Treball de Fi de Màster

Master's degree in Automatic Control and Robotics

Gaze-tracking-based interface for robotic chair guidance

MEMÒRIA

Autor: Paula Olvera Marín
Directors: Alícia Casals Gelpí / Manel Frigola Bourlon

Convocatòria: Setembre, 2023

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

Abstract

This research focuses on finding solutions to enhance the quality of life for wheelchair users, specifically by applying a gaze-tracking-based interface for the guidance of a robotized wheelchair.

For this purpose, the interface was applied in two different approaches for the wheelchair control system. The first one was an assisted control in which the user was continuously involved in controlling the movement of the wheelchair in the environment and the inclination of the different parts of the seat through the user's gaze and eye blinks obtained with the interface. The second approach was to take the first steps to apply the device to an autonomous wheelchair control in which the wheelchair moves autonomously avoiding collisions towards the position defined by the user. To this end, the basis for obtaining the gaze position relative to the wheelchair and the object detection was developed in this project to be able to calculate in the future the optimal route to which the wheelchair should move. In addition, the integration of a robotic arm in the wheelchair to manipulate different objects was also considered, obtaining in this work the object of interest indicated by the user's gaze within the detected objects so that in the future the robotic arm could select and pick up the object the user wants to manipulate.

In addition to the two approaches, an attempt was also made to estimate the user's gaze without the software interface. For this purpose, the gaze is obtained from pupil detection libraries, a calibration and a mathematical model that relates pupil positions to gaze.

The results of the implementations have been analysed in this work, including some limitations encountered. Nevertheless, future improvements are proposed, with the aim of increasing the independence of wheelchair users.

Acknowledgements

I would like to express my deepest gratitude to all those who have helped me in the completion of this master's thesis and my academic stage at the Polytechnic University of Catalonia.

To my tutors, Alícia Casals Gelpí and Manel Frigola Bourlon, for tutoring this work and guiding me.

To all my family and friends, who have supported and encouraged me in this process.

And finally, to my fellow master's students, especially my group of friends who have accompanied, encouraged and helped me in my master's studies.

Table of Content

ETSEIB

Chapter 1

1. Introduction

1.1 Origin of the project

This project arises from a reflection. If the future is moving towards a more autonomous world where humans delegate repetitive and simple tasks in areas such as industry, transportation, or health to automatic, computerised, or artificial intelligence systems, why not also use them to assist people who, due to mobility impairment, cannot move and comfortably carry out their daily routine?

This master's degree project is part of a research project that arises from the need to improve the mobility of people with physical disabilities. Specifically, to create a robotic wheelchair that can incorporate different control interfaces such as control of the wheelchair's movement based on brain, voice or eye signals and a robotic arm to help users interact with the environment. Furthermore, the use of sensors to perceive the environment and artificial intelligence to move autonomously are also envisaged.

As can be seen, the research project attempts to cover different technologies, some more innovative than others, but all with the peculiarity that they need time to be developed and incorporated into an actual robotised wheelchair. As it is so extensive, this final master's degree project focuses on the research, programming, and incorporation of a gaze-based interface for the control of a robotized wheelchair.

1.2 Motivation

The personal motivation that has led me to carry out this project is to be able to contribute my bit to such an important and interdisciplinary research project as the creation of a robotic wheelchair and indirectly or directly to be able to influence a society that considers and assists people with physical disabilities. In addition, the wheelchair control systems chosen for this project are directly related to some of the subjects of the master's degree in Automation and Robotics, such as computer vision, machine learning and medical robotics, helping me to strengthen and improve my knowledge. Furthermore, it is convenient to know Python language, as the software that incorporates the glasses that take images of the pupils and

extract information is based on this programming language. This requirement has helped me to practice and develop a project with this new language.

1.3 Project context

This master's thesis is part of a research project whose purpose is to implement different devices that allow wheelchair users, especially users with upper and lower limb mobility problems, to be able to control a robotic wheelchair. The type of control to be implemented in the research project is an autonomous control system in which the user does not intervene continuously in controlling the movement of the wheelchair but gives the desired position where he/she wants to go, and the wheelchair performs the movement autonomously avoiding any collision in an unknown environment. In addition, this robotic wheelchair also will have a robotic arm that allows the user to reach and manipulate different objects with the wheelchair.

Considering the overall goal of the research project, this master's thesis primarily is focused on integrating one of the devices that facilitates understanding the user's intentions for wheelchair control. This device in question is a set of eye-tracking glasses.

Since this was the first time that this device was used in the robotic wheelchair, the first mission of this master's thesis was to try to control the different parts of the wheelchair with the device in a simple way, without considering the autonomous control system. Specifically, to control the movement of the wheels of the wheelchair and the inclination of the different parts of the seat. The purpose of this was to make a first contact as simple as possible to get to know the eye-tracking glasses and the software they use.

Once this initial contact was established, the next step was to implement an autonomous control system in the wheelchair with the eye-tracking device. To implement this control system, it was necessary to know the position in which the user was looking with respect to the wheelchair, as well as to detect obstacles in the environment, to calculate, plan and execute the optimal route to reach the desired destination without colliding with any obstacles. Since implementing an autonomous control system with the eye-tracking device requires a lot of time and testing, this work aimed to take the first steps and obtain the necessary information to calculate the optimal route in the future. The robotic arm was also considered in this master thesis, but as in the autonomous control of the wheelchair movement, only the first steps were taken. Only the object that the user was looking at, and therefore the object that he/she would be interested in picking up with the robotic arm, was obtained from the detected objects.

Finally, it was observed that for the software of the glasses to work, it was necessary to display the images captured by the device on a screen. This screen was not designed to be incorporated into the wheelchair of the research project to which this work belongs, as it would

not be useful for the user to observe the images and it would also be annoying to have to carry the screen in the wheelchair. For this reason, it was proposed to carry out in this project the initial steps for the adaptation of the software without the need to use the graphic interface. This first step consisted of obtaining the user's 2D gaze.

1.4 Project objectives

Considering the explanation given in the context of the project, this master's thesis has three objectives:

The first objective is to allow the user to easily control the movement and the seat positions of the wheelchair using the eye-tracking device. The idea is that the user can steer the wheelchair where he/she wants and change the inclination of the different parts of the seat.

The second objective is to take the first steps towards the implementation of an autonomous control system for the wheelchair based on the eye-tracking interface. The aim is that in the future the wheelchair will be able to move autonomously to the user's desired position without requiring constant intervention. To advance in this implementation, objects or obstacles in the environment will be detected and the position of the user's gaze in relation to the wheelchair will be obtained. In addition, it is planned to incorporate a robotic arm to the wheelchair so that the user can pick up different objects with the robotic arm, for this purpose, in the detection of the objects, the object in which the user is interested will also be obtained.

The third objective is to estimate the 2D gaze of the user without the need to graphically display the images captured by the glasses. This improvement of the interface software is oriented towards the ability to use the eye-tracking device without the need to display information and images on a screen.

In summary, the final master's thesis focuses on adapting the eye-tracking glasses for use in a robotic wheelchair, allowing the user to control the movement and configuration of the chair in an assisted control, taking the first steps towards implementing the device in an autonomous control system and improving the software of the eye-tracking device to dispense with a graphical interface.

1.5 Prerequisites

The prerequisites to be able to replicate, develop or have this project as a basis are:

- Knowledge in installing drivers to access the cameras and creating sockets to exchange data flow.

- Intermediate knowledge of Python.
- Knowledge of computer vision libraries such as OpenCV to detect features and apply matching algorithms.
- Basic knowledge of artificial intelligence and how to use the "torch" library since the convolutional network, YOLOv5, is extracted from this library.

Although it is crucial to have the prerequisites mentioned above, it is not a mandatory condition as this document tries to explain each of these points to allow someone with insufficient skills to understand the project.

1.6 Document Outline

To understand how this document is organised, this section presents the general structure of the document divided into different chapters. Each chapter provides a summary of what it consists of. The structure is as follows:

- Chapter 2 - State of the art
  This chapter investigates current robotic wheelchairs and the architecture that most of them follow. In addition, some communication interfaces between the user and the wheelchair are explained and classified according to the type of user needs.

- Chapter 3 - Project design
  Chapter 3 discusses the design of the individual modules and their control architecture as well as the hardware, software and algorithms used.

- Chapter 4 - Project implementation
  This chapter explains all the necessary steps to be able to implement each of the modules.

- Chapter 5 – Results and analysis
  The results of the project are contained in this section. In addition, the results of the individual module implementations are analysed.

- Chapter 6 – Budget and time planning
  This chapter presents the economic costs of the development of this project. This budget contains staff, energy, material and rental costs. Furthermore, this chapter presents the organisation of the different tasks necessary to carry out the project and the time to complete them.

- Chapter 7 – Social, economic, and environmental impact
  Chapter 7 analyses the social, economic and environmental impact of this project during its implementation. In addition, it also analyses the impact it may have in the future.

- Chapter 8 – Conclusions and future work
  In this chapter, the technical and personal conclusions of this work are made. Also, some ideas are presented to improve the project and try to solve all the problems observed during the implementation of the modules.

  Finally, at the end of the document, there is a list of all the bibliographies used in this master's thesis as well as a list of the tables and figures in the annexes. The code used for the implementation of each of the modules can be found at the following link: https://github.com/paulaOlvera/tfm.git

Chapter 2

2 State of the Art

2.1 Robotic wheelchair

Although it may be thought that people with disabilities and mobility problems represent a tiny part of the population, the statistics show the opposite. The World Health Organization [4] estimated that 16% of the worldwide population had a significant disability in 2020. In Spain, the National Statistics Institute (INE) declared that the first cause of disability was mobility impairment affecting 54 out of every 1000 people in 2020 [5]. In addition, the world's population is ageing and there is an increase in the number of older people with mobility problems [6]. In response to the demand this situation poses, it is necessary to provide solutions to the limitations that this type of user has.

One solution that has been used and is used by people who are paralysed in all, or part of their body is the use of wheelchairs. This mechanical device was invented a long time ago. The first recorded wheelchair dates back to 1595 and was made for King Philip II. The first self-propelled wheelchair was invented by a person with paraplegia in 1655 [1]. The motorisation of a wheelchair with electric motors and batteries was patented in the 1950s [2]. And in the last decades, the robotisation of the wheelchair started to take place.

Normally, electric wheelchairs are controlled by the user via a joystick. The problem is that some users, such as quadriplegics, cannot use this control interface because they cannot move, or control dexter enough their upper limbs and a poor control can harm themselves or the environment [7]. Studies have also shown that navigating with joysticks can cause fatigue when users use them for long periods [8].

Emphasis is currently being placed on developing wheelchairs that merge the features of a powered wheelchair with robotic technology to address the limitations or vulnerabilities that the user has with a conventional power wheelchair [3]. Such wheelchairs, known as robotic wheelchairs, demand user supervision and interaction. However, they offer the advantage of being capable of semi-automatic or fully automatic navigation within the environment.

Robotic chairs are characterised by the fact that they are equipped with technologies that allow the user to move around the environment without needing to control the wheelchair to move continuously. For example, these types of wheelchairs detect the environment and the

presence of obstacles employing sensors and, with the use of artificial intelligence, they can move automatically [3]. In addition, integrating different technologies allows this type of wheelchair to have greater interoperability and control. Other advanced technologies, such as robotic arms or brain-signal control, can be integrated into robotic wheelchairs.

