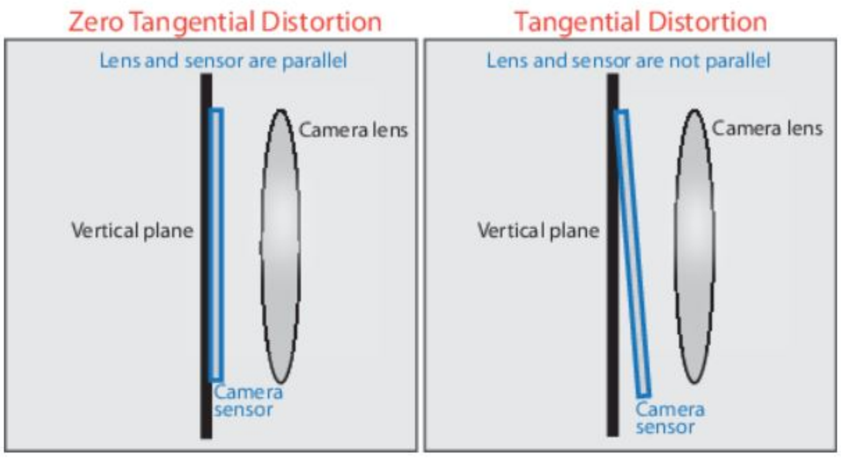


Figure 2.10: Cause of the tangential distortion.

2.7 Feature detection and tracking

Image features are the points of interest which provide rich information about the image which can be used for recognition, matching, reconstruction, among many other applications in computer vision. These interest points are preferably invariant to rotation, translation, intensity and scale changes (basically robust and reliable). There are different types interest points such as corners, edges, blobs etc. The process of identifying features in an image is called feature detection, and multiple detectors have been described in the literature, dependent on the features of interest. According to [32] ideal features should have the following qualities:

- **Distinctiveness:** the detected features should have characteristics that makes them easy to identify;
- **Locality:** to reduce the chances of getting occluded.
- **Quantity:** should be in enough quantity to describe the image;
- **Accuracy:** features should be accurate enough to be detected scale, shape or location;
- **Efficiency:** should be detected fast enough to be used real-time application;
- **Repeatability:** a high number of features should be detected in different images in the same regions;
- **Invariance:** deformative minimal effects on the features, due to scaling, rotation or translation;
- **Robustness:** features should be less sensitive to deformations due to noise, blur, compression, etc.

2.7.1 Corner detection

As mentioned above, corners can be considered interest points in an image. The Harris Corner Detector [33] is a famous example of a corner detector. It works using the following steps:

- Compute the x-wise (horizontal), $I_x(x, y)$, and y-wise (vertical), $I_y(x, y)$, partial image derivatives;
- Compute the second-order derivatives, $I_x^2(x, y)$, and, $I_y^2(x, y)$, and cross-derivatives, $I_x, I_y(x, y)$
- Compute the second-moment matrix $M(x, y)$ in a Gaussian window around each pixel
- Compute the Harris score defined by: $H(x, y) = \lambda_1\lambda_2 - k \times (\lambda_1 + \lambda_2)^2 = det(M) - k \times trace(M)^2$
- Detect local extrema whose Harris score is greater than the set threshold

An alternative to select corners is to analyse the eigenvalues, λ_1 and λ_2, of M. If both eigenvalues are below a minimum, there's no interesting features (flat region). If one is low, but the other is high, we are in the presence of and edge. Lastly, if both eigenvalues are high, the pixel is most likely a corner (Figure 2.11). Therefore, another solution to corner selection criterion, is to check the value of the lowest eigenvalue of $M(\lambda_{min})$, through the approximation Eq. (2.65).

$$\lambda_{min} \approx \frac{\lambda_1\lambda_2}{\lambda_1 + \lambda_2} = \frac{det(M)}{trace(M)} \tag{2.65}$$

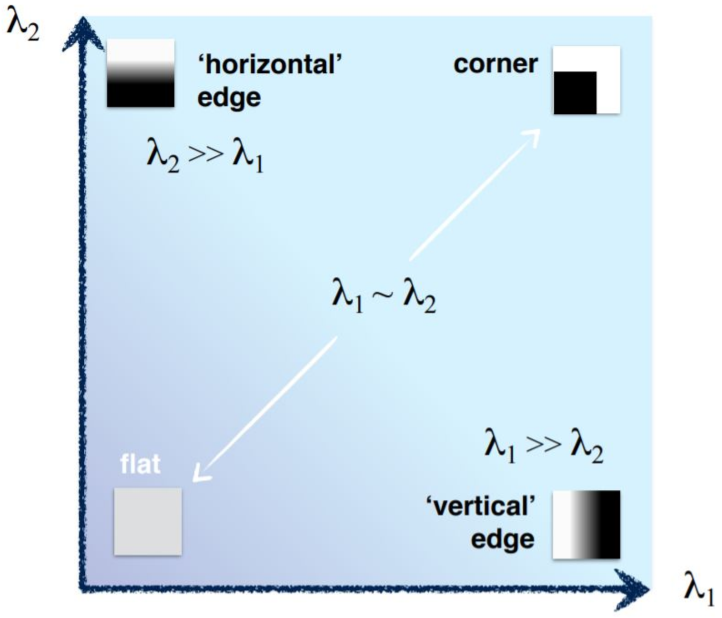


Figure 2.11: Diagram of corner selection according to the eigenvalues.

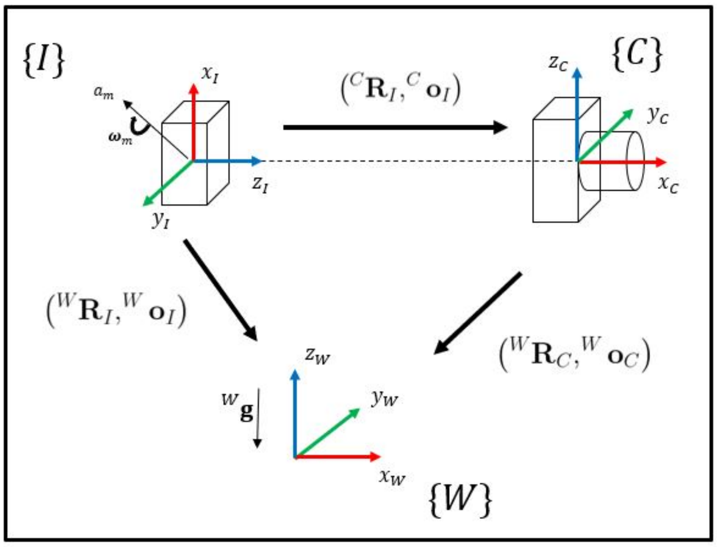
Chapter 3

Methods

This chapter contains all the mathematical models used to develop the fusion of the IMU sensor and the camera. The coordinates frames and the transformations between physical quantities are described in section 3.1 and the sensor models in section 3.2. The variable notation used from here on out was based on the work [26] and can also be consulted in the Nomenclature section.

3.1 System model

Firstly, we need to assign coordinate frames to each component of the system, as well as, define the geometric transformations between them. Figure 3.1 conceptually describes the kinematics of our visual-inertial sensor configuration. The IMU frame, denoted as $\{I\}$, is rigidly attached to the camera frame, $\{C\}$, and finally a static world frame, $\{W\}$ is created. The frame assignment of the IMU is done according to its datasheet B.2 and the camera frame is oriented based on the neuroscience convention.



$\{I\}$
a_m
ω_m
x_I
y_I
z_I
$({}^C\mathbf{R}_I, {}^C\mathbf{o}_I)$
z_C
$\{C\}$
y_C
x_C
$({}^W\mathbf{R}_I, {}^W\mathbf{o}_I)$
$({}^W\mathbf{R}_C, {}^W\mathbf{o}_C)$
z_W
${}^W\mathbf{g}$
y_W
x_W
$\{W\}$

Figure 3.1: System kinematics.

The next sections contain the continuous and the discrete time system models that describe the movement of the IMU and the camera.

3.1.1 System dynamics

In this model, we consider that the IMU is the main responsible for the dynamics of the system. The inertial measurements of the gyroscope and accelerometer are directly incorporated in the propagation of the state variables. These variables include the orientation, velocity and position of the IMU (from here on out called ”body”) relative to the world frame. As will be seen in sections 3.2.1 and 3.2.2, the IMU measurements are affected by bias and noise and so, in the state dynamics they are modelled as random walks. Lastly, the 3D positions of p static landmarks are included in the state and will be tracked by the camera.

Let's consider the following model of the system containing the IMU and camera [30]:

$$\text{Body (IMU):} \begin{cases} {}^W\dot{\mathbf{R}}_I(t) = {}^W\mathbf{R}_I(t)\left({}^I\boldsymbol{\omega}_{W,I}(t) + \mathbf{b}^{\mathbf{g}}(t) + \mathbf{w}^{\mathbf{g}}(t)\right)^\wedge = {}^W\mathbf{R}_I(t)\,{}^I\tilde{\boldsymbol{\omega}}^\wedge_{W,I}(t) \\ {}^W\dot{\mathbf{v}}_{W,I}(t) = {}^W\mathbf{R}_I(t)\left({}^I\mathbf{a}_{W,I}(t) + \mathbf{b}^{\mathbf{a}}(t) + \mathbf{w}^{\mathbf{a}}(t)\right) + {}^W\mathbf{g} = {}^W\mathbf{R}_I(t)\,{}^I\tilde{\mathbf{a}}_{W,I}(t) + {}^W\mathbf{g} \\ {}^W\dot{\mathbf{o}}_I(t) = {}^W\mathbf{v}_{W,I}(t) \\ \dot{\mathbf{b}^{\mathbf{g}}}(t) = \mathbf{w}^{\mathbf{bg}} \\ \dot{\mathbf{b}^{\mathbf{a}}}(t) = \mathbf{w}^{\mathbf{ba}} \end{cases} \tag{3.1}$$

$$\text{Landmarks:} \begin{cases} {}^W\dot{\mathbf{p}}_i = \mathbf{0}, \quad i = 1, ..., p \end{cases} \tag{3.2}$$

where ${}^W\mathbf{R}_I(t) \in SO(3)$ is the current orientation of the IMU frame relative to the world frame, ${}^W\mathbf{o}_I(r) \in \mathbb{R}^3$ and ${}^W\mathbf{v}_{W,I}(t) \in \mathbb{R}^3$ are the position and velocity of the IMU in the world frame, respectively. Finally, vectors $\mathbf{b}^{\mathbf{g}}(t), \mathbf{b}^{\mathbf{a}}(t) \in \mathbb{R}^3$ are the IMU biases that are related to the gyroscope and accelerometer measurements, ${}^I\tilde{\boldsymbol{\omega}}_{W,I}(t), {}^I\tilde{\mathbf{a}}_{W,I}(t) \in \mathbb{R}^3$. The multiple white Gaussian noise sources are modelled according to equation 3.3.

$$\begin{aligned} \mathbf{w}^{\mathbf{g}} &\sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_{\mathbf{g}}), \quad \boldsymbol{\Sigma}_{\mathbf{g}} = \sigma_g^2\mathbf{I}_3 \in \mathbb{R}^{3\times3} \\ \mathbf{w}^{\mathbf{a}} &\sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_{\mathbf{a}}), \quad \boldsymbol{\Sigma}_{\mathbf{a}} = \sigma_a^2\mathbf{I}_3 \in \mathbb{R}^{3\times3} \\ \mathbf{w}^{\mathbf{bg}} &\sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_{\mathbf{bg}}), \quad \boldsymbol{\Sigma}_{\mathbf{bg}} = \sigma_{bg}^2\mathbf{I}_3 \in \mathbb{R}^{3\times3} \\ \mathbf{w}^{\mathbf{ba}} &\sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_{\mathbf{ba}}), \quad \boldsymbol{\Sigma}_{\mathbf{ba}} = \sigma_{ba}^2\mathbf{I}_3 \in \mathbb{R}^{3\times3} \end{aligned} \tag{3.3}$$

3.1.2 Time discretization

The dynamic model that describes the evolution of the state in 3.1.1 is expressed in continuous time and, therefore, is now discretized using the first-order forward Euler method as in [27, 34]. The discrete time system can be expressed as

$$
\begin{aligned}
&{}^{W}\mathbf{R}_I(t+\Delta t) = {}^{W}\mathbf{R}_I(t)\,\mathrm{Exp}\left(\left({}^{I}\tilde{\boldsymbol{\omega}}_{W,I}(t) - \mathbf{b}^{\mathbf{g}}(t) - \mathbf{w}^{\mathbf{gd}}(t)\right)\Delta t\right) \\
&{}^{W}\mathbf{v}_{W,I}(t+\Delta t) = {}^{W}\mathbf{v}_{W,I}(t) + \left({}^{W}\mathbf{R}_I(t)\left({}^{I}\tilde{\mathbf{a}}_{W,I}(t) - \mathbf{b}^{\mathbf{a}}(t) - \mathbf{w}^{\mathbf{ad}}(t)\right) + {}^{W}\mathbf{g}\right)\Delta t \\
&{}^{W}\mathbf{o}_I(t+\Delta t) = {}^{W}\mathbf{o}_I(t) + {}^{W}\mathbf{v}_{W,I}(t)\Delta t + \frac{1}{2}\left({}^{W}\mathbf{R}_I(t)\left({}^{I}\tilde{\mathbf{a}}_{W,I}(t) - \mathbf{b}^{\mathbf{a}}(t) - \mathbf{w}^{\mathbf{ad}}(t)\right) + {}^{W}\mathbf{g}\right)\Delta t^2 \\
&\mathbf{b}^{\mathbf{g}}(t+\Delta t) = \mathbf{b}^{\mathbf{g}}(t) + \mathbf{w}^{\mathbf{bgd}} \\
&\mathbf{b}^{\mathbf{a}}(t+\Delta t) = \mathbf{b}^{\mathbf{a}}(t) + \mathbf{w}^{\mathbf{bad}}
\end{aligned}
\tag{3.4}
$$

The discrete time noise variables $\mathbf{w}^{\mathbf{gd}}(t)$ and $\mathbf{w}^{\mathbf{ad}}(t)$ have a covariance that is a function of the sampling time Δt: $\boldsymbol{\Sigma}_{\mathbf{gd}}(t) = \frac{1}{\Delta t}\boldsymbol{\Sigma}_{\mathbf{g}}(t)$ and $\boldsymbol{\Sigma}_{\mathbf{ad}}(t) = \frac{1}{\Delta t}\boldsymbol{\Sigma}_{\mathbf{a}}(t)$ for the gyroscope and accelerometer, respectively. The biases noises $\mathbf{w}^{\mathbf{bg}}, \mathbf{w}^{\mathbf{ba}}$ are modelled in a similar way: $\boldsymbol{\Sigma}_{\mathbf{bgd}} = \Delta t\boldsymbol{\Sigma}_{\mathbf{bg}}$ and $\boldsymbol{\Sigma}_{\mathbf{bad}} = \Delta t\boldsymbol{\Sigma}_{\mathbf{ba}}$. For convenience, we dropped the coordinate frame superscripts and subscripts and replaced the time interval from $[t, t+\Delta t]$ to $[k-1, k]$ to express the state variables before and after integration in 3.5.

$$
\begin{aligned}
&\mathbf{R}_k = \mathbf{R}_{k-1}\,\mathrm{Exp}\left(\left(\tilde{\boldsymbol{\omega}}_k - \mathbf{b}^{\mathbf{g}}_{k-1} - \mathbf{w}^{\mathbf{gd}}_k\right)\Delta t\right) \\
&\mathbf{v}_k = \mathbf{v}_{k-1} + \left(\mathbf{R}_{k-1}\left(\tilde{\mathbf{a}}_k - \mathbf{b}^{\mathbf{a}}_{k-1} - \mathbf{w}^{\mathbf{ad}}_k\right) + \mathbf{g}\right)\Delta t \\
&\mathbf{o}_k = \mathbf{o}_{k-1} + \mathbf{v}_{k-1}\Delta t + \frac{1}{2}\left(\mathbf{R}_{k-1}\left(\tilde{\mathbf{a}}_k - \mathbf{b}^{\mathbf{a}}_{k-1} - \mathbf{w}^{\mathbf{ad}}_k\right) + \mathbf{g}\right)\Delta t^2 \\
&\mathbf{b}^{\mathbf{g}}_k = \mathbf{b}^{\mathbf{g}}_{k-1} + \mathbf{w}^{\mathbf{bgd}} \\
&\mathbf{b}^{\mathbf{a}}_k = \mathbf{b}^{\mathbf{a}}_{k-1} + \mathbf{w}^{\mathbf{bad}}
\end{aligned}
\tag{3.5}
$$

3.2 Observation models

The measurement models used for the IMU are described in this section. A simplification was made by considering that the world frame is stationary relative to the inertial frame because the sensor doesn't travel significant distances when compared to the size of the earth. Therefore, all the IMU measurements are considered to be made with respect to the world frame.

