TÉCNICO LISBOA

Determining the orientation of a RGB camera embedded on an artificial eye

Adding a camera to obtain an unbiased real-time estimate for the three-dimensional orientation of a humanoid eye

Mariana Ribeiro dos Reis do Vale Martins

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Advisor(s)/Supervisor(s): Prof./Dr. Alexandre J. M. Bernardino
Prof./Dr José A. R. dos Santos Vitor
Prof./Dr John van Opstal

Examination Committee

Chairperson: Prof./Dr. João Pedro Gomes
Supervisor: Prof./Dr. Alexandre J. M. Bernardino

October 2019

Authorship Declaration

I declare that the present document is an original work of my authorship and fulfills all the requirements of the Code of Conduct and Good Practices of the University of Lisbon.

Acknowledgments

First and foremost, I would like to thank Professor Alexandre Bernardino for conceding me the opportunity of working alongside him for the last one year and a half in this project. Professor Alexandre genuinely dedicates himself to his work, to science and to helping others around him achieve their own goals and ambitions. In him, I have found comfort and great role model. Both him and Professor José Santos-Victor should feel proud for keeping such a warm and friendly working environment at Vislab.

Professor John Van Opstal is the brain (and the eye) behind the ORIENT project. I thank him for the great amount of time he has dispensed on helping our team in Lisbon, even when it's not his area of expertise. I seek inspiration in him, as every time he talks about a topic, he does it with enormous passion, confidence and happiness. He infects everyone else in the room with the same enthusiasm as he has for life.

The last piece of the project are my teammates, Carlos and Akhil that made everything much more wholesome by supporting, helping and making me laugh on the toughest nights. They are the most competent people I have met and I have no doubt they will succeed in any path they choose to follow as I stand to support them onto it. To the most recent members, Rui and Bernardo, I wish you all the luck and hope you enjoy the experience as much as I have.

In the day I arrived to Técnico, Professor Arlindo emphasized how the next five years would be the best of my life and he was right. But, he was right because during this time I have met the most incredible humans and collected numerous memories I hold very dear to my heart. I thank that to my group of friends that have kept by my side from the beginning and with whom I could not have done it, Firmino, João, Sara, Marco and Pedro. I wish to see you happy, fulfilled and on the next monthly dinner with enough alcohol in your blood.

As it should be implicit to them, I want to hugely thank my family and Bruno that through a lot of trouble down the road, have, nevertheless, always supported me in doing what I find suiting in my path.

Lastly, it feels right to mention my cat, Panda Félix, that through cruel punishments that consist of wrist and ankles biting, bowl throwing and alternative home decorations, has always looked for comfort and cuddles in me and I in him during tough times.

You all will be missed.

Abstract

The brain is a highly complex system, even a simple task like looking at an object is still not fully understood. The eye has six extra-ocular muscles, but only needs two degrees of freedom to orient the fovea at any far position in the visual environment. How does the brain determine which muscles to contract? Bernstein theorized that it tries to find the optimal control solution, but which cost is minimized, energy, speed, accuracy? Building a robotic eye can prove valuable to identify the fundamental mechanisms behind these questions.

A working mechanical model of a biologically inspired eye has been built in a previous project. An indispensable aspect of this prototype is determining its orientation with high accuracy, which is currently done by an IMU. However, this method suffers from significant inaccuracy, due to drift. Hence, we investigated whether adding a RGB camera to estimate the eye's 3D orientation could significantly improve accuracy and precision of the measurements.

The camera position on the prototype is such that when rotating the eye, there is also a translation associated to the movement, which is a fixed function of the rotation. Despite the long literature on orientation estimation using a camera with naturalistic visual features, there is not much literature concerning this constraint. We compare two adapted state-of-the art algorithms, and present a novel algorithm that optimally makes use of this constraint. We validated the algorithms with a simulator and on the real eye prototype, showing promising results when compared to the IMU.

Keywords: camera orientation, rotation estimation, epipolar geometry, procrustes, back projection

Resumo

O teu resumo aqui...

Keywords: as tuas palavras chave

Contents

List of Tables

List of Figures

Nomenclature

Camera Model

$\mathbf{m}$	Digital image point (in pixels)
u, v	Digital image horizontal and vertical coordinates (in pixels)
$\mathbf{m}'$	Image plane point (in meters)
x', y'	Image plane horizontal and vertical coordinates (in meters)
$\mathbf{m}'_\mathbf{d}$	Image plane point (in meters) distorted
x'_d, y'_d	Image plane horizontal and vertical coordinates (in meters distorted)
$\mathbf{M}$	3D point (in meters)
X, Y, Z	3D point coordinates (in meters), Z is perpendicular to the image plane, X is horizontal and Y is vertical
K	Intrinsics Matrix
P	Projection Matrix
λ	Depth
f	Focal Length
s	Skew
c_x, c_y	Camera's principal point offset
s_x, s_y	Pixel scalings

Epipolar Geometry

E	Essential Matrix
F	Fundamental Matrix

Representation of the eye's orientation

$\theta_X, \theta_Y, \theta_Z$	Euler angles
R	Rotation Matrix
$\mathbf{t}$	Translation vector
$\mathbf{r}$	Rotation vector
$\mathbf{q}$	Quaternion vector

Acronyms

FLANN Fast Library for Approximate Nearest Neighbour. 28, 30

GRAT Gradient-based technique. viii, 17, 21, 23

IMU Inertial Measurement Unit. viii, xiii, 4–6, 25–28, 30, 31, 33, 44

LPMS-CU LP-Research Motion Sensor CAN bus and USB version. 44

MBPE Minimization of the back projection error. viii, 21–23

OPPR Orthogonal Procrustes Problem. 7, 12, 13, 18, 19, 21, 23

PCB Printable Circuit Board. xiii, 43

RANSAC RANdom SAmple Consensus. vii, 18, 19

RGB Red, Green, Blue. 6, 12, 26

SIFT Scale Invariant Feature Transform. viii, xiii, 28–30, 37, 39

SURF Speeded Up Robust Features. viii, xiii, 28–30, 39, 40

SVD Singular Value Decomposition. i, 17, 23, 41

Glossary

gaussian blur The Gaussian blur is an image-blurring filter that uses a Gaussian function. It produces a smoother and less noisy image.. 30, 37

homogeneous coordinates In computer vision, an extra dimension is added to the coordinates of a point that makes perspective projection transformations easier to compute. For a point $[x\ y]^T$, any three numbers, $[a_1\ a_2\ a_3]^T$ for which $\frac{a_1}{a_3} = x$ and $\frac{a_2}{a_3} = y$ are homogenous coordinates. This three new coordinates are represented by "$\sim$", e.g. $\widetilde{a} = [a_1\ a_2\ a_3]^T$. vii, 11, 14

homography An homography is a 3 by 3 matrix that relates two images looking at the same plane from different angles. 14

saccade A saccade is a rapid eye movement that shifts the center of gaze from one part of the visual field to another.. xiii, 25, 26

singular value decomposition Singular Value Decomposition (SVD) is the factorization of a matrix A into the product of three matrices $U\Sigma V^T$ where the columns of U and V are orthonormal and Σ is a diagonal matrix with positive real entries. 12

skew symmetric matrix A skew-symmetric matrix is a square matrix whose transpose equals its negative. These matrices can be used to represent cross products as matrix multiplications.. vii, 13, 14, 41

Chapter 1

Introduction

1.1 Context

Even after years of research, the brain remains to this day a mysterious information-processing biological system. For example, from many of its puzzling abilities, it is still not completely understood how the brain determines which muscles to control in order to fulfill a particular movement task. This task could be something as simple as grasping an object, or even before that, looking at (and identifying) that object.

The required number of degrees of freedom to perform a particular movement is typically much smaller than the ones made available by the muscular apparatus, thus yielding a redundant control problem with infinitely many possibilities. For example, the eye has six extra-ocular muscles (6 DOF), but to point the eye in any given direction, only two coordinates are needed (the azimuth and the elevation angles). This aspect of motor control has become known as the ”Degrees of Freedom Problem” (DOF)[1], and was first articulated by Nikolai Bernstein in its current form. Bernstein theorized that the brain gradually tries to find the optimal control solution for a certain task, to constrain the DOF to the required motor space, which would result in consistent performance. Indeed, when different subjects perform the same task, their muscular activations are remarkably similar. In fact, as it will be seen below, the eye orientation is constrained by the so-called Donders' law.

Having to navigate through an environment in which multiple stimuli compete for attention, and being equipped with multiple sensory and motor systems to do so, severely complicates the challenges for the brain to rapidly select the optimal plan for a particular goal [2]. This leads to several questions: How does the normal brain decide which signals are ”goals”, and which are ”distractors” in those complex environments? How is the plan to reach the goal truly defined? And how does this differ for sensory-impaired brains?

When focusing on audiovisual stimuli, it is important to apply the scientifically acquired concepts from neuroscience and psychophysics to an approximate model of the human sensory-motor system (eyes, head, and ears), which is the main topic of the ORIENT research project for the Lisbon team[2]. The idea is to create an autonomous humanoid eye-head robot with foveal vision, realistic auditory inputs, three-dimensional nested eye and head motor systems, and rapid sensory-motor feedback controls and learning algorithms. This will hopefully contribute to a better understanding of the eye's control system, and consequently help at restoring impaired vision, which was, after all, Franciscus Donders' ultimate desire.

So far, a working mechanical model of a biologically inspired eye with six muscles has been built in

[1] N. Bernstein, “The Coordination and Regulation of Movements. Oxford : Pergamon Press,” 1967.
[2] http://www.mbfys.ru.nl/ johnvo/OrientWeb/orient.html

a previous work [3], and is shown in Figure 1.1. This model was constructed with Donders' and Listing's Laws in mind. Donders' law states that the 3D orientation of the eye, when looking in a specific direction, is always the same. Listing propounded that the law could be further constrained by the discovery that the rotation axes corresponding to different eye orientations all lie in a common plane, called Listing's plane. The component of the rotation axis normal to this plane quantifies the eye's torsional orientation component, and results to be close to zero in these conditions. The normal vector to Listing's Plane is directed into the so-called *primary direction*, and points about 15 degrees upwards relative to the straight-ahead viewing direction (head fixed) in primates[3].

It is still not well understood which aspects of the ocular motor system determine the primary orientation (and hence, Listing's law). Whether the rotation axis of eye orientation is programmed by the brain, fully determined by the mechanical properties of the eye muscles, or by both factors, is still not known for sure, and is highly debated in the literature. Yet, implementing this system in a humanoid robot might help at understanding this problem better. Note that for active eye-head movements, for vergence eye movements (near viewing), for vestibular eye movements (head rotations), and for eye-movements under tilted head orientations, Listing's law no longer holds true, which provides a strong counter argument against the eye muscle (pulley) hypothesis. [4]

1.2 Motivation

The current eye prototype uses an Inertial Measurement Unit (IMU) to estimate the eye's orientation. Although IMU units are good on acceleration and velocity measurements, they tend to have a significant position drift when determining the orientation, as seen in Figure 1.2), which is of course a considerable problem when the eye's orientation ideally needs to be established with better than $0.5\ degrees$ resolution. Therefore, this work's focus is to study the addition of a camera to the system, in order to determine the orientation of the eye in a more reliable manner.

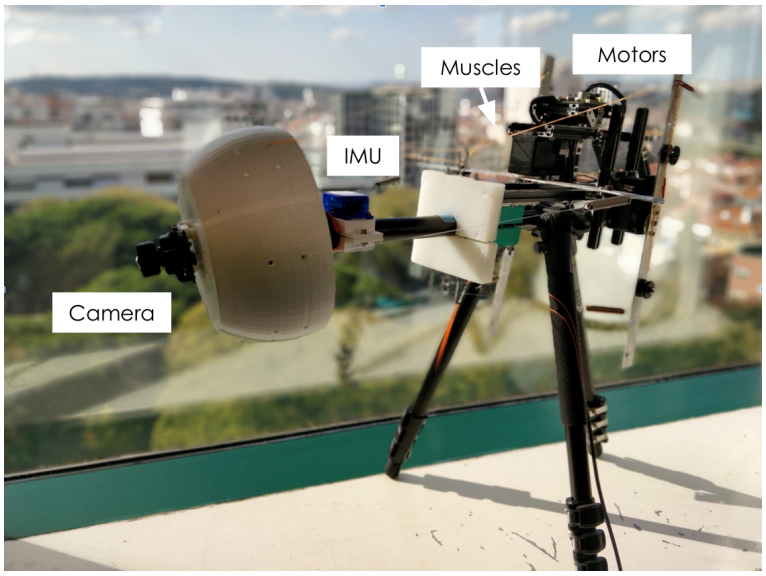


Figure 1.1: The current mechanical eye model. It is composed of an IMU, seen on the left, and connected to a supportive white eye mounted on a global joint, in turn connected to six elastics, representing the eye muscles, and finally controlled by three motors that pull at those elastics (paired by the aluminum strips). The three motors are controlled by the computer.

[3]See appendix A.1 for more background on this.

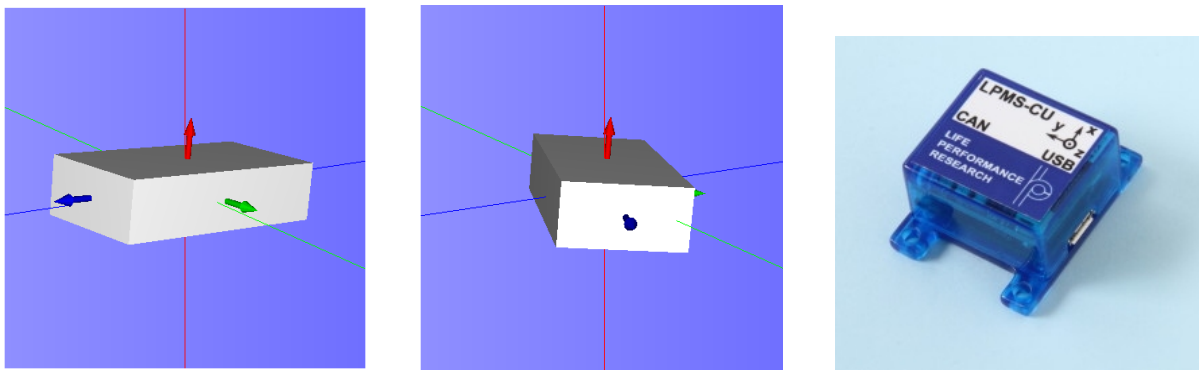


Figure 1.2: IMU sensor's drift. The two images on the left represent the IMU position on 3D space, between them it can be observed that, in 80 seconds, the IMU drifts about 60 degrees around the torsional axis. The coordinate system of the IMU is defined on the image on the right, where the axis in red is the torsional component, z, green is y, and blue is x.

1.3 Problem definition

As the eye model will eventually rely on accurate camera output, and to foster solid neuroscientifically inspired study with this model, it's indispensable to have the best possible estimates of the camera's orientation. It is also important to consider the computational speed versus accuracy trade off for real-time operations. The accuracy and computational time mainly depend on the complexity of detecting correspondences between consecutive images and on the algorithm responsible for calculating the camera's orientation difference between those images.

There is a long literature on the estimation of the orientation of a camera from visual features. However, the current prototype, seen on Figure 1.2, has a particularity. There is a translation movement associated to the camera movement that may be defined as a function of the rotation and the length from the camera's center to the rotational center that holds the camera. That length will be referred to as baseline and is shown on Figure 1.3.

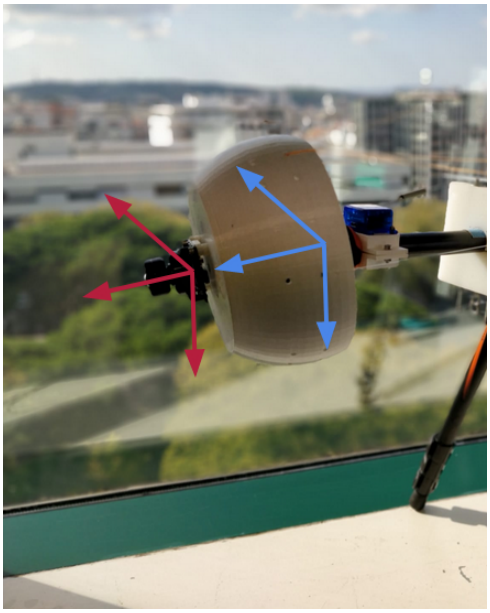


Figure 1.3: Baseline associated. It shows where the camera's center is, red coordinate system, versus where the eye's rotation center is, blue coordinate system, hence creating a baseline length on the Z coordinate.

Hence, the problem is finding the best algorithm to estimate the orientation of a RGB camera with a baseline length associated to the rotational movement.

1.4 Objectives

The main objective of this thesis is to develop an algorithm to determine the orientation of the camera in its particular context that works as accurately and as fast as possible. Beyond that, this work also aims to:

- Determine how much accuracy may be gained from using the baseline constraint versus the classical unconstrained or rotation only approaches;
- Understand how different algorithms behave in the real world;
- And evaluate the improvement of using the camera over the IMU.

1.5 Outline

The next chapters of this document are structured as follows.

- Chapter 2 gives insight on essential concepts necessary to understand this work, along with an overview of the state of the art on the thesis' topic.

  CHANGE CHANGE CHANGE

- Chapter 3 explains the methodology used to accomplish the objectives. It describes a novel algorithm to obtain the camera's orientation, how state of the art algorithms were implemented and how all of them were compared against each other.
- Chapter 4 presents the results gathered from comparing various estimation algorithms between each other, and with the IMU.
- Chapter 5 draws conclusions on which algorithm is best to use on the current prototype according to their behavior on the real world. Enlightens the reader on how this work attempts to contribute to science and on future work that shall be done.

Chapter 2

Background and State of the Art

In this chapter there will be an overview on essential concepts to understand this work, such as the representation of the eye orientation in the 3D space and the pinhole camera model. Some state of the art algorithms that can potentially solve the problem and enhance the solution will be presented. More specifically, the Orthogonal Procrustes Problem, optimization functions using the epipolar geometry relation and a robust estimation method are described.