Robotic wheelchairs integrate several technologies that still have the potential for development, such as vision, indoor navigation, outdoor navigation, mode selection, sensor fusion and user interfaces. The integration of these technologies is a major challenge because they do not have to fail for long periods and, in case of failure, do not endanger the user [9].

A search of the scientific literature on robotic wheelchairs shows that each research team integrates different robotic technologies. However, they all have in common that they follow the structure of Figure 1. There is human-computer interaction with multimodality of devices such as voice, EMG signals, joystick or other interfaces such as gaze tracking, the one used in this project. Then to move the wheelchair there is a planner, a controller, and actuators and to capture the environment and possible obstacles, a fusion of sensors such as bumpers, sonars, stereoscopic cameras or IR sensors [7] is used.

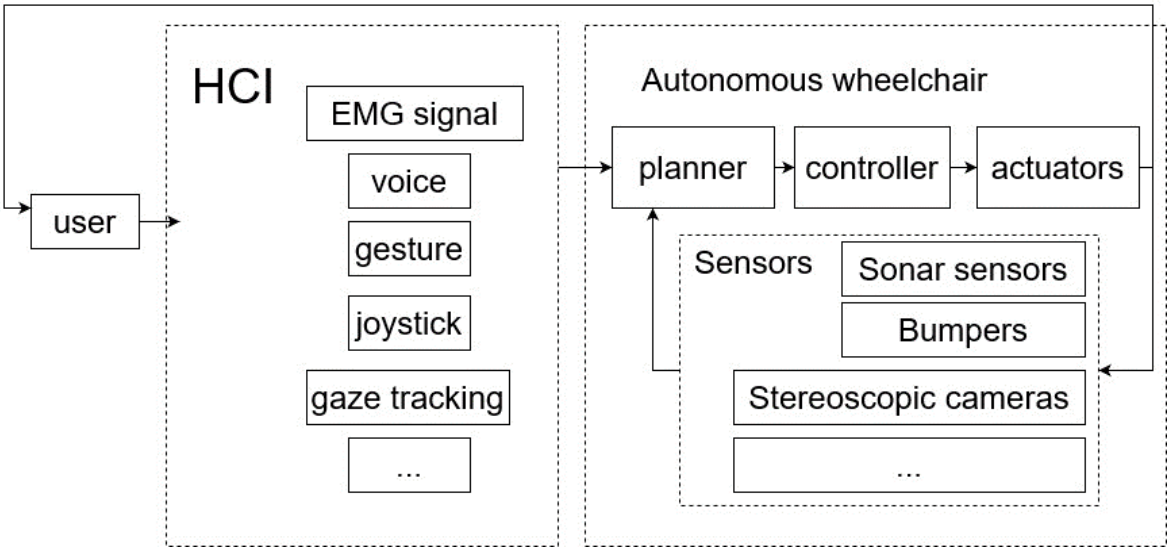

Figure 1. Architecture of a robotic wheelchair.

One point to consider when designing a robotic wheelchair is the level of autonomy to give to the user to control the wheelchair. Since both the user and the wheelchair make decisions, a control strategy is needed. The most used control mode is shared control, where the user, for example, gives a command and then the wheelchair automatically executes it in a safe way avoiding collisions and choosing a possible optimal path [8].

The general structure of a robotic wheelchair that moves in an unknown or partially known environment looks like the one shown in Figure 2. To enable movement and control of the

wheelchair, there are different elements at play.

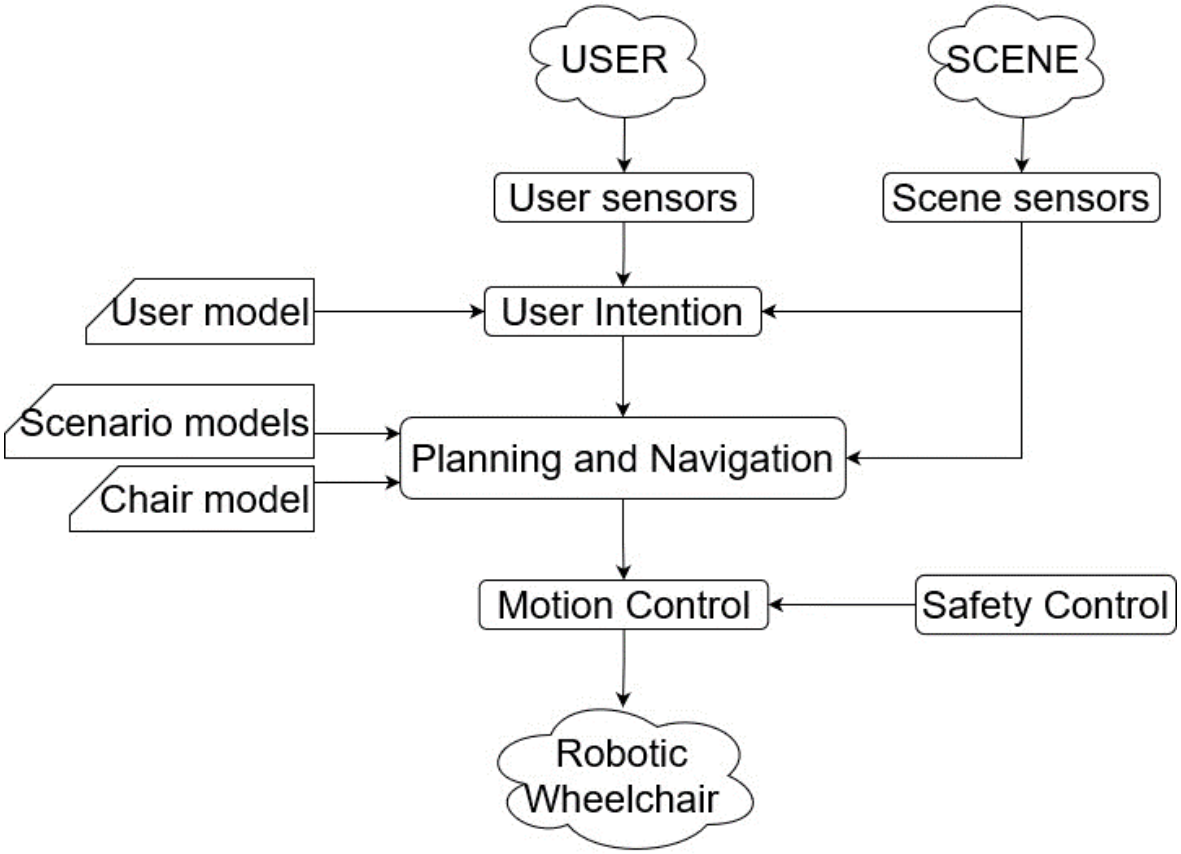

Figure 2. Robotic wheelchair structure.

One of them is sensors that capture the environment. Some of the robotic wheelchairs that have been created assume prior knowledge about the scenario or reduce the complexity of the scenario from restricted spaces or have pre-calculated the path the wheelchair must follow [8]. For a robotic wheelchair to navigate in highly changeable environments and unfamiliar scenarios, sensors are needed to sense the environment and navigate around static and dynamic obstacles. There are a wide variety of sensors that can capture the surrounding environment: ultrasonic, infrared or pulsed laser light (LiDAR) sensors, cameras or vision sensors such as stereo, depth or RGB cameras, tilt or acceleration sensors such as accelerometers or gyroscopes, contact sensors such as sensors that can measure pressure or retractable sensors such as bumpers and global positioning sensors such as GPS.

To obtain the user's intention on where they want to move or what action they want to perform, sensors are needed to capture this intention. User sensors and this would also include scene sensors, are responsible for capturing and providing information from the user and the environment to the human-computer interaction interface (HCI) and transforming it into a computerised output signal that captures the user's intention as closely and accurately as

possible to the user's true intention. In addition, human-computer interaction interfaces need the user model, which collects information about the user's characteristics, behaviours, limitations, or capabilities and serves the interfaces to modify and transform the output responses so that they are better tailored to the user to fulfil the user's will. In Section 2.2 there are explained in detail different human-computer interfaces: brain-computer, gesture-recognition, eye-tracking, and voice user interfaces.

Once the user's intention and information about the environment is obtained, the global and local path planners are executed. The global path planner is responsible for tracing the path that the robotic wheelchair will use to move in the environment or to move the robotic arm from a starting point to a destination while the local path is the path used to cope with face-to-face situations such as obstacles and avoid collisions. The essence of a local path planner is that it calculates small paths in real time and if there is any change in the scenario that makes unfeasible the current path it finds a new one [10]. In the case of this project, there is no global path planner because as the robot is used in unknown and highly changing environments it is very difficult to calculate global paths. There are different local path planners such as A*, Dijkstra's algorithm, Rapidly-exploring Random Trees (RRT), Potential Field Methods and Probabilistic Roadmaps (PRM), Dynamic Window Approach (DWA), etc. Some of them can be also used as global path planners.

For path calculation, the scenario model is used to obtain a snapshot of the current environment and relevant information on navigation decisions in real-time. For the snapshot, the scenario model is used to identify all nearby obstacles with their positions to generate and calculate an obstacle-free and feasible route, as well as to predict the velocities or accelerations of objects that are dynamic in the scenario with the help of sensors that capture the environment. Also, having a model of the scenario allows for smoother and more continuous routes, as well as the ability of the robot to quickly obtain a path when the current one is not feasible due to a change in the environment.

Once all feasible paths have been obtained, the robotic wheelchair needs to determine the trajectory and the path to follow. For this, the model of the wheelchair is needed, as it provides information about its limitations, such as the maximum speed and acceleration of the motor or the dimensions of the wheelchair. In addition, the model can provide information about the user's preferences in terms of speeds, accelerations, or trajectories to be used depending, for example, on weather conditions or time of day or any other factor that may affect the trajectory decision.

After obtaining the trajectory, the motion control executes the required movements in the wheelchair to follow the trajectory. For a safe user experience, a safety control unit is needed to stop the wheelchair in case of an emergency. For this purpose, there are different safety techniques. The best known is the emergency button which is usually mushroom-shaped and

serves to stop the wheelchair in case of an emergency immediately. When the button is pressed, priority is given to this button compared to other inputs, and it serves to stop any operation or movement of the wheelchair. Next, the bumpers are sensors around the wheelchair that are activated when the wheelchair touches an obstacle or a person due to a collision or impact. The response when the bumper signal is activated is to stop or reverse the robot's movement. The band sensor is practically the same as the emergency button, but in this case, instead of the emergency button on the wheelchair, the user has a wristband with an emergency button. Once the emergency button has been activated, a mechanism known as a rearm switch is needed to reset the wheelchair and get it moving again. The emergency sensors mentioned above do not have to be the same for all robotic wheelchairs but can be different depending on the environment and the users in which the wheelchair is to be operated.

2.2 Human-Computer Interfaces (HCI) for people with reduced mobility.

Depending on the residual capacity of the user, the assistive technologies that can be applied in the wheelchair to support the mobility of the user change. Therefore, to analyse assistive technologies, a general classification is made of the motor abilities that wheelchair users may have. In this classification, three types of users of robotic wheelchairs are distinguished:

- Users with full mobility in upper extremities and head.
- Users with partial mobility in upper extremities and head.
- Users with paralysis in all body.