3.2.1 Gyroscope observation model

The 3-axis gyroscope measures the angular velocity in rad/s of the body frame with respect to the world frame, expressed in the body frame. The measured rotation rate, ${}^{I}\tilde{\boldsymbol{\omega}}_{W,I}(t)$ is affected by additive Gaussian noise, $\mathbf{w}_{\mathbf{g}}(t)$, and bias, $\mathbf{b}_{\mathbf{g}}(t)$.

$$
{}^{I}\tilde{\boldsymbol{\omega}}_{W,I}(t) = {}^{I}\boldsymbol{\omega}_{W,I}(t) + \mathbf{b}_{\mathbf{g}}(t) + \mathbf{w}_{\mathbf{g}}(t)
\tag{3.6}
$$

3.2.2 Accelerometer observation model

The accelerometer provides measurements of the specific force (acceleration together with gravity) in m/s^2. Like the gyroscope, the accelerometer model assumes the measurements are corrupted by Gaussian noise, $\mathbf{w_a}(t)$, plus a bias, $\mathbf{b_a}(t)$.

$$\begin{aligned} {}^{I}\tilde{\mathbf{a}}_{W,I}(t) &= {}^{I}\mathbf{a}_{W,I}(t) + \mathbf{b_a}(t) + \mathbf{w_a}(t) \\ &= {}^{I}\mathbf{R}_{W}(t)({}^{W}\mathbf{a}(t) - {}^{W}\mathbf{g}) + \mathbf{b_a}(t) + \mathbf{w_a}(t) \end{aligned} \tag{3.7}$$

If we assume that the linear accelerations in the world frame are approximetly zero (${}^{W}\mathbf{a}(t) \simeq \mathbf{0}$), then the accelerometer model is the following:

$${}^{I}\tilde{\mathbf{a}}_{W,I}(t) = -{}^{I}\mathbf{R}_{W}(t){}^{W}\mathbf{g} + \mathbf{b_a}(t) + \mathbf{w_a}(t) \tag{3.8}$$

This assumption comes from the fact that the main purpose of this sensor is to measure inclination, not the change in position.

3.2.3 Camera observation model

In combination with the measurements from the gyroscope and accelerometer, images from a camera are also collected. This information is used to detect and track the p landmarks using the standard pinhole model [6, 20, 31]. The measurement of landmark $\mathbf{p}_i$ is given by,

$$\mathbf{z}^i = \frac{1}{z_w^i}\begin{bmatrix} z_u^i \\ z_v^i \end{bmatrix} + \mathbf{n}_z^i, \quad i = 1, \ldots, p \tag{3.9}$$

with $\mathbf{n}_z^i \sim \mathcal{N}(\mathbf{0}, \mathbf{R})$ and $\mathbf{R} = \sigma_z^2\mathbf{I}_2 \in \mathbb{R}^{2\times2}$. Each landmark observation, $\mathbf{z}^i$, is obtained through the camera model in equation 3.10 in homogeneous coordinates. A detailed diagram of the coordinate transformations between reference frames is depicted in Figure 3.2.

$$\begin{aligned} \begin{bmatrix} z_u^i \\ z_v^i \\ z_w^i \end{bmatrix} &= \boldsymbol{K}\begin{bmatrix} \mathbf{I}_3 & \mathbf{0}_{3\times1} \end{bmatrix}\begin{bmatrix} {}^{C}\mathbf{p}_i \\ 1 \end{bmatrix} \\ &= \boldsymbol{K}\left[{}^{C}\mathbf{R}_I\left({}^{W}\mathbf{R}_I^{T}{}^{W}\mathbf{p}_i + {}^{I}\mathbf{o}_W\right) + {}^{C}\mathbf{o}_I\right] \end{aligned} \tag{3.10}$$

where ${}^{C}\mathbf{p}_i$ is the position of landmark $\mathbf{p}_i$ expressed in the camera coordinate frame and $\boldsymbol{K}$ is the intrinsic parameter matrix (for more information about the estimation of this matrix, check appendix B.1.2). From equation 3.10, the variables ${}^{C}\mathbf{R}_I$ and ${}^{C}\mathbf{o}_I$ refer to the relative pose between the IMU and the camera and result from the extrinsic calibration of the two sensors (check appendix B.3).

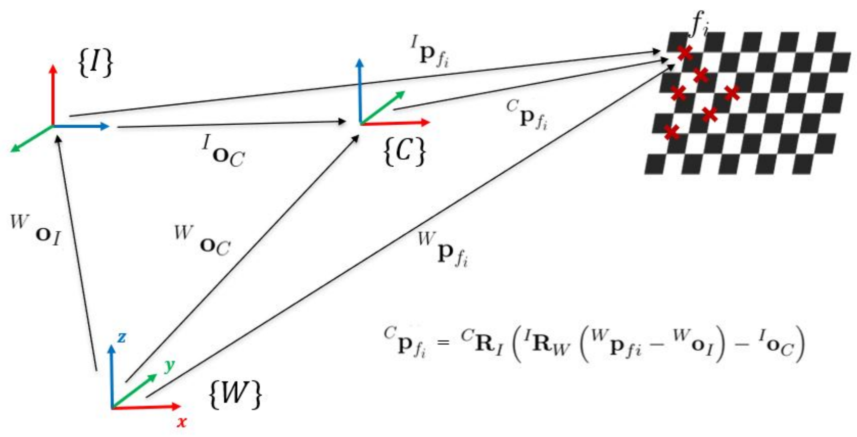


Figure 3.2: Coordinate transformations. Each landmark $\mathbf{p}_i$ is projected to the image plane according to 3.9.

3.3 Unscented Kalman Filter using Lie Group structure

This section contains the main object of study of this thesis, which includes using a variation of the standard Unscented Kalman Filter described in section 2.4. The method uses the Lie Group state structure introduced in earlier sections and the recent works involved in sensor fusion [30, 31, 35]. In the previous sections of this chapter we modelled the system dynamics through equations 3.1 and 3.5 and now we want use UKF formulation to compute the probability distribution of the states variables. However, we cannot use the standard version of the UKF which assumes that the state is in the form $\mathbf{x} \in \mathbb{R}^n$. We'll define a different parametrization of the state which involves the use of Lie group theory introduced in chapter 2.

3.3.1 State structure

Like with any Gaussian filter, the state distribution is defined by its mean, $\hat{\mathbf{x}}_k$, and covariance, $\mathbf{P}_k$. For our problem, the general mean state and covariance are given by

$$\hat{\mathbf{x}}_k = \left(\hat{\mathbf{R}}_k, \hat{\mathbf{v}}_k, \hat{\mathbf{o}}_k, \hat{\mathbf{b}}^{\mathbf{g}}_k, \hat{\mathbf{b}}^{\mathbf{a}}_k, \hat{\mathbf{p}}_1, \ldots, \hat{\mathbf{p}}_p\right) \in SO(3) \times \mathbb{R}^3 \times \mathbb{R}^3 \times \mathbb{R}^3 \times \mathbb{R}^3 \times \mathbb{R}^{3p} \tag{3.11}$$

and

$$\mathbf{P}_k = \begin{bmatrix} \mathbf{P_R} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{P_v} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{P_o} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{P_{p_1}} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{P_{p_p}} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{P_{b_g}} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{P_{b_a}} \end{bmatrix} \in \mathbb{R}^{(15+3p)\times(15+3p)} \tag{3.12}$$

where each member of the diagonal represents the covariance matrices of the orientation, velocity, position, landmark positions and gyroscope and accelerometer biases, respectively.

Another possible way to write the state is to use a combination of the Lie group structure described in section 2.5.5 that is commonly found in SLAM problems, with elements of $\mathbb{R}^3$. Under this topology, the orientation, velocity and position of the body (IMU) are combined with the landmark locations in a matrix of the form $\hat{\chi}_k \in SE(3)_{2+p}$. The biases $\hat{\mathbf{b}}_k^{\mathbf{g}}, \hat{\mathbf{b}}_k^{\mathbf{a}}$ are considered to be independent from this structure and maintain the vector form. Thus, the final state is represented by

$$\begin{aligned} \hat{\mathbf{x}}_k &= \left(\hat{\chi}_k, \hat{\mathbf{b}}_k\right) \in SE(3)_{2+p} \times \mathbb{R}^6 := \mathcal{S} \\ \hat{\chi}_k &= \begin{bmatrix} \hat{\mathbf{R}}_k & \hat{\mathbf{v}}_k & \hat{\mathbf{o}}_k & \hat{\mathbf{p}}_1 \ldots \hat{\mathbf{p}}_p \\ \multicolumn{3}{c}{\mathbf{0}_{(2+p)\times 3}} & \mathbf{I}_{(2+p)} \end{bmatrix} \in SE(3)_{2+p} \\ \hat{\mathbf{b}}_k &= \begin{bmatrix} \hat{\mathbf{b}}_k^{\mathbf{g}} \\ \hat{\mathbf{b}}_k^{\mathbf{a}} \end{bmatrix} \in \mathbb{R}^6 \end{aligned} \tag{3.13}$$

where $\mathcal{S}$ denotes the defined group for the state of the filter. The corresponding Lie algebra is denoted by $\mathfrak{s}$.

3.3.2 Uncertainty and state covariance

In order to define random variables for Lie groups we have to resort to another strategy since our state is not a vector. For the state $\mathbf{x}_k \in \mathcal{S}$, its uncertainty is represented by

$$\mathbf{x}_k \sim \mathcal{N}(\hat{\mathbf{x}}_k, \mathbf{P}_k) \rightarrow \mathbf{x}_k = \varphi(\hat{\mathbf{x}}_k, \boldsymbol{\xi}_k) = \left(\hat{\chi}_k \operatorname{Exp}(\boldsymbol{\xi}_k^{\chi}), \hat{\mathbf{b}}_k + \boldsymbol{\xi}_k^{b}\right), \quad \boldsymbol{\xi}_k = \begin{bmatrix} \boldsymbol{\xi}_k^{\chi} \\ \boldsymbol{\xi}_k^{b} \end{bmatrix} \sim \mathcal{N}(\mathbf{0}, \mathbf{P}_k) \tag{3.14}$$

where we have two Gaussian noise perturbations $\boldsymbol{\xi}_k^{\chi}$ and $\boldsymbol{\xi}_k^{b}$, for the matrix part $\hat{\chi}_k$, and for the vector part $\hat{\mathbf{b}}_k$ of the state, respectively. The two uncertainties in equation 3.15 are stacked together to give the total uncertainty of the state. Notice that the covariance matrix of $\boldsymbol{\xi}_k$ yields $\mathbf{P}_k$, defined in 3.12. For the bias vector, the Gaussian noise is additive as one would expect, but for the case of the matrix state we have to use the exponential map which is computed according to equations 2.56 and 2.57.

$$\begin{aligned} \boldsymbol{\xi}_k^{\mathcal{X}} &= \left[\boldsymbol{\xi}_{\mathbf{R}}^T\, \boldsymbol{\xi}_{\mathbf{v}}^T\, \boldsymbol{\xi}_{\mathbf{o}}^T\, \boldsymbol{\xi}_{\mathbf{p}_1}^T \cdots \boldsymbol{\xi}_{\mathbf{p}_p}^T\right]^T \in \mathbb{R}^{9+3p} \\ \boldsymbol{\xi}_k^{b} &= \left[\boldsymbol{\xi}_{\mathbf{b_g}}^T\, \boldsymbol{\xi}_{\mathbf{b_a}}^T\right]^T \in \mathbb{R}^{6} \end{aligned} \tag{3.15}$$

where the distribution of each uncertainty $\boldsymbol{\xi}$ is given by,

$$\begin{aligned} &\boldsymbol{\xi}_{\mathbf{R}} \sim \mathcal{N}(\mathbf{0}, \mathbf{P_R}), \quad \mathbf{P_R} \in \mathbb{R}^{3\times 3} \\ &\boldsymbol{\xi}_{\mathbf{v}} \sim \mathcal{N}(\mathbf{0}, \mathbf{P_v}), \quad \mathbf{P_v} \in \mathbb{R}^{3\times 3} \\ &\boldsymbol{\xi}_{\mathbf{o}} \sim \mathcal{N}(\mathbf{0}, \mathbf{P_o}), \quad \mathbf{P_o} \in \mathbb{R}^{3\times 3} \\ &\boldsymbol{\xi}_{\mathbf{b_g}} \sim \mathcal{N}(\mathbf{0}, \mathbf{P_{b_g}}), \quad \mathbf{P_{b_g}} \in \mathbb{R}^{3\times 3} \\ &\boldsymbol{\xi}_{\mathbf{b_a}} \sim \mathcal{N}(\mathbf{0}, \mathbf{P_{b_a}}), \quad \mathbf{P_{b_a}} \in \mathbb{R}^{3\times 3} \\ &\boldsymbol{\xi}_{\mathbf{p}_i} \sim \mathcal{N}(\mathbf{0}, \mathbf{P}_{\mathbf{p}_i}), \quad i = 1, ..., p, \quad \mathbf{P}_{\mathbf{p}_i} \in \mathbb{R}^{3\times 3} \end{aligned} \tag{3.16}$$

3.3.3 Propagation

Now that we have all the necessary definitions about the structure of the state we can now proceed to the propagation/dynamics of the system. Letting f be the discrete time function that propagates the state variables described in 3.5

$$\check{\mathbf{x}}_k = \left(\check{\boldsymbol{\chi}}_k, \check{\mathbf{b}}_k\right) = f\left(\hat{\mathbf{x}}_{k-1}, \mathbf{u}_k, \mathbf{w}_k\right) \tag{3.17}$$

we remind that $\check{\mathbf{x}}_k$ has the same notation as the one used to represent the predicted belief at time instant k, in the standard UKF (section 2.4.2). The input vector $\mathbf{u}_k$ is defined as 3-axis gyroscope and accelerometer readings, $\tilde{\boldsymbol{\omega}}_k$ and $\tilde{\mathbf{a}}_k$, respectively. Vector $\mathbf{w}_k$ denotes the Gaussian noise of the process and the respective covariance matrix $\mathbf{Q}_k$ is represented in the expression below.

$$\mathbf{w}_k = \begin{bmatrix} \mathbf{w}_k^{\mathbf{gd}} \\ \mathbf{w}_k^{\mathbf{ad}} \\ \mathbf{w}_k^{\mathbf{bgd}} \\ \mathbf{w}_k^{\mathbf{bad}} \end{bmatrix} \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_k) \quad \text{with} \quad \mathbf{Q}_k = \begin{bmatrix} \boldsymbol{\Sigma}_{\mathbf{g}} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\Sigma}_{\mathbf{a}} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{\Sigma}_{\mathbf{bg}} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{\Sigma}_{\mathbf{ba}} \end{bmatrix} \in \mathbb{R}^{12\times 12} \tag{3.18}$$

The propagation of the state starts by propagating the noiseless mean $\hat{\mathbf{x}}_{k-1}$ and augment the covariance to include the process noise. Then, sigma points for the augmented state uncertainty are generated and are passed through the dynamic model in 3.17. However these sigma points are first mapped to the Lie group using 3.14. The propagated sigma points are finally transformed back to the tangent space where we then retrieve the covariance matrix $\check{\mathbf{S}}_k$. More details are described in Algorithm 1 and in appendix A.2. It's important to point out that both the propagation and the update steps follow the square-root implementation of the UKF, which propagates the covariance matrix through its Cholesky factor for being numerically more stable and computationaly less expensive [36].

Algorithm 1: Propagation of the state [31]

```
Input: $\hat{\mathrm{x}}_{k-1} = \left(\hat{\chi}_{k-1}, \hat{\mathbf{b}}_{k-1}\right)$,   $\hat{\mathbf{P}}_{k-1} \to \hat{\mathbf{S}}_{k-1}$   (Cholesky decomposition),
       $\mathbf{Q}$   (Process noise),   $\mathbf{u}_k = [\tilde{\boldsymbol{\omega}}_k^T\ \tilde{\mathbf{a}}_k^T]^T$   (IMU measurements)
1 $\mathbf{Q} \to \mathbf{S}_Q$,   (Cholesky decomposition)                           // Process noise covariance
2 $\hat{\mathbf{S}}_{k-1}^A = \begin{bmatrix} \hat{\mathbf{S}}_{k-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{S}_Q \end{bmatrix}$                // Create augmented state covariance
3 $\check{\mathrm{x}}_k = f\left(\hat{\mathrm{x}}_{k-1}, \mathbf{u}_k, \mathbf{0}\right) = \left(\check{\chi}_k, \check{\mathbf{b}}_k\right)$      // Propagate noiseless mean (Propagated belief)
  /* Generate sigma points                                                              */
4 $\boldsymbol{\mathcal{X}}_{k,i} = \gamma\, \mathrm{col}_i \left(\hat{\mathbf{S}}_{k-1}^A\right)$,   for $i = 1, ..., L^A$
5 $\boldsymbol{\mathcal{X}}_{k,i} = -\gamma\, \mathrm{col}_i \left(\hat{\mathbf{S}}_{k-1}^A\right)$,   for $i = 1 + L^A, ..., 2L^A$
6 $\boldsymbol{\mathcal{X}}_{k,i} = \pm \left[\boldsymbol{\xi}_{k,i}^{\chi}\ \boldsymbol{\xi}_{k,i}^{b}\ \mathbf{w}_{k,i}\right]$,   $i = 1, \ldots, L^A$
  /* Propagate sigma points through the process model                                   */
7 $\begin{cases} \check{\boldsymbol{\xi}}_i^b = \boldsymbol{\xi}_i^b + \Delta t \mathbf{w}_i^b & \text{// add bias noise} \\ \check{\mathcal{X}}_i^{\chi} = f\left(\hat{\chi}_{k-1} \operatorname{Exp}\left(\boldsymbol{\xi}_i^{\chi}\right), \mathbf{u}_k - \check{\boldsymbol{\xi}}_i^b, \mathbf{w}_i\right) \end{cases}$,   $i = 1, \ldots, 2L^A$     // Omitting time instant $k$
8 $\check{\boldsymbol{\xi}}_i^{\chi} = \operatorname{Log}\left(\left(\hat{\chi}_{k-1}\right)^{-1} \check{\mathcal{X}}_i^{\chi}\right)$,   $i = 1, \ldots, 2L^A$      // Transform points back to original space
9 $\check{\mathbf{S}}_k \leftarrow \mathrm{qr}\left(\check{\boldsymbol{\xi}}_i^{\chi}, \check{\boldsymbol{\xi}}_i^{b}, i = 1, \ldots, 2L^A, \mathbf{S}_Q\right)$      // Compute propagated covariance.  See Appendix or [31]
Output: $\check{\mathrm{x}}_k = \left(\check{\chi}_k, \check{\mathbf{b}}_k\right)$,   $\check{\mathbf{S}}_k$   (Propagated belief)
```

3.3.4 Update

For the update step of the filter, we'll consider the camera measurements for each of the static landmarks. These measurements are the 2D pixel locations of the features tracked using the Kanade-Lucas-Tomasi (KLT) tracker and take the form of equation 3.9. To facilitate reading, we compact all the camera observations in the following expression

$$\mathbf{Z}_k = \begin{bmatrix} \mathbf{z}_k^1 \\ \vdots \\ \mathbf{z}_k^p \end{bmatrix} := H\left(\chi_k, \mathbf{n}_z\right) \tag{3.19}$$

it's noticeable that this model only "observes" part of the state contained in χ_k. $\mathbf{n}_z \sim \mathcal{N}(\mathbf{0}, \mathbf{R})$ is the Gaussian noise for one landmark, with $\mathbf{R}$ being the measurement covariance matrix.