2.1 Representation of eye orientation in 3 dimensions

In computer vision, it's customary to use the coordinate system on the left of Figure 2.1, but in neuroscience a different convention, represented on the right of the Figure, is used. In order to be coherent with computer vision concepts, the first mentioned convention will be the one in use throughout the remaining of this report.

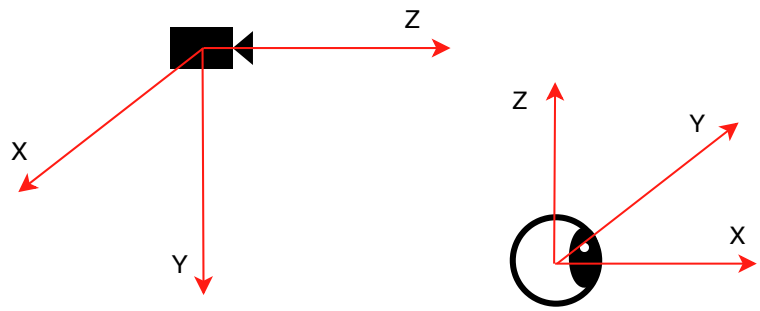


Figure 2.1: On the left, the coordinate system used in computer vision with the torsional component coming out of the front of the camera. On the right, the coordinate system used in the neuroscience field with the torsional component coming out of the front of the eye.

There are several ways of representing a rotation in 3D, the most common being Euler angles and rotation matrices. Euler angles are three angles that describe the orientation of a rigid body with respect to a fixed coordinate system. For this work, maintaining the coordinate system said above, the sequence of rotations used to represent a given orientation can be defined by the angles $[\theta_Z\ \theta_Y\ \theta_X]$, which corresponds to rotating around Z, Y, X axes, respectively.

2.1.1 Rotation matrices

The human eye is a purely rotative body (no translations), and thus any orientation can be described by a unique series of three rotations around each of the axis defined in three dimensional space as

$$R_z(\theta_Z) = \begin{bmatrix} \cos\theta_Z & -\sin\theta_Z & 0 \\ \sin\theta_Z & \cos\theta_Z & 0 \\ 0 & 0 & 1 \end{bmatrix}, \; R_y(\theta_Y) = \begin{bmatrix} \cos\theta_Y & 0 & \sin\theta_Y \\ 0 & 1 & 0 \\ -\sin\theta_Y & 0 & \cos\theta_Y \end{bmatrix}, \; R_x(\theta_X) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_X & -\sin\theta_X \\ 0 & \sin\theta_X & \cos\theta_X \end{bmatrix}, \tag{2.1}$$

where each angle is defined in counter-clockwise direction around each axis. An arbitrary rotation, R, may then be defined as some multiplication (i.e. a serial order) of those three, noting that the order by which they are multiplied matters (rotations are non-commutative). A matrix originated from any combination of those is a rotation matrix because it satisfies the following conditions,

$$R^{-1} = R^T \tag{2.2}$$

$$det(R) = 1. \tag{2.3}$$

Instead of using multiple rotations done after each other around three different axes, Euler's theorem states that the orientation of the rotating body can also be parametrized by a single rotation with an angle, ρ, about an axis in 3D, $\hat{\mathbf{n}} = (n_1, n_2, n_3)$. That rotation is denoted as $R(\hat{\mathbf{n}}, \rho)$ and is given by

$$R(\hat{\mathbf{n}}, \rho) = \begin{pmatrix} \cos\rho + n_1^2(1-\cos\rho) & n_1 n_2(1-\cos\rho) - n_3\sin\rho & n_1 n_3(1-\cos\rho) + n_2\sin\rho \\ n_1 n_2(1-\cos\rho) + n_3\sin\rho & \cos\rho + n_2^2(1-\cos\rho) & n_2 n_3(1-\cos\rho) + n_2\sin\rho \\ n_1 n_3(1-\cos\rho) - n_2\sin\rho & n_2 n_3(1-\cos\rho) + n_1\sin\rho & \cos\rho + n_3^2(1-\cos\rho) \end{pmatrix}. \tag{2.4}$$

Still, rotation matrices are not the most efficient way to define rotations given they have nine elements and only three are actually necessary to describe the rotation uniquely. Hence, in oculomotor literature more suitable representations are used, such as quaternions, or rotation vectors that are more intuitive. [5][6]

2.1.2 Quaternions

Quaternions are 4-dimensional complex algebraic objects that are related to a rotation around an axis, $\hat{\mathbf{n}}$ by an angle ρ as follows,

$$q = \cos(\rho/2) + \mathbf{i}\sin(\rho/2)\hat{\mathbf{n}} \equiv q_0 + \mathbf{i}\cdot\mathbf{q}, \tag{2.5}$$

where q_0 and $\mathbf{q} = [q_1 \; q_2 \; q_3]$ are the scalar and vectorial parts of the quaternion, respectively, and $\mathbf{i}$ is the complex 3D vector.

Like rotation matrices, a quaternion can be defined as a sequence of rotations by multiplying each corresponding quaternion in sequence. The product of a quaternion $\mathbf{q}$ with a quaternion $\mathbf{p}$ is

$$q_0 p_0 - \mathbf{q}\cdot\mathbf{p} + q_0\mathbf{p} + p_0\mathbf{q} + \mathbf{q}\times\mathbf{p}. \tag{2.6}$$

The rotation between two quaternions can be obtained by

$$\mathbf{p}\cdot\mathbf{q}^{-1}, \tag{2.7}$$

where $\mathbf{q}^{-1} = \frac{q^*}{|q|}$ with q^* being the conjugate of q.

Moreover, quaternions can be related to rotation matrices through

$$R = \begin{bmatrix} 1-2\left(q_2^2+q_3^2\right) & 2\left(q_1q_2-q_0q_3\right) & 2\left(q_1q_3+q_0q_2\right) \\ 2\left(q_1q_2+q_0q_3\right) & 1-2\left(q_1^2+q_3^2\right) & 2\left(q_2q_3-q_0q_1\right) \\ 2\left(q_1q_3-q_0q_2\right) & 2\left(q_2q_3+q_0q_1\right) & 1-2\left(q_1^2+q_2^2\right) \end{bmatrix} \tag{2.8}$$

For the description of eye movements, only the vectorial part, $\mathbf{q}$, is necessary, since the scalar part does not contain any information not already given by the vectorial one. Thus, it can be eliminated by using rotation vectors, which are related to quaternions by $\mathbf{r} = \mathbf{q}/q_0$.

2.1.3 Rotation vector in head-fixed coordinates

The rotation vector, $\mathbf{r}$, is directed along the rotation axis, $\hat{\mathbf{n}}$, and its length varies with the amount of rotation, ρ, around it. It can be described by

$$\mathbf{r} = \tan(\frac{\rho}{2})\hat{\mathbf{n}}, \tag{2.9}$$

where $\hat{\mathbf{n}}$ can be obtained through a rotation matrix as

$$\begin{aligned} n_1 &= \sin\rho\left(R_{32}-R_{23}\right)/2 \\ n_2 &= \sin\rho\left(R_{13}-R_{31}\right)/2 \\ n_3 &= \sin\rho\left(R_{21}-R_{12}\right)/2, \end{aligned} \tag{2.10}$$

and ρ through

$$\cos\rho = 1/2\cdot\left(R_{11}+R_{22}+R_{33}-1\right). \tag{2.11}$$

It's important to note that in this format, the units are given in half-radians.

2.2 Camera Model

This section introduces the essential concepts on the pinhole camera model that follow. Digital cameras are equipped with a sensor (mostly charge-coupled devices, CCD, and CMOS') that transforms light into discrete electrical signals. When taking a picture, some of the light reflecting on the object is directed towards the camera pinhole (tiny aperture on the camera), forming a 180º inverted image of the object on the sensor, as shows on Figure 2.2. All the light rays entering the camera converge into the pinhole and diverge on the other side (with a unique one-to-one correspondence). Therefore the pinhole is also named the **center of projection**, principal point, or focal point.

In cameras the aperture size and light-exposure time can be controlled. A smaller aperture produces a sharper image but needs a longer exposure time to collect enough light, whereas a larger aperture allows more light to be captured, but at the expense of producing blurred images.

The image of the object is projected onto the so-called **image plane**. When moving this plane closer to the pinhole (zoom out), the projected object gets smaller, and vice-versa. The distance between the plane and the aperture is the **focal length**, or focal distance. The view-angle of a camera depends on the focal distance, and on the sensor size. The image resolution depends solely on the number of pixels that fit on the sensor.

A useful coordinate system, called the **camera reference system**, is centered at the pinhole, and has its z-axis perpendicular to the image plane, as mentioned in section 2.1. The line defined between the center of projection and the center of the image plane is called the **optical axis**.

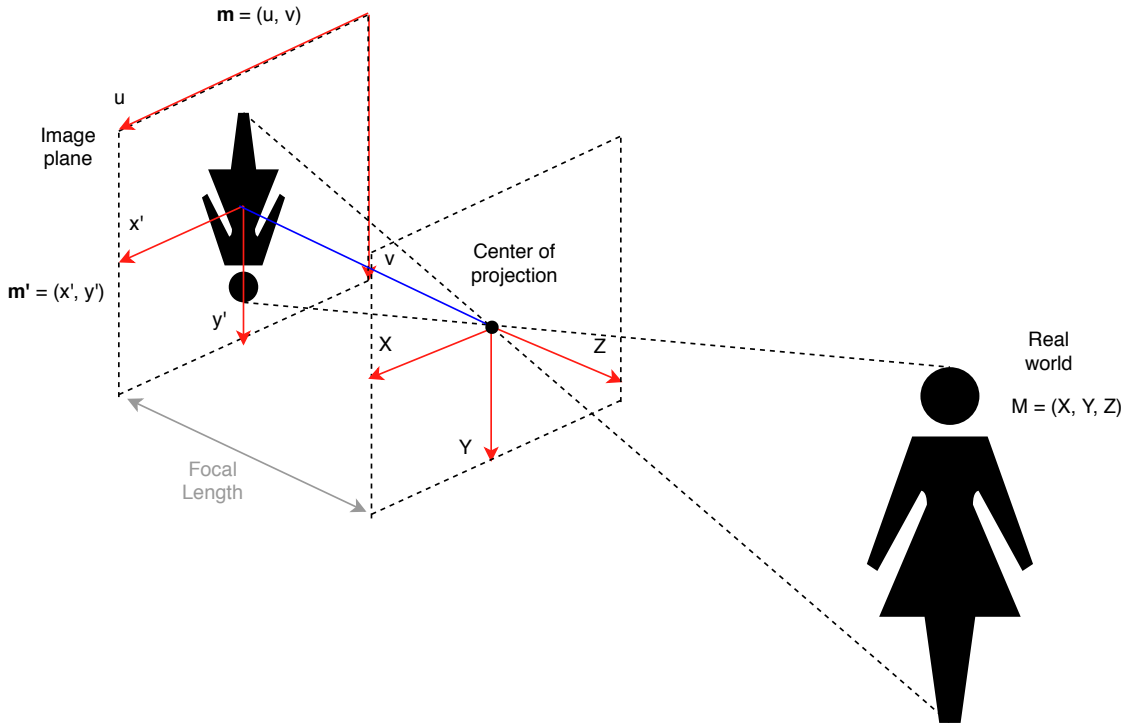


Figure 2.2: Pinhole Camera Model. The girl defined by the generic set of points M is projected onto the image plane (defined by the set of generic points $\mathbf{m_p}$) through the light rays passing by the camera's pinhole, called the center of projection. The focal length is the distance from the center of the image plane to the center of projection. The digital image's coordinate system (u, v) is defined on top right corner of the image plane. The optical axis is the blue line.

Figure 2.2 summarizes the concepts explained above, and helps at understanding the upcoming details. It's possible to establish a relationship between a real point in space and a point in an image in pixels, through the camera model as follows.

1. **Passing from the world frame to the camera reference frame**

   An arbitrary real point in space $M_W = [X_W\ Y_W\ Z_W]^T$ may be represented in the world reference frame, so it needs to be be converted into the camera reference frame, becoming M.

   This conversion consists on expressing the origin of the world reference frame on the camera frame, through a translation, $\mathbf{t}$, and a rotation, that can be represented by a matrix, R. These are called the **extrinsic parameters**.

   $$M = [X\ Y\ Z]^T = R\,[X_W\ Y_W\ Z_W]^T + \mathbf{t} \tag{2.12}$$

2. **Project the 3D points into the image plane reference frame**

   As may be observed in the figure, a correspondence between the real world points $M = [X\ Y\ Z]^T$ and the points projected onto the image plane $\mathbf{m}' = [x'\ y']^T$ can be established by the following triangular similarity as

   $$x' = f\frac{X}{Z},\ y' = f\frac{Y}{Z}, \tag{2.13}$$

   where f is the focal length in meters, and x', y', X, Y and Z are also represented in meters.

3. **Transform image plane points into pixel coordinates**

The digital image obtained is actually expressed in pixel units, whereas till now everything was described in metric units, requiring the points to be converted by scalings (s_x, s_y), as

$$u_s = s_x x',\ v = s_y y', \tag{2.14}$$

where (u, v) are the scaled image coordinates in pixels, denoted by $\mathbf{m} = [u_s\ v_s]^T$.

Moreover, the image plane center does not coincide with the center of the digital image reference frame, as shown in Figure 2.2, thus the image points also have to be translated by $(c_x\ c_y)$ pixels, which is camera's principal point offset.

$$u_{sc} = s_x x' + c_x,\ v = s_y y' + c_y. \tag{2.15}$$

Furthermore, due to sensor errors, there might be a deformation skew, s, associated to the camera reference frame, which means that it may not have exactly perpendicular axes.

$$u = s_x x' + s y' + c_x,\ v = s_y y' + c_y. \tag{2.16}$$

Combining all the parameters, it yields the so-called **intrinsic parameters** matrix defined as,

$$K = \begin{bmatrix} f s_x & s & c_x \\ 0 & f s_y & c_y \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.17}$$

Hence, using homogeneous coordinates, $\mathbf{m}$ may be expressed as,

$$\begin{array}{l} \widetilde{\mathbf{m}} = KR\frac{M}{Z} + \mathbf{t} \\ \lambda\widetilde{\mathbf{m}} = KRM + \mathbf{t} \end{array}, \tag{2.18}$$

where $\lambda = Z$. Or, more commonly,

$$\begin{array}{l} \lambda\widetilde{\mathbf{m}} \sim K[R \mid \mathbf{t}]\widetilde{M} \\ \lambda\widetilde{\mathbf{m}} \sim P\widetilde{M} \end{array}, \tag{2.19}$$

where P is a 3×4 matrix, named **camera matrix**.

In practice, the camera model contains several nonlinear effects, generically denoted as "distortion", since real cameras have lenses instead of a pinhole. The lens allows all the emitted rays of light to be refracted into one converging point, the focal point. Although the focal length will now increase the distance from the lens to the converging point, the camera model explained above still applies, but the distortions must be accounted for. These distortions are incorporated in (2.20). Radial distortion is modeled by the k_i coefficients on the left-hand side of the polynomial transformation in (2.20) (taken from Zernike's model of aberrations), and the tangential distortion (which occurs when the image plane is not parallel to the camera lens) is defined by the p_j coefficients in the right-hand term.[1]

$$\begin{array}{l} x'_d = x' \frac{1+k_1 r^2+k_2 r^4+k_3 r^6}{1+k_4 r^2+k_5 r^4+k_6 r^6} + (2p_1 x' y' + p_2(r^2 + 2x'^2)) \\ y'_d = y' \frac{1+k_1 r^2+k_2 r^4+k_3 r^6}{1+k_4 r^2+k_5 r^4+k_6 r^6} + (p_1(r^2 + 2y'^2) + 2p_2 x' y') \end{array}, \tag{2.20}$$

where $r^2 = x'^2 + y'^2$ and x'_d and y'_d are the distorted coordinates of the image point, which should then be converted into digital image coordinates as shown previously.

[1] Born, Max; Wolf, Emil. Principles of Optics: Electromagnetic Theory of Propagation, Interference and Diffraction of Light. Chapter 5.

2.3 Orthogonal Procrustes Problem and Lack of Depth

Assuming that the image points are obtained from the RGB camera installed on the eye prototype, using the camera model, this 2D points can be converted to corresponding points in space.

The 3D points can then be used to estimate the rotation executed by the eye. There are several ways to do this, a potential one is Orthogonal Procrustes Problem (OPPR). This algorithm finds the optimal linear transformation between two 3D clouds of points.

Procrustes, the son of Poseidon in Greek mythology, made his victims fit in a wonderful all-fitting bed, either by stretching their limbs, or by cutting them off. Ultimately, he was fitted into his own bed by Theseus.[2] This is similar to the analysis held here, there are 3 elements to Procrustes's problem: M_2, the bed, M_1, the unlucky adventurer, and T, the fitting treatment. The solution to the problem is a matrix, T, that minimizes

$$\|M_2 - TM_1\|^2, \tag{2.21}$$

which transforms M_1 into M_2. M_1 and M_2 are clouds of 3D points obtained from the images took from the perspectives before and after a rotation, defined as R. Hence, the function to minimize is

$$\|M_2 - RM_1\|^2 \tag{2.22}$$

that through the Frobenius norm can be expanded as

$$\|M_2 - RM_1\|^2 = \operatorname{trace}(M_2^T M_2 + M_1^T M_1) - 2\operatorname{trace}(M_2^T R M_1). \tag{2.23}$$

This means that minimizing (2.21) with respect to R is equivalent to maximizing the second term of (2.23). By applying singular value decomposition (SVD) to $M_1 M_2^T$, the latter term can be further simplified into

$$\operatorname{trace}(M_2^T R M_1) = \operatorname{trace}(M_1 M_2^T R) = \operatorname{trace}(U\Sigma V^T R) = \operatorname{trace}(\Sigma V^T R U) = \operatorname{trace}(\Sigma H) = \sum_{i=1}^{N} \sigma_i h_{ii}. \tag{2.24}$$

The singular values of σ_i are all non-negative, and so the expression becomes maximum when $h_{ii} = 1$ for $i = 1, 2, ..., N$, since H, a product of orthogonal matrices, is an orthogonal matrix itself, thus having its maximal value when $H = I$. This results in $I = V^T R U$, obtaining $R = VU^T$ as the rotation of the eye.