For a robotic wheelchair the assistive technologies for the user are those based on human-computer interfaces (HCI) due to the interaction of the wheelchair (computer) with the user (human). Below, the interfaces that may be most appropriate depending on the motor skills of the users are discussed in more detail.

2.2.1 Full mobility in upper extremities and head.

Users with full upper limb mobility can use their hands, arms, and fingers to move the wheelchair. Therefore, this type of user can use a wide variety of HCI interfaces. The most suitable may be a joystick, keyboard, touch screen and gesture recognition interfaces, although other more complex interfaces may be used.

For example, the joystick, Figure 3, is a traditional input device that provides a fast interaction to the user, and it is easy to use [11].

Figure 3. Joystick control [12].

The keyboards and touch screens are used so that the user can control the direction of the wheelchair and the speed of the wheelchair easily and intuitively by pressing the keys or the screen.

Gesture recognition interfaces are devices or systems that interpret and recognise with algorithms human gestures. A gesture is an expressive physical movement to provide meaningful information to a receptor or interact with the environment [13]. There are different types of gestures: hand and arm, head and face and body gestures. Furthermore, a gesture can provide different information: spatial location, the path taken, and symbolic and emotional expression. There are different devices to capture information about the human body: gloves, magnetic and optical trackers, cameras, etc.

2.2.2 Partial mobility in upper extremities and head.

For this type of user, the most suitable interfaces are those in which the user does not depend on the use of hands to control the wheelchair. The most suitable interfaces are sip/puff switch, head array control, voice-user and eye-tracking interface.

The sip/puff switch, Figure 4, is a pneumatic device which depending on the air pressure produced in the inlet and outlet sip the user can drive the wheelchair [16]. The number of sips and puffs can be programmed depending on the user's capabilities.

Figure 4. Sip/Puff Switch [18].

The head array control, Figure 5, is a type of driving method for an electric wheelchair that uses the head of the user. The device design is pleasing since compared with other wheelchair devices such as the sip/puff switch there is nothing in front of the face. The downside is that the user needs to have good head control.

Figure 5. Head array control [17]

A voice user interface (VUI) is an interface that enables a person to interact with a device or system through speech [19]. The advantages of this type of interface are that can be controlled without the need to pay a lot of attention to the device since users can do another task at the same time, and it can be used hands-free. Furthermore, speaking is much faster than other types of inputs such as typing and it is flexible, there is a wide range of synonyms and expressions that mean the same thing [20]. One disadvantage of this interface is that the user does not have privacy. For instance, when a user speaks to a voice interface other people can also listen. Furthermore, the VUIs can also overhear other conversations that are not addressed to the interface. In this kind of interface is necessary to keep in mind that some users with physical impairments have difficulties speaking clearly. Therefore, a voice interface should understand broken speech and, be patient with the pauses of the speaker [21].

The eye-tracking interface is used to detect the movement of the eyes and extract information about where and how long the user is looking at some point measuring the eye position, the

movement, and the pupil size [22]. There are different techniques to track eye movement: scleral search coil, infrared oculography (IOG), electro-oculography (EOG) and video oculography (VOG). The scleral search coil is a contact lens that rests on the sclera of the eye and has two magnetic fields and two coils to measure horizontal, vertical, and torsional eye movements [23]. This technique is highly accurate but, it is an invasive method that needs anaesthesia of the eye, the lens cannot be used for long periods, and it cannot capture head movement [22]. Infrared oculography (IOG) measures the intensity of the reflection of an infrared beam to the sclera of the eye. It can be carried in dark environments but, it cannot measure torsional movement. The electro-oculography (EOG) is a technique that measures with sensors the electric voltage produced by the fluctuation of the skin when the eyes rotate. This method cannot be used in daily life, and it is sensitive to other types of noise around the eyes. And finally, video oculography (VOG) uses video cameras to detect the eye's movement. This technique allows head movement and recording but if the eyes are not close.

2.2.3 Full-body paralysis.

This type of user has the disadvantage that they cannot move their head, upper and lower limbs. In this case, the interfaces that might be best suited are brain computer interfaces or interfaces specifically created for the user's disability.

The brain-computer interface (BCI), acquires and interprets the user's intention from the brain. There are different types of BCIs: non-invasive (electroencephalography (EEG), magnetoencephalography (MEG), electrooculography (EOG), partially invasive (electrocorticography (ECoG) and invasive (microelectrode arrays (MEAs). The different phases to acquire the brain signal are: first measure the brain activity or signal with the use of a headset, cap or tiny electrode arrays implanted in specific regions of the brain to be able to detect and record the signals. The second is to digitise, process and analyse the recorded signals. With the use of algorithms, the brain activity is interpreted and therefore, the user's intention. The third is to send the signal of the user's intention to an application or device to execute the command. The last phase is the feedback that lets the user know if the command sent matches the intended action [24].

2.3 Project’s interface

This final master project is based on an eye-tracking interface. Specifically, an eye-tracking device that uses video cameras (video oculography) to capture the movement of the eyes to move a wheelchair. This interface has been selected among others because the type of users for whom the wheelchair is intended are users who have no mobility from the neck down and therefore, need to move the wheelchair without using arms, hands, or fingers. In addition, the eye-tracking device allows the wheelchair to be moved with a greater range of control,

precision, and speed than other devices such as sip/puff, head array or voice interfaces where control actions are more limited and the time for the user to indicate intent is longer. Another reason for choosing this device is that the fatigue and effort for the user is less than using other devices that require head, mouth or breathing movements. In addition, the interface used is non-invasive for the user as the eye movement is done through video cameras which are located externally to the user and are only attached to the glasses.

Chapter 3

3 Project Design

As mentioned in the introduction, this master's thesis aims to achieve three objectives. To meet them, this work has been divided into four modules. Each of the modules is explained below, following the outline of the project in Figure 6.

Movement and seat position control of the robotic wheelchair (M1)

The first module is responsible for making the first contact with the eye-tracking device. For this purpose, an easy control system is created in which, through the position of the gaze, the user can control the movement of the wheelchair as well as the different reclining positions of the different parts of the seat. To be able to switch from controlling the movement of the wheels of the wheelchair to the position of the different parts of the seat, the blinking of the eyes is used so that the user can continuously control the wheelchair without having to use the limbs.

To obtain the user's gaze in 3D, following Figure 6, it is necessary to use the software of the glasses. With this software, the pupil positions of the left and right eye images captured by the cameras of the eye-tracking device worn on the user's head are obtained. To obtain the corresponding gaze position using the pupil positions of each eye, a calibration routine is required. The user must look at different markers that appear on the screen of a monitor to obtain the corresponding gaze position. With this routine, the relationship between the position of the user's pupils and the position of the markers (gaze) is obtained. Once the reference positions have been obtained, the gaze is determined from the pupil positions. In addition to gaze, eye blinking is also required, which is also detected by the software of the device.

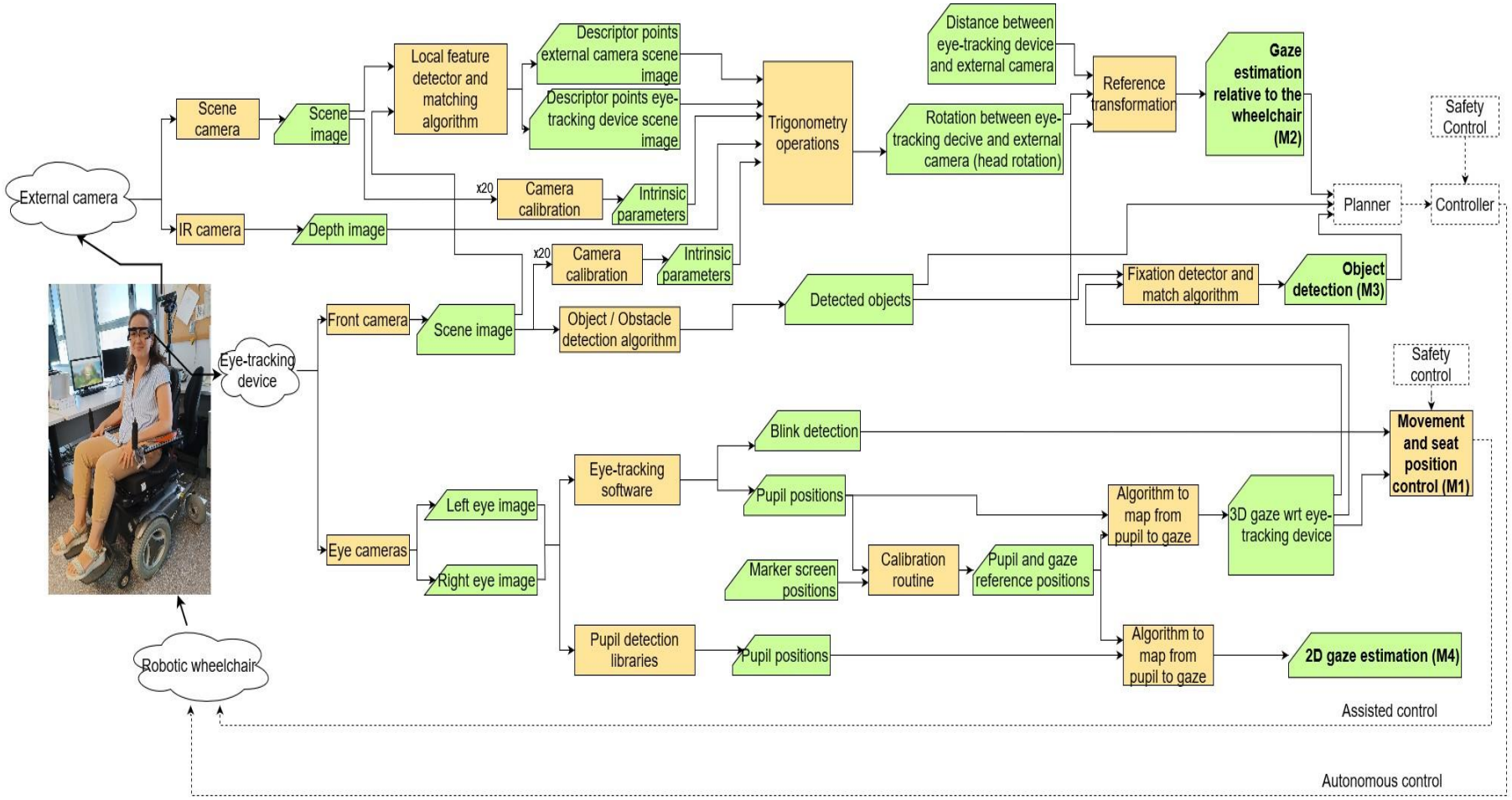

Figure 6. Outline of the project and the different modules.

Gaze estimation relative to the wheelchair (M2)

Module 2, along with Module 3, are dedicated to fulfilling the second objective, which is to initiate the development of autonomous wheelchair control. The primary role of this module is to obtain the gaze position relative to the wheelchair, providing the groundwork for the use of motion planners and controllers in the future. For the planner to effectively trace trajectories, it needs to know the user's intended destination. The user's intended destination is obtained from the gaze position provided by the eye-tracking glasses. This position needs to be transformed to obtain it with respect to the wheelchair because otherwise the trajectories would not be well calculated by the planner and therefore, the wheelchair could collide with obstacles, people, or objects.