Similarly to the propagation step, the update starts by computing the predicted measurement using the propagated noiseless state $\check{\chi}_k$, and constructing the augmented state covariance by incorporating the measurement covariance. The $2L^A$ tangent space sigma points, $\xi_{k,i}^{\chi}$, are then mapped to the respective Lie group and passed through the observation model H to give the sigma points of the predicted measurements, $\check{\mathcal{Z}}_i$. Next the predicted measurement mean, $\check{\mathbf{Z}}_k$, the cholesky factor of the measurement covariance, $\mathbf{S}_{zz}$, and the cross covariance, $\mathbf{P}_{xz}$ are computed so that we can get the Kalman gain, $\mathcal{K}$, and the correction terms (innovation), $\delta\bar{\boldsymbol{\xi}}^{\chi}$ and $\delta\bar{\boldsymbol{\xi}}^{b}$. More details can be found in Algorithm 2 and in appendix A.3.

Algorithm 2: State update [31]

```
Input: $\check{\mathbf{x}}_k = \left(\check{\boldsymbol{\chi}}_k, \check{\mathbf{b}}_k\right), \quad \check{\mathbf{S}}_k,$
       $\mathbf{Z}_k = \left[\mathbf{z}_k^{1\,T} \ \dots \ \mathbf{z}_k^{p\,T}\right]^T$  (Camera observation of the $p$ landmarks),
       $\mathbf{R}$  (Observation noise)
1  $\mathbf{R} \rightarrow \mathbf{S}_R,$  (Cholesky decomposition)                    // Observation noise covariance
2  $\check{\mathbf{S}}_k^A = \begin{bmatrix} \check{\mathbf{S}}_k & \mathbf{0} \\ \mathbf{0} & \mathbf{S}_R \end{bmatrix}$                    // Create augmented state covariance
3  $\check{\boldsymbol{\mathcal{Z}}}_0 = H\left(\check{\boldsymbol{\chi}}_k, \mathbf{0}\right)$            // Pass propagated mean through observation model $H$
   /* Generate sigma points                                                   */
4  $\boldsymbol{\mathcal{X}}_{k,i} = \gamma\, \mathrm{col}_i\left(\check{\mathbf{S}}_k^A\right), \quad$ for $i = 1, ..., L^A$
5  $\boldsymbol{\mathcal{X}}_{k,i} = -\gamma\, \mathrm{col}_i\left(\check{\mathbf{S}}_k^A\right), \quad$ for $i = 1 + L^A, ..., 2L^A$
6  $\boldsymbol{\mathcal{X}}_{k,i} = \pm\left[\boldsymbol{\xi}_{k,i}^{\chi}\ \boldsymbol{\xi}_{k,i}^{b}\ \mathbf{n}_{k,i}\right], \quad i = 1, \dots, L^A$
   /* Propagate sigma points through the observation model                    */
7  $\check{\boldsymbol{\mathcal{Z}}}_i = H\left(\check{\boldsymbol{\chi}}_k \operatorname{Exp}\left(\boldsymbol{\xi}_i^{\chi}\right), \mathbf{n}_i\right), \quad i = 1, \dots, 2L^A$      // Omitting time instant $k$
8  $\check{\mathbf{Z}}_k = W_0^{(c)} \check{\boldsymbol{\mathcal{Z}}}_0 + \sum_{i=1}^{L^A} W_i^{(m)}\left(\check{\boldsymbol{\mathcal{Z}}}_i^{+} + \check{\boldsymbol{\mathcal{Z}}}_i^{-}\right)$      // Compute predicted measurement mean
9  $\mathbf{S}_{zz}, \mathbf{P}_{xz} \leftarrow$ computeMeasurementCovariances $\left(\check{\mathbf{Z}}_k, \check{\boldsymbol{\mathcal{Z}}}_0, \check{\boldsymbol{\mathcal{Z}}}_i^{\pm}, {\boldsymbol{\xi}_{k,i}^{\chi}}^{\pm}, {\boldsymbol{\xi}_{k,i}^{b}}^{\pm}\right)$   // Compute measurement
     covariance cholesky factor and cross covariance.  Check appendix
10 $\boldsymbol{\mathcal{K}} = \mathbf{P}_{xz}\left(\mathbf{S}_{zz}^T \mathbf{S}_{zz}\right)^{-1}$                    // Kalman gain
11 $\delta\overline{\boldsymbol{\xi}} = \left[\delta\overline{\boldsymbol{\xi}}^{\chi T},\ \delta\overline{\boldsymbol{\xi}}^{b T}\right]^T = \boldsymbol{\mathcal{K}}\left(\mathbf{Z}_k - \check{\mathbf{Z}}_k\right)$          // Innovation
   /* Update state mean and covariance (posterior belief)                     */
12 $\begin{cases} \hat{\boldsymbol{\chi}}_k = \check{\boldsymbol{\chi}}_k \operatorname{Exp}\left(\delta\overline{\boldsymbol{\xi}}^{\chi}\right) \\ \hat{\mathbf{b}}_k = \check{\mathbf{b}}_k + \delta\overline{\boldsymbol{\xi}}^{b} \end{cases}$
13 $\hat{\mathbf{S}}_k^{'} \leftarrow$ SeqCholUpdate $\left(\check{\mathbf{S}}_k,\ \boldsymbol{\mathcal{K}}\mathbf{S}_{zz}^T,\ -1\right)$             // Check appendix
14 $\hat{\mathbf{S}}_k = \hat{\mathbf{S}}_k^{'}\mathbf{J}$
Output: $\hat{\mathbf{x}}_k = \left(\hat{\boldsymbol{\chi}}_k, \hat{\mathbf{b}}_k\right), \quad \hat{\mathbf{S}}_k$  (Posterior belief)
```

Chapter 4

Implementation

In this section, an overview of the implemented system will be made. The implementation comprises the practical application of the methods and algorithms described in chapter 3. The UKF on Lie group algorithm was implemented in MATLAB [31] and was tested in a simulation environment [37] as well as in real data (offline). All the steps involved in the design and development of the experiments will also be described in the following sections.

4.1 Full system architecture

The implemented system revolves around the UKF on Lie groups with the objective of fusing information from the IMU and camera. Figure 4.1 shows a simple flow diagram with the main functions of the designed system. The measurements from the inertial sensor, i.e, angular velocity (rad/s) and acceleration (m/s^2), are directly used to propagate the state of the filter and infer the motion of the body through simple Euler integration as this is simpler and easier to implement [27]. In the update step, the visual information provided by the camera is used to correct the trajectory and update the error covariances. It does so by detecting and tracking p static landmarks in the scene using the standard pinhole camera model in 3.2.3. The filter uses the KLT tracker using minimum eigenvalue feature detection, both implemented in MATLAB [38].

In order to evaluate the performance of the filter, the orientation estimation is compared against a ground-truth. In the case of the simulator, as it'll be seen, this ground-truth corresponds to the generated trajectory. In the real case, the ground-truth is given by the joint angle feedback of the Kinova Gen3 robot that is used to infer the trajectory of the end-effector, where the camera and IMU are mounted 4.3. In the following sections, the procedures involved in the planning and the setup for the experiments will be described.

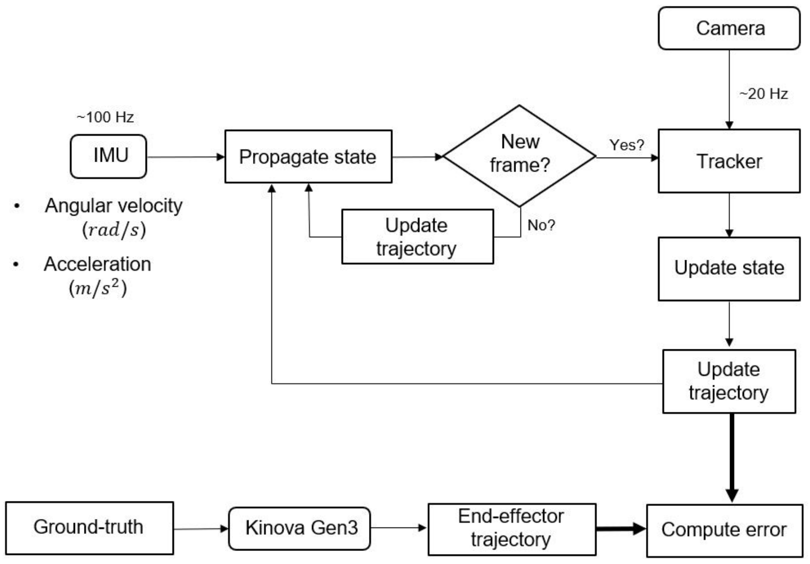


Figure 4.1: Simplified flow diagram of the system. The gyroscope and accelerometer readings propagate the state of the filter at a high rate (100 Hz), while the camera updates and corrects the state whenever there's a frame available (20 Hz). This illustration only contains the ground truth for the real case scenario.

4.2 Simulation (ESIM simulator)

In the previous work [6] done for this project, a MATLAB simulator that generates image points matches (with or without noise) with known rotations was developed. However, for this thesis, since the goal is to have a sense of the whole trajectory of the camera, this simulator becomes a bit short because we're not only interested in the initial and final orientation of the camera. Plus, this simulator skips the whole image processing task usually involved in visual odometry, such as feature detection and tracking.

For these reasons, the option was taken to use the open event camera simulator, ESIM [37], which was already being used in the context of the ORIENT project. The object of this work does not involve event cameras, however, this simulator (depicted in Figure 4.2) is still able to provide regular camera frames, IMU readings and ground-truth.

The rendering engine is able to capture the scene from the viewpoint of the camera, at a constant frame rate and can also simulate exposure times which results in motion blur of the images (making a more realistic approximation of an actual camera). The user inputs the trajectory of the virtual camera and the simulator generates the accelerometer and gyroscope readings as well as the ground-truth trajectory. This trajectory is a result of a spline interpolation, in order to give a smooth trajectory (continuous and differentiable). This also is important for the generation of the inertial sensor readings because they can be derived using double differentiation, in the case of the accelerometer. Then, the simulator adds some bias and noise, resulting in more realistic data. The same logic can be applied for the generation of angular velocity readings for the gyroscope by taking the first derivative of the rotation along the trajectory.

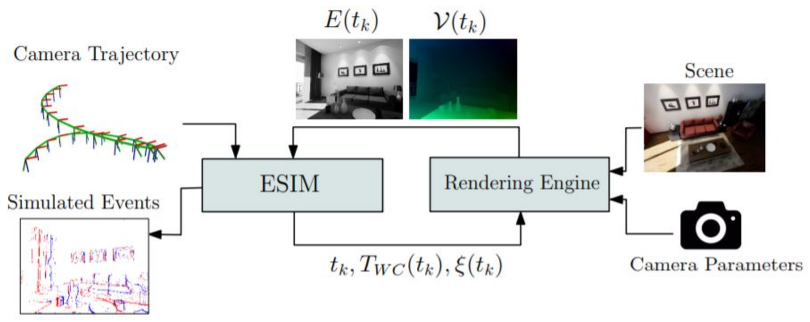


Figure 4.2: Architecture of ESIM. It shows the tight coupling between the rendering engine and the simulator. The user defines the camera trajectory (pose, $T_{WC}(t_k)$, and twist, $\xi(t_k)$) which is then passed through the rendering engine, resulting in a new irradiance map $E(t_k)$ and a motion field map $\mathcal{V}(t_k)$ at time t_k. For more information consult [37].

4.3 Real system - Kinova Gen3

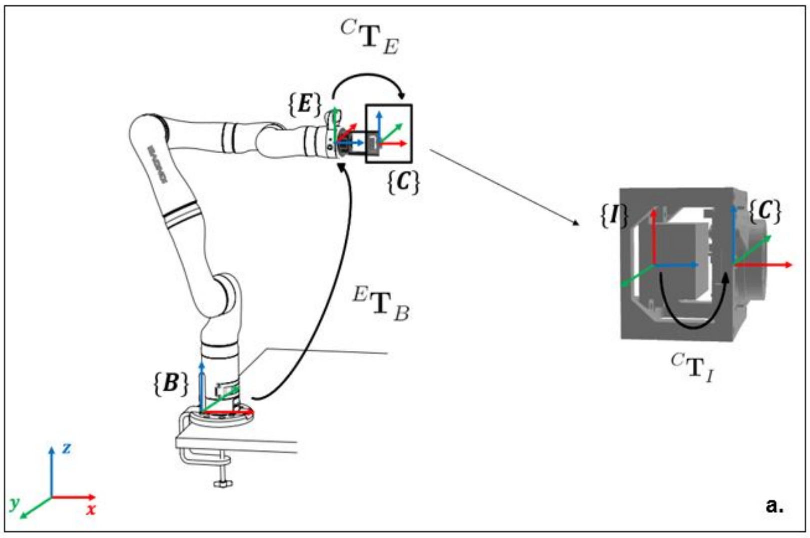
One of the main problems to solve in the previous implementation [6] was the aspect of the ground-truth when dealing with real world data. The ground-truth orientation could only be estimated up to an error of 4 degrees which compromised the comparison with the rotation estimation algorithms. For this work, the option was to use the Kinova Gen3 robotic arm with 7 degrees of freedom, where each joint has a precision of around 1 degree. Besides this very important feature, with the Kinova and its available API [1] (in C++, Python, MATLAB and ROS) it's possible to communicate with the robot via Ethernet connection and develop scripts to configure parameters, read sensor feedback, send joint commands, etc. The only downside of using the Kinova is its built in joint speed limitations of around 50 deg/s for the smaller joints and 57 deg/s for the bigger joints (see datasheet in [39]). As mentioned in the beginning in section 1.2, eye saccades have very high peak velocities (around 700 deg/s) and so, the ideal case would be a benchmarking system that is able to handle these kinds of speeds. However, for the purpose of this thesis, it's better to start with a simpler and perhaps more reliable setup to test the algorithm and slowly improve in future works.

Like with the experiment using the ESIM simulator, the purpose of working with the robotic arm was to send pre-computed, well defined trajectories that serve as ground-truth, and at the same time, collect the camera frames and inertial sensor readings (from gyroscope and accelerometer). This way, we'll have a real world dataset where it's possible to test visual inertial odometry algorithms. The details of the dataset recording and format are described in section 4.3.3.

[1] https://www.kinovarobotics.com/en/resources/gen3-technical-resources

4.3.1 Sensor setup

The setup used for the recording of the dataset is represented in Figure 4.3.a and consists of the Kinova Gen3 robot, with a custom gripper designed in SolidWorks where the visual-inertial system is attached. The 3D model of the gripper is shown in Figure 4.3.b. The camera and IMU are attached to the gripper using hex screws in such a way so that vibrations are as low as possible. In Figure 4.3.a are also illustrated the different transformations that are relevant to problem. ${}^{E}\mathbf{T}_{B}$ refers to transformation between the base frame, $\{B\}$, and the end-effector frame of the robot, $\{E\}$, and is specified by the Denavit–Hartenberg parameters in the Kinova datasheet [39]. ${}^{C}\mathbf{T}_{E}$ expresses the pose of the camera frame, $\{C\}$, relative to the end-effector frame. This transformation is obtained through the known dimensions of the designed gripper, and as it'll be seen in section 4.3.2, the planning of trajectories that are sent to the robot are based in the camera frame. Last but not least, ${}^{C}\mathbf{T}_{I}$ is the transformation between the IMU frame, $\{I\}$, and the camera frame. The knowledge of this transformation is very important for the implementation of our the visual-inertial system and was obtained using the *Kalibr* Toolbox from ETH Zurich [2]. All of these matrices can be consulted in appendices B.4 and B.3.