When M_1 and M_2 are associated with different origins, it is customary to consider better fits of the configurations by allowing shape-preserving translations of the origin. When translating the points clouds by t_1 and t_2, respectively, (2.21) becomes

$$\left\|(M_2 - \mathbf{1}\mathbf{t_2}^T) - R(M_1 - \mathbf{1}\mathbf{t_1}^T)\right\|^2, \tag{2.25}$$

which can also be written as

$$\left\|M_2 - RM_1 - \mathbf{1}\mathbf{t}^T\right\|^2, \tag{2.26}$$

where $\mathbf{t}^T = \mathbf{t_2}^T - R\mathbf{t_1}^T$. The expression is minimized when $\mathbf{t}^T$ corresponds to the column-means of $M_2 - RM_1$, therefore a simple way of effecting this translation is to remove the column-means of M_1 and M_2, separately, before minimizing, which is called centering. The data then becomes expressed in terms of the mean deviations. [7]

However, in order to obtain the 3D point clouds from the RGB images, taken before and after rotation,

[2] Karl Kerenyi, The Heroes of the Greeks, 1959:223, noting pseudo-ApollodorusDiodorus Siculus, 4.59.5.

it is necessary to have the depth λ that corresponds to the Z coordinate in the real world, as seen on section 2.2. Assuming pure rotation, any depth provides the same information, so it's possible to define an artificial depth by projecting the 2D points in a sphere with enough radius compared to the focal length, as follows

$$\|(\lambda \mathbf{m_1'})\| = \|(\lambda x_1', \lambda y_1', \lambda)\| = r \tag{2.27}$$

where r is the radius of the sphere and λ then becomes

$$\lambda = \frac{r}{\sqrt{x'^2 + y'^2 + 1}}, . \tag{2.28}$$

Yet, this makes centering inapplicable. Thus to use this algorithm, the translation associated to the eye movement in the current eye prototype has to be ignored, making the rotation derived from OPPR a mere approximate estimation.

2.4 Epipolar Geometry

Epipolar geometry provides an alternative yet powerful tool to obtain image transformations using various different approaches. It describes the relation between two images, before and after a transformation, through a 3x3 singular, non-invertible, matrix called the **essential matrix**, E, if the camera matrix is known, or the **fundamental matrix**, F, otherwise. These matrices are singular because they express an under-constrained relationship between a point in one image and its possible location in the other image, which is not unique due to depth ambiguity. The essential matrix is the product of a rotation matrix and a skew symmetric matrix, whose determinant is zero, as it will be shown below. If a point in the real world, M, is projected as a point $\mathbf{m_1}$ in the first image, and point $\mathbf{m_2}$ in the second, then those points satisfy the epipolar relation

$$\tilde{\mathbf{m}}_\mathbf{2}^T F \tilde{\mathbf{m}}_\mathbf{1} = 0, \tag{2.29}$$

which can be easily deduced with projective geometry. In Figure 2.3, c_1 and c_2 are the centers of

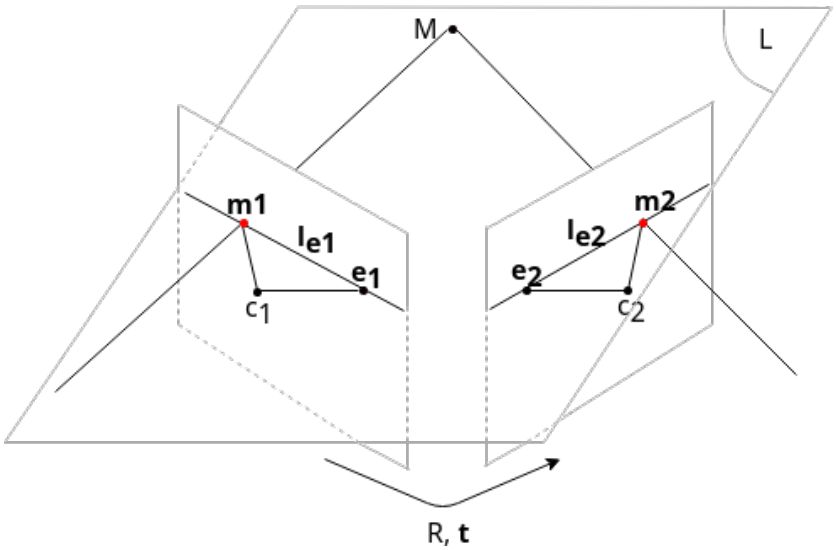


Figure 2.3: Epipolar geometry. The two small vertical planes correspond to the image planes after the rotation and translation of the camera, R and $\mathbf{t}$, respectively. c_1 and c_2 are the centers of projection before and after the transformation. M is the fixed point gazed at in the real world. $\mathbf{m_1}$ and $\mathbf{m_2}$ are the projections of point M into the respective image planes, whose rays are lying on the plane L. $\mathbf{l_{e1}}$ and $\mathbf{l_{e2}}$ are the epipolar lines, that also lie on the plane and $\mathbf{e_1}$ and $\mathbf{e_2}$ are the epipoles.

projection before and after the camera has been displaced by $(R, \mathbf{t})$. Given the point $\mathbf{m_1}$ on the first image, its correspondence, $\mathbf{m_2}$, in the second image is constrained to a line, the epipolar line, $\mathbf{l_{e2}}$. The latter is defined as the intersection of the plane L and the second image plane. Furthermore, plane L is delineated by the two centers of projection and $\mathbf{m_1}$. The epipoles, $\mathbf{e_1}$ and $\mathbf{e_2}$, are born from the intersection of the line $c_1 - c_2$ with the epipolar lines, $\mathbf{l_{e1}}$ and $\mathbf{l_{e2}}$.

2.4.1 Projective geometry concepts

To understand the upcoming deductions, let's introduce the following concepts.

Representation of lines in homogeneous coordinates

A line in a plane is represented by $ax + by + c = 0$, which can also be defined by a vector $(a, b, c)^T$. For any non-zero constant, k, the vectors $(a, b, c)^T$ and $k(a, b, c)^T$ represent the same line and are equivalent. An equivalence class of vectors is known as an homogenous vector.

Point lying on a line

A point $\mathbf{p} = (p_1, p_2)^T$ lies on a line $\tilde{\mathbf{l}} = (a, b, c)^T$, if and only if $ap_1 + bp_2 + c = 0$, which can be written as $(p_1, p_2, 1)(a, b, c)^T = \tilde{\mathbf{p}}^T\tilde{\mathbf{l}} = \tilde{\mathbf{l}}^T\tilde{\mathbf{p}} = 0$.

Line joining points

A line passing through any two points $\tilde{\mathbf{p}}$ and $\tilde{\mathbf{q}}$ can be defined by the cross-product of those points as $\tilde{\mathbf{l}} = \tilde{\mathbf{p}} \times \tilde{\mathbf{q}}$.

Cross product's skew symmetric matrix representation

The previous cross product, $\tilde{\mathbf{l}} = \tilde{\mathbf{p}} \times \tilde{\mathbf{q}}$, can be written as $\tilde{\mathbf{l}} = [\tilde{\mathbf{p}}]_\times \tilde{\mathbf{q}}$, where $[\mathbf{p}]_\times$ is a skew symmetric matrix defined as

$$\begin{bmatrix} 0 & -p_3 & p_2 \\ p_3 & 0 & -p_1 \\ -p_2 & p_1 & 0 \end{bmatrix} \tag{2.30}$$

.

2.4.2 Deducing the fundamental matrix algebraically

The derivation of the fundamental matrix uses the following facts,

1. there is an homography mapping via L plane from the points of the first to the second image, characterized by $\tilde{\mathbf{m}}_\mathbf{2} = H_M\tilde{\mathbf{m}}_\mathbf{1}$;
2. because $\mathbf{m_2}$ lies on $\tilde{\mathbf{l}}_{\mathbf{e_2}}$, then $\tilde{\mathbf{m}}_\mathbf{2}^T\tilde{\mathbf{l}}_{\mathbf{e_2}} = 0$;
3. and since $\mathbf{e_2}, \mathbf{m_2} \in \tilde{\mathbf{l}}_{\mathbf{e_2}}$, then $\tilde{\mathbf{l}}_{\mathbf{e_2}} = [\tilde{\mathbf{e}}_\mathbf{2}]_\times\tilde{\mathbf{m}}_\mathbf{2}$.

From 1) and 3),

$$\tilde{\mathbf{l}}_{\mathbf{e_2}} = [\tilde{\mathbf{e}}_\mathbf{2}]_\times H_M\tilde{\mathbf{m}}_\mathbf{1} \tag{2.31}$$

can be deduced and using conclusion 2) and (2.31),

$$\tilde{\mathbf{m}}_\mathbf{2}^T\tilde{\mathbf{l}}_{\mathbf{e_2}} = 0 = \tilde{\mathbf{m}}_\mathbf{2}^T[\tilde{\mathbf{e}}_\mathbf{2}]_\times H_M\tilde{\mathbf{m}}_\mathbf{1} = \tilde{\mathbf{m}}_\mathbf{2}^T F\tilde{\mathbf{m}}_\mathbf{1}, \tag{2.32}$$

where F, the fundamental matrix, is an homogeneous matrix of rank-2 (since $[\tilde{\mathbf{e}}_\mathbf{2}]_\times$ is rank-2 and H_M is rank-3) with 7 degrees of freedom, and $det(F) = 0$.

2.4.3 Deducing the fundamental matrix from camera motion

Another way of obtaining this relation is through the camera motion. If the point in the real world is expressed in the eye's perspective before rotating, M_1, under the camera model studied in section 2.2, (2.33) and ((2.34) are true and (2.35) can be derived from them.

$$\lambda_1 \tilde{\mathbf{m}}_\mathbf{1} = K\tilde{M}_1, \tag{2.33}$$

$$\lambda_2 \tilde{\mathbf{m}}_\mathbf{2} = K[R\ \mathbf{t}]\tilde{M}_1 \tag{2.34}$$

$$\lambda_2 \tilde{\mathbf{m}}_\mathbf{2} = K[R\ \mathbf{t}]K^{-1}\lambda_1 \tilde{\mathbf{m}}_\mathbf{1} \tag{2.35}$$

For the sake of simplicity, the intrinsics matrix will be considered the identity, $K = I$, and the scale factors λ_1 and λ_2 will be dropped. Hence, by eliminating $\tilde{M}_1$, the previous equations become

$$\tilde{\mathbf{m}}_\mathbf{2} = R\tilde{\mathbf{m}}_\mathbf{1} + \mathbf{t}, \tag{2.36}$$

and because the cross product of two vectors is orthogonal to them both,

$$\tilde{\mathbf{m}}_\mathbf{2}^T \cdot (\tilde{\mathbf{m}}_\mathbf{2} \times \mathbf{t}) = 0 \tag{2.37}$$

$$\text{and } (\tilde{\mathbf{m}}_\mathbf{2}^T \cdot ((R\tilde{\mathbf{m}}_\mathbf{1} + \mathbf{t}) \times \mathbf{t}) = 0. \tag{2.38}$$

Therefore, the fundamental matrix, F, can be determined by

$$\begin{aligned} \tilde{\mathbf{m}}_\mathbf{2}{}^T [\mathbf{t}]_\times R\tilde{\mathbf{m}}_\mathbf{1} &= 0 \\ \tilde{\mathbf{m}}_\mathbf{2}{}^T F\tilde{\mathbf{m}}_\mathbf{1} &= 0. \end{aligned} \tag{2.39}$$

When the intrinsic parameters are not the identity matrix, then

$$\begin{aligned} \tilde{\mathbf{m}}_\mathbf{2}{}^T K^{-T} E K^{-1} \tilde{\mathbf{m}}_\mathbf{1} &= 0 \\ \text{and finally } F &= K^{-T} E K^{-1}, \end{aligned} \tag{2.40}$$

where E is the essential matrix.

2.4.4 Estimating the fundamental matrix

Having seen how the fundamental matrix is obtained from the known rotation and translation of a camera, the question now is how to determine F, and consequently the rotation, given two images with matching points. With the points referred to as $\mathbf{m_1} = [u_1\ v_1]^T$ and $\mathbf{m_2} = [u_2\ v_2]^T$, the epipolar equation (2.29) can

then be written as

$$\begin{aligned}
&\begin{bmatrix} u_2 & v_2 & 1 \end{bmatrix} \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix} \begin{bmatrix} u_1 \\ v_1 \\ 1 \end{bmatrix} \\
&= \begin{bmatrix} f_{11}u_1u_2 + f_{12}v_1u_2 + f_{13}u_2 + f_{21}u_1v_2 + f_{22}v_1v_2 + f_{23}v_2 + f_{31}u_1 + f_{32}v_1 + f_{33} \end{bmatrix} \\
&= \begin{bmatrix} u_1u_2 & v_1u_2 & u_2 & u_1v_2 & v_1v_2 & v_2 & u_1 & v_1 & 1 \end{bmatrix} \begin{bmatrix} f_{11} & f_{12} & f_{13} & f_{21} & f_{22} & f_{23} & f_{31} & f_{32} & f_{33} \end{bmatrix}^T \\
&= \mathbf{u} \cdot \mathbf{f} \\
&= 0
\end{aligned} \tag{2.41}$$

For n pairs of points the expression becomes

$$Uf = 0, \tag{2.42}$$

where $U = [\mathbf{u_1}, \cdots, \mathbf{u_n}]^T$. This linear homogeneous equation, and the rank-2 constraint over F, that restrains the matrix to 7 degrees of freedom (since F is also defined up a scalar factor), will permit its unique identification. Hence, the estimation of F may be done using only 7 point matches, or by using 8 or more if enough data points are available. In the latter case, the solution produced is in general unique, and there are several techniques to obtain it. In Zhang 1996's review on the issue [8], the conclusion was that linear techniques are usually sensitive to noise and not very stable, because they ignore the constraints of F and the minimization criterion is not physically meaningful. However, the results could be improved by using normalized data points instead of pixel coordinates.

Nevertheless, non-linear optimization techniques seem to yield better results to the estimation problem of F. Three nonlinear algorithms are mentioned in the review paper [8] and in R. Hartley and A. Zisserman [9]: (i) a minimization of the re-projection errors, (ii) a minimization of the symmetric epipolar error, (iii) and a minimization of the first order geometric error. The first one is the most time-consuming, thus not recommended. From the other two most promising approaches, the second seems to give the worst results and the last algorithm has been proposed to give the best results in the least amount of computational time.

Algorithm (i), also called the Gold Standard, estimates F by minimizing the distances between the points in the image and its re-projections. Having M_1 as a point in space, the corresponding points in the images before and after rotating are $\widetilde{\mathbf{m}}_1$ and $\widetilde{\mathbf{m}}_2$, respectively, obtained through (2.33) and ((2.34). The re-projections of $\widetilde{\mathbf{m}}_1$ and $\widetilde{\mathbf{m}}_2$ are then,

$$\lambda_2 \tilde{\mathbf{m}}_\mathbf{2}^\mathbf{r} = K[R\ \mathbf{t}]K^{-1}\lambda_1 \tilde{\mathbf{m}}_\mathbf{1} \tag{2.43}$$

$$\lambda_1 \tilde{\mathbf{m}}_\mathbf{1}^\mathbf{r} = K[R^T\ -R^T\mathbf{t}]K^{-1}\lambda_2 \tilde{\mathbf{m}}_\mathbf{2}, \tag{2.44}$$

where R and $\mathbf{t}$ are obtained from the factorization of F, that will be approached briefly on section 2.4.5, K are the known intrinsic parameters of the camera and λ_1 and λ_2 are given by the projection of the points on a sphere mentioned on section 2.3. The distance between $\widetilde{\mathbf{m}}_1$ and $\widetilde{\mathbf{m}}_1^r$ is defined by $||\widetilde{\mathbf{m}}_1 - \widetilde{\mathbf{m}}_1^r||$, leading to the following cost function to minimize

$$\min_{\mathbf{F}} \sum_i ||\widetilde{\mathbf{m}}_{1i} - \widetilde{\mathbf{m}}_{1i}^r||^2 + ||\widetilde{\mathbf{m}}_{2i} - \widetilde{\mathbf{m}}_{2i}^r||^2, \tag{2.45}$$

where $i = 1, ..., n$.

Algorithm (ii) minimizes the epipolar error symmetrically, which is the distance from a point to its

epipolar line. Having the epipolar line of the first image as $\widetilde{\mathbf{l}}_{e_1} = F\widetilde{\mathbf{m}}_2$ and the second's as $\widetilde{\mathbf{l}}_{e_2} = F\widetilde{\mathbf{m}}_1$, as shown on 2.32, the quantity to minimize is given by

$$\min_{\mathbf{F}} \sum_i \left(d^2\left(\mathbf{m}_{2i}, \mathbf{l}_{e_2}\right) + d^2\left(\mathbf{m}_{1i}, \mathbf{l}_{e_1}\right)\right), \tag{2.46}$$

where $d\left(\mathbf{p}, \mathbf{l}\right) = \frac{ap_1+b_2+c}{\sqrt{a^2+b^2}}$, with $\widetilde{\mathbf{l}} = (a, b, c)^T$ and $\mathbf{p} = \left(p_1, p_2\right)^T$, is the distance of a point to a line.