According to Figure 6, an external camera and an eye-tracking device are needed to obtain this gaze estimation. This external camera is attached to the wheelchair as shown in the figure and focuses on the environment. This external camera has two cameras, an RGB camera and an infrared camera. The infrared camera captures depth images while the RGB camera captures colour images of the scene. To obtain the gaze position with respect to the wheelchair, a transformation of the reference system of the front camera of the eye-tracking device is necessary, as it is the only device that captures the user's gaze, but not with respect to the wheelchair. This transformation is a rotation and a translation of the reference system of the front camera of the eye-tracking device. The translation is obtained by measuring the distance between the cameras, while the rotation requires several steps.

The first step to be performed is to calibrate the front eye-tracking camera as well as the external camera capturing the colour scene to obtain the intrinsic parameters of the cameras that will be necessary to perform the trigonometric operations that will obtain the rotation angles. This calibration is performed by capturing 20 or more images of a chessboard in different positions. Once the calibrations are performed, the scene images taken by the external camera are compared with the scene images taken by the eye-tracking device using a local feature detector and a matching algorithm. With these algorithms, the most characteristic points known as descriptor points shared by the two images are obtained. The positions of these points in each image are used together with the intrinsic parameters and the depth image, and by means of trigonometric operations, the rotation angles are determined. Once the rotation angles have been obtained, the rotation matrix is constructed and, together with the translation matrix, the estimation of the gaze with respect to the wheelchair is calculated. As can be seen in the figure, the 3D gaze estimation relative to the eye-tracking device is obtained with the same steps discussed in the previous module, using the device’s software, and performing a calibration with markers before obtaining the gaze.

Object detection (M3)

This module is a further step in the creation of autonomous control. To be able to know the environment and apply autonomous control, it is necessary to detect entities such as objects or people. Thus, in planning, if obstacles are considered, they can be avoided, or a different route can be planned depending on the type of obstacles observed. In addition, in the case of the robotic arm, the recognition of what kind of object it is or whether it is a person is fundamental because it identifies whether it is something it can pick up or not.

As can be seen in Figure 6, for object detection only the front camera of the eye-tracking device is used and after applying the detection algorithm the objects of the scene image captured by the front camera are obtained. These detected objects can be considered obstacles if they impede the movement of the wheelchair towards the target position given by the user. These obstacles will be provided to the planner so that it can calculate the different routes to avoid them. Once the objects have been detected, the object on which the user fixes his or her gaze and is most interested is also obtained to in a future use this information in the control system of the robotic arm. For this, the user's gaze is obtained with the software of the glasses and the fixations are determined from the user's gaze. If the user fixes his gaze on a detected object, the result is the class name of the object.

2D Gaze estimation (M4)

This module is connected to the attainment of objective 3. This module has been created because the software of the eye-tracking device needs to have an open graphical user interface to work. As having a screen next to the wheelchair user does not add functionality and can be uncomfortable, the code of the software has been adapted to estimate the 2D gaze of the user without the graphical user interface.

Following Figure 6, to estimate the user's gaze without the software, first of all, images of the eyes need to be obtained from the eye-tracking device. Once obtained, the position of the pupils in these images is detected using pupil detection libraries. To apply the algorithm that relates the pupil positions to the gaze, a marker calibration is needed to obtain the reference positions between gaze and pupil positions. This marker calibration is the same as the one used to obtain the gaze in the previous modules and is carried out with the glasses software.

Chapter 4

4 Project implementation

4.1 Movement and seat position control of the robotic wheelchair (M1)

4.1.1 Hardware

Pupil Labs Core headset

To control the movement and the seat positions of the wheelchair in this module, Pupil Labs Core headset is used. These glasses have three cameras: one that observes the scene or environment and two that capture the eyes. The Pupil Core headset is connected to the laptop via USB. Some of the most important specifications of these glasses are:

- **Pupil Labs Core glasses:**
  - **Weight:** 22,75 g
  - **Sampling Frequency:**
    Eye Camera: 200 Hz with a resolution of 192x192 px.
    World Camera: 30 Hz with 1920 x 1080 px resolution, 60 Hz with 1280 x 720 px resolution, 120 Hz with 720 x 480 px resolution.
  - **Scene Camera FOV (world camera):**
    1920 x 1080 px: 139ºx83º, 1280 x 720 px: 99ºx53º, 720 x 480 px: 100ºx74º

In this case, the sample frequency selected for the glasses is 30 Hz of sampling frequency with a resolution of 1920 x 1080 px of the world camera. This resolution has been chosen because the speed at which the data is obtained is good for this module and it is the maximum resolution. The dimensions of the headset and how they look like are shown in Figure 7:

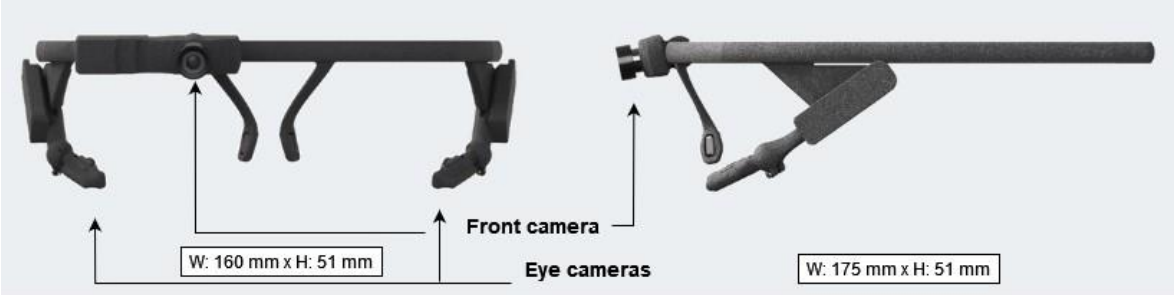

Figure 7. Pupil Labs Core headset.

4.1.2 Software

Pupil Labs Software

To perform eye tracking and capture and extract information from a person's eye movement, these glasses have an open-source software called Pupil Core. This software has two components: Pupil Capture and Pupil Player. Pupil Capture is in charge of capturing, recording and storing all the information related to the gaze, the pupil or recording specific events (for example, performing an action on the screen) in real-time. Since the robotic wheelchair is used in real-time, only the Pupil Capture component is employed. Pupil Player performs a post-hoc visualisation and analysis of the data previously saved with Pupil Core. Running the Pupil Capture programme through the app or from source generates two processes, the “Eye Process” and the “World Process”. The “Eye Process” has three steps: grab eye images from the camera video stream that records the eyes, identify the pupil position in the image and finally, broadcast or stream the detected pupil position. The “World Process” is breakdown in: first grab world camera images from world camera video stream, receive information about the pupil positions from “Eye Process”, performs a calibration mapping (converts the pupil positions to the corresponding gaze position), load plugins such as detect fixations, tracks surface or the custom plugin and records video and data. The process that Pupil Player does can be broken down in three steps: visualize the recorded data allowing users to view the recorded data, perform post-hoc analyses and computations after the data has been recorded and export video and CSV data to be able to share and save the results of the analysis.

For users who do not want to modify the code but want to perform basic functions that are already provided by the Pupil Core software, they only need to download the app and launch Pupil Capture where they will see the world window and the eye window on the screen. Then, to obtain a correspondence between pupil and gaze position, a calibration is necessary. There are different calibration routines such as one single marker or five markers. These markers are visual labels, which in the case of Pupil Labs software have a circular pattern, see Figure 9.

On the other hand, if someone wants to create a specific program and the default Pupil Core app does not have it, there are three different ways to interact with the software: using the

Pupil Core network API, creating a new Plugin or modifying the source code [29].

The Pupil Core network API allows to the user to remotely control the Pupil Core software and access all the data that the Pupil Core software provides in real time, such as gaze, fixations, or video data. In addition to receiving real-time data, you can also send data to the API. To communicate with the Pupil Core network API it is needed to create a file containing any programming language supported by *zeromq* and *msgpack*. *Zeromq* is a high-speed, low-latency socket library that allows messages to be sent efficiently within a single machine or across the network, while *msgpack* handles serialisation and deserialization of binary data. This module 1, which consists of obtaining eye gaze and eye blink and transforming it into control actions that modify the configuration of the robotic wheelchair and its movement, could be implemented using the Pupil Core network API. This could be done by subscribing to the topic that generates the eye blink and gaze information and transforming this information into the necessary control actions in another file.

It can also be done by creating a new Plugin within the Pupil Core software and when the software is running, the Plugin will run automatically in conjunction with the default plugins of Pupil Capture or Pupil Player. This module has been implemented by creating a new plugin, although the Pupil Core network API could also have been used. The source code has not been modified because the module can already be created with the network-based API or the plugin.

There are two ways to run the created plugins and the Pupil Labs software. The first is to download the Pupil Labs desktop app and place the created plugin in the "plugins" subdirectory. On the next run of Pupil Core the programme will load this file. The other option is to download the Pupil Labs code and all its dependencies to run it from source. To add the plugin, it is only necessary to access the "shared_modules" folder by navigating to "pupil->pupil_src->shared_modules" and place the plugin file there. Also, inside “launchables" folder ("pupil->pupil_src->launchables"), in the "world.py" file, it is required to declare the name of the class created in the plugin. The second option has been used for this master's thesis. To be able to run the Pupil Labs code from source, it is important to note that Python 3.11 has been used in this project although Python 3.7 or newer would also have worked. In both cases, the information as well as the link to download the software can be found in [30].

Developing a plugin

A plugin in Pupil Core is a Python file that is loaded and executed in runtime [29]. To create a new plugin and have it recognised by the Pupil Core software, it must follow the Pupil Core framework. The new plugin is managed within the main process of the application that runs the plugins (the new plugin and the default plugins of the software) via callbacks. The new plugin has to inherit the Plugin class, which can be seen on Github [31].

Then, the structure in Python is as Figure 8:

```
from plugin import Plugin

class MyCustomPlugin(Plugin):
    pass
```

Figure 8. New plugin that inherits the root class Plugin [29].

Explaining all the possible options that exist within Pupil Core software to create a new Plugin can be found in [29]. In this case, in order not to repeat the information that can be found on the internet, only the elements that are used to program the plugin for the modules and that are important are mentioned.

Plugin class attributes

The class attributes in this case determine how a plugin behaves in general.

Table 1. Plugin class attributes.

Name	**Values**	**Selected for this module 1**	**Description**
uniqueness	“not_unique”, “by_class”, “by_base_class”	“by_class”	While “not unique” plugins can have multiple instances, “unique by class” are restricted to have only one instance at a time. “by_base_class” is a plugin that if another share the same base class, only one can be activated (mutual exclusion).

One-time callbacks

These types of callbacks are only called one time in the life of the plugin.

Table 2. One-time callbacks.

Callback name	**Description**
__init__(self, g_pool, **kwargs)	It’s the first callback called when the plugin is started. It is used also, to initialise the

	variables of the program. 'g_pool' provides access to the application's functionalities. In 'kwargs' the preferences of the user can be stored.
init_ui(self)	It is called after _init_ if the plugin has a user interface. Allows the user to setup the setting menu and quick access buttons. For example, in this module it has been used so that the user can activate or deactivate the plugin created from the main programme by means of a button
deinit_ui(self)	This callback removes any UI element added during 'init_ui'.