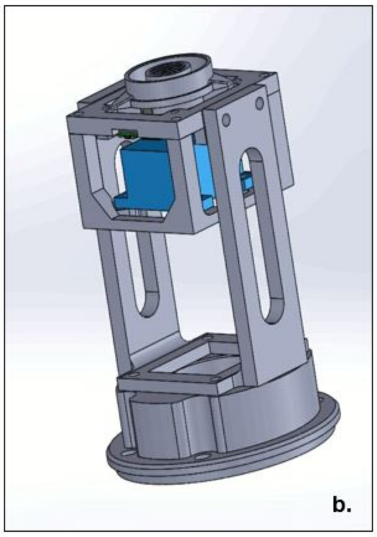


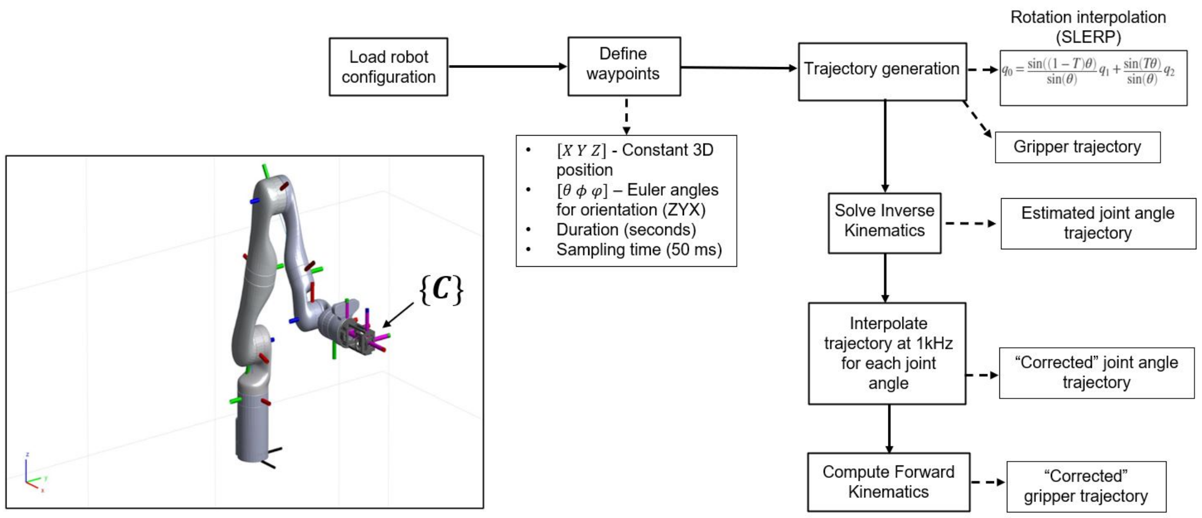
Figure 4.3: **a.** Sensor setup used during the dataset collection. The Kinova image was taken from the datasheet [39]. **b.** 3D model of the custom gripper designed for the Kinova.

4.3.2 Trajectory planning

One of the concerns with this experiment was to record movements that resemble human eye saccades and therefore we were only interested in rotational trajectories in the 3D space (the eye doesn't translate). With this in mind, the next step was to generate 3D rotations that the Kinova arm could perform. To accomplish this, a MATLAB script was developed to define waypoints of the trajectory, perform an interpolation at a constant rate (50 ms) between those points using SLERP (described in section 2.2.4)

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

and solve the numerical inverse kinematics (BFGS Gradient Projection algorithm) in order to obtain the trajectory in the robot joint space. A more detailed diagram of this procedure is illustrated in Figure 4.4, alongside the robot model with the custom grip in its initial configuration.



Rotation interpolation (SLERP)
Load robot configuration
Define waypoints
Trajectory generation
$q_0 = \frac{\sin((1-T)\theta)}{\sin(\theta)} q_1 + \frac{\sin(T\theta)}{\sin(\theta)} q_2$
Gripper trajectory
$[X\ Y\ Z]$ - Constant 3D position
$[\theta\ \phi\ \varphi]$ – Euler angles for orientation (ZYX)
Duration (seconds)
Sampling time (50 ms)
$\{C\}$
Solve Inverse Kinematics
Estimated joint angle trajectory
Interpolate trajectory at 1kHz for each joint angle
"Corrected" joint angle trajectory
Compute Forward Kinematics
"Corrected" gripper trajectory

Figure 4.4: Simplified trajectory planning diagram

All the trajectories planned for the experiments were computed based on the camera frame, using the transformations mentioned in the previous section to describe the kinematics of the system. It's important to point out that, while the trajectory planning is solved at 50 ms, it's necessary to re-interpolate the trajectory to 1 kHz due to software constraints in the Kinova API. This interpolation was performed in each of the 7 joint trajectories using the Akima spline algorithm [40]. Even after this last step, the trajectory has to be verified to guarantee that the algorithm didn't hit any singularity of the robotic arm.

An example of a rotation trajectory around the Z-axis of the camera frame with a maximum amplitude of 20 degrees is shown below in Figure 4.5.

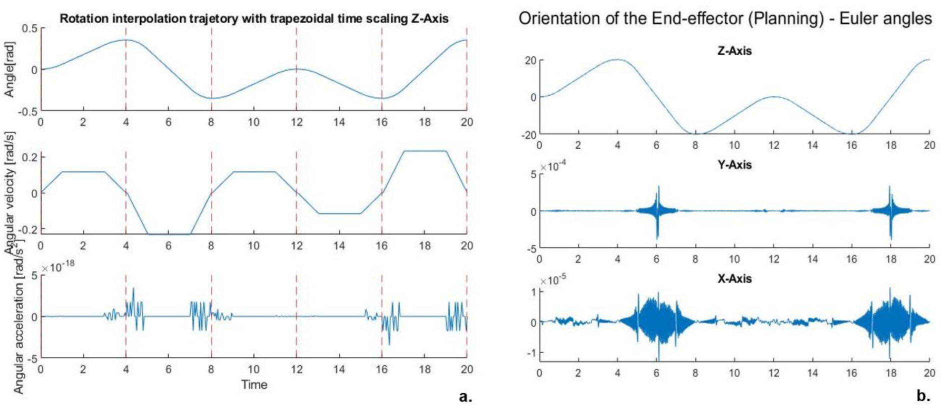


Figure 4.5: Example of trajectory planning. **a.** Result of the SLERP interpolation using a trapezoidal velocity profile. **b.** Orientation of the end-effector (camera frame) in Euler angle notation after the re-interpolation at 1kHz, ready to be sent to the robot.

4.3.3 Dataset format and recording

The data collected comes in the format of a rosbag. It contains the images, IMU measurements and the Kinova joint angles using the standard `sensor_msgs/Image`, `sensor_msgs/Imu` and `sensor_msgs/JointState` message types, respectively. For convenience, we also saved all the .mat files that contain the information about the planning. The pre-computed trajectory is sent to the robot through its MATLAB API [3]. An overview diagram of the recording system is given in Figure 4.6.

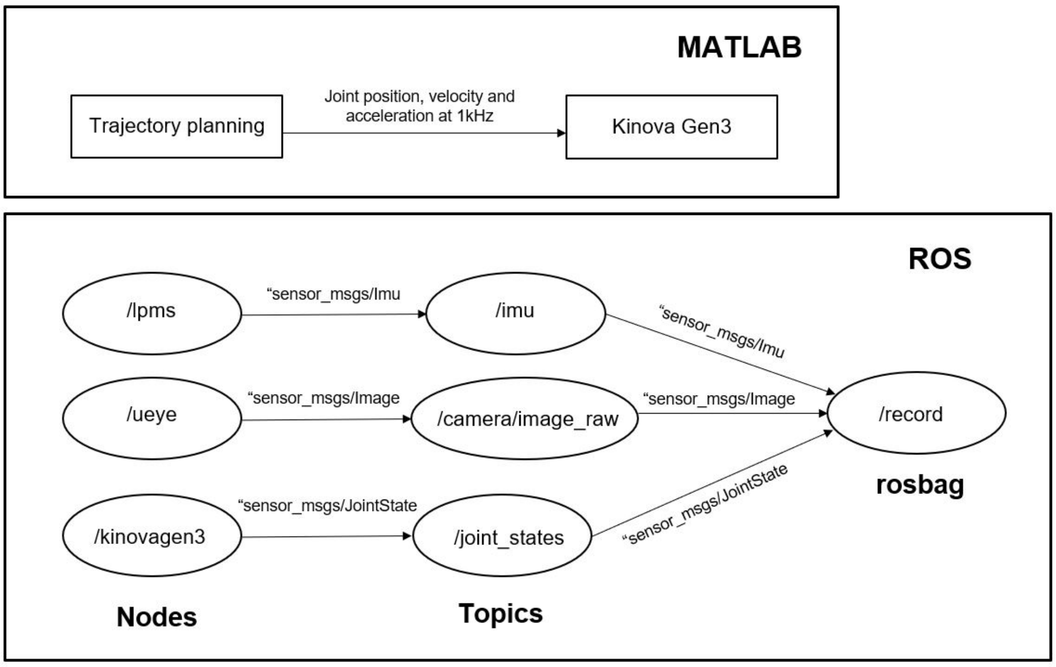


Figure 4.6: Simplified diagram of the implemented system.

The various sensors and their summarized specifications are described in Table 4.1. The measurements collected by the rosbag have the following characteristics:

- **Camera** - intensity frames at rate of 20 Hz.
- **IMU** - inertial measurements (angular velocity in rad/s and linear acceleration in m/s^2) at 100 Hz.
- **Kinova** - joint angle feedback in radians at 10 Hz.

Table 4.1: Overview of sensor characteristics. For more information about each one, refer to appendices B.1, B.2 and B.4

Sensor	Type	Rate	Characteristics
Camera	IDS uEye USB3 UI-3271LE-C-HQ-VU	20 Hz	CMOS sensor, Global shutter, 2056x1542
IMU	LPMS-CU	100 Hz	3-axis gyroscope, 3-axis accelerometer
Kinova	Actuator sensors KA-75+ and KA-58	10 Hz	Position sensor precision at start-up $\pm 1.5^{o}$

[3] `https://github.com/Kinovarobotics/matlab_kortex`

4.3.4 Ground truth

The ground-truth for this dataset was considered to be given by the joint angle feedback of the Kinova, due to the high precision of its actuators. The joint-wise trajectory is then transformed into the camera frame trajectory to give our ground-truth trajectory.

4.3.5 Calibration

Both the camera calibration and the extrinsic calibration between the camera and IMU were done recurring to the open-source toolbox, *Kalibr*. More details about this procedure are found in appendices B.1.2 and B.3.

4.3.6 Limitations

This dataset has a few issues that worth mentioning:

- The vibrations of the actuator joints may throw-off and cause poor readings, specially in the IMU.
- It's assumed that the trajectory planning done in MATLAB matches the movement performed by the Kinova. Specially because we define a location of the principal point that might correspond to the reality.
- There's error in the estimation of the relative pose between the IMU and the camera.
- The ROS timestamps may not be accurate and synchronised.
- The IMU is known to have poor signal-to-noise ration in slow movements.

4.4 Error metrics

To evaluate the performance of the filter, we consider the geodesic distance between two points in $SO(3)$, at time instant k

$$\epsilon_k = || \operatorname{Log}(\hat{\mathbf{R}}_k^T \mathbf{R}_k^{GT}) || \tag{4.1}$$

where $\mathbf{R}_k^{GT}$ is the ground-truth orientation calculated using the feedback joint angles of the Kinova.

Chapter 5

Experiments and results

This chapter contains the various experiments performed (in simulation and real world data) and the respective results. Firstly, we test the performance of the filter in terms of the orientation estimation error for a few trajectories sent to the ESIM simulator. Then, the UKF on Lie groups is tested on real world data acquired using the Kinova robotic arm. An overall discussion of the obtained results is also done in the end. All experiments were performed using the coordinate system convention defined in section 3.1.

5.1 Simulation (ESIM simulator)

The experiments with the ESIM were based on simple rotations around each one of the axes. The camera trajectories sent to the simulator had amplitudes of around 18 degrees. An example of a ground-truth trajectory is shown in Figure 5.1, alongside the initial frame of the rendered scene.

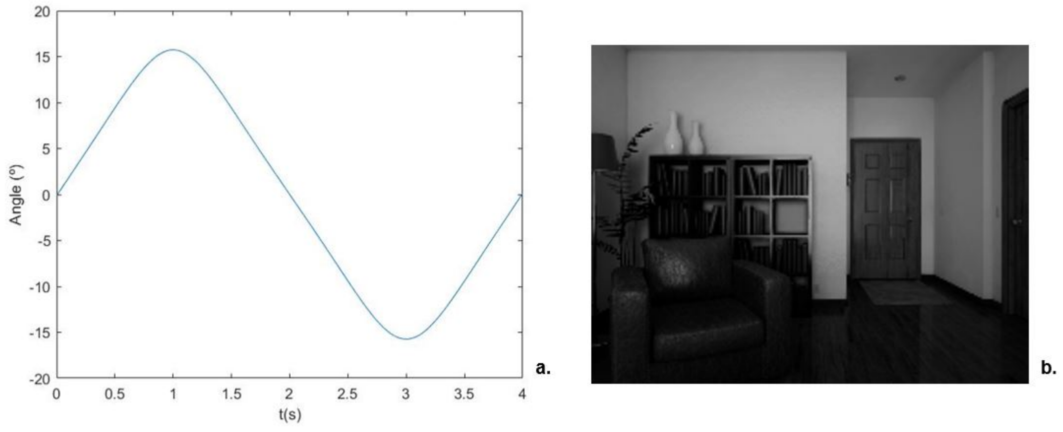


Figure 5.1: (a.) Example of the ground-truth rotational trajectory from the ESIM. (b.) Initial frame of the scene for the dataset.

The returned data (in the format of a .txt file) from the simulator consisted of the inertial measurements and ground-truth sampled at 1kHz, camera frames at around 15 Hz with timestamps, and the intrinsic and distortion parameters of the camera. The initial state of the filter was considered to be the first element of the ground-truth. The measurement noise for one landmark was set to 2 pixels standard deviation and the IMU noises were set to values similar to the ones used in the real experiment and can be found in appendix B.2.1. The plots for each individual rotation (in Euler angles) around the X, Y and Z axis are found in Figures 5.2 and 5.3, respectively.

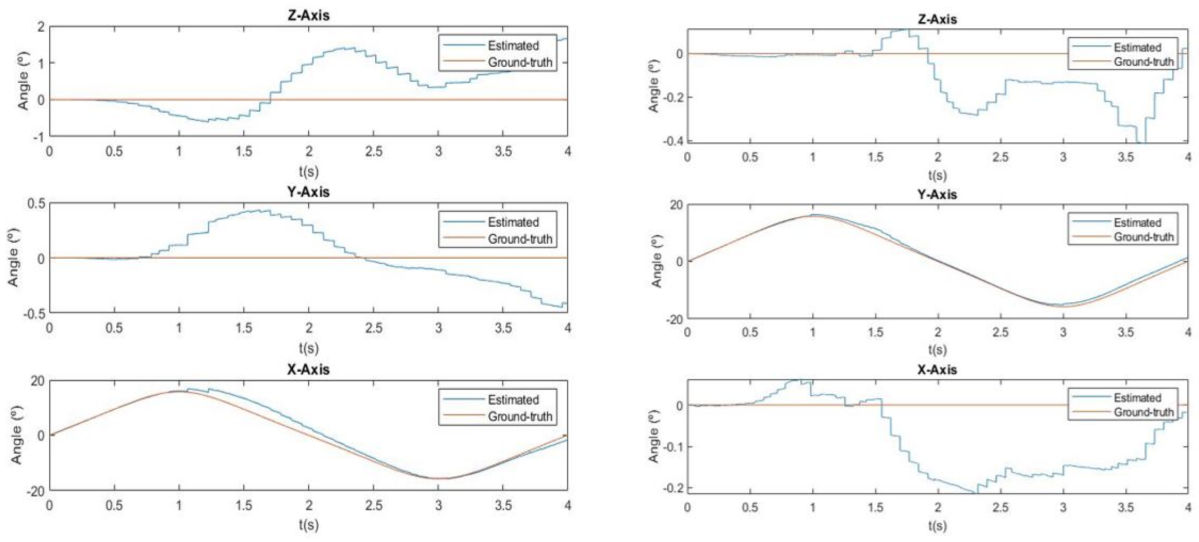


Figure 5.2: Rotation around X (left) and Y (right).

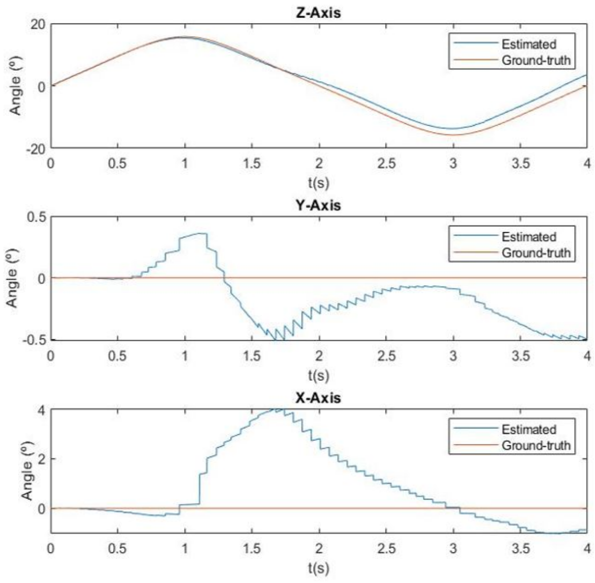


Figure 5.3: Rotation around Z.