Minimizing $\sum_i (\widetilde{\mathbf{m_i}}'^T F \widetilde{\mathbf{m_i}})^2$ doesn't yield a good result because the variance of each i term is not the same and the least-squares technique produces an optimal solution if each term has the same variance. So one possibility is to determine the fundamental matrix the way it's given by algorithm (iii),

$$\min_F \sum_i \frac{(\widetilde{\mathbf{m}}_{\mathbf{2i}}^T F \widetilde{\mathbf{m}}_{1i})^2}{\sigma_i^2}, \tag{2.47}$$

where σ_i^2 is the variance given by

$$\sigma_i^2 = \sigma[l_{e1i_x}^2 + l_{e1i_y}^2 + l_{e2i_x}^2 + l_{e2i_y}^2]. \tag{2.48}$$

This corresponds to minimizing the derivative of the epipolar relation, thus its also called Gradient-based technique (GRAT), or the Sampson error. Because multiplying each term by a constant makes no difference, σ can be dropped.

2.4.5 Factorization of the essential matrix

Once the fundamental matrix is obtained through one of those methods, assuming the intrinsic parameters are known, which is the case for this work, the essential matrix may be obtained by $E = K^T F K$ as seen on (2.40). From here, it's possible to retrieve the camera matrix, $P = [R \mid \mathbf{t}]$, since $E = [\mathbf{t}]_\times R$, so that the rotation, R, and thus the orientation of the camera becomes known. To find the camera matrix, one can apply the SVD to the essential matrix, which generates 4 possible solutions,

$$P = \left[UWV^T | + u_3\right] \quad \text{or} \quad \left[UWV^T | - u_3\right] \quad \text{or} \quad \left[UW^TV^T | + u_3\right] \quad \text{or} \quad \left[UW^TV^T | - u_3\right], \tag{2.49}$$

that are represented by (a), (b), (c) and (d), respectively, on Figure 2.4 [3]. U and V are the outputs of SVD and

$$\mathrm{W} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad . \tag{2.50}$$

The real world point is only in front of both cameras in one of the four solutions, thus it is enough to use a single point to decide which of the different solutions produces the correct camera matrix. [9]

To summarize, the steps for obtaining the camera matrix from epipolar geometry are:

1. Determining the fundamental matrix, F, using one of the algorithms described on section 2.4.4;
2. Obtain the essential matrix, E, from the intrinsic parameters (which is possible in this case);
3. Retrieve the four possible solutions for $P = [R \mid \mathbf{t}]$ by doing a SVD decomposition on E;
4. Choose the feasible solution.

[3] The way this solutions are obtained is further detailed on appendix A.4.

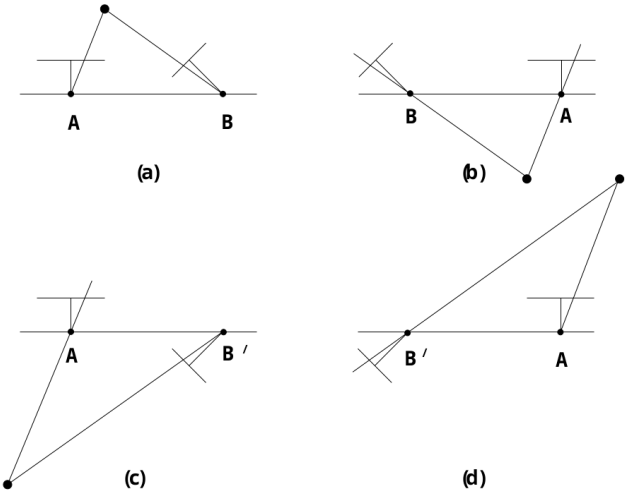


Figure 2.4: Four possible solutions retrieved from the essential matrix E. The points A and B correspond to the centers of projection before and after rotating. Between the right side, (a) and (c), and the left side, (b) and (d), figures, there is a translation direction inversion. Between top, (a) and (b), and bottom, (c) and (d), figures, there is a rotation of 180 degrees around the baseline. [9]

2.4.6 Pure rotation

As a disadvantage of epipolar geometry, if the eye's movement is not constrained by a translation component, the epipolar relation will not work, since $[\mathbf{t}]_\times$ would be a $3x3$ matrix of zeros and, consequently, $E = [\mathbf{t}]_\times R$ would yield the same, making it impossible to retrieve the rotation.

2.5 Robust Estimation and Rejection of Image Sections

Until now, the image points were assumed as given, yet it is necessary to use strong feature detection and matching algorithms to determine them. Wrongly matched pairs of points between the two images, or points with large location errors, can severely affect the precision of the rotation estimation. The reason for this is that all methods are least-square techniques that assume the noise which corrupts the data has zero mean. Hence, it is relevant to look into techniques of robust estimation.

In computer vision, it is very common to estimate the parameters of a model from image data. Robust estimation eliminates non-gaussian noise from the data. Points that don't conform to a model are called outliers and are eliminated.

RANdom SAmple Consensus

One technique of robust estimation, is RANdom SAmple Consensus (RANSAC), first introduced by Martin A. Fischler and Robert C. Bolles in 1981 [10], where a small set of inliers is used to find a model and test all the other points against it. In this manner, it's possible to discover which points fit the model or not, and if they don't consider them outliers. The final model is the one that has more inliers.

However, it is necessary to have a way of defining the said model. In the case of this work, the model could be obtained through Orthogonal Procrustes Problem applied to the point matches. More specifically, 3 point matches would be enough to estimate the rotation between images that, applied to the remaining matches, could define which ones are outliers or inliers, leading to more matching accuracy.

Besides the accuracy, there is another interesting effect produced using this technique:

1. Points that are closer to the camera are the ones that are more affected by the baseline/translation component, mentioned in section 1.3. The ones further way are a closer fit to pure rotations.
2. OPPR derives pure rotations as explained in section 2.3.

Therefore by using RANSAC and OPPR together, points matches closer to the camera, are naturally eliminated. This corresponds to image sections that would be more affected by translation and might prove beneficial to eliminate them. [11]

To summarize RANSAC works as following:

- selection of a small random sample, ”maybe inliers”;
- fit of a model to that sample;
- test of the model for the rest of the points matches. The ones that fit are ”also inliers”;
- if ”maybe inliers” + ”also inliers” are enough points for a model to be considered good, then this inlier point matches are potential outcome;
- repeat all the steps for as many tries as desired in order to gather the biggest number of inliers.

Chapter 3

Methods

This chapter presents the three most propitious strategies to determine the rotation performed by the eye, and consequently its orientation, when having two sets of matching pixel points on the images captured before and after rotating. Two of these methods are taken from the state of the art, Orthogonal Procrustes Problem (section 2.3) and Gradient-based technique (section 2.4.4), the latter rather modified. The third method, Minimization of the back projection error (MBPE), is novel, and as the second, uses the baseline constraint mentioned on section 1.3 to its advantage.

It's relevant to note for the following deductions that the baseline length, $\mathbf{b}$ and the intrinsic parameters, K, are known values taken from the current eye prototype.

3.1 Translation derivation

Because the current eye prototype is a coupled system, the translation can be obtained through the knowledge of the rotation and the baseline length associated. This length is defined as the distance from the center of the camera's sensor to the center of rotation, as shown on Figure 3.1.

The translation can be easily derived as follows using frame to frame transformations[1]. Having the world reference frame, W, set at the center of rotation, the transformation from the world to the first position of the camera, C_1, is

$$ {}^{W}_{C_1}T = \begin{bmatrix} I & -\mathbf{b} \\ \mathbf{0} & 1 \end{bmatrix}, \tag{3.1} $$

where I is the identity matrix and $\mathbf{b}$ is the baseline length expressed in each axis, X, Y and Z, as $\mathbf{b} = [b_X \; b_Y \; b_Z]$. The transformation from the world reference frame to the second position of the camera, C_2, is

$$ {}^{W}_{C_2}T = \begin{bmatrix} R & -\mathbf{b} \\ \mathbf{0} & 1 \end{bmatrix}, \tag{3.2} $$

where R is the rotation. Hence, the transformation from the first position of the camera to the second can be obtained as

$$ {}^{C_1}_{C_2}T = {}^{W}_{C_2}T \, {}^{C_1}_{W}T = {}^{W}_{C_2}T \, {}^{W}_{C_1}T^{-1} = \begin{bmatrix} R & -\mathbf{b} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} I & \mathbf{b} \\ \mathbf{0} & 1 \end{bmatrix} = \begin{bmatrix} R & R\mathbf{b} - \mathbf{b} \\ \mathbf{0} & 1 \end{bmatrix}. \tag{3.3} $$

[1] Consult Chapter 2 of Introduction to ROBOTICS mechanics and control [12] on Spatial descriptions and transformations

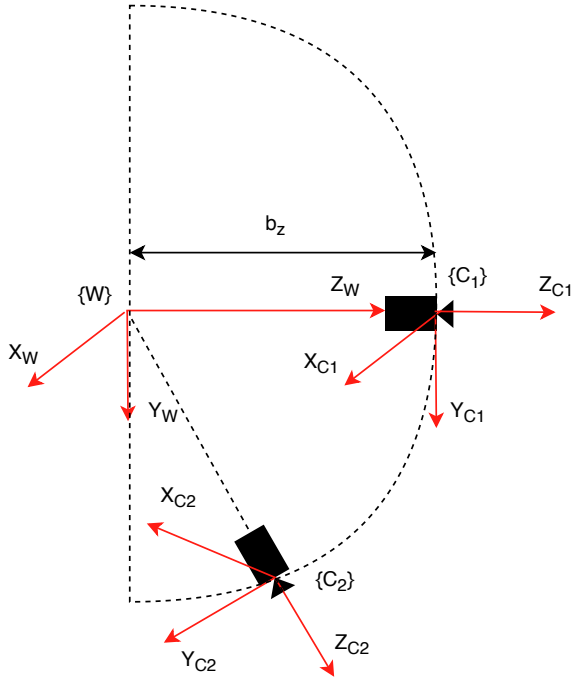


Figure 3.1: Translation derivation. Effect of a rotation on the current eye prototype's coupled system. The rotation and the baseline length define the translation. W is the world reference frame centered on the rotation center, C_1 is the reference frame of the camera in the first position and C_2 is its frame on the second position, after rotating. b_z is the baseline length on the Z axis.

And finally, the translation ends up as

$$\mathbf{t}(R, \mathbf{b}) = R\mathbf{b} - \mathbf{b} = (R - I)\mathbf{b}. \tag{3.4}$$

3.2 Minimization of the back projection error

This algorithm tries to minimize the error between the real image points and its back projections, similarly to algorithm (i) in section 2.4.4, except that it estimates the variables R and $\mathbf{t}$ directly instead of using the fundamental matrix, F, as an intermediate. This seems to prove advantageous in relation to algorithm (i), because solely 3 parameters have to be estimated rather than the whole F matrix. Three parameters are enough to define the rotation (see section 2.1) and consequently the translation, $\mathbf{t}(R, \mathbf{b})$, as is dependent.
It can be seen by the equations 2.43 and 2.44 used for the re-projections of algorithm (i),

$$\lambda_2 \tilde{\mathbf{m}}_\mathbf{2}^\mathbf{r} = K[R\ \mathbf{t}]K^{-1}\lambda_1 \tilde{\mathbf{m}}_\mathbf{1} \tag{3.5}$$

$$\lambda_1 \tilde{\mathbf{m}}_\mathbf{1}^\mathbf{r} = K[R^T\ -R^T\mathbf{t}]K^{-1}\lambda_2 \tilde{\mathbf{m}}_\mathbf{2}, \tag{3.6}$$

that λ_1 and λ_2, the depth of the corresponding 3D points, is unknown. Therefore, it has to be estimated for each point, making a total of $3 + n$ variables to estimate in the minimization. The cost function to this algorithm is then

$$\min_{\theta, \lambda_{11}^r, \ldots, \lambda_{1n}^r} \sum_{i=1}^{n} [(u_{1i}^r - u_{1i})^2 + (v_{1i}^r - v_{1i})^2 + (u_{2i}^r - u_{2i})^2 + (v_{2i}^r - v_{2i})^2] \tag{3.7}$$

where $[u_{1i}\ v_{1i}] = \mathbf{m_{1i}}$ are the pixel points in the image before rotating, $[u_{2i}\ v_{2i}] = \mathbf{m_{2i}}$ are the pixel points after rotating, $[u^r_{1i}\ v^r_{1i}] = \mathbf{m^r_{1i}}$ and $[u^r_{2i}\ v^r_{2i}] = \mathbf{m^r_{2i}}$ are its corresponding back projections, $\lambda^r_{11}, ..., \lambda^r_{1n}$ are the estimated depths and $\theta = [\theta_Z\ \theta_Y\ \theta_X]$ are Euler angles used to determine the rotation.

The back projections should be obtained by

$$\tilde{\mathbf{m}}^{\mathbf{r}}_{\mathbf{1}} = \frac{KR(\theta)^T(\lambda^r_2 K^{-1}\tilde{\mathbf{m}}_{\mathbf{2}}) - R(\theta)^T\mathbf{t}(R(\theta), \mathbf{b}))}{\lambda^r_1}$$

$$\tilde{\mathbf{m}}^{\mathbf{r}}_{\mathbf{2}} = \frac{KR(\theta)(\lambda^r_1 K^{-1}\tilde{\mathbf{m}}_{\mathbf{1}}) + \mathbf{t}(R(\theta), \mathbf{b}))}{\lambda^r_2}.$$

In order to start running the algorithm it's necessary to give it initialization parameters, λ^r_1 and λ^r_2 can be acquired from the projection of the image points in a 3D sphere, explained in section 2.3, and the Euler angles, θ, can be given by Orthogonal Procrustes Problem as it's a fast and light-weight approximation.

3.3 Gradient-based technique

On section 2.4.5, a brief summary explains that to obtain the camera matrix, R and $\mathbf{t}$, using epipolar geometry, the procedure is to determine the fundamental matrix, F, followed by determining the essential matrix, E, and finally the camera matrix through the SVD of E, choosing the most feasible solution. However, instead of obtaining R and $\mathbf{t}$ through a factorization, this can be estimated rather estimating F. Furthermore, given that $\mathbf{t}$ depends on R, like before only the 3 parameters have to be estimated.

These constraint can be used in combination with the most promising epipolar method explained on section 2.4.4, the Gradient-based technique. A similar approach is taken by Vasconcelos F. et al [13] to calibrate a camera using multiple sets of pairwise correspondences, in section "8.2 Bundle Adjustment".

The cost function for this algorithm is then

$$\min_\theta \sum_i \frac{(\widetilde{\mathbf{m}}^T_{\mathbf{2i}} F \widetilde{\mathbf{m}}_{1i})^2}{\sigma^2_i} \text{ with} \tag{3.8}$$

$$F = K^{-T}[\mathbf{t}]_\times R(\theta), K^{-1} \text{ and} \tag{3.9}$$

$$\sigma^2_i = [l^2_{e1i_x} + l^2_{e1i_y} + l^2_{e2i_x} + l^2_{e2i_y}], \tag{3.10}$$

where σ^2_i is the variance, $l_{e1i} = F\widetilde{\mathbf{m}}_2$ and $l_{e2i} = F\widetilde{\mathbf{m}}_1$ are the epipolar lines for each point i, θ are the Euler angles and $R(\theta)$ is a rotation matrix obtained through 2.1. Once again, the initialization of θ can be done using Orthogonal Procrustes Problem.

As opposed to MBPE, in GRAT the depths of the 3D points are not necessary. An explicit representation of 3D points can be avoided by minimizing the perpendicular distances between point correspondences and their epipolar lines [13] [14]. However, because this is based on the fundamental matrix, it won't work for pure rotations, as explained on section 2.4.6, where OPPR should prove more suitable. The latter algorithm should yield worst results than the previous two when there is translation associated to the rotation of the eye, as that particularity is ignored.

Chapter 4

Implementation

This chapter will start with a brief overview on how the whole system for this thesis is implemented and flows, detailing then further each of the procedures. It goes through the implementation of a simulator that is used to test the estimation algorithms in a safe and closed environment, the specifications of the real system setup and also the conception of a C++ library that deals with all of this.

As a reminder, the coordinate system convention that will be used throughout this chapter is the computer vision one mentioned in section 2.1.

4.1 Flow Overview

Figure 4.1 shows the flow of the procedures that accompany the determination of the eye rotation. Firstly two images are collected, before and after rotating. In each of the images, points that make the image unique and identifiable, called feature points, are detected. These points are then matched together between the images. Because some may be wrongly matched, as mentioned on section 2.5, they need to go through a filtering process. Afterwards, the rotation can be determined using one the algorithms presented on section 3. In order, to understand which one is best they are all put to test by determining their error against the ground truth. Besides that, the IMU data is collected to compare its performance against the camera.

This workflow is what is necessary to determine the rotation of the camera under real data sets. However, in order to evaluate the three algorithms in a controlled environment, a simulator was implemented and what it does is generate image points matches with defined rotations. This points may have added Gaussian noise, or even false matches to test the performance of the outlier filtering and the resilience of the estimator.

Both the simulator and the procedures to deal with real data were implemented in Matlab [1] and can be found in the thesis github repository[2]. A C++ library was also created to deal with the data on the real system for convenience and computational speed, and can also be found at the repository[3].