Processing callbacks

As it has been commented before, Pupil Core executes different processes at the same time, such as plugins. In each process there is an infinite loop that processes the data in each iteration called event cycle. The data is acquired, generated, and processed by invoking the processing callbacks of Table 3.

Table 3. Processing callbacks.

Callback name	**Description**
recent_events(self, events)	It's called once per application event cycle. The 'events' variable is a dictionary containing string keys and values with built-in Python data types, like lists, dictionaries, and so on. In this module the variable 'events' is accessed to obtain information about eye blinking and 3D gaze direction.
gl_display(self)	It's invoked once for every application event cycle when the calling process has a user interface. It renders the custom visual elements declared in this callback using

	OpenGL.

Calibration routine

To obtain accurate gaze data, an eye-tracking calibration process is required [32]. This process ensures that the gaze that is picked up by the glasses matches or is as close as possible to the actual gaze of the user in the real world. The calibration process can be divided into five steps:

1. Setup: The user must place the Pupil Core headset on the head so that it is comfortable and stable on the face, as well as position the cameras so that the eyes and the scene are clearly visible in the camera images.

2. Selection of the calibration routine: As discussed above, there are different calibration routines such as single marker or five markers. In this step, the user selects the calibration routine that suits him best. In the case of the 5 markers, as can be seen in Figure 9, the markers are in different parts of the screen.

3. Marker calibration: the user is asked to look at each marker in turn. The calibration correlates the calibration markers' positions in the scene camera coordinates with certain parameters of the eye like the pupil's centre position.

4. Gaze mapping: Pupil Labs has two approaches to obtain the gaze from the calibration data: one obtains the gaze in 2D and the other in 3D. The 2D gaze mapper creates a polynomial regression model with the data obtained during the marker calibration: the input to the regression is the pupil location in normalised eye coordinates and the reference target is the direction of the gaze in the coordinates on the screen. Once the polynomial model is obtained to obtain the gaze in 2D it is only necessary to give as input the pupil location in normalised eye coordinates to the model and it will give as a result the gaze with respect to the coordinates of the scene camera in pixels. To obtain the 3D gaze, the marker calibration in this case performs a bundle adjustment to estimate the physical relationship between the eye and scene cameras and as a result, gives two matrices (for the left and right eye) that transform the eye to world coordinates. Out of the calibration, the 3D gaze is obtained using a geometrical method that intersects visual axes. This requires the eyeball position and normalised gaze in scene camera coordinates which are determined by having the eyeball position and pupil location in normalised eye coordinates and the matrices calculated in the marker calibration.

   It is essential to note that gaze tracking accuracy is optimal only at the distance for

which calibration has been performed. If the user moves or changes his/her position with respect to the calibration set reference points, the accuracy of the gaze estimation decreases, as the spatial relationship between the gaze and the calibrated reference points is altered.

The 2D gaze mapping is explained in more depth in module 4 because instead of getting the gaze directly through the Pupil Labs software as in modules 1, 2 and 3, the gaze mapper had to be created from scratch in the fourth module, see Section 0.

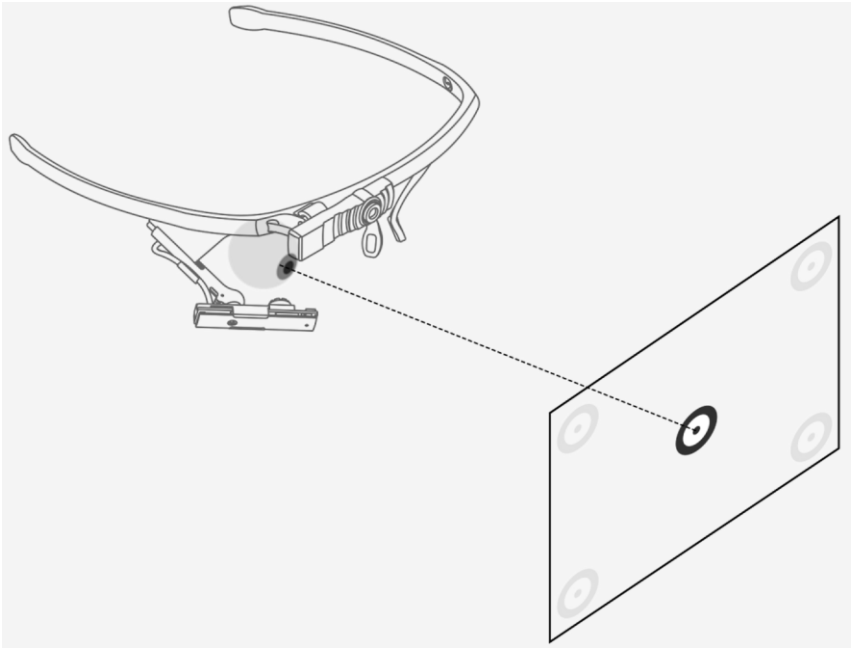

Figure 9. Calibration method in Pupil Labs [32].

4.1.3 Implementation steps

As mentioned above, it is desired to extract two types of information from the user through the eye-tracking glasses, the blinking of the eyes to change the wheelchair configuration and the position of the gaze to control the movement and the seat position of the wheelchair.

In more detail and as can be seen in Figure 10, the wheelchair in this project has four configurations or modes. The first of these, configuration "0", is the one that allows control of the wheels. The other three, control the seat position of the wheelchair. Configuration "1" controls the inclination of the backrest, configuration "2" the inclination of the middle seat and configuration "3" the inclination of the lower seat.

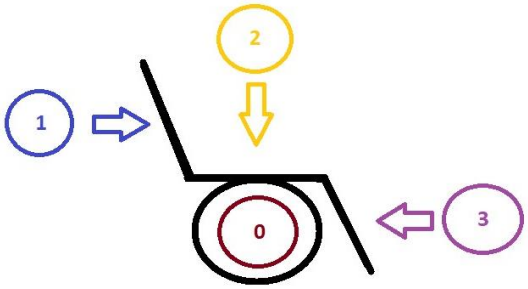

Figure 10. Modes of the robotic wheelchair in module 1.

Within each configuration, the wheelchair can perform different movements. In the case of “configuration 0" if the user's gaze is upwards but centred in the central part of the FOV (field of view), the wheelchair will move forwards. If the gaze is downwards, but centred in the central part of the FOV, it will move backwards and if the gaze is centred the wheelchair will not move. If looking up is looking to the left, the wheelchair will move forward but turning to the left. If looking up is looking to the right, the chair will move to the right. And if the user looks down when turning, the turn will be backwards. As an example, in Figure 11, the user looks up (F – Forward) and to the left (L – Left) and this will cause the wheelchair to turn to the left but move forward.

In the case of configurations "1", "2" and "3" which are used to control the positions of the different parts of the seat, given that the inclination of the seat can only be forward or backward, even if the gaze positions (left, right, neutral) are sent through a socket to the wheelchair, only those related to forward, backward and neutral are the ones that the wheelchair will use to do the motion control. For example, considering Figure 11, looking upwards to the left, if the user is in configuration "2", it will mean that the seat in the middle position will move forward but not to the left.

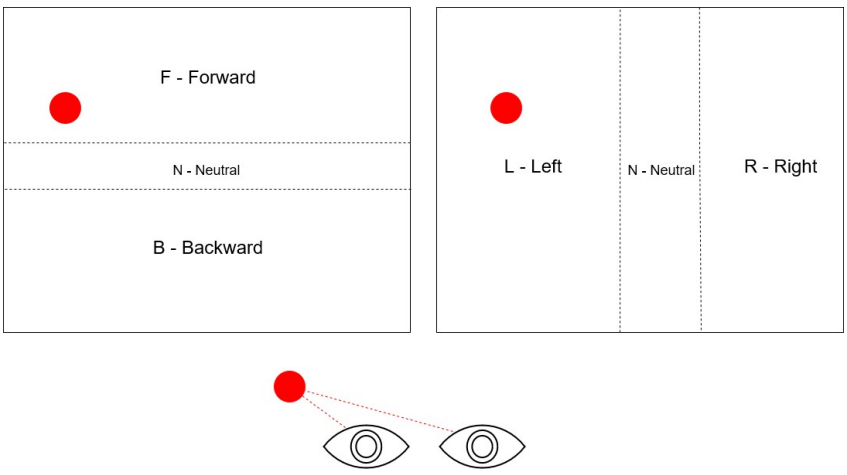

Figure 11. Gaze to control movement.

The plugin created for this module 1 is a file wrote in Python as programming language and is called or loaded by the "World Process" which is responsible for activating all plugins including the one created in this module.

The class created in this plugin collects two variables inside "events" which is a dictionary with string keys and values provided by the Pupil Labs app when executed. These two variables are eye blinking and 3D gaze. The programme must detect both eyes well through the cameras to provide the desired information, so if only one of the eyes is captured, no information can be obtained.

To obtain the eye blinks, the variable "events["blinks"]" is accessed. This variable provides different data such as the confidence of the blink or the time stamp at which the blink was made. Voluntary blinks need to be distinguished from non-voluntary blinks because as human beings our eyes need to blink. To know that the user is blinking voluntarily, he/she must blink twice in a row, because with only one blink it is impossible to distinguish whether it is a voluntary or involuntary blink. When the program is running, the wheelchair operates in "configuration 0" by default. To switch to the next configuration ("configuration 1"), the individual only needs to perform two consecutive blinks. However, if the person does not blink voluntarily, they will remain in the current configuration. It is important to note that the user cannot switch from one configuration to another that are not consecutive. For example, from configuration 1 to configuration 3. Furthermore, when the user is in "configuration 3" and wants to switch to the next configuration, he/she switches to the initial configuration, configuration 0.

The 3D gaze is obtained by accessing the variable "events["gaze"]. This variable gives information about the user's gaze in x, y and z coordinates with respect to the front camera of the Pupil Labs glasses in millimetres. To view the reference coordinates of the Pupil Labs front camera, see Figure 16. In addition to obtaining the gaze, it is possible to obtain the confidence of the gaze. The confidence gives information about how reliable the data of the gaze is. Low confidence values suggest higher tracking errors and inaccuracies while higher confidence values mean a higher degree of certainty about the accuracy of the gaze tracking. Because the gaze data must be very reliable as the wheelchair moves directly with it and a mistake can even cause a collision, only those gaze data with a very high confidence value are collected. To translate the gaze coordinates into directions (forward, backward, neutral, left, right, neutral), certain thresholds are defined. For example, if the gaze is to the right of a specific threshold, it is labelled as "right"; if it falls between an upper and a lower threshold, it is considered "neutral", and so on. Figure 11 shows how the different segments have been delineated.

Finally, the orders of the motion control are sent to the wheelchair control via a UDP (User Datagram Protocol) socket, which is characterised as a fast communication protocol between two devices in a computer network as it does not formally establish the connection before

transferring the data [40]. Three pieces of data are sent to the wheelchair via the socket. The first one is the direction of gaze if it has been Forward (F), Backward (B) or Neutral (N). Only the capital letter in brackets is sent. The next letter sent is the gaze direction if Left (L), Right (R) or Neutral (N). And finally, the number of the configuration mode that the wheelchair currently has. These three data are encoded to bytes to ensure a correct transmission through the communication channel. Apart from being sent over the socket these three data are also displayed on the Pupil Labs "world" process screen using the processing callback “gl_display”. This callback, as it has been seen in Section 4.1.2, renders visual elements which, in this case, are of text type showing the current configuration of the wheelchair and the gaze directions.