The Root Mean Squared Error (RMSE) for the camera rotation was calculated for each trajectory and is represented in Table 5.1.

Axis	X	Y	Z
RMSE (º)	0.5411	0.5590	1.0257

Table 5.1: Root Mean Squared Error in degrees for the rotation estimated by the filter.

From the plots and the table above, we can conclude that the filter is able to keep track of trajectory and have a low error. However, for the axis where supposedly there is no rotation, the estimation seems to drift (it's specially noticeable for the case of the Z axis rotation). This may be due to error accumulation from the IMU, but also from the fact that there's ambiguity when it comes to depth of the landmarks which then introduces errors when computing the predicted measurement mean (prediction of feature position).

One of the main issues found with this simulator is that, for our specific application, the image resolutions were quite low (which affects the tracking of features across the image) and we couldn't test faster movements (that resemble saccades for instance).

5.2 Dataset - Kinova Gen3

For the real world dataset, the procedure was very similar to the one used in the simulation. We generated rotation trajectories for each axis as was explained in section 4.3.2 and sent them to the Kinova. The obvious difference was the rate at which we sampled the data. The IMU captured at 100 Hz, the camera at 20 Hz and the joint angles for the ground-truth were sampled at 10 Hz. The reason for this value to be so slow resides in software constraints in the Kinova API, which restricts the feedback communication depending on the complexity of the commands that it has to execute. The scene used to captured the data is shown in Figure 5.4. We made sure that the scene had good lighting conditions and that there were enough textures to facilitate the tracking of features for the filter.

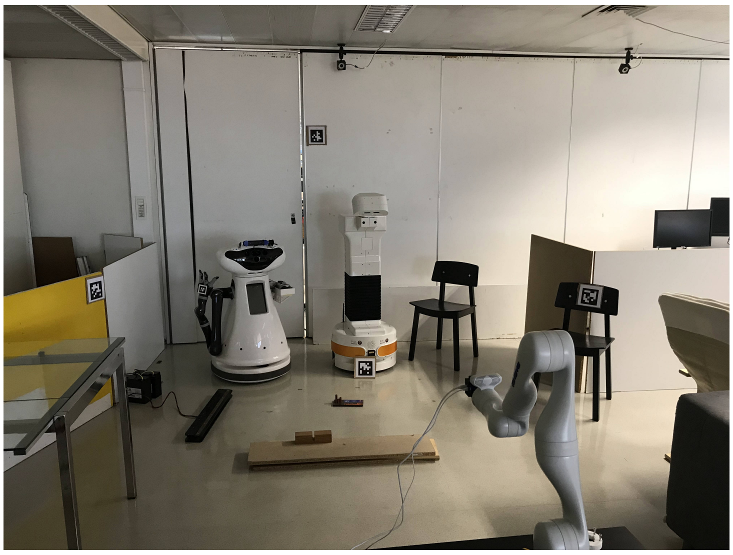


Figure 5.4: Picture of the experiment setup used when recording the Kinova dataset.

As with the ESIM, the filter was initialized to the ground-truth trajectory. The noise parameters were the same for the process and update part, as well as the IMU noises (appendix B.2.1). It was early noticed that the real raw IMU measurements were extremely noisy and contained some outliers that were damaging the estimation. With this in mind, we decided to test the algorithm without, and with a prefiltering of the inertial measurements. We used the built-in MATLAB function that implements a median filter with window sizes of 10 or 15 samples.

We performed experiments for three different amplitudes of rotation (10, 15 and 20 degrees), for each individual axis. For each amplitude, we also tested the estimation against three different joint velocities (given by the trajectory planning). The procedure was repeated to include the cases where there's no prefiltering of the inertial measurements and pre-filtering with a median-filter of window sizes of 10 or 15 samples. Tables 5.2, 5.3 and 5.4 summarize the results of the estimation in terms of the root mean squared error for orientation (difference between the estimated trajectory and the ground-truth provided by the Kinova). The initials MF10 and MF15 stand for "Median Filter size 10" and "Median Filter size 15", respectively. We also plotted the estimated trajectories against the ground-truth for three different experiments, as well as the evolution of the estimation error. They can be found in Figures 5.5, 5.6 and 5.7.

Amplitude	**Max. speed**	No pre-filter	MF10	MF15
10°	9°/s	5.7740°	6.06210	6.4316°
	13°/s	10.3822°	8.9449°	6.6748°
	18°/s	1.1646°	1.4568°	1.3571°
15°	13°/s	8.3587°	10.0005°	6.9050°
	20°/s	21.2131°	12.1310°	21.3327°
	27°/s	13.5712°	5.1181°	7.3425°
20°	18°/s	2.0388°	1.8295°	2.7774°
	21°/s	1.3059°	1.7515°	3.5968°
	27°/s	1.2968°	2.0826°	1.8405°

Table 5.2: Root Mean Squared Error in degrees for the rotation around the X-axis estimated by the filter.

Amplitude	**Max. speed**	No pre-filter	MF10	MF15
10°	9°/s	5.6552°	2.0862°	4.5036°
	11°/s	5.6254°	0.9170°	1.4776°
	13°/s	4.1176°	1.3798°	1.3497°
15°	11°/s	10.1208°	2.2902°	4.2037°
	13°/s	10.9695°	2.9958°	4.6353°
	16°/s	10.7740°	10.3929°	10.2110°
20°	13°/s	7.0344°	3.6176°	2.3387°
	15°/s	18.7596°	3.0261°	3.5340°
	18°/s	9.9611	2.2135	3.0578°

Table 5.3: Root Mean Squared Error in degrees for the rotation around the Y-axis estimated by the filter.

Amplitude	**Max. speed**	No pre-filter	MF10	MF15
10^o	9^o/s	7.0316^o	3.0881^o	4.2551^o
	11^o/s	7.7438^o	2.1107^o	4.4832^o
	13^o/s	7.0317^o	2.5079^o	11.5581^o
15^o	11^o/s	12.9176^o	2.5718^o	2.6900^o
	13^o/s	8.5640^o	3.2799^o	1.6522^o
	16^o/s	19.4217^o	2.6324^o	2.5665^o
20^o	13^o/s	8.7397^o	3.5558^o	4.6629^o
	15^o/s	11.6063^o	2.1810^o	6.0337^o
	18^o/s	24.2505^o	3.6255^o	4.3504^o

Table 5.4: Root Mean Squared Error in degrees for the rotation around the Z-axis estimated by the filter.

The results from the rotation around the X-axis, represented in Table 5.2, show that the best performances were achieved for the case of greater amplitude of movement (20 degrees). For amplitudes of 15 degrees the filter revealed to be quite inconsistent and poor in terms of estimation error. With 10 degrees amplitude, the estimation was also not the best, except for the case where the joint speed was 18^o/s. It's noticeable that the pre-filtering rarely improves the error with the exception of a few cases. It would be expected, since the rotation around the torsonial axis is the simplest one, that the filter would better estimate the orientation, however that is not always the case. With visible landmarks in almost every frame, it's possible that the source of these errors might come from fine tuning in parameters of the filter or bad initial guess of trajectory and biases.

For rotations around the Y and Z axis the results seem to improve sightly, especially in terms of consistency. In these experiments, the effect of pre-filtering the raw gyroscope and accelerometer measurements, has, overall, significant impact on the RMSE. If we increase the size of the window to 15 samples, the estimation starts to worsen, probably because we're filtering out reliable measurements.

Finally, from the inspection of the plots, it's possible to see that, in general, the filter does a good job of tracking the rotation around the "main" axis. A large amount of error comes from the drift around the axis where there is no rotational movement. This drift shows that filter is not able to keep track of the bias correctly, leading to bad estimations.

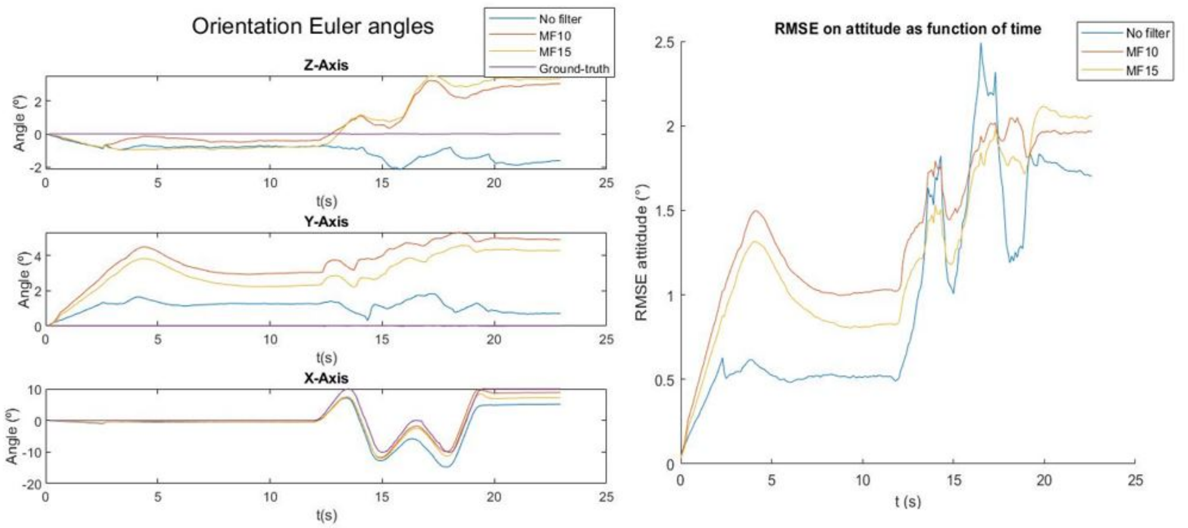


Figure 5.5: Example experiment: Rotation of 10 degrees around the X-axis with maximum joint speed of 18 degrees per second for 3 different scenarios (left). Root mean squared error on orientation as function of time for 3 different scenarios (right).

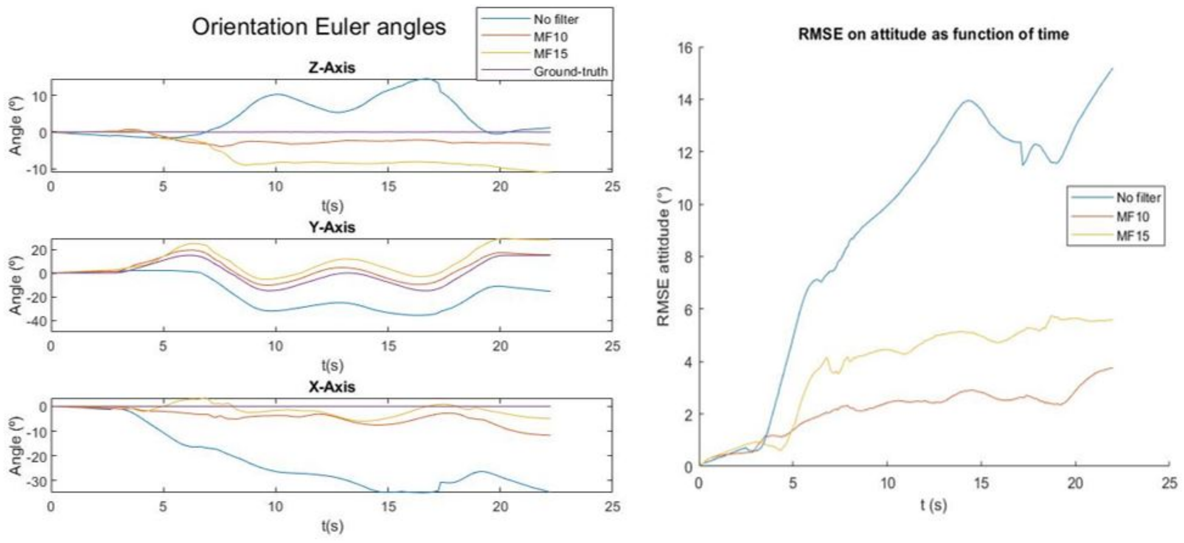


Figure 5.6: Example experiment: Rotation of 15 degrees around the Y-axis with maximum joint speed of 11 degrees per second for 3 different scenarios (left). Root mean squared error on orientation as function of time for 3 different scenarios (right).

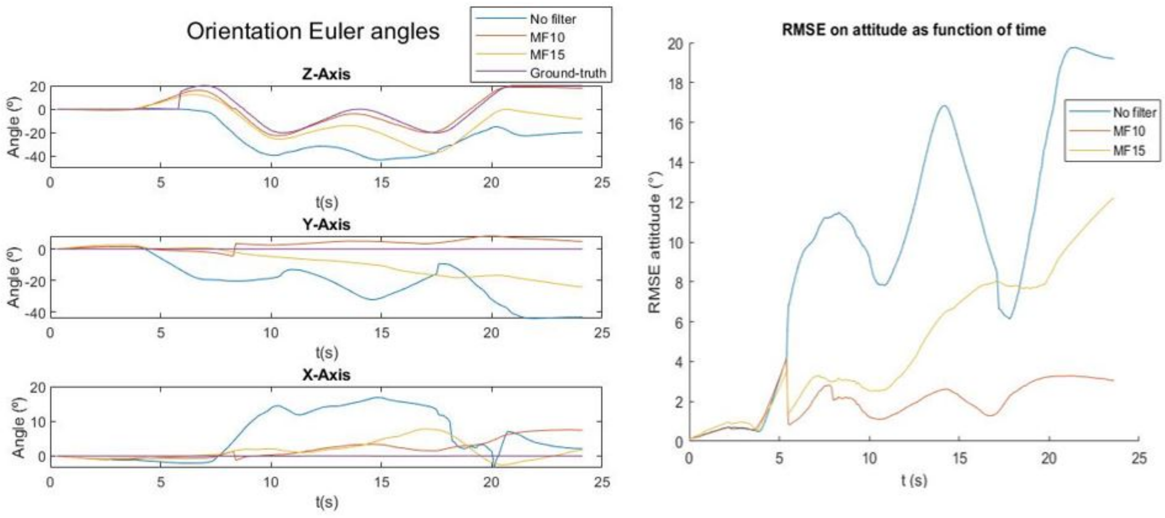


Figure 5.7: Example experiment: Rotation of 20 degrees around the Z-axis with maximum joint speed of 15 degrees per second for 3 different scenarios (left). Root mean squared error on orientation as function of time for 3 different scenarios (right).

Chapter 6

Conclusions

The objective of this thesis was to develop an algorithm that could fuse sensor information from an IMU and a camera to accurately estimate the orientation of a camera that would be embedded in the current eye prototype. The proposed approach was the Unscented Kalman Filter on Lie groups, which can be leveraged in visual-inertial odometry. We found that the filter can indeed be used to track rotational trajectories, both in simulation and in real world scenarios, with reasonably good accuracy. However, there are still quite a few problems, namely in terms of consistency (the filter is sensitive to the tuning of noise parameters of the IMU) and computational efficiency (as of now, the filter is still slow and needs to be optimized and adapted to fit a real-time application). The fact that we were only able to perform slow movements for the benchmarking process, most likely also compromised the consistency of the filter because of the poor signal-to-ratio of the IMU at low frequencies. Faster movements would improve the readings, but would at the same time affect the camera frames. The depth ambiguity can also become a problem since it may cause the filter to diverge.

Overall, this method can definitely be used to estimate orientations for our existent eye prototype.

6.1 Achievements/contributions

The main contributions added to the project are the following:

- Provide an overview on how to deal with visual-inertial odometry problems.
- Full characterization of the sensors used in the prototype for benchmarking purposes.
- Design and development of a real world experiment with reliable ground-truth.
- Implementation in MATLAB of a sensor fusion algorithm that combines inertial measurements with camera frames in a versatile way (UKF-LG).

6.2 Future Work

To finish this work, it's possible to identify several ways to improve or new paths to explore. Unfortunately, this work was not able to include minimization of the back projection error (MBPE) approach studied in [6]. So, a possibility is to try to incorporate the latter in the UKF-LG. However, instead of using a tightly-coupled framework like the one proposed in this work, one could try a loosely-coupled approach, briefly mentioned in the state of the art. To the best of our knowledge there isn't really a dataset for this application of rotation only movements (especially eye saccades). For future work, it would be interesting to develop both a simulator and a real world setup specifically designed to help the problem of eye saccades. A prototype gimbal that could be used in this new benchmarking process is presented in Figure 6.1. As for the simulator, it could also be interesting to develop an ESIM type simulator but for eye movements. The user would define a saccade trajectory and the result would be images with resolution close to what would be expected from the human eye, as well as, IMU measurements and ground-truth.