[1] https://www.mathworks.com/products/matlab.html
[2] https://github.com/Mrrvm/Orient/tree/master/Matlab
[3] https://github.com/Mrrvm/Orient/tree/master/C++

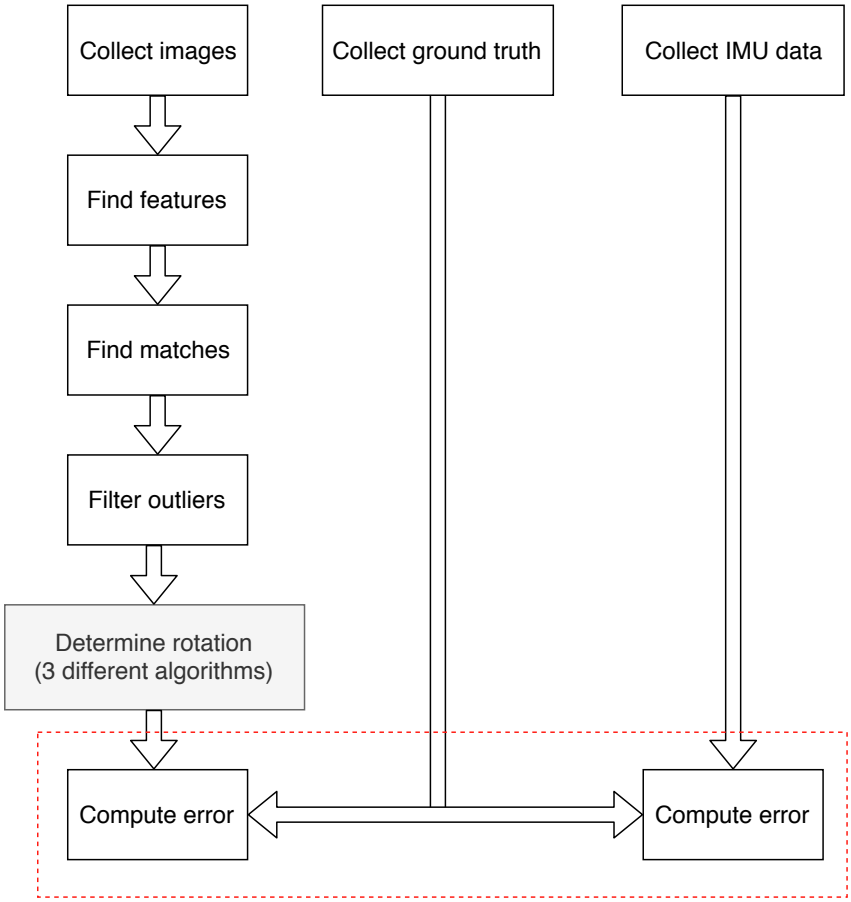


Figure 4.1: Flow Diagram. Using the RGB camera, two images are collected, before and after rotation. In each image, features are detected and matched between them. Some matches might be false or have too much noise, so they are filtered out. Using the matches information, the executed rotation is determined using an estimation algorithm. 3 algorithms will be put to test. Finally, the rotation determined is compared against the ground truth. The IMU's orientation output is also compared against the ground truth to evaluate the improvement in relation to the camera.

4.2 Simulator

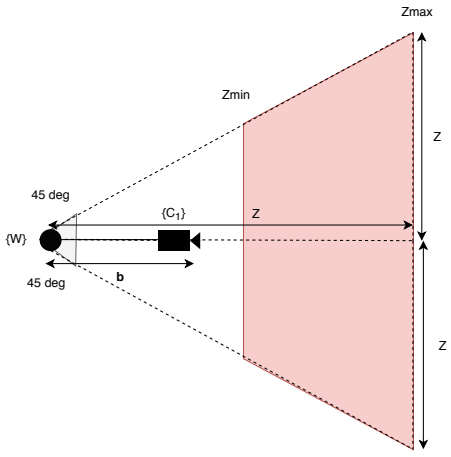


Figure 4.2: Range in which simulated points are generated

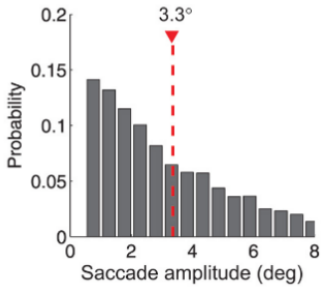


Figure 4.3: Probability of executing a specific saccade amplitude when free viewing [1]

To generate corresponding points in two simulated images, first it's necessary to pick the rotation amplitude between them. This amplitude is defined by a normal distribution with a certain variance, so

that it simulates the real environment. Poletti, M. et al [1], provides information on the probability of specific saccade amplitudes when free viewing. Free viewing is measured during normal examination of a scene, the observers freely view pictures of natural scenes, each presented for $10\ s$. The mean of the distribution is $3.3\ degrees$, shown on figure 4.3.

Afterwards, 3D points represented in the world reference frame, W, are generated within a camera viewing range of $90\ degrees$, as represented on figure 4.2 in both Y and X directions, and within a range of depth $Zmin$ and $Zmax$, in order to represent a real environment where objects are at certain distances, that may affect the accuracy of the estimation. This is enough range to simulate real eye saccades, given the amplitudes are small as seen before.

Having the points in the world frame, they are then represented in the camera reference frame before rotating, C_1, and after rotating, C_2, according to the amplitude designated between them. Finally, the 3D points are projected into the 2D image plane, and it's verified if they fit the image dimensions chosen for the simulated images.

It's possible to input Gaussian noise and also false matches into the system. For the former, random values defined in a Gaussian distribution, with a chosen variance for the simulation, are added randomly to one of the images, image before or after rotating. In the case of false matches, they are set according to the quantity chosen for the simulation, and defined as a random 2D coordinate within the image dimensions that will replace an existing correct coordinate in a random match in one of the images.

The 2D point matches are now ready to be sent to the rotation estimator.

4.3 Real system

4.3.1 Setup

As shown on Figure 1.1, the current eye prototype is composed by 6 motors that move the elastics, working as muscles. To the center of rotation seen on Figure 4.4 is attached a camera, and an IMU. The distance from the center of rotation to the camera, defined previously as baseline length, is $[0\ 0\ 53.7] \pm 3\ mm$. The camera reference frame and the IMU reference frame differ. This setup was fixed $5\ m$ away from the object with the most depth.

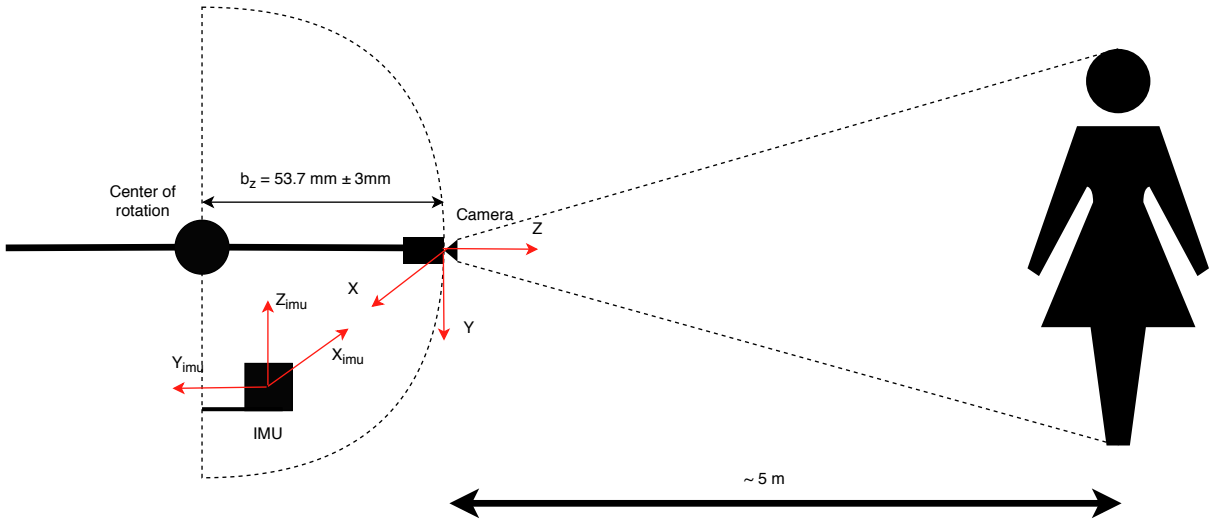


Figure 4.4: Eye prototype scheme used on the experiments. The camera is displaced from th center of rotation by b_z

The camera is calibrated[4] and the images are collected using an uEye LE USB3 camera[5] in grayscale before and after a certain rotation.

4.3.2 Collect ground truth

Using the same camera in the same positions, the two images are taken but positioning a chessboard [6] on the scenery, which is used to determine the ground truth through its regular pattern. The ground truth is then utilized to evaluate the estimation algorithms performance and also the IMU's performance over the camera. On Figure 4.5, an example of the two image pairs that are collected can be observed.

Figure 4.5: Example of the two pairs of images. The first and second images are used for finding feature matches and estimating the orientation. The third and fourth images are the same as the first and second ones, respectively, but with a chessboard positioned on the scenery, and serve as ground truth to evaluate the algorithms and the IMU's performance.

To determine the ground truth from the images with the chessboard using Matlab, the following steps are taken.

1. Checkboard points are detected using `detectCheckerboardPoints`[7] by Geiger A. et al [15] for each of the images;
2. The extrinsic parameters are determined using `extrinsics`[8] for each of the images. This function outputs a rotation matrix that allows the transformation from the world coordinate to the camera coordinate frame before rotating, ${}^{C_1}_{W}R$ and after rotating, ${}^{C_2}_{W}R$;
3. The rotation from the camera's position before rotating to its position after rotating is computed as

$$ {}^{C_1}_{C_2}R = ({}^{W}_{C_1}R)^T \cdot {}^{W}_{C_2}R \tag{4.1} $$

and used as ground truth to compare with the rotations determined by the estimation algorithms.

4.3.3 Feature detection, matching and filtering

It is necessary to gather corresponding features in consecutive images taken before and after a rotation, and to use an algorithm to estimate the orientation using those features. The latter, also referred to as keypoints or interest points, are spatial locations of an image that "stand out", allowing it to be identifiable. This points should be such that even after rotating, translating, shrinking or distorting the image, they can be found.

It is absolutely needed to have a solid detection and trust-worthy correspondence of image features (matching) when doing an "eye" movement to eventually estimate the its current orientation. Two main

[4] See appendix B.1.3 for more details on this
[5] See appendix B.1 for camera specifications.
[6] See appendix B.1.2 for chessboard specifications.
[7] https://www.mathworks.com/help/vision/ref/detectcheckerboardpoints.html
[8] https://www.mathworks.com/help/vision/ref/extrinsics.html

approaches exist to the problem of feature detection and matching. The first method finds features in one image and matches these features in the other image. The second method, which is more suitable for points in the near space, independently detects features in both images and subsequently matches them on the basis of local correspondences.

It might not be appropriate to extract features from an image having high detail (i.e. high spatial bandwidth) on the finest stable scale possible, because when matching features with another image at a different (coarser) scale those details may no longer be detectable. Therefore, it's important to extract features at a variety of scales to ensure their invariance.

Besides scale, preserving rotation and orientation invariance is also a concern. One way to deal with this problem is to define a descriptor. This is a scaled and oriented patch around the chosen point with the local orientation and scale, from which the dominant orientation can be extracted to guarantee its invariance. [9]

There exist a multitude of algorithms that can do the job. However, none of these algorithms is optimal for all images, as each of them is optimized for particular computer vision applications, with performance depending heavily on the environmental properties (illumination, image quality, contrast, ...). A comprehensive survey on the state of the art of feature detection and description [16] (by Salahat E. and Qasaimeh M. in 2017), proposes that ideal features should have the following properties:

- Distinctiveness: the gradient variations surrounding the point should be sufficiently large;
- Locality: to avoid obstruction and deformation between the two images.
- Quantity: enough features to describe the content;
- Accuracy: features should be accurate enough to be detected independently of image scale, shape or pixel location;
- Efficiency: be detected fast enough for real-time systems;
- Repeatability: a high percentage of features should be detected in both images on the overlapping regions;
- Invariance: deformative effects on the features, due to scaling, rotation or translation are minimized;
- Robustness: features should be less sensitive to deformations due to noise, blur, compression, and so on.

The review paper also presents an overview on the recent algorithms proposed on this matter, comparing them in terms of the metrics defined above. The analysis in that article suggests that Maximally stable extremal regions (MSER) and Scale Invariant Feature Transform (SIFT) algorithms enhance performance on computational complexity, accuracy and execution time. From the scale and rotation invariant algorithms, Speeded Up Robust Features (SURF), proved to be faster than SIFT, although not as robust.

MSER, by Matas J. et al in 2002 [17], are areas of uniform intensity outlined by contrasting backgrounds. They are constructed by trying multiple thresholds on the input image. The ones that remain unchanged in shape over the range of thresholds are the potential features to be selected as areas of interest. The centroids of these areas can subsequently be used to create a feature descriptor for the matching step.

SIFT (by Lowe, David G. et al in 2004) [18] algorithm detects image features using a Gaussian filter by generating increasingly blurred images, and subtracting each image to each other. This is done in

several scales in order to provide scale invariance. Afterwards, a descriptor for each feature at a certain scale is created and includes information about the orientation and gradient magnitude around the point, which also grants rotation invariance.

SURF (by Bay, Herbert et al in 1999) [19] uses an integral image, which is an intermediate representation of the original image where the value of a location is the sum of all pixels of a rectangular region formed around that location. This is called box filter and serves as an approximation to the gaussian blur, speeding up the process in relation to SIFT.

A more comprehensive explanation on both SIFT and SURF can be found at appendix A.2 and A.3, respectively. Because SURF is faster and more accessible due to patenting concerns regarding SIFT, it will be be used on this work.

4.3.4 Matching step

Regarding the matching of feature descriptors, one of the most common search algorithms used, that will also be used here, is the Nearest Neighbor Search. Fast Library for Approximate Nearest Neighbour (FLANN) [20] provides a library for performing this kind of searches in high-dimensional spaces. It contains a collection of algorithms that the authors found to work best for nearest neighbor search and a system for automatically choosing the best algorithm and optimum parameters depending on the dataset.

4.3.5 Collect IMU data

The Inertial Measurement Unit [9] obtains two quaternions (see section 2.1.2) with the current camera's orientation before and after rotating, coherently with the images. To estimate the rotation between the two positions of the camera, the rotation quaternion is simply computed by 2.7.

The LPMS-CU IMU offers a C++ library, the LpSensor library, that contains classes that allow the integration of these devices into their own applications, and also wrappers for other platforms, yet the Matlab one is still under development. Hence, the IMU data is obtained through C++ as it will be seen further ahead on section 4.7.

4.4 Filtering matches

As referred on section 2.5, robust estimation is essential to guarantee an accurate estimation by eliminating outliers that can interfere with the estimation. On that note, RANSAC with OPPR is used to filter the matches.

Regarding the summary on section 2.5, RANSAC was implemented the following way.

- 3 matches are selected for the small random sample, as only three are necessary to estimate the rotation through OPPR.
- For the matches to be inliers, the projection of the points from the camera's first position to the second through the defined model must not differ more than $5cm$.
- The quantity of matches necessary for the the model to be considered good and the quantity of iterations needed to, in principle, estimate it are designated as following.

[9] See appendix B.2 for IMU specifications.

If the whole set of matches contains a fraction ε of outliers, the probability, P, of at least one of the subsets of "inliers" being tested generating a good model is

$$P = 1 - [1 - (1 - \varepsilon)^p]^{iters}, \tag{4.2}$$

where p is the quantity of needed matches for the model to be good and $iters$ is the number of iterations. [8] Making the probability near 1, the quantity of subsets - iterations - to test is

$$iters = \frac{\log(1 - P)}{\log[1 - (1 - \varepsilon)^p]}. \tag{4.3}$$

P was considered 40% and $p = 15$, which is half of the quantity of the maximum number of matches allowed, due to speed concerns.

4.5 Determine the rotation

To estimate the rotation, the three different methods OPPR, MBPE and GRAT are tried out. For the last two, it is necessary to initialize them with OPPR as mentioned on section 3.2, and for OPPR and MBPE it is necessary to projected the points into a sphere of enough radius. OPPR is easily implemented using SVD. However, the others are non-linear unconstrained minimization problems, thus a non-linear solver is necessary.

For that, Matlab's `fminsearch`[10] was utilized. This function is an adaptation of the Nelder-Mead simplex algorithm, described by Lagarias et al. [21] `fminunc`, another possible function to use, implements the Newton method and deals with explicit boundaries. It usually is more suitable for problems with a large number of parameters and it needs to know the gradient of the cost function. However, `fminsearch` was chosen as it is simpler, gives less trouble to implement and should be enough for the current problem.

OPPR outputs a rotation matrix from the SVD. GRAT requires the fundamental matrix to be computed, thus it is necessary to have a rotation matrix previously, as expressed on 3.10. For MBPE, it's also easier to use a rotation matrix to calculate the points back projections. Hence, this rotation representation is the most suited for the problem. Yet, in order to estimate only 3 parameters instead of 9, the Euler angles representation (see section 2.1) in the ZYX convention is used as the variable to estimate. This has some more advantageous, as debug it's easier to understand how the results of the estimations differ from each other and the ground truth, and during the optimizations by converting the Euler angles into a rotation matrix, it guarantees the constraint that the matrix being determined is in fact, a rotation matrix, meaning it satisfies the conditions 2.2 and 2.3.

4.6 Compute the error

The error is then calculated as

$$error = ||\theta_{gt} - \theta_{est}||, \tag{4.4}$$

where θ_{gt} are the ground truth Euler angles and θ_{est} are the Euler angles of the estimation.

[10] https://www.mathworks.com/help/matlab/ref/fminsearch.html

4.7 C++ Library

So that in the future, it becomes more convenient to estimate the orientation of the camera on the actual prototype, which is an essential aspect of this work, a C++ library was implemented. It provides a number of classes that wrap and simplify the code, along with documentation and basic usage examples. The idea was to create it open-source, in such a way that it may help whomever uses the same hardware, or wants to do similar estimations.

It is organized into 4 different classes, `Camera`, `Sensor`, `Image` and `Rotation`, that provide all the data in the same computer vision friendly format. In other words, it makes everything compatible with the very popular Open Source Computer Vision Library[11], that contains up to 2500 optimized algorithms for this type of work.

- Class `Camera` deals with all the little details of using uEye camera software, reducing it to a couple of easy-to-use functions: `Connect()` and `Disconnect()`, `Capture()` for capturing images, `SpawnCameraError()` for debugging camera errors and finally `Calibrate()`, for calibrating the camera given a bunch of chessboard images[12].

- Class `Sensor` again wraps all the work fo dealing with the LpSensor Library, into 3 functions, `Connect()`, `Disconnect()` and `GetOrientation()`.