4.2 Gaze estimation relative to the wheelchair (M2)

4.2.1 Hardware

Pupil Labs Core headset

In this module, two devices are needed in addition to the robotic wheelchair. The first of these is the Pupil Labs Core headset, whose specifications in this module will be the same as in module 1, see Section 4.1.1, except that the resolution of the front camera is 320 x 240 pixels in order to get the images from this camera and send them as quickly as possible. Furthermore, a calibration of the front camera is necessary to know the intrinsic parameters: the focal length and the principal points in x and y.

Asus Xtion Pro Live

In addition of the Pupil Labs Core headset, it is also necessary to perform the camera calibration of the other device used in this module, the Asus Xtion Live Pro camera. The focal length and the principal points in x and y coordinates of both cameras are important because after some mathematical manipulation together with other parameters, they give the rotation angles (rotation matrix) of the Pupil Core headset front camera with respect to the scene camera located on the wheelchair. This rotation matrix is used to transform the gaze, to express the gaze with respect to the wheelchair. The implementation section discusses in more detail how the intrinsic parameters have been derived, Section 4.2.4.1.

The Asus Xtion Live Pro camera, Figure 12, is positioned on the wheelchair and is used to obtain images of the environment. The most important specifications of this camera are:

- **Asus Xtion Live Pro Camera:**
  - **Software:** Open NI SDK bundled
  - **Dimensions:** 180 x 40 x 25 mm
  - **Depth range:** 0.8 to 3.5 m

- **Field of View (FOV):** 58º H; 45º V
- **Frame rate:** 30 fps
- **Resolution:** SXVGA (1280x1024)
- **Depth Image Size:** VGA (640 x480); QVGA (320x240)

As can be seen in the specifications, the camera captures two types of images, one is the RGB image and the other a depth image. The depth image is obtained because the camera has an infrared sensor that emits the infrared light signal and then to know the depth it picks up the time it takes for the light pulse to travel from the camera to the object and back. In this module, the chosen resolution in this case is QVGA (320x240) at 30 fps, the same as the RGB image. To obtain RGB images quickly because of the need for a fast process the camera resolution is chosen to be as low as possible, the resolution chosen is 320x240px.

Figure 12. Asus Xtion Pro Live Camera.

4.2.2 Software

Pupil Labs Software

The software used for this module, as Pupil Labs Core glasses are used, is the same as in the previous module, the Pupil Labs software. A new plugin has been created for this module, which follows the same structure as mentioned in Section 4.1.2 above. The class attribute is still "by_class". The difference with the previous plugin is that now the variable "events" is not accessed to obtain the blink and the 3D gaze, this variable is accessed to obtain the images made by the front camera of the glasses and the 3D gaze.

OpenNI2

In order to access the images taken by the Asus Xtion Live Pro camera, the installation of a driver called OpenNI2 is required. To install it on Windows the following link is used: [33]. Once installed, this project accesses the images via a file using the Python programming language. This file is the same one used to develop and implement this module 2, which transforms the

3D gaze with respect to the eye-tracking device to the 3D gaze with respect to the wheelchair. The procedure to access the images first consists of declaring in the Python file to be accessed the openni2 module inside the PrimeSense library. Once the module is declared, the first step is its initialization, then search for and detect the Asus Xtion camera connected to the computer and open a connection with the RGB camera and the depth camera. In addition, it is necessary to specify the type of resolution desired which, as mentioned above, is 320 x 240 pixels in both cases. Once these steps have been taken, the only thing left to do is to initialise the data capture.

OpenCV

In order to compare the images and extract the information captured by the Asus Xtion Pro Live camera with the Pupil Labs front camera, the computer vision library known as OpenCV (Open-Source Computer Vision Library) is used to apply the different computer vision algorithms: SURF (Speeded-Up Robust Features), RANSAC (RANdom SAmple Consensus) and FLANN (Fast Library for Approximate Nearest Neighbors). The version of OpenCV used is version 4.6.0. Pupil Labs software also uses this library, so the version chosen has to be compatible with Pupil software and have the necessary algorithms. The problem is that for newer versions of OpenCV the SURF algorithm is disabled due to licensing restrictions, so “pip install opencv” with the specific version can’t be used to install it. To download the library, it’s necessary to acquire it from the source code by enabling the option of non-free algorithms. After downloading, the OpenCV library is imported into the python file of module 2 using “cv2”.

VPython

VPython [34] is a library for the creation of three-dimensional animations that allow real-time rendering. Since the syntax of this library is very simple, as it is written in Python, it allows an easy and fast creation of three-dimensional objects and scenes. In this module 2, not only the 3D coordinates of the gaze with respect to the Asus Xtion camera attached to the wheelchair are calculated, but by means of VPython this module also graphically displays the rotation of the reference axes of the Pupil Labs front camera, the Asus Xtion camera and the user's head. In addition, the position of the 3D gaze with respect to the Pupil Labs front camera is also displayed graphically. This library allows the creation of different 3D objects: arrows, boxes, cones, curves, cylinders, ellipsoids, pyramids, rings, etc. To create them inside a scene it’s necessary to indicate the position, colour, size and other characteristics. The renderization of the scene in Vpython is done through a browser using WebGL [35].

Matlab

To calibrate the two cameras and obtain the intrinsic parameters needed in this module, the programming and calculation platform known as Matlab [39] has been used. Matlab has

different families of apps and one of them is the apps related to computer vision and image processing. Specifically, the app used to perform the calibration of the Pupil Labs front camera and the Asus Xtion camera was "Camera Calibrator". In the implementation section, Section 4.2.4.1, it is explained in more detail how the calibration was performed with this program.

4.2.3 Algorithms

This section explains the algorithms used for the implementation of module 2.

Speed-Up Robust Features (SURF) and matching algorithms.

Feature and matching algorithms are useful to extract image content and match images. They are used for a variety of purposes in computer vision and image processing such as automated object tracking, object detection, 3d reconstruction, etc [45]. The feature and matching algorithms can be of two types: global or local. The difference between them is that global features describe the entire image such as the colour, texture or shape while the local features describe a set of regions or interest points called key points inside an image [46]. Global feature detectors are much faster and require less memory than local feature detectors since they don't detect a huge number of features, but they are variant to illumination, occlusions, noise, and clutter and the performance is lower. Therefore, as in this project, the aim is to detect features with the highest possible performance and be invariant to the presence of different variations of an image, a local feature and matching algorithm have been chosen. There are a wide variety of local feature detector and matching algorithms but some of the most common are Harris Corner Detector, SIFT (Scale-Invariant Feature Transform), SURF (Speeded Up Robust Features), ORB (Oriented FAST and Rotated BRIEF), FAST (Features from Accelerated Segment Test, etc. Among all the possibilities, the algorithm selected to extract local features and match the images is the Speeded Up Robust Features (SURF) since it is real-time, scale and rotation invariant, robust and with good performance [47].

The SURF algorithm has two main parts: a feature extraction and a feature description [48].

Feature extraction

The feature extraction or detection is based on the determinant of the Hessian matrix [49]. The Hessian is a matrix composed of all the second-order partial derivatives of a scalar function [50] and for a pixel $X = \{x, y\}$ in an image is represented as in Equation (1).

$$H(f(x,y)) = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix} \quad (1)$$

The use of the second order derivate in x, y and x-y direction enables finding points with maximum values, points that have a local change. With the determinant of the Hessian matrix, the points of interest are maximal considering all the directions.

The image is convolved with a Gaussian second-order derivative instead of doing the second-order derivate directly from the image, Equation (2). The use of a Gaussian kernel allows to compute at the same time the location and the scale of a point when the determinant of the Hessian matrix is applied. Therefore, the maximal points are obtained in any scale of the image, with different resolutions.

$$H(X,\sigma) = \begin{bmatrix} L_{xx}(X,\sigma) & L_{xy}(X,\sigma) \\ L_{xy}(X,\sigma) & L_{yy}(X,\sigma) \end{bmatrix} \quad (2)$$

In Equation (2), $X = \{x, y\}$ is the location of the point and σ the scale. $\mathrm{L_{xx}(X,\sigma)}$ is the convolution of the Gaussian second-order derivative in the x direction with the image at point X and scale σ. The same in the y-direction and x-y direction with $\mathrm{L_{yy}(X,\sigma)}$ and $\mathrm{L_{xy}(X,\sigma)}$, respectively.

What makes SURF faster than other local feature detectors and descriptors is that it uses box filters as an approximation of the Gaussian second-order derivative and the calculation of the convolution between the filter kernel and the image is done using the integral image. Furthermore, this calculation can be done in parallel with different scales as it is explained in the following paragraph.

Scale-space representation

As it had been seen the interest points can be found at different scales σ or spatial resolutions. The representation in all the scales, scale-space, is normally represented using an image pyramid. In other methods, to obtain the pyramid, the images are repeatedly convolved with a Gaussian second-order partial derivative filter and reduced to get the next pyramid level and the next image to filter. Therefore, to reach a higher level of the pyramid, the previously filtered image must be obtained. In SURF, different filter kernel sizes are utilised on the original image to generate the image pyramid, enabling concurrent processing. In this method, the lowest filter kernel size is a 9x9 filter which corresponds to the lowest scale σ equal to 1.2 and higher spatial resolution. The next masks have sizes of 15x15,21x21,27x27… due to the integral image and the approximation of the Gaussian second-order derivative kernel nature. The next filter size is also double the scale of the previous mask. For instance, if the mask size is 21x21 the scale is $\sigma = 2 \text{ x } 1.2 = 2.4$. To detect interest points in images and over scales, the maximal point is selected over a 3x3x3 neighbourhood. Then, the maxima points are interpolated in scale and image space with the method of Brown *et al*. [51] to accurately localise the key points at sub-pixel localisation

using intensity and gradient information of the neighbour pixels.

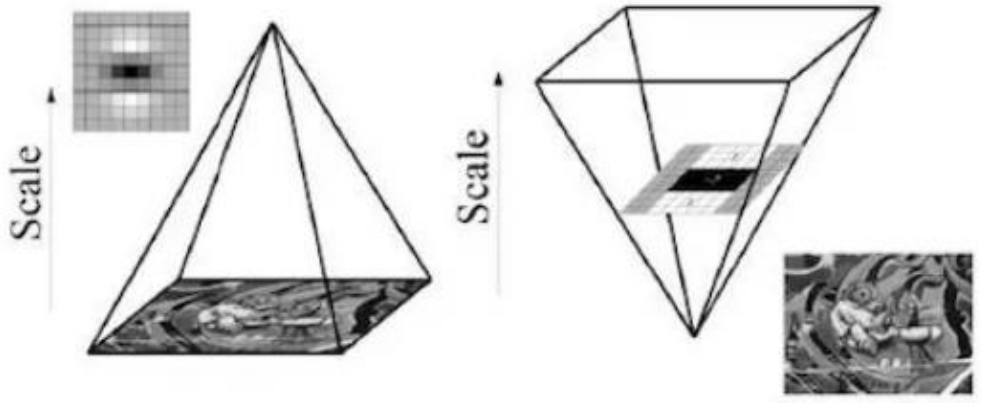

Figure 13. Scale-space with other methods reducing image size (left), SURF method up-scaling the filter (right) [52].