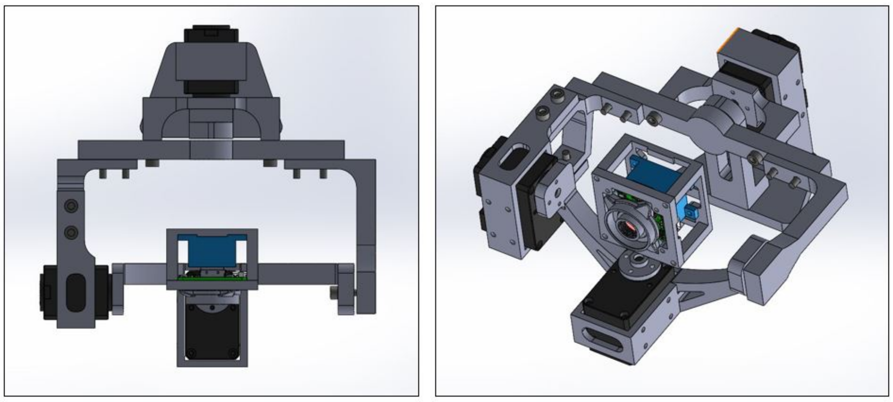


Figure 6.1: 3D model of the prototype gimbal system.

Bibliography

[1] The extraocular muscles. URL https://teachmeanatomy.info/head/organs/eye/extraocular-muscles/.

[2] A. John, C. Aleluia, A. J. Van Opstal, and A. Bernardino. Modelling 3d saccade generation by feedforward optimal control. *PLOS Computational Biology*, 17(5):1–35, 05 2021. doi: 10.1371/journal.pcbi.1008975. URL https://doi.org/10.1371/journal.pcbi.1008975.

[3] B. das Chagas e Silva Colaço Dias. Modeling, simulation, analytic linearization and optimal control of a 6 tendon-driven biomimetic eye: a tool for studying human oculomotor control. Master's thesis, Instituto Superior Técnico, 2020.

[4] C. Aleluia. Control of Saccades with Model of Artificial 3D Biomimetic Eye. Master's thesis, Instituto Superior Técnico, 2019.

[5] R. E. M. Cardoso. Feedback control of saccades on a model of a 3D biomimetic robot eye. Master's thesis, Instituto Superior Técnico, 2019.

[6] M. Martins. Determining the orientation of a RGB camera embedded on an artificial eye. Master's thesis, Instituto Superior Técnico, 2019.

[7] D. Scaramuzza and Z. Zhang. Visual-inertial odometry of aerial robots. *CoRR*, abs/1906.03289, 2019. URL http://arxiv.org/abs/1906.03289.

[8] P. Corke, J. Lobo, and J. Dias. An introduction to inertial and visual sensing. *The International Journal of Robotics*, 26:519–535, 2007.

[9] S. Lynen, M. W. Achtelik, S. Weiss, M. Chli, and R. Siegwart. A robust and modular multi-sensor fusion approach applied to mav navigation. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3923–3929, 2013. doi: 10.1109/IROS.2013.6696917.

[10] L. Kneip, M. Chli, and R. Y. Siegwart. Robust real-time visual odometry with a single camera and an imu. In *BMVC*, 2011.

[11] M. Li and A. I. Mourikis. High-precision, consistent ekf-based visual-inertial odometry. *Int. J. Rob. Res.*, 32(6):690–711, May 2013. ISSN 0278-3649. doi: 10.1177/0278364913481251. URL https://doi.org/10.1177/0278364913481251.

[12] K. Eckenhoff, Y. Yang, P. Geneva, and G. Huang. Tightly-coupled visual-inertial localization and 3-d rigid-body target tracking. *IEEE Robotics and Automation Letters*, 4:1541–1548, 2019.

[13] K. Hepp. On listing's law. *Communications in Mathematical Physics*, 132(1):285–292, 1990.

[14] A. M. Wong. Listing's law: clinical significance and implications for neural control. *Survey of Ophthalmology*, 49(6):563–575, 2004. ISSN 0039-6257. doi: https://doi.org/10.1016/j.survophthal.2004.08.002. URL `https://www.sciencedirect.com/science/article/pii/S0039625704001341`.

[15] J. Van Opstal. *The auditory system and human sound-localization behavior*. Academic Press, 2016.

[16] R. W. Baloh, A. W. Sills, W. E. Kumley, and V. Honrubia. Quantitative measurement of saccade amplitude, duration, and velocity. *Neurology*, 25(11):1065–1065, 1975.

[17] H. P. Goldstein. The neural encoding of saccades in the rhesus monkey. 1984.

[18] T. D. Barfoot. *State Estimation for Robotics*. Cambridge University Press, 2017. doi: 10.1017/9781316671528.

[19] M. Kok, J. D. Hol, and T. B. Schön. Using inertial sensors for position and orientation estimation. *CoRR*, abs/1704.06053, 2017. URL `http://arxiv.org/abs/1704.06053`.

[20] R. Szeliski. *Computer Vision*. Springer London, 2011. doi: 10.1007/978-1-84882-935-0. URL `https://doi.org/10.1007%2F978-1-84882-935-0`.

[21] J. Diebel. Representing attitude: Euler angles, unit quaternions, and rotation vectors. *Matrix*, 58, 01 2006.

[22] E. B. Dam, M. Koch, and M. Lillholm. *Quaternions, interpolation and animation*, volume 2. Citeseer, 1998.

[23] K. Shoemake. Animating rotation with quaternion curves. In *Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pages 245–254, 1985.

[24] S. J. Julier and J. K. Uhlmann. New extension of the kalman filter to nonlinear systems. In *Signal processing, sensor fusion, and target recognition VI*, volume 3068, pages 182–193. International Society for Optics and Photonics, 1997.

[25] E. A. Wan and R. Van Der Merwe. The unscented kalman filter for nonlinear estimation. In *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No.00EX373)*, pages 153–158, 2000. doi: 10.1109/ASSPCC.2000.882463.

[26] S. Traversaro and A. Saccon. *Multibody dynamics notation (version 2)*. Technische Universiteit Eindhoven, Nov. 2019. Dept. of Mechanical Engineering. Report locator DC 2019.100.

[27] C. Forster, L. Carlone, F. Dellaert, and D. Scaramuzza. On-Manifold Preintegration for Real-Time Visual–Inertial Odometry. *IEEE Transactions on Robotics*, 33(1):1–21, 2017. doi: 10.1109/TRO.2016.2597321.

[28] R. M. Murray, S. S. Sastry, and L. Zexiang. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, Inc., USA, 1st edition, 1994. ISBN 0849379814.

[29] A. Barrau and S. Bonnabel. An EKF-SLAM algorithm with consistency properties. *CoRR*, abs/1510.06263, 2015. URL `http://arxiv.org/abs/1510.06263`.

[30] M. Brossard, S. Bonnabel, and A. Barrau. Unscented Kalman Filter on Lie Groups for Visual Inertial Odometry. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 2018*, Madrid, Spain, Oct. 2018. URL `https://hal.archives-ouvertes.fr/hal-01735542`.

[31] M. Brossard, S. Bonnabel, and A. Barrau. Invariant Kalman Filtering for Visual Inertial SLAM. In *21st International Conference on Information Fusion*, 21st International Conference on Information Fusion, Cambrige, United Kingdom, July 2018. University of Cambridge. URL `https://hal.archives-ouvertes.fr/hal-01588669`.

[32] E. Salahat and M. Qasaimeh. Recent advances in features extraction and description algorithms: A comprehensive survey. *CoRR*, abs/1703.06376, 2017. URL `http://arxiv.org/abs/1703.06376`.

[33] C. G. Harris and M. J. Stephens. A combined corner and edge detector. In *Alvey Vision Conference*, 1988.

[34] G. Loianno, M. Watterson, and V. Kumar. Visual inertial odometry for quadrotors on SE(3). In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1544–1551, 2016. doi: 10.1109/ICRA.2016.7487292.

[35] M. Brossard, A. Barrau, and S. Bonnabel. A Code for Unscented Kalman Filtering on Manifolds (UKF-M). In *International Conference on Robotics and Automation (ICRA)*, Paris, France, May 2020. URL `https://hal.archives-ouvertes.fr/hal-02463553`.

[36] R. Van der Merwe and E. Wan. The square-root unscented kalman filter for state and parameter-estimation. In *2001 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No.01CH37221)*, volume 6, pages 3461–3464 vol.6, 2001. doi: 10.1109/ICASSP.2001.940586.

[37] H. Rebecq, D. Gehrig, and D. Scaramuzza. ESIM: an open event camera simulator. *Conf. on Robotics Learning (CoRL)*, Oct. 2018.

[38] J. Shi and Tomasi. Good features to track. In *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 593–600, 1994. doi: 10.1109/CVPR.1994.323794.

[39] *KINOVA Gen3 Ultra lightweight robot*. Kinova Inc. URL `https://artifactory.kinovaapps.com/ui/api/v1/download?repoKey=generic-documentation-public&path=Documentation%2FGen3%2FTechnical%20documentation%2FUser%20Guide%2FUser%20Guide%20Gen3%20-%20R06.pdf`.

[40] H. Akima. A new method of interpolation and smooth curve fitting based on local procedures. *J. ACM*, 17(4):589–602, Oct. 1970. ISSN 0004-5411. doi: 10.1145/321607.321609. URL `https://doi.org/10.1145/321607.321609`.

[41] P. Furgale, J. Rehder, and R. Siegwart. Unified temporal and spatial calibration for multi-sensor systems. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1280–1286, 2013. doi: 10.1109/IROS.2013.6696514.

Appendix A

Square-root Unscented Kalman filter on Lie groups

A.1 Initialization

- Initialization of orientation, velocity and position with the ground-truth.
- IMU bias initialization at zero.
- Initialization of landmarks done by detecting the first set of points in the first image and setting the depth at $z = 1$.

The initial covariance matrix, $\mathbf{P_0}$, associated with the state, $\chi_\mathbf{0}$, is given by equation A.1, where each element of the diagonal is also a 3×3 diagonal matrix containing the variances of the state variables. This uncertainty can be fine tuned depending on how well the initial state is known.

$$\mathbf{P_0} = \begin{bmatrix} \mathbf{P_R} & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & \mathbf{P_v} & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & \mathbf{P_o} & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{P_{p_1}} & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \mathbf{P_{p_\mathit{N}}} & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & \mathbf{P_{b_g}} & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & \mathbf{P_{b_a}} \end{bmatrix} \in \mathbb{R}^{(15+3p)\times(15+3p)} \tag{A.1}$$

$$\hat{\mathbf{x}}_0 = \left(\hat{\chi}_0, \hat{\mathbf{b}}_0\right), \quad \hat{\mathbf{P}}_0 \rightarrow \hat{\mathbf{S}}_0 \text{ (Cholesky decomposition)} \tag{A.2}$$

A.2 Propagation of the state

$$\begin{aligned} &\hat{\mathbf{x}}_{k-1} = \left(\hat{\mathcal{X}}_{k-1}, \hat{\mathbf{b}}_{k-1}\right), \quad \hat{\mathbf{P}}_{k-1} \rightarrow \hat{\mathbf{S}}_{k-1} \text{ (Cholesky decomposition)} \\ &\mathbf{u}_k = \begin{bmatrix} \tilde{\boldsymbol{\omega}}_k \\ \tilde{\mathbf{a}}_k \end{bmatrix} \quad \text{(IMU measurements)} \end{aligned} \tag{A.3}$$

Propagate noiseless mean (Predicted belief).

$$\check{\mathbf{x}}_k = f\left(\hat{\mathbf{x}}_{k-1}, \mathbf{u}_k, \mathbf{0}\right) = \begin{cases} \check{\mathbf{R}}_k = \hat{\mathbf{R}}_{k-1} \operatorname{Exp}\left(\left(\tilde{\boldsymbol{\omega}}_k - \hat{\mathbf{b}}^{\mathbf{g}}_{k-1}\right) \Delta t\right) \\ \check{\mathbf{v}}_k = \hat{\mathbf{v}}_{k-1} + \left(\hat{\mathbf{R}}_{k-1}\left(\tilde{\mathbf{a}}_k - \hat{\mathbf{b}}^{\mathbf{a}}_{k-1}\right) + \mathbf{g}\right) \Delta t \\ \check{\mathbf{o}}_k = \hat{\mathbf{o}}_{k-1} + \hat{\mathbf{v}}_{k-1}\Delta t + \frac{1}{2}\left(\hat{\mathbf{R}}_{k-1}\left(\tilde{\mathbf{a}}_k - \hat{\mathbf{b}}^{\mathbf{a}}_{k-1}\right) + \mathbf{g}\right) \Delta t^2 \\ \check{\mathbf{b}}^{\mathbf{g}}_k = \hat{\mathbf{b}}^{\mathbf{g}}_{k-1} \\ \check{\mathbf{b}}^{\mathbf{a}}_k = \hat{\mathbf{b}}^{\mathbf{a}}_{k-1} \end{cases} \tag{A.4}$$

$$\check{\mathbf{x}}_k = \left(\check{\mathcal{X}}_k, \check{\mathbf{b}}_k\right) \tag{A.5}$$

Generate sigma points. First augment the covariance matrix to include the process noise. The covariance of the state has size $L = 15 + 3p$ and the noise covariance has size $M = 12$. The total size of the augmented state covariance is $L^A = L + M$.

$$\begin{aligned} &\mathbf{Q} \rightarrow \mathbf{S}_Q, \quad \text{(Cholesky decomposition)} \\ &\hat{\mathbf{S}}^A_{k-1} = \begin{bmatrix} \hat{\mathbf{S}}_{k-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{S}_Q \end{bmatrix} \in \mathbb{R}^{L^A \times L^A} \end{aligned} \tag{A.6}$$

Scale parameters: $\lambda = \left(\alpha^2 - 1\right) L^A$, $\gamma = \sqrt{L^A + \lambda}$, $\alpha = 1 \times 10^{-3}$ (usual value according to [25]) and $\beta = 2$. Compute weights according to equation 2.29.

$$\begin{aligned} &\boldsymbol{\mathcal{X}}_{k,i} = \gamma \operatorname{col}_i\left(\hat{\mathbf{S}}^A_{k-1}\right), \qquad \text{for } i = 1, ..., L^A \\ &\boldsymbol{\mathcal{X}}_{k,i} = -\gamma \operatorname{col}_i\left(\hat{\mathbf{S}}^A_{k-1}\right), \qquad \text{for } i = 1 + L^A, ..., 2L^A \end{aligned} \tag{A.7}$$

$$\boldsymbol{\mathcal{X}}_{k,i} = \pm\left[\boldsymbol{\xi}^{\mathcal{X}}_{k,i} \ \boldsymbol{\xi}^b_{k,i} \ \mathbf{w}_{k,i}\right], \quad i = 1, \ldots, L^A \tag{A.8}$$

Sigma point propagation:

$$\mathbf{u}_k \leftarrow \mathbf{u}_k - \hat{\mathbf{b}}_{k-1} \tag{A.9}$$

The sigma points, $\boldsymbol{\xi}^{\mathcal{X}}_i$, are first mapped to the Lie group and then propagated through the system model, f. Omitting time instant k:

$$\begin{cases} \check{\boldsymbol{\xi}}^b_i = \boldsymbol{\xi}^b_i + \Delta t \mathbf{w}^b_i \\ \check{\mathcal{X}}^{\mathcal{X}}_i = f\left(\hat{\mathcal{X}}_{k-1} \operatorname{Exp}\left(\boldsymbol{\xi}^{\mathcal{X}}_i\right), \mathbf{u}_k - \check{\boldsymbol{\xi}}^b_i, \mathbf{w}_i\right) \end{cases}, \quad i = 1, \ldots, 2L^A \tag{A.10}$$

Recollect propagated sigma points in the original space.

$$\check{\boldsymbol{\xi}}_i^{\mathcal{X}} = \operatorname{Log}\left(\left(\hat{\boldsymbol{\chi}}_{k-1}\right)^{-1} \check{\mathcal{X}}_i^{\mathcal{X}}\right) \tag{A.11}$$

Recalculate the Cholesky factor for the propagated covariance, $\check{\mathbf{S}}_k$.

$$\check{\mathbf{S}}_k \leftarrow \mathcal{QR}\left(\sqrt{W_i^{(c)}}\begin{bmatrix} \check{\boldsymbol{\xi}}_1^{\mathcal{X}} & \cdots & \check{\boldsymbol{\xi}}_{L^A}^{\mathcal{X}} & -\check{\boldsymbol{\xi}}_1^{\mathcal{X}} & \cdots & -\check{\boldsymbol{\xi}}_{L^A}^{\mathcal{X}} \\ \check{\boldsymbol{\xi}}_1^{b} & \cdots & \check{\boldsymbol{\xi}}_{L^A}^{b} & -\check{\boldsymbol{\xi}}_1^{b} & \cdots & -\check{\boldsymbol{\xi}}_{L^A}^{b} \\ \mathbf{0} & & \mathbf{S}_Q & \mathbf{0} & & -\mathbf{S}_Q \end{bmatrix}\right) \tag{A.12}$$

A.3 State update

Observation model

$$\mathbf{z}_k^i = h\left(\boldsymbol{\chi}_k\right) + \mathbf{n}_z^i, \quad i = 1, \ldots, p \tag{A.13}$$

$$\begin{bmatrix} z_u^i \\ z_v^i \\ z_w^i \end{bmatrix} = \boldsymbol{K}\left[{}^{C}\mathbf{R}_I\left(\mathbf{R}_k^T \mathbf{p}_i + \mathbf{o}_k\right) + {}^{C}\mathbf{o}_I\right] \tag{A.14}$$

Compact version

$$\mathbf{Z}_k = \begin{bmatrix} \mathbf{z}_k^1 \\ \vdots \\ \mathbf{z}_k^p \end{bmatrix} := H\left(\boldsymbol{\chi}_k, \mathbf{n}_z\right) \tag{A.15}$$

$$\begin{aligned} &\check{\mathbf{x}}_k = \left(\check{\boldsymbol{\chi}}_k, \check{\mathbf{b}}_k\right), \quad \check{\mathbf{S}}_k \rightarrow \textbf{Predicted belief} \\ &\mathbf{Z}_k = \begin{bmatrix} \mathbf{z}_k^1 \\ \vdots \\ \mathbf{z}_k^p \end{bmatrix} \quad \text{(Camera observation of the } p \text{ landmarks)} \rightarrow \text{Actual measurements} \end{aligned} \tag{A.16}$$

where each landmark observation $\mathbf{z}_k^i$ is given by the model in equation 3.9.