- Class `Image` holds an image object with which it's possible to execute several functions within the range of this work, such as `FindKeypoints()`, `FindMatches()`, `ShowMatches()` and `DetectChessboardRotation()` for obtaining the ground truth as mentioned on section 4.3.2. It essentially process the data comming from the `Camera` class.

- Class `Rotation` implements the three estimation algorithms and some auxiliary functions: `RansacByProcrustes()`, `ProjectToPlane()`, `ProjectToSphere()`, `MakeHomogeneous()` and `Estimate()` that estimates the rotation according to the algorithm chosen.
  To implement the MBPE and GRAT into C++, it was again necessary to use a non-linear optimizer. For that it was made use of Ceres Solver[13], which is a very complete and well-documented library produced by Google since 2010, that can solve non-linear least squares.

[11] https://opencv.org/
[12] See appendix B.1.3
[13] http://ceres-solver.org/

Chapter 5

Results and Discussion

5.1 Simulator

As it was specified on Figure 1.1, the current eye prototype contains a baseline length of $53.7\ mm$ on the torsional axis. To make the comparisons easier, this baseline was also used in the simulator.

5.1.1 Rotation estimation error

On Experiment 1, to simulate the saccade amplitudes, the variance used on the Gaussian distribution was $4°$, which is realistic in relation to the human eye. On experiment 2, the variance was $15°$, in order to understand how the estimation algorithm works in wider amplitude ranges.

Simulation Parameters

The following parameters were used to run the simulation. Zmax and Zmin refer to the range in which points are simulated explained on section 4.2.

- Maximum Matches : 30
- Maximum error: $0.05\ m$
- False Matches : 10% of maximum matches
- Good Matches : 50% of the maximum matches
- Gaussian Noise σ^2 : $10\ px$
- Number of saccades: 45
- Zmin : $0.05m$
- Zmax : $5m$

The lines on the graphs below represent the best fitting of the points.

Experiment 1 - Saccade amplitudes generated with $\sigma^2 = 4°$

Figures 5.1, 5.2 and 5.3, show the error, computed as 4.4, from the estimation of the rotation per saccade amplitudes on the horizontal, vertical and torsional axis, respectively. The amplitudes were generated randomly around a specific axis in a normal distribution with variance $\sigma^2 = 4°$. Table 5.1, shows the

Method	X Mean	Y Mean	Z Mean
OPPR	0.33 °	0.54 °	0.46 °
MBPE	0.31 °	0.52 °	0.44 °
GRAT	0.77 °	1.05 °	0.64 °

Table 5.1: Mean error (in degrees) per method and per axis for saccade amplitudes with variance $\sigma^2 = 4°$ under simulation.

corresponding mean error in degrees per axis for each of the methods.

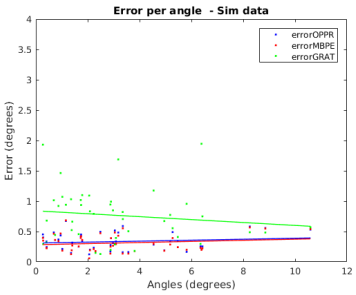


Figure 5.1: Error per saccade amplitude (in degrees) under simulation on the horizontal axis.

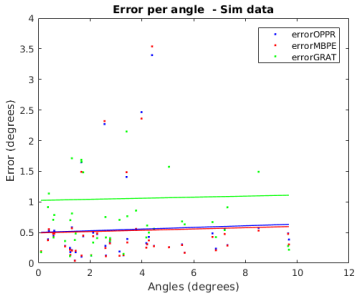


Figure 5.2: Error per saccade amplitude (in degrees) under simulation on the vertical axis.

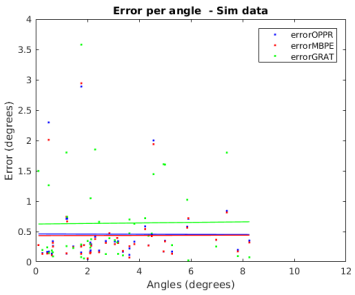


Figure 5.3: Error per saccade amplitude (in degrees) under simulation on the torsional axis.

Figure 5.4 shows that the error from the estimation of the rotation per saccade amplitudes randomly executed around all axis, simultaneously. Table 5.2 shows the mean error and standard deviation for this experiment for each of the methods.

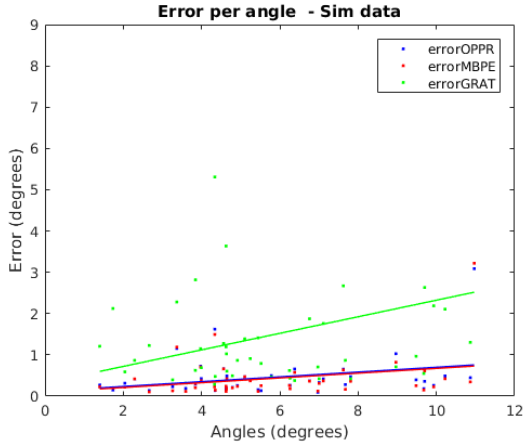


Figure 5.4: Error per saccade amplitude (in degrees) under simulation for saccade amplitudes with variance $\sigma^2 = 4°$

Method	Mean	Standard Deviation
OPPR	0.45 °	0.49 °
MBPE	0.42 °	0.50 °
GRAT	1.47 °	1.76 °

Table 5.2: Mean error and standard deviation (in degrees) of the experiment on the left per each method tested for saccade amplitudes with variance $\sigma^2 = 4°$ under simulation.

Robust estimation was applied to filter out wrong matches and detected a 15.55% of them from the whole set of matches.

Experiment 2 - Saccade amplitudes generated with $\sigma^2 = 15°$

Just like Experiment 1, Figure 5.5 shows the error per amplitude in degrees from random saccades around all axis, simultaneously, generated in a normal distribution with variance $\sigma^2 = 15°$. Table 5.3 presents the respective mean error and standard deviation. Robust estimation detected a 22.55% of bad matches.

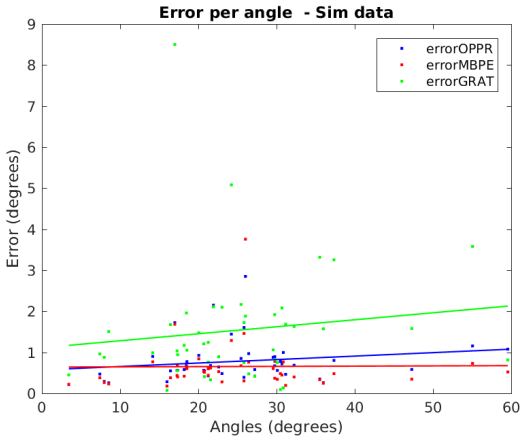


Figure 5.5: Error per saccade amplitude (in degrees) under simulation for saccade amplitudes with variance $\sigma^2 = 15°$

Method	Mean	Standard Deviation
OPPR	0.77 °	0.49 °
MBPE	0.65 °	0.60 °
GRAT	1.53 °	1.43 °

Table 5.3: Mean error and standard deviation (in degrees) of the experiment on the left per each method tested for saccade amplitudes with variance $\sigma^2 = 15°$ under simulation.

Overview

Doing the saccades per isolated axis on Experiment 1, demonstrated that there is not a significant difference in the error around a specific coordinate by looking at Table 5.1. Given that, the experiments can comfortably be executed with random saccades around all axis at the same time, which represents the real human eye more accurately.

Table 5.2 and Figure 5.4 on Experiment 1, show that for angles under $12°$ the mean error is under $0.5°$ for the best two methods, OPPR and MBPE, whereas for GRAT the error is greater. MBPE seems to be the best method in this case. However, the standard deviation is comparable to the mean for all the methods, alerting instability.

In Table 5.3 and Figure 5.5 (Experiment 2), the conditions are all the same except for the amplitude range. Still MBPE and OPPR continue to thrive over GRAT, this time having OPPR become worse as the saccade amplitude increases. This may be due to the fact that for a bigger saccade, the associated translation creates a larger effect that is not counter acted when using OPPR. Overall, the error increases as the saccade amplitude increases proportionally for all methods.

There might be several reasons as to why GRAT performs worst than the other algorithms. Because the baseline length is small, the skew translation matrix, that generates the essential matrix together with the rotation, expressed in 2.39, will have values very close to zero, that during intermediate calculus in the minimization can reduce the accuracy of the estimation. Other reason could be that, because the depth, λ, is not being estimated, opposite to MBPE (see 3.7), the leeway to estimate the rotation is reduced, ending up with a poorer guess.

5.1.2 Variable noise

To test how Gaussian noise affects the algorithms, all the same parameters as the previous section were kept and the amplitude of the saccades was set to a variance of $4°$ again, while the added noise varies from 0 to $100\ px$.

Simulation Parameters

- Maximum Matches : 30

- Maximum error: $0.05\ m$
- Good Matches : 50% of the maximum matches
- False Matches : 10% of maximum matches
- Noise σ^2 : $0 - 100\ px$ incrementing $10\ px$ per iteration
- Number of saccades: 45
- Zmin : $0.05m$
- Zmáx : $5m$
- Saccade amplitude σ^2 : $4°$

Figure 5.6 shows the error increases with the pixel noise. It can be seen that GRAT is more sensitive to noise than the others.

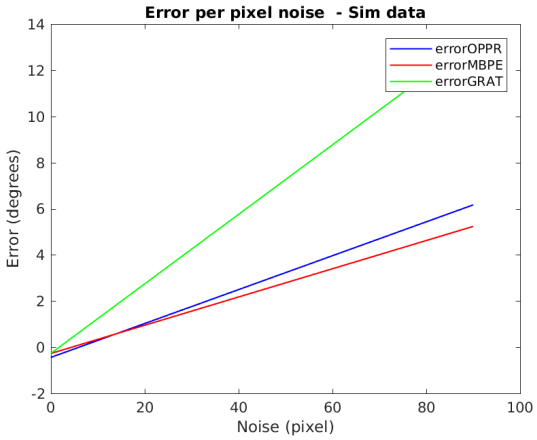


Figure 5.6: Error (in degrees) per Gaussian noise in the simulated image (in pixels).

5.1.3 Variable distance

To see how the distance from the furthest object to the camera interferes with the estimation, a simulation with the following parameters was run, with a varying distance from $5\ cm$ to $10\ m$.

Simulation Parameters

- Maximum Matches : 30
- Maximum error: $0.05\ m$
- Good Matches : 50% of the maximum matches
- False Matches : 10% of maximum matches
- Noise σ^2 : $10\ px$ incrementing $10\ px$ per iteration
- Number of saccades: 45
- Z : $0.05m$ to $10\ m$ incrementing $1\ m$ per iteration.

- Saccade amplitude $\sigma^2 : 4°$

The further away the points are, the more approximate to a pure rotation the movement becomes as explained in section 2.5. Figure 5.7 exhibits the error in degrees according to the the distance to the furthest points from the camera. OPPR and MBPE, improve with distance, probably due to rejecting image sections, which will be studied in more detail on next section. GRAT makes use of translation to get a good estimation, doesn't really gain from distance.

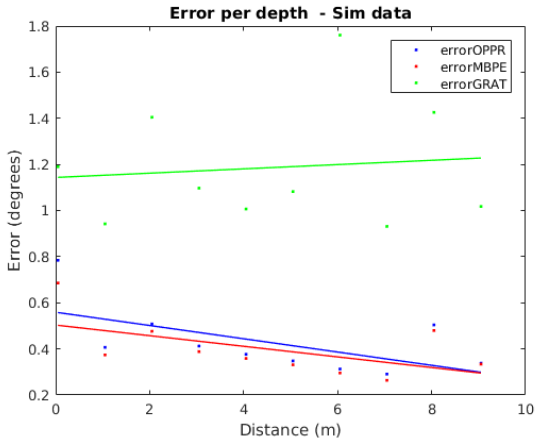


Figure 5.7: Error (in degrees) per distance to the camera in the simulated scene in meters.

5.1.4 Effect of rejecting image sections

To understand how by doing robust estimation with OPPR, the rejection of image sections is affecting the estimation of the orientation, a simulation was run with the following parameters.

Simulation Parameters

- Maximum Matches : 30
- Maximum error: $0.05\ m$
- Good Matches : 50% of the maximum matches
- False Matches : 0% of maximum matches
- Noise $\sigma^2 : 0px$
- Number of saccades: 45
- Zmin : $0.05m$
- Zmáx : $5m$
- Saccade amplitude $\sigma^2 : 15°$

By having no noise or false matches to eliminate, the effect of rejecting image section is easier to study, under simulation. The variance for saccade amplitude generation is now $\sigma^2 = 15°$ to emphasize the effect, as bigger translations are executed.

Experiment 1 - Without robust estimation

Figure 5.8 and Table 5.4 correspond to running the algorithms without robust estimation.

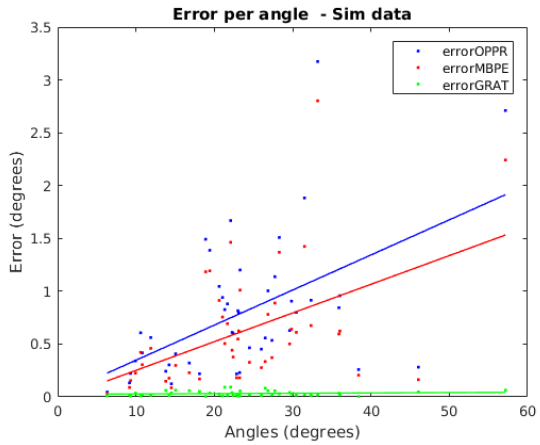


Figure 5.8: Error per saccade amplitude (in degrees) under the simulation **without** robust estimation.

Method	Mean	Standard Deviation
OPPR	0.79 °	0.63 °
MBPE	0.61 °	0.56 °
GRAT	0.03 °	0.02 °

Table 5.4: Mean error and standard deviation (in degrees) of the experiment on the left per each method tested **without** robust estimation.

Experiment 2 - With robust estimation

Figure 5.9 and Table 5.5 were ran with robust estimation.

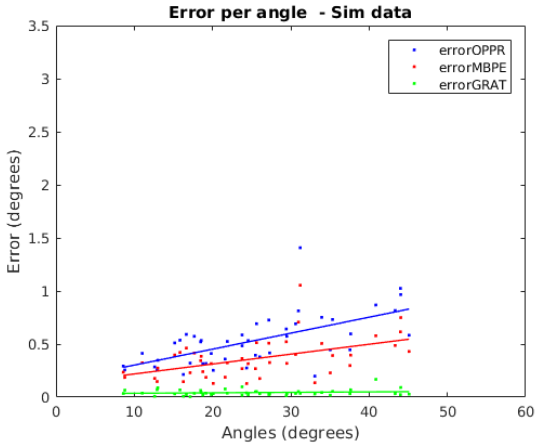


Figure 5.9: Error per saccade amplitude (in degrees) under the simulation **with** robust estimation

Method	Mean	Standard Deviation
OPPR	0.53 °	0.24 °
MBPE	0.36 °	0.18 °
GRAT	0.04 °	0.03 °

Table 5.5: Mean error and standard deviation (in degrees) of the experiment on the left per each method tested **with** robust estimation.

Overview

As it was mentioned on section 2.5, doing RANSAC+OPPR for robust estimation, yields point matches that are more approximate to a pure rotation.

When looking at the results from Experiment 1 and 2, it's possible to see that it has slightly improved by using robust estimation for the methods OPPR and MBPE. For the former, this was very expected, as it is most accurate when dealing with pure rotations, as for MBPE by taking OPPR as an initialization parameter, it is normal to also have its results improved. GRAT has a really small error now compared to the results in Figure 5.5, which probably indicates that it is extremely sensitive to Gaussian noise in the images, that is not present here. This in fact makes sense, as this algorithm uses pixel coordinates directly to estimate the rotation.

5.2 Real System

5.2.1 Data collection

To test the algorithms under the real system, 10 images of a scenery, with the setup mentioned on 4.4, were took with and without a chessboard at the same time as the IMU collected the current orientation of the eye prototype, for each experiment. Joining the images/orientations with each other made up to 45 different saccades.

5.2.2 Rotation estimation error

The algorithms were then run with the following parameters.

System Parameters

- Maximum Matches : 30
- Maximum error: $0.05\ m$
- Good Matches : 50% of the maximum matches
- Zmin : $0.05m$
- Zmáx : $5m$
- Saccade amplitude : random

The lines on the graphs below represent the best fitting of the points.

Experiment 1

In experiment 1, all the 45 rotations obtained from the ground truth were valid to be analyzed.
Figure 5.10 presents the error per saccade amplitude on the real system in degrees for the camera, and Figure 5.11 shows the same for the IMU. Table 5.6 displays the corresponding mean error and standard deviation for each method and for the IMU.

Robust estimation applied to the images detected 94.96% of wrongly paired matches.

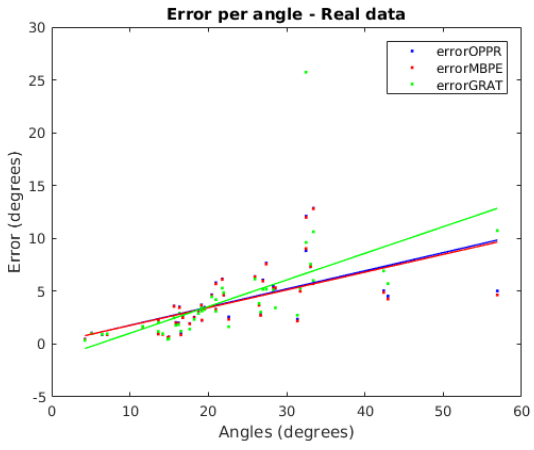


Figure 5.10: Error per saccade amplitude (in degrees) under the real system for the camera estimation.