Feature description

Then, the feature descriptor is applied to obtain local image information such as structure, gradients and intensity variations surrounding a key point. The process of creating a SURF descriptor relies on two steps: assign an orientation and extract the descriptor [53].

Orientation Assignment

The orientation of the interest points must be identified to be rotation invariant. To accomplish this, the first step is to convolve the image patches with Haar-like filters in both the x and y directions within a circular area. The radius of this circle is determined by multiplying the scale in which the interesting point was found by 6, and the centre of the circle is positioned at the location of the key point. Since with higher scales, the size of the Haar wavelet is bigger, the integral image is used again for fast computation. The result of applying the Haar-like filter is then weighted with a Gaussian of $\sigma = 2.5$ times the scale and centred in the key point. The responses are represented as vectors in a two-dimensional space with the horizontal response strength on the x-axis and, the vertical response strength on the y-axis. To determine the dominant orientation, the wavelet responses are summed within a scanning area that covers an angle of 60 degrees. The result is a local orientation vector. After scanning all the circle, different local orientation vectors are acquired. The dominant orientation of the key point will correspond to the longest vector.

Descriptor components extraction

To obtain a descriptor around a key point, a square region is constructed with a size of 20 times the scale in which the key point was found. The centre is around the key point, and it is

oriented following the dominant orientation of the interest point. Then, this region is split into smaller sub-regions of 4x4 square. In each sub-region, a set of sample points are divided into a 5x5 grid pattern evenly spaced. For each sample point, the Haar-like wavelet is applied to get responses in the horizontal direction d_x and in the vertical direction d_y. These responses are weighted with a Gaussian of 3.3 times the scale and centre the interest point to increase robustness to deformations and localisation errors. The responses and the absolute value of the response are then summed up for each sub-region. The absolute value is analysed to have information about the polarity of the intensity changes. The result is a vector of dimension four in a sub-region, $v = (\sum d_x, \sum d_y, \sum |d_x|, \sum |d_y|)$ and for all the regions, a vector of length 64.

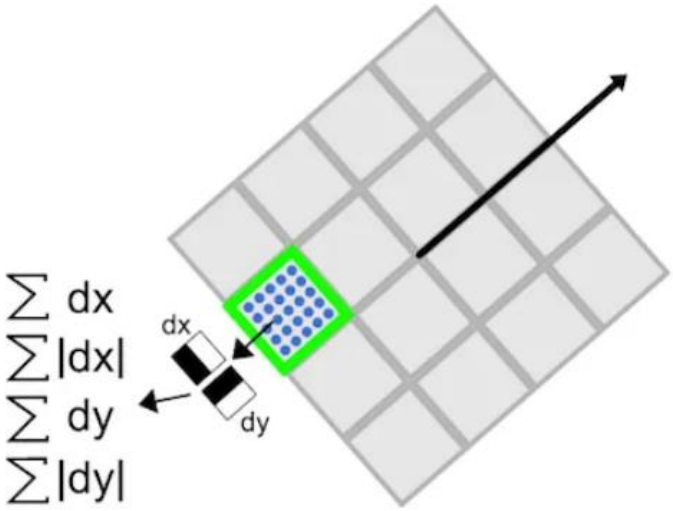

Figure 14. Haar wavelet convolution in a sub-region of the image [52].

Furthermore, the descriptor vector is not affected by a constant offset in the overall brightness or intensity level of an image. If the descriptor vector is transformed to be unit length, the descriptor is also invariant to scale or changes in contrast.

- Matching algorithms

As it has been seen, SURF can detect features and descriptors even if images are rotated, scaled or a bias illumination has been applied to them. Furthermore, as the Gaussian weighting is applied in the responses of the feature extraction, SURF is also more robust to deformations, noise, and localisation errors. After having the key points and their descriptors for each image (reference and query image) the library FLANN (Fast Library for Approximate Nearest Neighbors) is used to obtain the most similar features between two images [54]. This library contains different algorithms and the one being used in this project is the KD-tree (k-dimensional tree) [55].

K-d tree construction

As the name can suggest, it is a data structure that organises the descriptor vectors in a hierarchical binary tree to search similar features faster [56]. The construction of the tree is a recursive process, and the steps are shown in Figure 15.

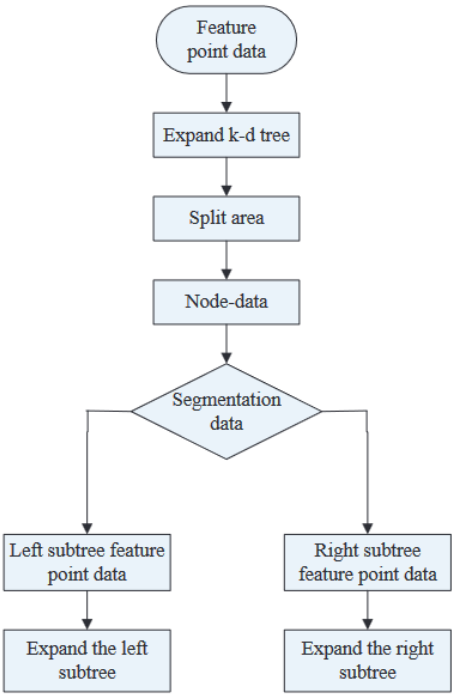

Figure 15. k-d tree construction process SURF [56].

The first step is to determine the split domain. In SURF, each key point is a unitary vector of dimension 64. The variance is calculated for each dimension of all the key points. The result is a 64-dimensional vector that contains the variance of each dimension. Then, the dimension with the largest variance of the vector is chosen as the split dimension. After obtaining the dimension with the largest variance, the split position (node-data) is calculated by sorting the descriptor vectors in an ascendant way in the split dimension and selecting the median value. Next, the descriptor vectors are split into two subsets based on their values along the split dimension. The left subtree will have values lower or equal to the split position and the right subtree the rest values. These two first steps are repeated recursively until the tree is constructed. The tree is only built for one of the images, the reference image.

Nearest Neighbor Search

After having the tree, the Nearest Neighbor Search is applied [57]. This method consists in selecting a reference vector of the query image and finding the nearest neighbour among the tree of the reference image. The nearest neighbour is obtained after calculating the Euclidean

distance between the query descriptor and the descriptors of the reference image tree and selecting the descriptor with the shortest distance. In this project and as OpenCV does in the SURF algorithm, two nearest neighbours' references are selected for each query descriptor.

Lowe's ratio test

There are features in the images that were not correct match. Lowe's ratio test is used to discard these matches. It consists of using the two nearest neighbours' key points and applying a threshold to the ratio between the distance of the closest neighbour and the second closest distance since correct matches have a probability distribution function (PDF) centred in a much lower ratio than false matches [58].

Random Sample Consensus (RANSAC)

Even though Lowe's ratio test tries to discard false matchings, there are still some key points that remain incorrectly matched. For instance, maybe two key points are matched because they have the same local appearance but, they do not correspond to the same feature [59]. Therefore, to filter the points that remain incorrectly matched, the Random Sample Consensus (RANSAC) is used [60]. It consists of selecting a minimum subset of matched key points and estimating a mathematical model for them. In the case of estimating a model with the homography transformation, the minimum sample of key points is 4. After fitting the homography model to the randomly chosen points, the number of key points that fit the model within a measuring error (inliers) are counted. This process of fitting the model to the randomly chosen data points is repeated a fixed number of times. Then, the estimated model that has the largest number of inliers is selected and the outlier points are disregarded.

4.2.4 Implementation steps

As mentioned above, this module aims to express the user's gaze in 3D with respect to the wheelchair, specifically with respect to the Asus Xtion camera attached to the wheelchair. To do this, it is first necessary to obtain the 3D gaze through the eye-tracking device that the user wears on his or her head. This gaze is relative to the front camera of the Pupil Labs headset, so a transformation is necessary to express it correctly. This transformation simply consists of changing the reference system of the user's gaze. To do this, it is necessary to know the distance between the two cameras and the rotation between them. Since the user wears one of the cameras on the head (the front camera of the Pupil Core headset) and the user's head moves with respect to the other camera attached to the wheelchair, the rotation is not fixed but varies in real-time. The position and orientation of the Asus Xtion camera and the Pupil Labs front camera reference frame can be observed in Figure 16. If the two references coincide with the same orientation as it happens in Figure 16, it means that the head has not rotated, and it is only necessary to make a translation between the two references since the positions of the

cameras are different.

Figure 16. Position and orientation of the reference frames.

To obtain this rotation, the images from the cameras are needed and by applying computer vision algorithms together with trigonometric operations, the yaw, pitch and roll angles are obtained, which are actually the angles of rotation of the user's head with respect to the wheelchair. These angles are needed to construct the rotation matrix while the distance between cameras is used to construct the translation matrix. Once the matrices are available, the transformation, Figure 17, can be obtained by multiplying the 3D gaze coordinates with the rotation matrix and then adding the translation matrix. As a result, the 3D gaze in relation to the Asus Xtion camera, and consequently, in relation to the wheelchair, is obtained.

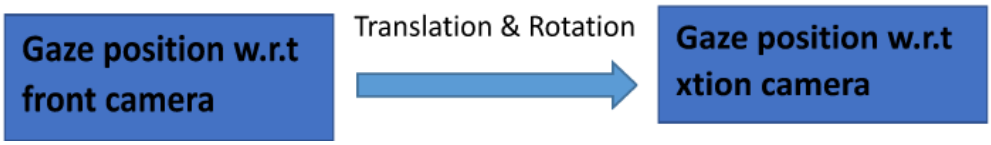

Figure 17. Transformation of reference frames.

The steps explained below are those followed in the implementation of the module and can be followed through Figure 18. Two files with the Python programming language have been used. In the first file called "module2_tfm.py" a plugin has been created using the Pupil Labs Software framework. This custom plugin collects information about the Pupil Core headset by performing two actions, the first one is to access the variable "events[gaze]" to obtain information about the user's gaze in x, y and z coordinates. Also, from this variable, the confidence values of the gaze are obtained, so only those gaze data whose confidence values are high will be selected. The other action is to obtain the images from the front camera of the

Pupil Core headset by accessing the variable "events["frame"]. These two data are sent to the other file, called "module2_tfm_2.py" using a TCP (Transmission Control Protocol) socket that establishes the connection between the sender and the receiver before sending the information. In this other Python file, the first action to be performed is to obtain the data that has been sent through the socket and, obtain the image that the Asus Xtion camera makes. Afterwards, the images are compared using computer vision techniques and mathematical operations are applied to obtain the rotation of the user's head (yaw, pitch, roll) with respect to the wheelchair, see Section 4.2.4.2. With these angles, the rotation matrix is constructed and with the distance between cameras, the translation matrix, see Section 4.2.4.3. Once the matrices have been obtained, the user's 3D gaze with respect to the wheelchair can be transformed. The second file is also responsible for graphically representing the reference axes of the cameras, the movement of the user's head and the user's gaze using the VPython