Create sigma points. Firstly, pass the propagated mean without noise through the observation model. The other measurement sigma points are computed separately.

$$\check{\boldsymbol{\mathcal{Z}}}_0 = H\left(\check{\boldsymbol{\chi}}_k, \mathbf{0}\right) \tag{A.17}$$

Augment the covariance matrix to include the observation noise. The covariance of the state has size $L = 15 + 3p$ and the noise covariance has size M. The total size of the augmented state covariance is $L^A = L + M$. Note that the observation covariance matrix, $\mathbf{R}$, represents the pixel image noise for one landmark.

$$\begin{aligned}
&\mathbf{R} \to \mathbf{S}_R, \quad \text{(Cholesky decomposition)} \\
&\check{\mathbf{S}}_k^A = \begin{bmatrix} \check{\mathbf{S}}_k & \mathbf{0} \\ \mathbf{0} & \mathbf{S}_R \end{bmatrix} \in \mathbb{R}^{L^A \times L^A}
\end{aligned} \tag{A.18}$$

Scale parameters: $\lambda = \left(\alpha^2 - 1\right) L^A$, $\gamma = \sqrt{L^A + \lambda}$, $\alpha = 1 \times 10^{-3}$ (usual value according to [25]) and $\beta = 2$. Compute weights according to equation 2.29.

$$\begin{aligned}
\boldsymbol{\mathcal{X}}_{k,i} &= \gamma \, \text{col}_i \left(\check{\mathbf{S}}_k^A\right), \qquad \text{for } i = 1, ..., L^A \\
\boldsymbol{\mathcal{X}}_{k,i} &= -\gamma \, \text{col}_i \left(\check{\mathbf{S}}_k^A\right), \qquad \text{for } i = 1 + L^A, ..., 2L^A
\end{aligned} \tag{A.19}$$

$$\boldsymbol{\mathcal{X}}_{k,i} = \pm \left[\boldsymbol{\xi}_{k,i}^{\chi} \; \boldsymbol{\xi}_{k,i}^{b} \; \mathbf{n}_{k,i}\right], \quad i = 1, \ldots, L^A \tag{A.20}$$

Sigma point propagation:

$$\left\{ \begin{aligned} &\boldsymbol{\chi}_i{}^{\pm} = \check{\boldsymbol{\chi}}_k \operatorname{Exp}\left(\boldsymbol{\xi}_i^{\chi\pm}\right) \\ &\check{\boldsymbol{\mathcal{Z}}}_i = H\left(\boldsymbol{\chi}_i{}^{\pm}, \pm\mathbf{n}_i\right) \end{aligned} \right. , \quad i = 1, \ldots, 2L^A \tag{A.21}$$

Compute the predicted measurement mean and weighted deviations:

$$\begin{aligned}
\check{\mathbf{Z}}_k &= W_0^{(c)} \check{\boldsymbol{\mathcal{Z}}}_0 + \sum_{i=1}^{L^A} W_i^{(m)} \left(\check{\boldsymbol{\mathcal{Z}}}_i^+ + \check{\boldsymbol{\mathcal{Z}}}_i^-\right) \\
\mathbf{e}_0 &= \sqrt{|W_0^{(c)}|} \left(\check{\boldsymbol{\mathcal{Z}}}_0 - \check{\mathbf{Z}}_k\right) \\
\mathbf{e}_i^{\pm} &= \sqrt{W_i^{(c)}} \left(\check{\boldsymbol{\mathcal{Z}}}_i^{\pm} - \check{\mathbf{Z}}_k\right), \quad i = 1, \ldots, L^A
\end{aligned} \tag{A.22}$$

Compute the Cholesky factors of the measurement covariance, $\mathbf{P}_{zz}$, and the cross covariance, $\mathbf{P}_{xz}$.

$$\begin{aligned}
&\mathbf{S}_{zz}^{'} \leftarrow \mathcal{QR}\left(\begin{bmatrix}\mathbf{e}_1^+ & \cdots & \mathbf{e}_{L^A}^+ & -\mathbf{e}_1^- & \cdots & \mathbf{e}_{L^A}^- & \mathbf{S}_R^{'}\end{bmatrix}\right) \\
&\mathbf{S}_{zz} \leftarrow \texttt{CholUpdate}\left(\mathbf{S}_{zz}^{'}, \, \mathbf{e}_0, \, \text{sign}\left(W_0^{(c)}\right)\right) \\
&\mathbf{P}_{zz} = \mathbf{S}_{zz}^T \mathbf{S}_{zz} \\
&\mathbf{P}_{xz} = \sum_{i=1}^{L^A} \sqrt{W_i^{(c)}} \left(\begin{bmatrix} \boldsymbol{\xi}_i^+ \\ \mathbf{b}_i^+ \end{bmatrix}^T \mathbf{e}_i^+ + \begin{bmatrix} \boldsymbol{\xi}_i^- \\ \mathbf{b}_i^- \end{bmatrix}^T \mathbf{e}_i^-\right)
\end{aligned} \tag{A.23}$$

Using the current measurement vector, $\mathbf{Z}_k$, and the predicted belief, compute the Kalman gain and the innovation.

$$\begin{aligned}
&\boldsymbol{\mathcal{K}} = \mathbf{P}_{xz} \left(\mathbf{S}_{zz}^T \mathbf{S}_{zz}\right)^{-1} \to \text{Kalman gain} \\
&\delta\bar{\boldsymbol{\xi}} = \begin{bmatrix} \delta\bar{\boldsymbol{\xi}}^{\chi} \\ \delta\bar{\boldsymbol{\xi}}^{b} \end{bmatrix} = \boldsymbol{\mathcal{K}} \left(\mathbf{Z}_k - \check{\mathbf{Z}}_k\right) \to \text{Innovation}
\end{aligned} \tag{A.24}$$

Finally, update the state mean and covariance (posterior belief).

$$\begin{cases} \hat{\boldsymbol{\mathcal{X}}}_k = \check{\boldsymbol{\mathcal{X}}}_k \operatorname{Exp}\left(\delta\bar{\boldsymbol{\xi}}^{\mathcal{X}}\right) \\ \hat{\mathbf{b}}_k = \check{\mathbf{b}}_k + \delta\bar{\boldsymbol{\xi}}^{b} \end{cases} \tag{A.25}$$

$$\begin{aligned} &\hat{\mathbf{S}}_k^{'} \leftarrow \texttt{SeqCholUpdate}\left(\check{\mathbf{S}}_k,\ \boldsymbol{\mathcal{K}}\mathbf{S}_{zz}^{T},\ -1\right) \\ &\hat{\mathbf{S}}_k = \hat{\mathbf{S}}_k^{'}\mathbf{J} \end{aligned} \tag{A.26}$$

$$\hat{\mathbf{x}}_k = \left(\hat{\boldsymbol{\mathcal{X}}}_k, \hat{\mathbf{b}}_k\right), \quad \hat{\mathbf{S}}_k \rightarrow \textbf{Posterior belief} \tag{A.27}$$

Appendix B

Hardware

B.1 Camera: *uEye LE USB3*

The camera used in the project is an uEye USB3 from Imaging Development Systems GmbH. It's a PCB camera used commonly for medical and industrial applications and has a S-Mount for the lens B.1.1. The basic datasheet is represented in Figure B.1. For more information about the technical data, refer to the suppliers page [1].

B.1.1 Lens - Lensation BM4018S118C

The specifications of the lens can be found in the datasheet below in Figure B.2.

B.1.2 Calibration

The camera calibration was done using the *Kalibr* Toolbox from ETH Zurich [2] with an Aprilgrid with the following dimensions (represented in Figure B.3):

- Tag columns: 6
- Tag rows: 6
- Tag size: 0.055 [m] (size of the apriltag, edge to edge)
- Tag spacing: 0.3 (ratio of space between tags to tag size)

[1] https://en.ids-imaging.com/store/ui-3271le.html
[2] https://github.com/ethz-asl/kalibr

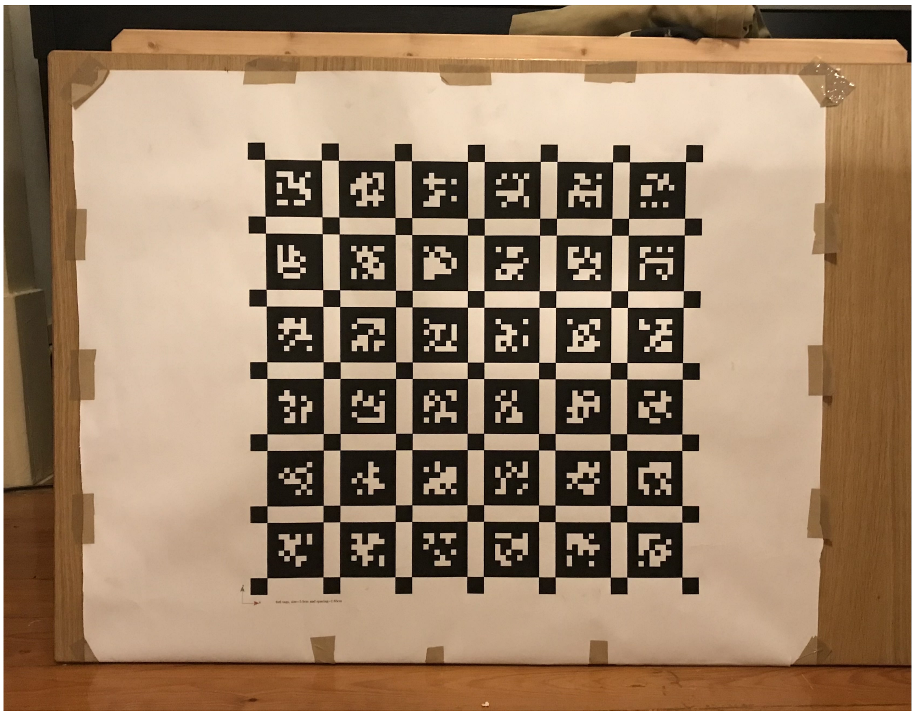


Figure B.3: Aprilgrid calibration target.

The estimated intrinsic parameters are:

$$\boldsymbol{K} = \begin{bmatrix} 1129.723097 & 0 & 978.656074 \\ 0 & 1130.712255 & 773.305876 \\ 0 & 0 & 1 \end{bmatrix} \tag{B.1}$$

Since the ideal camera model doesn't include a lens, to accurately represent a real camera, it's necessary to include the radial and tangential distortion of the lens. These parameters are also estimated and are given by

$$\begin{aligned} k_1, k_2 &= [-0.269803 \quad 0.068608] \\ p_1, p_2 &= [0.000538 \quad 0.000525] \end{aligned} \tag{B.2}$$

B.2 IMU: *LPMS-CU*

The IMU sensor used in this project is the LPMS-CU developed by Life Performance Research [3]. Its specifications can be consulted in Figure B.4.

B.2.1 IMU noise model parameters estimation

The Allan Standard Deviation (ASD) plots obtained for the gyroscope and accelerometer are represented in Figures B.5(a.) and B.5(b.), respectively. They were obtained by recording a rosbag of the IMU

[3] `https://lp-research.com/`

(standing still) measurements for around 4 hours. Then, the Allan deviations were plotted and interpreted as suggested by *Kalibr* [41].

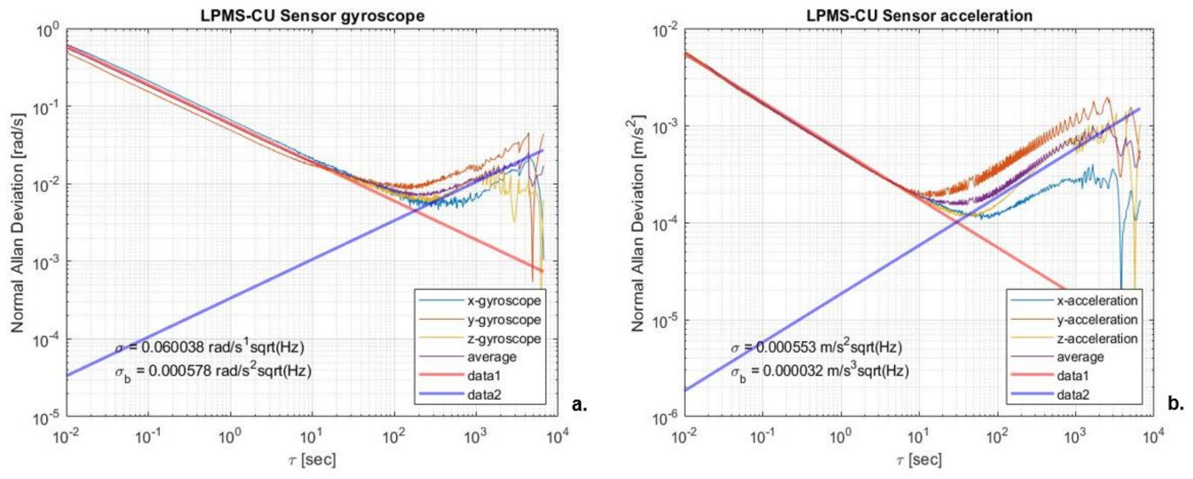


Figure B.5: Allan Standard Deviation for both gyroscope (a.) and accelerometer (b.) for the LPMS-CU sensor determined by *Kalibr* [41].

B.3 IMU-Camera extrinsic calibration

To calibrate the extrinsics, i.e, estimate the relative pose between the IMU and camera, we used once again the open source software *Kalibr* [41]. The procedure implies that both camera and IMU are individually calibrated and after that we have collected a dynamic dataset by exciting all axes of the IMU (in rotation and acceleration). The calibration process resulted in the following transformation from the IMU fram to the camera frame:

$$
{}^{C}\mathbf{T}_{I} = \begin{bmatrix} 0.02380328 & 0.99967821 & 0.00876872 & -0.02115072 \\ -0.9996539 & 0.02370258 & 0.01141338 & -0.00470428 \\ 0.01120186 & -0.00903736 & 0.99989642 & 0.00017564 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{B.3}
$$

where the rotation matrix encodes the difference of orientation between the IMU and the camera frames and the translation of approximately 2 cm along the optical axis of the camera. This translation can be easily confirmed because the IMU is right behind the camera lens (Figure B.6).

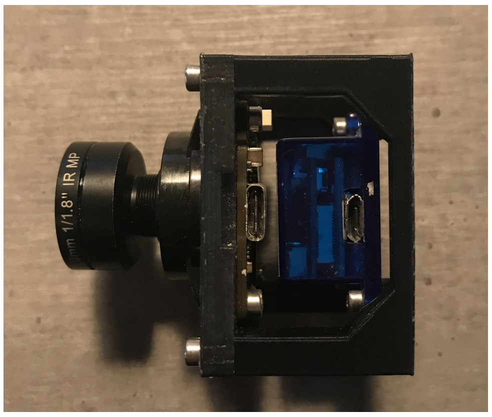


Figure B.6: Visual-inertial sensor configuration used.

B.4 Kinova Gen3

The specifications of the Kinova Gen3 robotic arm used to record the real world dataset can be found in [39]. The homogeneous transformation matrix in the initial configuration, ${}^{E}\mathbf{T}_{B}$, and the pose of the end-effector frame relative to the camera frame, ${}^{C}\mathbf{T}_{E}$, are specified in B.4.

$$\begin{aligned} {}^{E}\mathbf{T}_{B} &= \begin{bmatrix} 0 & 0.0124 & 0.9999 & 0.6288 \\ 1 & 0 & 0 & 0.0014 \\ 0 & 0.9999 & -0.0124 & 0.3559 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ {}^{C}\mathbf{T}_{E} &= \begin{bmatrix} 0 & 0 & 1 & -0.0970 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{B.4}$$

In Figure B.7 there's also a table with the technical specifications of the joints actuators.