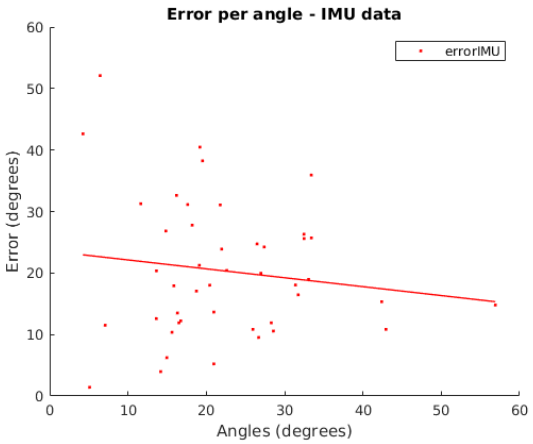


Figure 5.11: Error per saccade amplitude (in degrees) under the real system for the IMU estimation.

Method	Mean	Standard Deviation
OPPR	3.90 °	2.75 °
MBPE	3.83 °	2.75 °
GRAT	4.13 °	4.09 °
IMU	20.34 °	10.85 °
IMU without the vertical axis	11.79 °	3.57 °

Table 5.6: Mean error and standard deviation (in degrees) of the experiment on Figure 5.10 and 5.11 per each method tested.

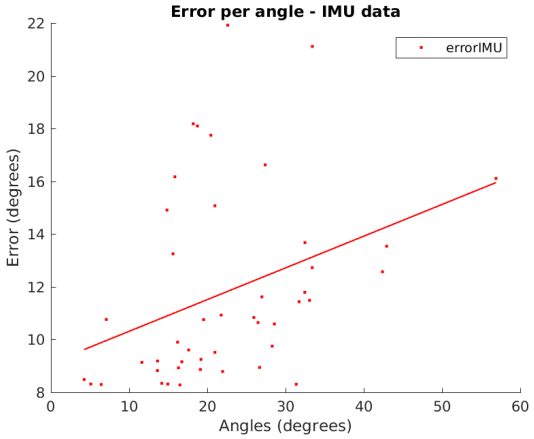


Figure 5.12: The same experiment as 5.11 but only visualizing the error on the horizontal and torsional axis per saccade amplitude (in degrees).

Experiment 2

In experiment 2, 13 of the saccades were invalid, thus only 32 are assessed. Figure 5.13 and 5.14 exhibit the error per saccade amplitude in degrees for the camera estimation and for the IMU estimation, respectively. Table 5.7 shows the corresponding mean error and standard deviation for each method and the IMU.

Robust estimation detected a 93.68% of bad matches.

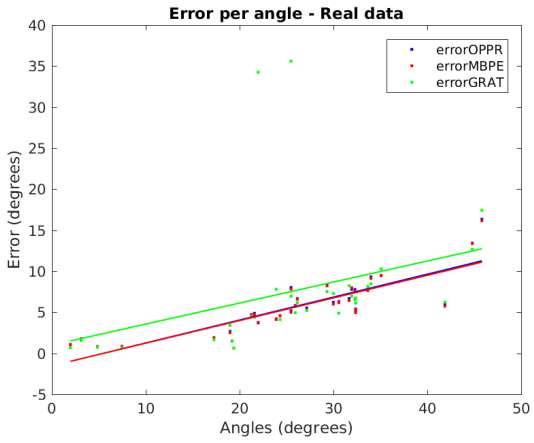


Figure 5.13: Error per saccade amplitude (in degrees) under simulation for the camera estimation.

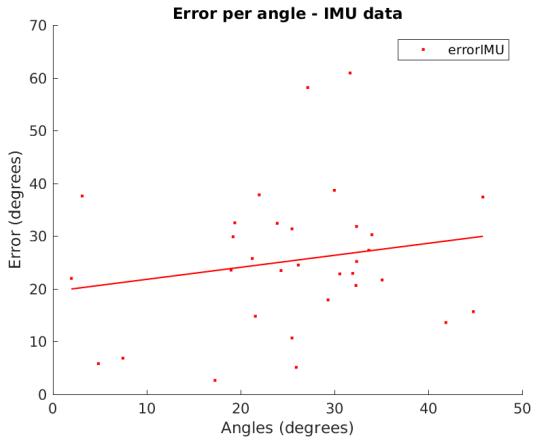


Figure 5.14: Error per saccade amplitude (in degrees) under simulation for the IMU estimation.

Method	Mean	Standard Deviation
OPPR	5.63 °	3.43 °
MBPE	5.55 °	3.40 °
GRAT	7.57 °	7.79 °
IMU	25.36 °	13.13 °

Table 5.7: Mean error and standard deviation (in degrees) of the experiment on Figure 5.13 and 5.14 per each method tested

Overview

Both Experiments 1 and 2 were the same but with two different sets of images, in order to validate the results.

Similarly to the simulation results in section 5.1.1, as seen on Figures 5.10 and 5.13, OPPR and MBPE keep being better than GRAT.

When comparing the camera estimation algorithms, in Experiment 1, the best mean error for the camera is $3.83°$, from MBPE with a standard deviation of $2.75°$. This error is probably due to noise in the image or even false matches that were not filtered out in the robust estimation. As it is real data, there is always noise associated and imposing stricter parameters for the quantity of good matches or maximum error for RANSAC (see section 4.4) will eventually not yield enough point matches to work with.

Experiment 1's estimation errors are smaller than Experiment 2, that is because the points in 1 are more concentrated around smaller angles where the error is less.

The IMU estimation results, shown on Figures 5.11 and 5.14 and on the respective Tables, manifest a huge error in relation to the camera. Besides that, this error doesn't seem to have a pattern like the camera's, where the error increases with the saccade amplitude. This outcome may be caused by unstable measurements during the experiment, as the IMU needs to stabilize its position for some time until it gives a correct orientation. It could also be justified by the drift specially when associated to rotations around the vertical axis (as it cannot use the force of gravity for the measurements). This last motivation can be backed up by Figure 5.12, that shows the same as Figure 5.11 without the rotation estimation around the vertical axis, presenting half the error as previously and an increasing pattern.

5.2.3 Effect of robust estimation

Figure 5.15 and Table 5.16 show the same Experiment 1 described in section 5.2.2, but without applying robust estimation. The mean error went up a lot compared to the same Experiment using RANSAC, most probably due to false matches.

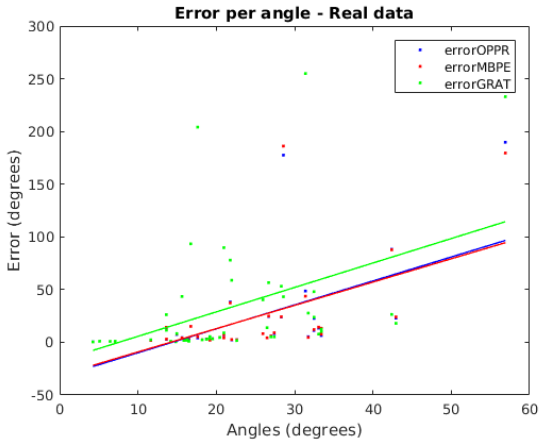


Figure 5.15: Error per saccade amplitude (in degrees) under the real system without robust estimation.

Method	Mean	Standard Deviation
OPPR	18.07 °	38.47 °
MBPE	18.09 °	38.17 °
GRAT	34.24 °	57.54 °

Figure 5.16: Error per saccade amplitude (in degrees) under the real system without robust estimation.

5.2.4 Computational time

When using the created C++ library (see section 4.7), the computational times seen on Table 5.8 were obtained for each of the processes that have to be executed to estimate the eye prototype's orientation and for the IMU estimation.

Process	Time
Image Capture	$120\ ms$
Undistort + Find Keypoints	$421\ ms$
Find Matches	$16\ ms$
Robust Estimation	$10\ ms$
OPPR	$13\text{e}{-2}\ ms$
MBPE	$212\ ms$
GRAT	$51\ ms$
IMU Estimation	$40\text{e}{-4}\ ms$

Table 5.8: Computational time for each of the necessary processes to estimate the orientation of the eye prototype in C++.

Obviously, the IMU takes much less time estimating the orientation than the camera. For the quickest algorithm, OPPR, the total amount of time taken is $567\ ms$.

5.2.5 Final remarks

Chapter 6

Conclusion

Conclusion here...

6.1 Contributions

- Empirical study on the best method to estimate orientation with the current prototype's particular constraints
- Derivation of the baseline constraint
- An orientation estimation method with better precision than an IMU
- An open-source C++ library for similar contexts within the community
- A matlab simulator

6.2 Future Work

Bibliography

[1] M. Poletti, D. Burr, and M. Rucci, "Optimal multimodal integration in spatial localization," *The Journal of neuroscience : the official journal of the Society for Neuroscience*, vol. 33, pp. 14259–68, 08 2013.

[2] J. Van Opstal. `http://www.mbfys.ru.nl/~johnvo/OrientWeb/orient_1.html#Abstract`. Accessed on October 9th, 2019.

[3] M. Lucas, "Construction and Characterization of a Biomimetic Robotic Eye Model with Three Degrees of Rotational Freedom : A Testbed for Neural Control of Eye Movements," no. October, 2017.

[4] J. Van Opstal, "200 years Franciscus Cornelis Donders," *Strabismus*, vol. 26, no. 4, pp. 159–162, 2018.

[5] J. van Opstal, A. Berthoz, "Multisensory Control of Movement: Representaion of the eye position in three dimensions," pp. 27–41, 1993.

[6] T. Haslwanter, "Mathematics of three-dimensional eye rotations," *Vision Research*, vol. 35, no. 12, pp. 1727–1739, 1995.

[7] D. G. Gower, J., "Procrustes Problems," *Technometrics*, vol. 47, no. 3, pp. 376–376, 2005.

[8] Z. Z., "Determining the Epipolar Geometry and its Uncertainty: A Review," *International Journal of Computer Vision*, vol. 27, no. 2, pp. 161–195, 1998.

[9] R. Hartley and A. Zisserman, "Multiview Geometry in Computer Vision," 2003.

[10] M. A. Fischler and R. C. Bolles, "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography," vol. 24, no. 6, 1981.

[11] D. Burschka and E. Mair

[12] J. Baillieul, "Introduction to ROBOTICS mechanics and control," 2004.

[13] F. Vasconcelos, J. P. Barreto, and E. Boyer, "Automatic Camera Calibration Using Multiple Sets of Pairwise Correspondences," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 4, pp. 791–803, 2018.

[14] J. K. Y. Ma, S. Soatto and S. Sastr, "An invitation to 3-d vision: from images to geometric models," *Springer*, 2004.

[15] A. Geiger, F. Moosmann, Ö. Car, and B. Schuster, "Automatic camera and range sensor calibration using a single shot," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 3936–3943, 2012.

[16] E. Salahat and M. Qasaimeh, “Recent advances in features extraction and description algorithms: A comprehensive survey,” *Proceedings of the IEEE International Conference on Industrial Technology*, pp. 1059–1063, 2017.

[17] J. Matas, O. Chum, M. Urban, and T. Pajdla, “Robust wide baseline stereo from maximally stable extremal regions,” *Image and Vision Computing*, vol. 22, pp. 761–767, 09 2004.

[18] D. G. Lowe, “Distinctive Image Features from Scale-Invariant Keypoints,” *International Journal of Computer Vision*, pp. 1–28, 2004.

[19] H. Bay, A. Ess, T. Tuytelaars, Gool, and L. Van, “Speeded-Up Robust Features (SURF) Herbert,” *Revue du Praticien - Medecine Generale*, no. 465 SUPPL., pp. 44–45.

[20] M. Muja and D. Lowe, “Fast approximate nearest neighbors with automatic algorithm configuration,” 2009.

[21] C. Lagarias, Jeffrey, A. Reeds, James, H. Wright, Margaret, and E. Wright, Paul, “Convergence properties of the nelder–mead simplex method in low dimensions,” *SIAM Journal on Optimization*, vol. 9, no. 1, pp. 112–147, 1998.

[22] D. Mistry and A. Banerjee, “Comparison of Feature Detection and Matching Approaches: SIFT and SURF,” *GRD Journals- Global Research and Development Journal for Engineering*, vol. 2, no. March, pp. 7–13, 2017.

Appendix A

Background

A.1 Donder's and Listing's laws

Donders' law states that the orientation of the eye, when looking in a specific direction at infinity, is always the same; in other words, as described by Donders: "with the head erect and looking at infinity, any gaze direction has a unique torsional angle, regardless the path followed by the eye to get there". Thus, the eye has 2 degrees of freedom for pointing.

Moreover, the 3D orientations of the eye can be uniquely described by head-fixed Cartesian coordinate system of single-axis rotations (section 2.1.3) that bring the eye from the primary position to the current orientation. Described in this way, Listing's law holds that the rotation axis corresponding to all possible eye orientations is confined to the yz-plane, called Listing's plane. The primary position of the eye is an unique position from which any other eye position can be reached through rotation around an axis lying in Listing's plane. This hypothetical rotation axis is defined by a vector, $\mathbf{r}$, the torsional component perpendicular to the Listing's plane, is zero. Figure A.1 illustrates that this law is very precise, by showing 3500 different eye orientations made by a head-restrained monkey, where the left image is the frontal view on the rotation axis, representing the gazed points coordinates of the horizontal and vertical components of the axis, and the right-hand plot is a side view, representing the coordinates expressed by the horizontal and torsional components. A plane is clearly seen around $0\ degrees$ of torsion with little deviation (≈ 0.6 deg). [4]

A.2 Scale Invariant Feature Transform

The SIFT algorithm consists of a local feature detector and local histogram-based descriptor. At first, SIFT takes the original image, and generates progressively blurred images through a Gaussian filter as $L(u, v, \sigma) = G(u, v, \sigma) * I(u, v)$, where (u, v) are the pixel coordinates, L is the blurred image, I is the original image, G is the gaussian blur, and finally σ is the amount of blur. Then, it re-sizes the original image to half its size, and generates the blurred images again, and repeats. Each new re-sizing of the previous image is scaled by an octave, and each octave can be blurred several times. This process ensures the scale invariance of potential features.

The set of blurred images obtained, then goes through the Difference of Gaussians (DoG) calculation, to find points of interest, or features, later on. To that end, the images from each octave (scale) are subtracted from each other pairwise, as illustrated in Figure A.2. Mathematically, what happens is obtaining $D(u, v, \sigma) = L(u, v, k\sigma) - L(u, v, \sigma)$, where k is a constant multiplicative (scaling) factor and D is the difference.

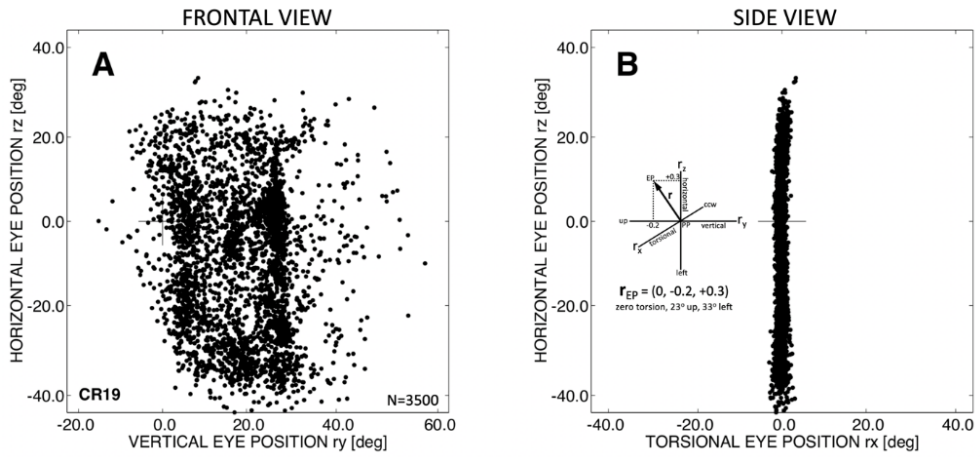


Figure A.1: Listing's law, illustrated for 3500 eye orientations made by a head-restrained monkey, freely looking around in the laboratory room. Eye fixations are represented by the head-fixed Cartesian components of the 3D rotation axes, expressed in half-radians, and calculated as $angle = 2 \cdot atan(component)$. [4]

Then, the search for local maxima/minima on the resulting images takes place. Each point is compared to its eight neighbors in the current image and nine neighbors in the image of one scale above and below, as Figure A.3 shows. The points are only selected if they are larger than all of its neighbors or smaller than all of them.

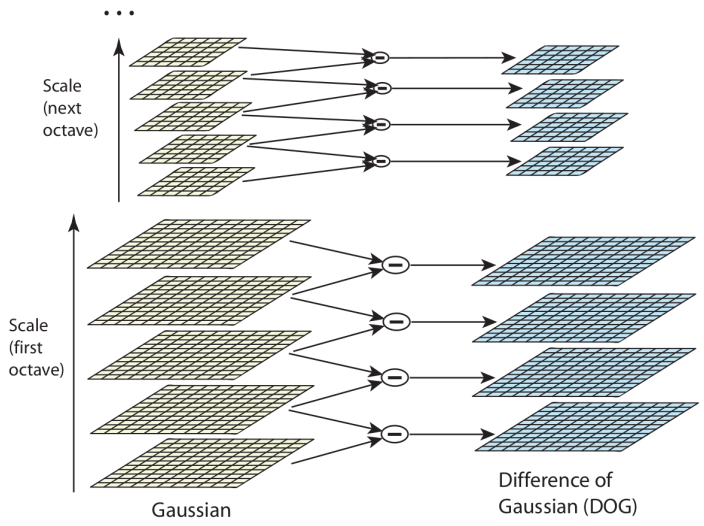


Figure A.2: Difference of Gaussians (DoG) by octave. To find points of interest (features), the Gaussian-blurred images at each scale (octaves) are subtracted pairwise. This process is computationally more efficient than computing the Laplacian of the Gaussians. [18]

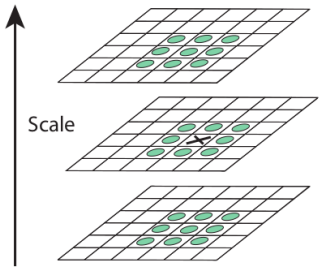


Figure A.3: Search for the maxima and minima of the DoG. After the DoG, each point is compared with its eight neigbours, and with the 9 corresponding neighbours of the images of the scale above and below. [18]

The location of this maxima/minima is an approximation since it almost never lies on the exact pixel but somewhere between pixels. Thus, the next step is to search for the feature's exact location by using a Taylor expansion (up to the quadratic terms) of the scale-space function $D(u, v, \sigma)$, shifted from the origin, according to

$$D(\mathbf{x}) = D + \frac{\partial D^T}{\partial \mathbf{x}}\mathbf{x} + \frac{1}{2}\mathbf{x}^T\frac{\partial^2 D}{\partial \mathbf{x}^2}x. \tag{A.1}$$

D and its derivatives are determined at each point by using $\mathbf{x} = (u, v, \sigma)^T$ as an offset.

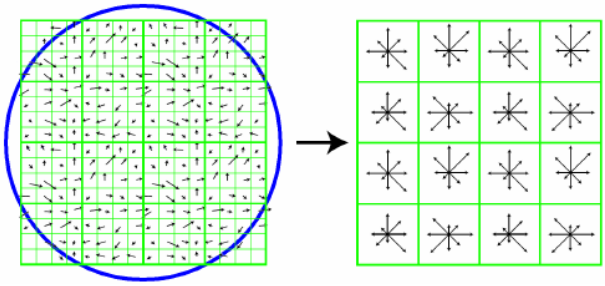


Figure A.4: SIFT feature descriptor. A window of 16×16 squares is centered around the point of interest. The blue circle represents the Gaussain window. In each 4×4 sub-region, a histogram of orientations is determined, represented on the image on the right. In total there are $8 \times 16 = 128$ values to describe the feature. [18]

Apart from removing low-contrast sub-pixels just determined, it's also important to remove edges since the DoG has a strong response to them (delta functions). An unwanted peak in the DoG will have a large gradient perpendicular to the edge but a small gradient along the edge direction, and because of that property it can be easily detected and eliminated. This is accomplished by computing a Hessian matrix on the point.

Finally, to have a rotation invariant feature, it's necessary to create an oriented patch around the point. For each image, $L(u, v)$, at a certain scale, the gradient magnitude, m_g and orientation, θ, are determined as

$$\begin{aligned} m_g(u,v) &= \sqrt{(L(u+1,v) - L(u-1,v))^2 + (L(u,v+1) - L(u,v-1))^2} \\ \theta(u,v) &= \tan^{-1}((L(u,v+1) - L(u,v-1))/(L(u+1,v) - L(u-1,v))) \end{aligned} . \tag{A.2}$$

This information is used to create a histogram of orientations. Peaks on the histogram correspond to dominant directions of local gradients. Local peaks that are within 80% of the highest peak are used to create more features with that orientation.

For the descriptor of the feature, a 16×16 window around the point of interest is set. For each 4×4 region, the orientations are put into an 8 bin histogram (each bin with a 45 degree range). This can be seen through Figure A.4. The length of each arrow corresponds to the sum of the gradient magnitudes near that direction within the region. Doing this for the 16 regions, there will be 128 values describing the feature. [18]

A.3 Speeded Up Robust Features

SURF starts from an integral image, which is an intermediate representation of the original image. The entry of an integral image $I_\Sigma(\mathbf{x})$ at a location $\mathbf{x} = (u, v)$ represents the sum of all pixels in the input image I of a rectangular region formed by the point $\mathbf{x}$ and the origin as

$$\mathrm{I}_\Sigma(\mathbf{x}) = \sum_{\mathrm{i}=0}^{\mathrm{i}\leq\mathrm{u}} \sum_{\mathrm{j}=0}^{\mathrm{j}\leq\mathrm{v}} \mathrm{I}(\mathrm{i},\mathrm{j}), \tag{A.3}$$

allowing a faster computation with box-type convolution filters.

The applied box filter is an approximation of Gaussian second-order derivatives. Instead, of having to apply the Gaussian filter iteratively to the output image and to different scales as seen on SIFT, this can be done directly with the box filter by up-scaling it and altering its masks in parallel. Figure A.5 makes a comparison between the Gaussian second-order derivatives, Laplacians L_{yy} and L_{xy}, and a box filter

applied on a point on the y and xy-direction, D_{yy} and D_{xy}.

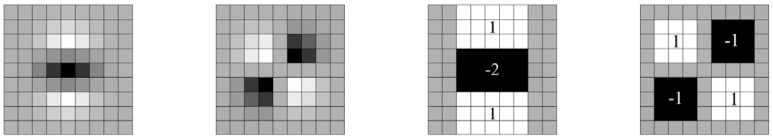


Figure A.5: Comparison between Gaussian second order derivatives (from the left, the first image in y-direction - L_{yy} - and second image in xy-direction - L_{xy}) and the corresponding images (from the left, the third image in y-direction - D_{yy} - and fourth image in xy-direction - D_{xy}) using a box filter. [19]

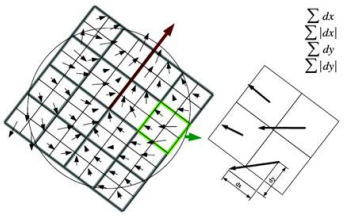


Figure A.6: SURF feature descriptor. The sampling window is rotated towards the dominant orientation. The vector $(\sum d_x, \sum d_y, \sum_x |d_x|, \sum |d_y|)$ is generated per each region through the Haar-wavelet responses in x and y direction. [19]

The Hessian matrix at point $\mathbf{x}$ and scale σ is defined as

$$\mathcal{H}(\mathbf{x}, \sigma) = \begin{bmatrix} L_{xx}(\mathbf{x}, \sigma) & L_{xy}(\mathbf{x}, \sigma) \\ L_{xy}(\mathbf{x}, \sigma) & L_{yy}(\mathbf{x}, \sigma) \end{bmatrix}, \tag{A.4}$$

where $L_{xx}(\mathbf{x}, \sigma)$ is the convolution of the Gaussian second-order derivative $\frac{\partial^2}{\partial \mathbf{x}^2} g(\sigma)$ with the image I at point $\mathbf{x}$, and similarly for $L_{xy}(\mathbf{x}, \sigma)$ and $L_{yy}(\mathbf{x}, \sigma)$, which are weighted on the calculation of the determinant by $w = \frac{|L_{xy}(1.2)|_F |D_{yy}(9)|_F}{|L_{yy}(1.2)|_F |D_{xy}(9)|_F} \approx 0.9$. The determinant is then defined as

$$\det\left(\mathcal{H}_{approx}\right) = D_{xx} D_{yy} - \left(w D_{xy}\right)^2, \tag{A.5}$$

and is used to find the local change around a point. Points where the determinant is maximal are chosen as features.

SURF's feature descriptor was designed to be scale and rotation invariant. The descriptor is sampled over a window that is proportional to the image scale, so that when a scaled version of the feature in another image is found, the descriptor will be sampled relatively, satisfying the scale-invariance requirement.

To obtain rotation invariance, the sampling window is rotated towards the dominant orientation. The window size is $20s$, where s is the scale at which the point was detected, and is divided into a quadratic grid with 4×4 square sub-regions. For each region, the Haar-wavelet responses are computed in the x and y directions. The descriptor is the sum of the x responses over its four quadrants, sum of the absolute values of the x responses and similarly for y, hence it may be defined as $(\sum d_x, \sum d_y, \sum_x |d_x|, \sum |d_y|)$. Having 4 values per region and 16 regions, yields a total of 64 values to describe the feature. Figure A.6 helps at visualizing the process.

The dominant orientation, here lightly and carelessly explained, is captured through the calculation of Haar-wavelet responses over a circular region around the point of interest with a radius of $6s$. Those responses are then summed within a sliding orientation window covering an angle of $\frac{\pi}{3}$. [19] [22]

A.4 Factorization of the essential matrix

To obtain the camera matrix from the essential matrix, the following should be considered. A 3×3 matrix is an essential matrix if and only if two of its singular values are equal, and the third is zero. This is easily proven by the decomposition of $E = SR$, where S is a skew symmetric matrix. Considering the matrices

$$\mathrm{W} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{and} \quad \mathrm{Z} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \tag{A.6}$$

where W is orthogonal and Z is a skew symmetric matrix, S may be written as $S = kUZU^T$[1], where U is orthogonal and $Z = \operatorname{diag}(1, 1, 0)W$, up to sign. Thus, $S = U \operatorname{diag}(1, 1, 0)WU^T$, up to scale, and

$$E = SR = U \operatorname{diag}(1, 1, 0) \left(WU^T R\right) = U \operatorname{diag}(1, 1, 0)V^T, \tag{A.7}$$

proving the initial statement. Because the singular values have to be equal, the SVD of E is not unique, in fact

$$E = U \operatorname{diag}(1, 1, 0)V^T, \tag{A.8}$$

or

$$E = U \operatorname{diag}(1, 1, 0)(-V)^T. \tag{A.9}$$

Considering (A.8), because

$$\begin{aligned} ZW &= \operatorname{diag}(1, 1, 0) \\ \text{and } ZW^T &= -\operatorname{diag}(1, 1, 0), \end{aligned} \tag{A.10}$$

$E = SR$ may be decomposed into two forms,

$$S = -UZU^T, \quad R = UW^T V^T, \tag{A.11}$$

or

$$S = UZU^T, \quad R = UWV^T. \tag{A.12}$$

(A.11) is a rotation matrix and can be proven as such. Since it's orthogonal,

$$R^T R = \left(UW^T V^T\right)^T UW^T V^T = VWU^T UW^T V^T = I, \tag{A.13}$$

and

$$\det\left(UW^T V^T\right) = \det(U) \det\left(W^T\right) \det\left(V^T\right) = \det(W) \det\left(UV^T\right) = 1, \tag{A.14}$$

which are enough conditions to prove it. Furthermore, S is a skew symmetric matrix because,

$$-S^T = \left(UZU^T\right)^T = UZ^T U^T = -UZU^T = S. \tag{A.15}$$

The same applies to (A.12).[2] A proof that these are the only solutions is given in Result 9.18 of R. Hartley and A. Zisserman [9].

Now regarding how to obtain the translation, since $S\mathbf{t} = [\mathbf{t}]_\times \mathbf{t} = 0$, it follows that $t = U(0, 0, 1)^T = u_3$, the last column of U, which can be explained by the following. Assuming $S\mathbf{t} = UZU^T\mathbf{t} = 0$, $ZU^T\mathbf{t}$

[1] A proof of this is given in Result A4.1 of R. Hartley and A. Zisserman,Multiple View Geometry in Computer Vision [9]

[2] C. Olsson. http://www.maths.lth.se/matematiklth/personal/calle/datorseende13/notes/forelas6.pdf. Lund Institue of Technology. Computer Vision 2013. Lecture 6.

should be equal to zero to satisfy the equation. Because Z is an orthogonal matrix,

$$ZU^T\mathbf{t} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} U^T\mathbf{t} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ s \end{bmatrix} = 0, \tag{A.16}$$

where s is any non-zero constant, such as 1. Therefore, $\mathbf{t} = U[0\ 0\ 1]^T = u_3$, as U is orthogonal as well.

Finally, having R and $\mathbf{t}$, there are 4 possible solutions for the camera matrix,

$$P = \left[UWV^T | + u_3\right] \quad \text{or} \quad \left[UWV^T | - u_3\right] \quad \text{or} \quad \left[UW^TV^T | + u_3\right] \quad \text{or} \quad \left[UW^TV^T | - u_3\right], \tag{A.17}$$

that are represented by (a), (b), (c) and (d), respectively, on Figure 2.4.

Appendix B

Material specifications

B.1 Camera

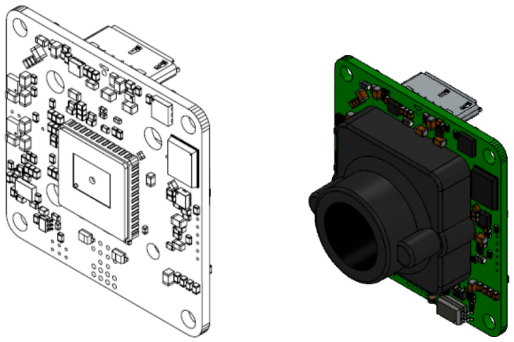


Figure B.1: On the left, the camera uEye LE's PCB. On the right, the camera mounted with the lens holder.

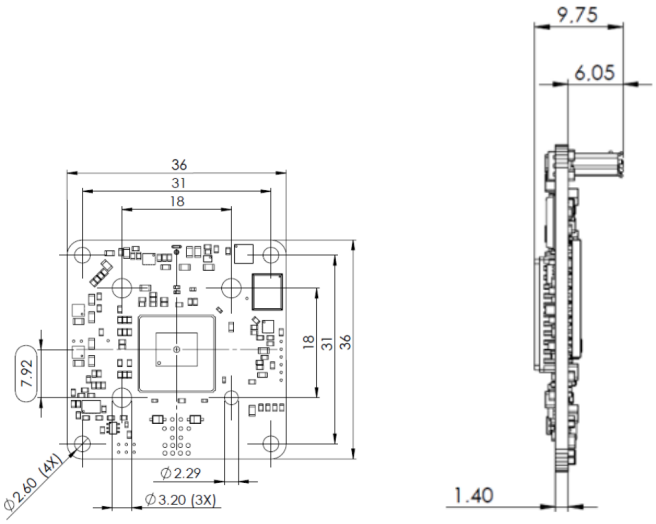


Figure B.2: Camera uEye LE USB3's PCB front view on the left and side view on the right. Measurements in millimeters.

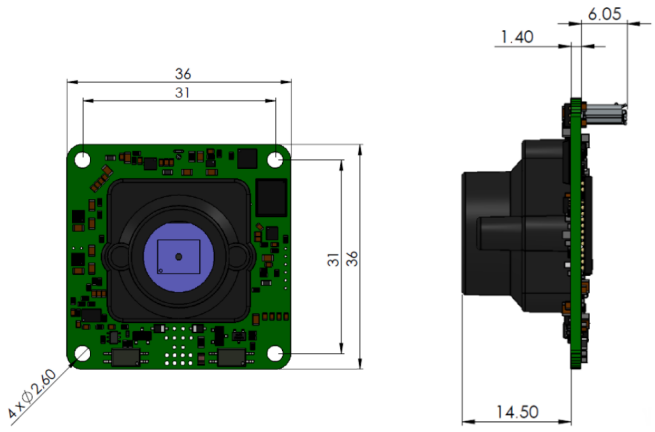


Figure B.3: Camera uEye LE USB3 with lens holder from front view on the left and side view on the right. Measurements in millimeters.

B.1.1 Lens

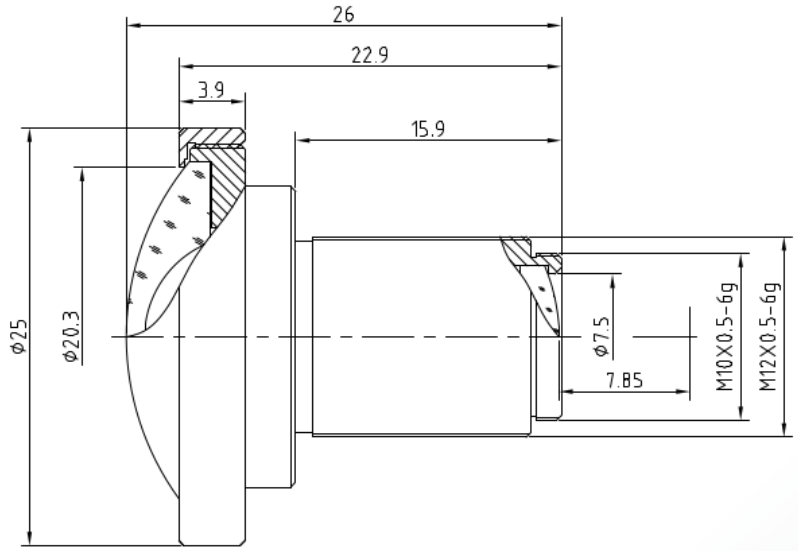
B.1.2 Chessboard

B.1.3 Calibration

intrinsics = [1.1253e+03 0.945541684501329 9.960929796822086e+02; 0 1.125630873153923e+03 7.543243830849593e+02; 0 0 1]; tanDist = [4.7003e-04 -3.3083e-04]; radialDist = [-0.3007 0.1288 -0.0318];

B.2 Inertial Measurement Unit

LP-Research Motion Sensor CAN bus and USB version (LPMS-CU) The LpSensor library contains classes that allow a user to integrate LPMS devices into their own applications



The S-Mount lens Lensagon BM2920S118 is suitable for common sensors up to 1 megapixel with an image format of 1/1.8 inch. Chief Ray Angle: 13.2°

Technical Data	
Mount	S-Mount
Sensor	1/1.8 - 1/2 inch Sensor
Focal Length	2.95 mm
Aperture	2.8
MOD	0.15 m
Back Focal	7.85 mm
Angle of View	D:178° H:138° V:104°
Megapixel	1-2 MP
IR Correction	Yes
IR Cut Filter	No
Optical Distortion	-97 %
Image Circle	8.93 mm
Weight	13 g
Iris	fixed
Focus	fixed
Typ	Standard

Figure B.4: Datasheet from lens BM2920S118 with $2.95\ mm$ of focal length.

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° about all axes	
Resolution	< 0.05 °	
Accuracy	< 2 °RMS (dynamic), < 0.5 °(static)	
Accelerometer	3-axis, ±20 / ±40 / ±80 / ±160 m/s², 16 bits	
Gyroscope	3-axis, ±250 / ±500 / ±2000 °/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 °C	
Software	C++ library for Windows, Java library for Android, LpmsControl utility software for Windows, Open Motion Analysis Toolkit (OpenMAT) for Windows	

Figure B.5: IMU specifications.