UI-3271LE-C-HQ-VU (AB02201)

Specification

Sensor	
Sensor type	CMOS Color
Shutter	Global Shutter
Sensor characteristic	Linear
Readout mode	Progressive scan
Pixel Class	3 MP
Resolution	3.17 Mpix
Resolution (h x v)	2056 x 1542 Pixel
Aspect ratio	4:3
ADC	12 bit
Color depth (camera)	12 bit
Optical sensor class	1/1.8"
Optical Size	7.093 mm x 5.320 mm
Optical sensor diagonal	8.87 mm (1/1.8")
Pixel size	3.45 µm
Manufacturer	Sony
Sensor Model	IMX265LQR-C
Gain (master/RGB)	24x/4x
AOI horizontal	same frame rate
AOI vertical	increased frame rate
AOI image width / step width	256 / 8
AOI image height / step width	2 / 2
AOI position grid (horizontal/vertical)	4 / 2
Binning horizontal	-
Binning vertical	-
Binning method	-
Binning factor	-
Subsampling horizontal	same frame rate
Subsampling vertical	increased frame rate
Subsampling method	M/C automatic
Subsampling factor	2

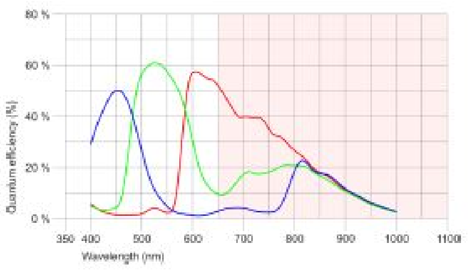
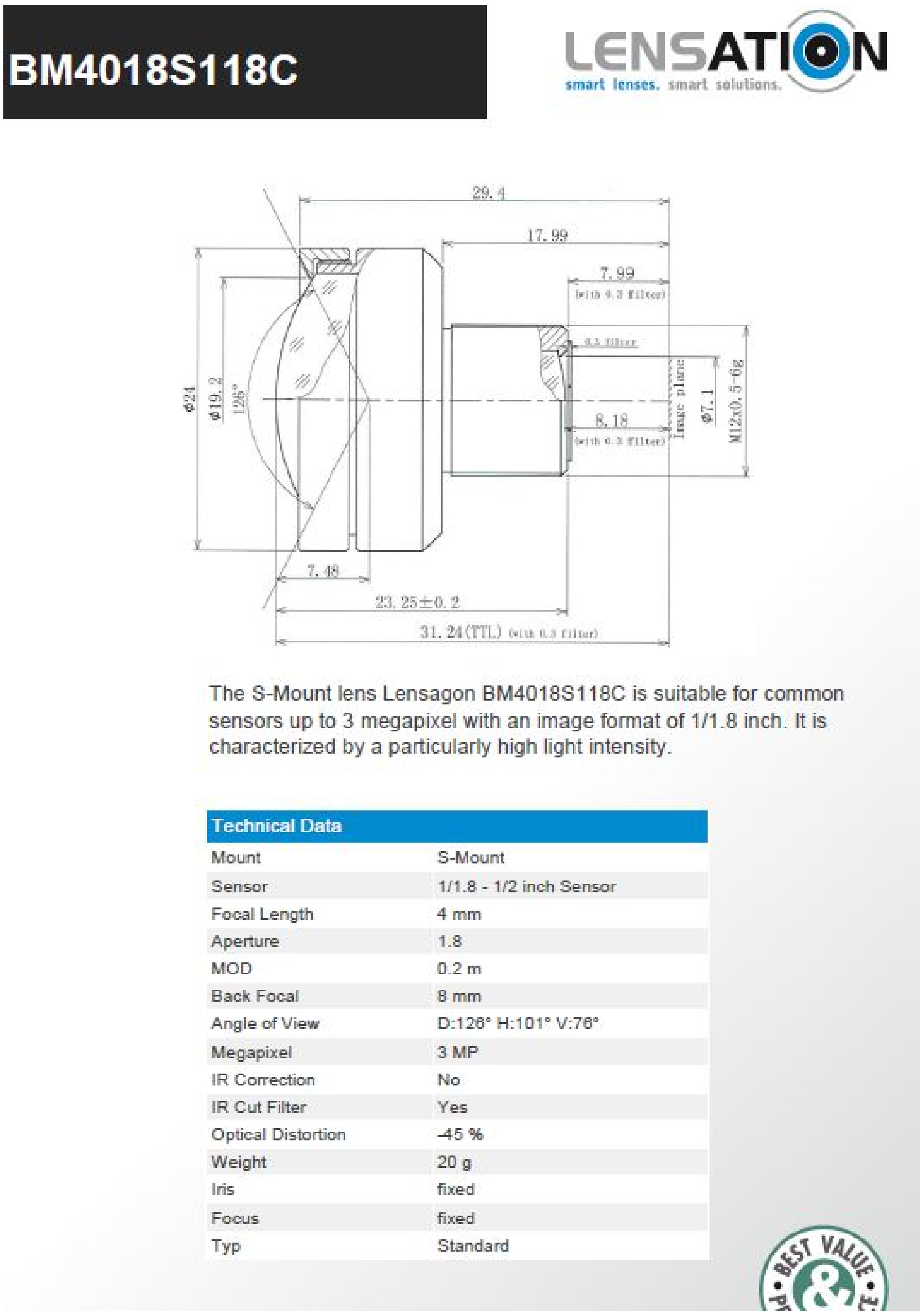
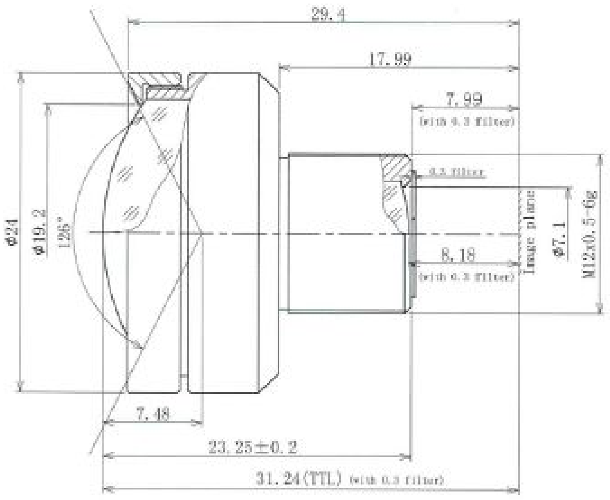


Figure B.1: Datasheet of the uEye USB3 camera used in the project.



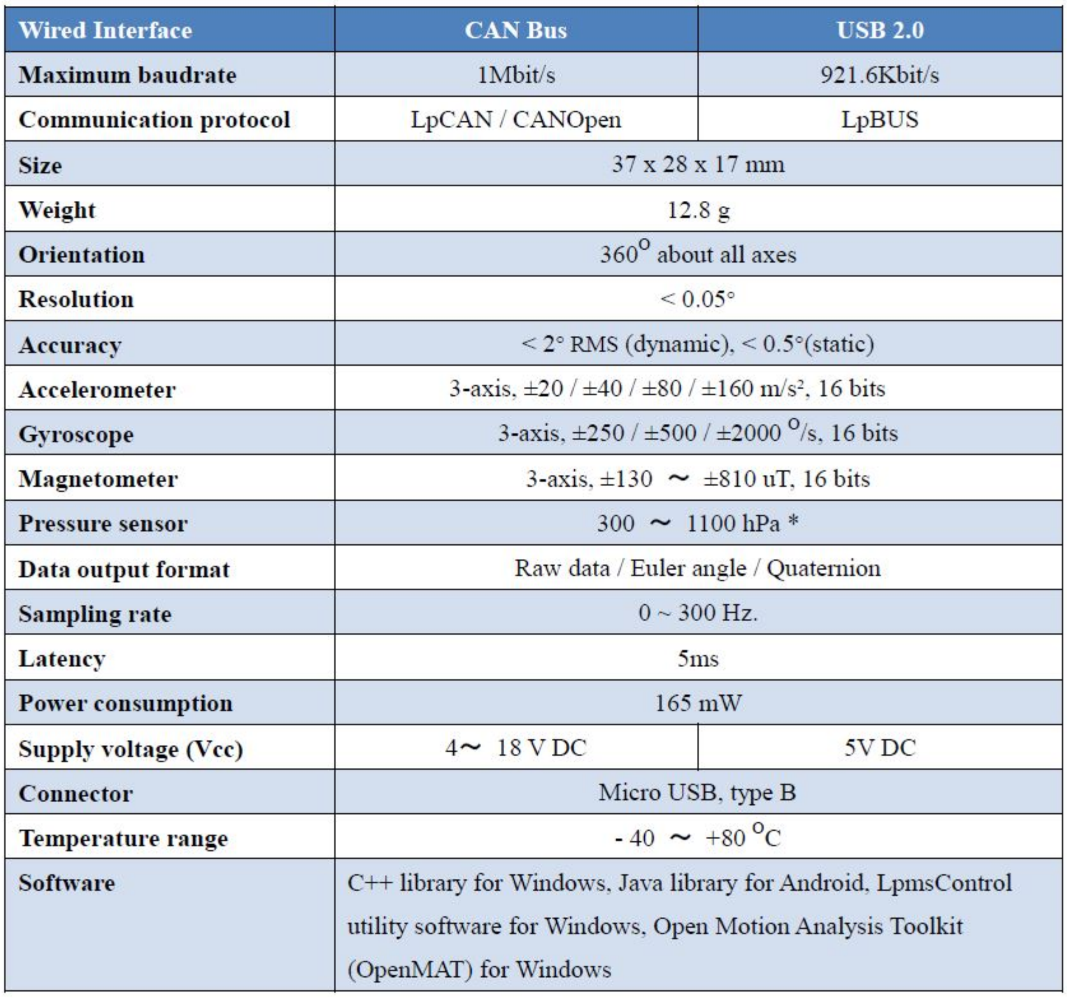
BM4018S118C



The S-Mount lens Lensagon BM4018S118C is suitable for common sensors up to 3 megapixel with an image format of 1/1.8 inch. It is characterized by a particularly high light intensity.

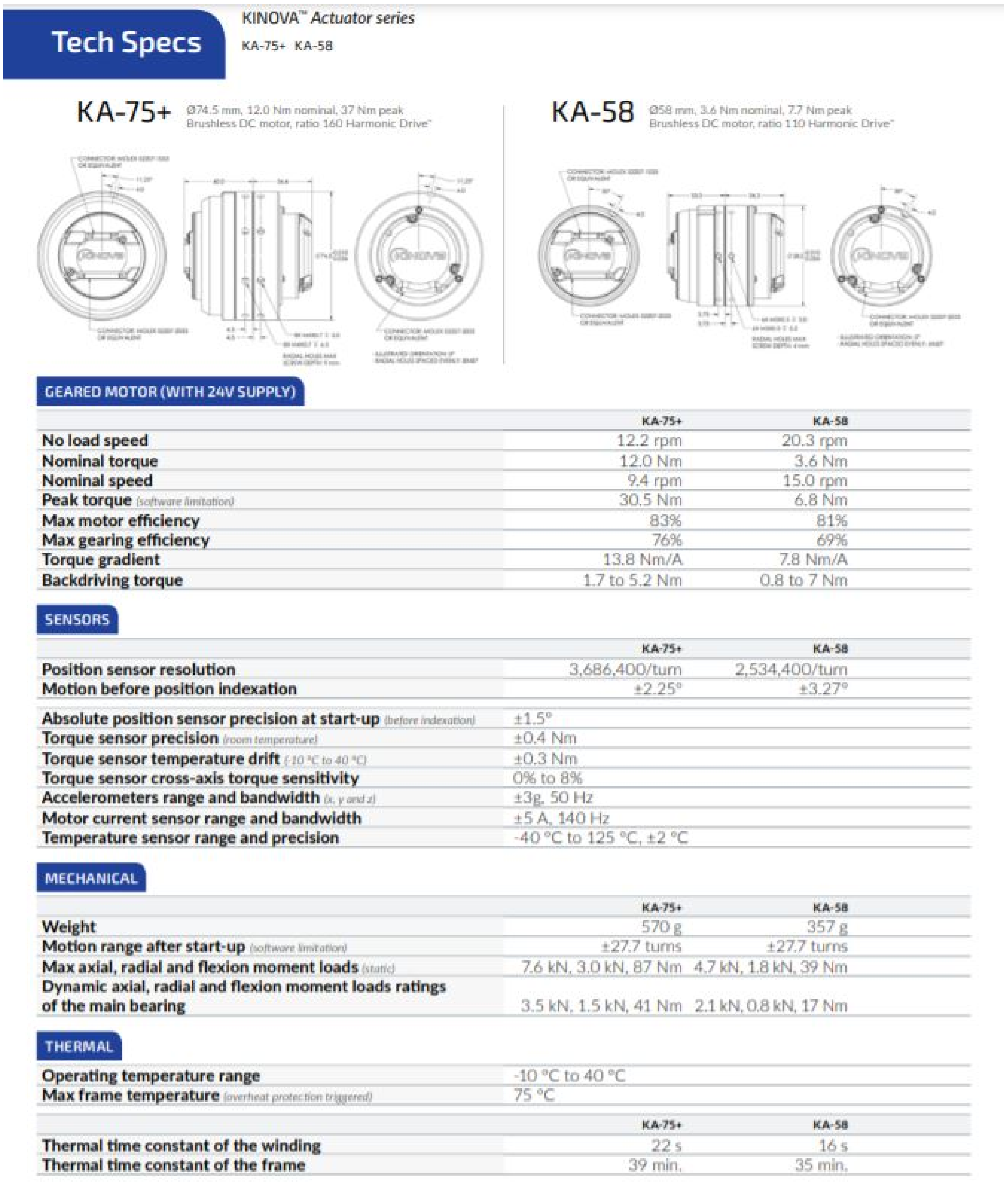
Technical Data	
Mount	S-Mount
Sensor	1/1.8 - 1/2 inch Sensor
Focal Length	4 mm
Aperture	1.8
MOD	0.2 m
Back Focal	8 mm
Angle of View	D:126° H:101° V:76°
Megapixel	3 MP
IR Correction	No
IR Cut Filter	Yes
Optical Distortion	-45 %
Weight	20 g
Iris	fixed
Focus	fixed
Typ	Standard

Figure B.2: Technical specifications of the lens BM4018S118C.



Wired Interface	CAN Bus	USB 2.0
Maximum baudrate	1Mbit/s	921.6Kbit/s
Communication protocol	LpCAN / CANOpen	LpBUS
Size	37 x 28 x 17 mm	
Weight	12.8 g	
Orientation	360^{o} about all axes	
Resolution	< 0.05°	
Accuracy	< 2° RMS (dynamic), < 0.5°(static)	
Accelerometer	3-axis, ±20 / ±40 / ±80 / ±160 m/s^2, 16 bits	
Gyroscope	3-axis, ±250 / ±500 / ±2000 $^{o}/s$, 16 bits	
Magnetometer	3-axis, ±130 ~ ±810 uT, 16 bits	
Pressure sensor	300 ~ 1100 hPa *	
Data output format	Raw data / Euler angle / Quaternion	
Sampling rate	0 ~ 300 Hz.	
Latency	5ms	
Power consumption	165 mW	
Supply voltage (Vcc)	4~ 18 V DC	5V DC
Connector	Micro USB, type B	
Temperature range	- 40 ~ +80 ^{o}C	
Software	C++ library for Windows, Java library for Android, LpmsControl utility software for Windows, Open Motion Analysis Toolkit (OpenMAT) for Windows	

Figure B.4: Technical specifications of the LPMS-CU inertial sensor.



Tech Specs

KINOVA™ *Actuator series*

KA-75+ KA-58

KA-75+ Ø74.5 mm, 12.0 Nm nominal, 37 Nm peak
Brushless DC motor, ratio 160 Harmonic Drive™

KA-58 Ø58 mm, 3.6 Nm nominal, 7.7 Nm peak
Brushless DC motor, ratio 110 Harmonic Drive™

GEARED MOTOR (WITH 24V SUPPLY)

	KA-75+	KA-58
No load speed	12.2 rpm	20.3 rpm
Nominal torque	12.0 Nm	3.6 Nm
Nominal speed	9.4 rpm	15.0 rpm
Peak torque *(software limitation)*	30.5 Nm	6.8 Nm
Max motor efficiency	83%	81%
Max gearing efficiency	76%	69%
Torque gradient	13.8 Nm/A	7.8 Nm/A
Backdriving torque	1.7 to 5.2 Nm	0.8 to 7 Nm

SENSORS

	KA-75+	KA-58
Position sensor resolution	3,686,400/turn	2,534,400/turn
Motion before position indexation	±2.25°	±3.27°

Absolute position sensor precision at start-up *(before indexation)*	±1.5°
Torque sensor precision *(room temperature)*	±0.4 Nm
Torque sensor temperature drift *(-10 °C to 40 °C)*	±0.3 Nm
Torque sensor cross-axis torque sensitivity	0% to 8%
Accelerometers range and bandwidth *(x, y and z)*	±3g, 50 Hz
Motor current sensor range and bandwidth	±5 A, 140 Hz
Temperature sensor range and precision	-40 °C to 125 °C, ±2 °C

MECHANICAL

	KA-75+	KA-58
Weight	570 g	357 g
Motion range after start-up *(software limitation)*	±27.7 turns	±27.7 turns
Max axial, radial and flexion moment loads *(static)*	7.6 kN, 3.0 kN, 87 Nm	4.7 kN, 1.8 kN, 39 Nm
Dynamic axial, radial and flexion moment loads ratings of the main bearing	3.5 kN, 1.5 kN, 41 Nm	2.1 kN, 0.8 kN, 17 Nm

THERMAL

Operating temperature range	-10 °C to 40 °C
Max frame temperature *(overheat protection triggered)*	75 °C

	KA-75+	KA-58
Thermal time constant of the winding	22 s	16 s
Thermal time constant of the frame	39 min.	35 min.

Figure B.7: Specifications of the Kinova actuators.