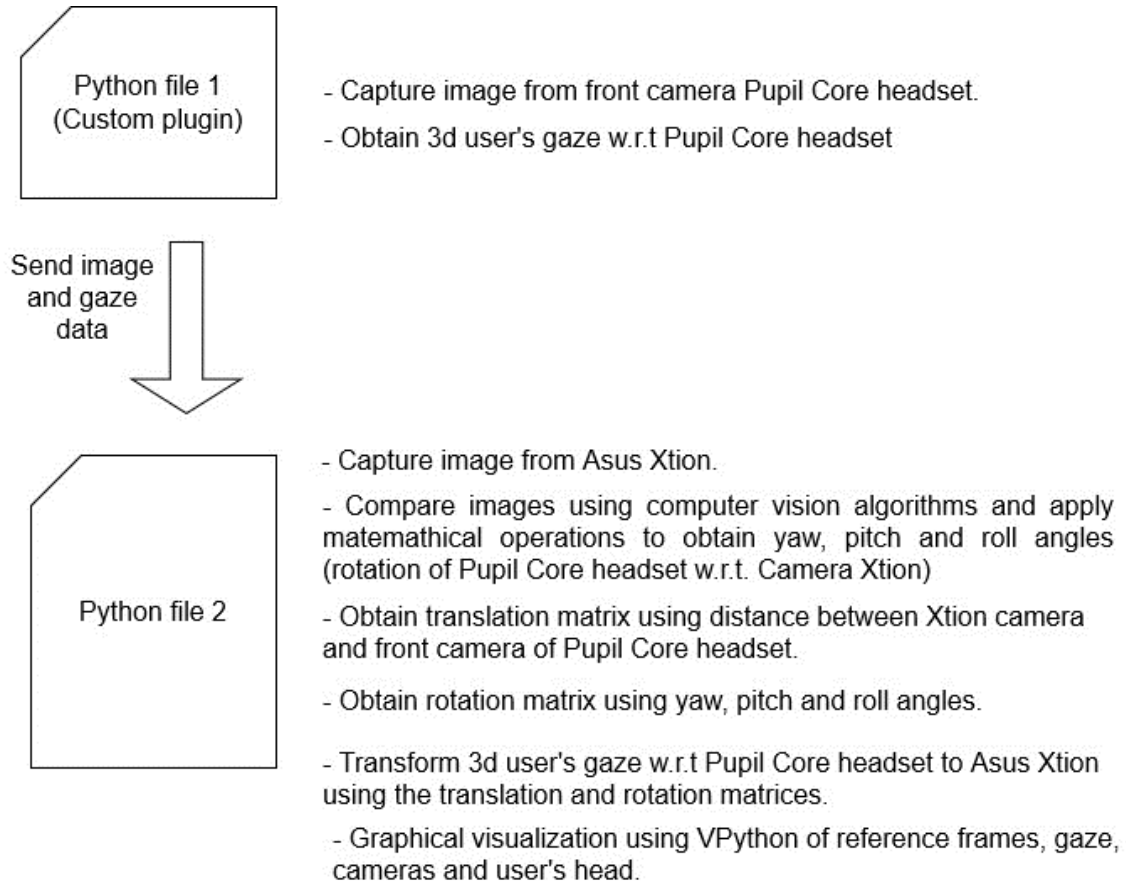

Figure 18. Module 2 implementation process.

4.2.4.1 Camera Calibration

To calculate the rotation of the user's head, apart from other data, it is necessary to know the intrinsic parameters of the Asus Xtion Live Pro camera and the front camera of the Pupil Labs headset. These parameters are the focal length and the principal point and are obtained by calibrating the cameras. The focal length is the distance between the optical centre of the lens and the camera sensor [41], see Figure 19. Two focal lengths are obtained for each camera, one in coordinate x and another in coordinate y. On the other hand, the principal point also known as optical center is that point which is pixel (0,0), the pixel at the centre of the image. In other words, it is the point from which the focal length is measured [42]. In Figure 20, the principal point can be seen visually.

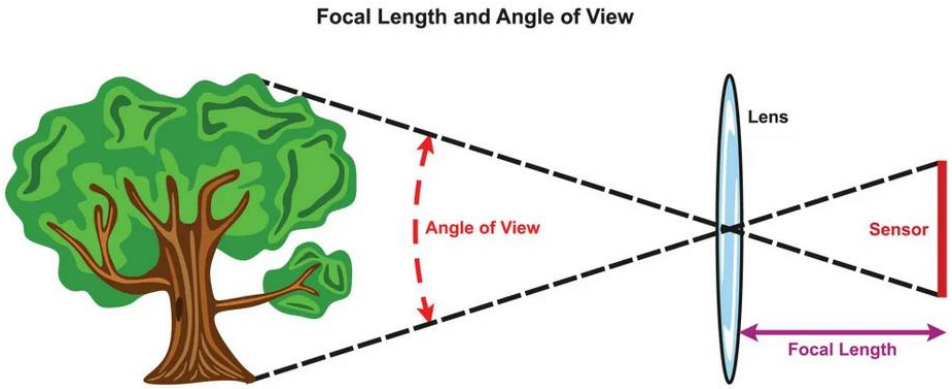

Figure 19. Focal Length [41].

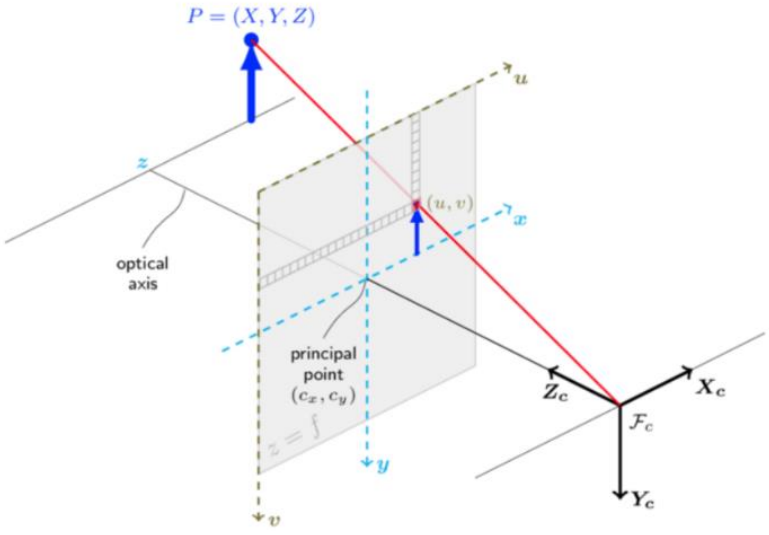

Figure 20. Pinhole camera model [43].

The calibration has been performed using Matlab, the "Camera Calibrator" app. The calibration consists of giving at least two images containing a chessboard in this case, with a square

dimension of 25 millimetres, Figure 21. For best results, the more images given, the better the calibration. The photos need to be captured using the same camera for calibration, and the image resolution should remain consistent across all photos. The internal characteristics of the camera, known as intrinsic parameters, adjust based on the image resolution. In this case, more than 20 chessboard images have been taken for each camera.

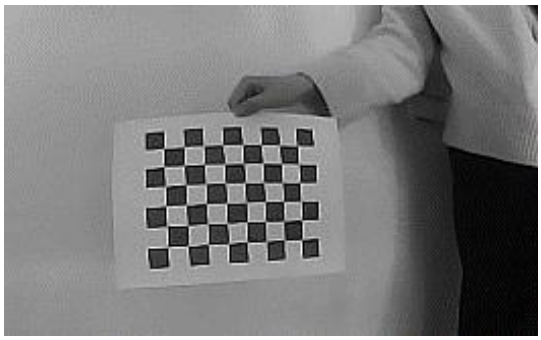

Figure 21. Camera calibration using chessboard 25 x 25 mm.

As a result, the three-dimensional representation of the chessboard positions in the different images is obtained (left Figure 22) as well as the mean pixel error of the calibration in each image (right Figure 22). It is desired that the mean error of the calibration be as low as possible for each image, so those images that have a large mean error must be deleted and must be retaken.

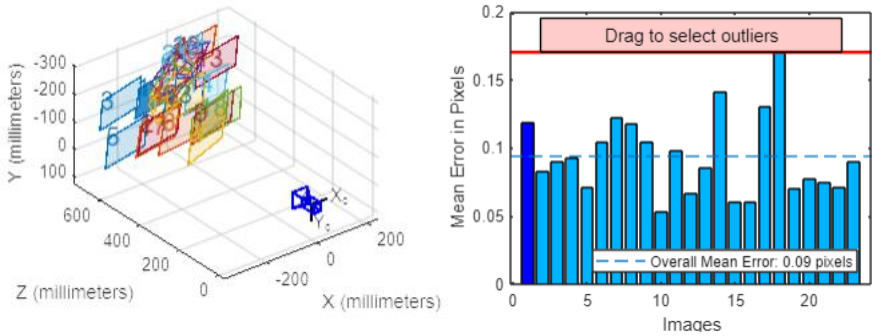

Figure 22. Camera calibration using more than 20 chessboard images.

In addition, the app also provides the intrinsic and extrinsic parameters of the camera. The extrinsic parameters represent the location of the camera in the 3D scene and these parameters are used to transform the world points to camera coordinates [44]. The intrinsic parameters map the camera coordinates onto the image plane. Figure 23 shows the results of the intrinsic parameters for the Pupil Labs core front camera and Figure 24 shows the intrinsic parameters for the Asus Xtion Live Pro camera. The first column in the two figures are the values in the x-coordinate values while the second column in the y-coordinate. Both cameras have been calibrated with a resolution of 320 x 240 pixels. In addition, the application also

provides the radial distortion of the camera that occurs when light rays are bent closer to the edges than at the optical centre. As the radial distortion in our cameras is very small, it is not considered for module 2.

```
Intrinsics
----------
Focal length (pixels):    [  204.1089 +/- 1.9094      205.2635 +/- 1.9297  ]
Principal point (pixels):[  156.7208 +/- 0.5466      132.7568 +/- 0.7134  ]
Radial distortion:        [   -0.4809 +/- 0.0109        0.2129 +/- 0.0120  ]
```

Figure 23. Intrinsic parameters of the front camera of Pupil Labs Core headset at 320 x 240 px resolution.

```
Intrinsics
----------
Focal length (pixels):    [  287.0591 +/- 6.8453      286.5934 +/- 6.8163  ]
Principal point (pixels):[  156.6400 +/- 0.4923      119.5049 +/- 0.9773  ]
Radial distortion:        [    0.1956 +/- 0.0128       -0.7696 +/- 0.0777  ]
```

Figure 24. Intrinsic parameters of Asus Xtion Live Pro Camera at 320 x 240 px resolution.

4.2.4.2 Obtaining head rotation.

There are several ways to obtain the rotation of the user's head. One of them is through image matching, which is what has been used for this module, see Section 4.2.4.2.1. Apart from this alternative, two other alternatives were also thought of, which are through an inertial sensor, Section 4.2.4.2.2, and the optical flow of the cameras, Section 4.2.4.2.3. Although the latter were not developed, they have been presented in this work.

4.2.4.2.1 Obtaining head rotation from image matching

In this Section 4.2.4.2.1, the head rotation is obtained with the matching of images. An image taken by the Asus Xtion camera is compared with an image taken by the Pupil Labs front camera. The images are compared by applying the SURF algorithm, a local feature detector and descriptor and then, matching algorithms. The results are feature points (key points) of the reference and the query image. These key points have pixel coordinates in each image. These key points will be used to get the head movement of the user in the wheelchair. The interest point locations are averaged to obtain a mean key point that represents the descriptor cloud in each image. In Figure 25 the key points are shown in green with their respective matches in the two images and the mean key point as a big black dot in each image.

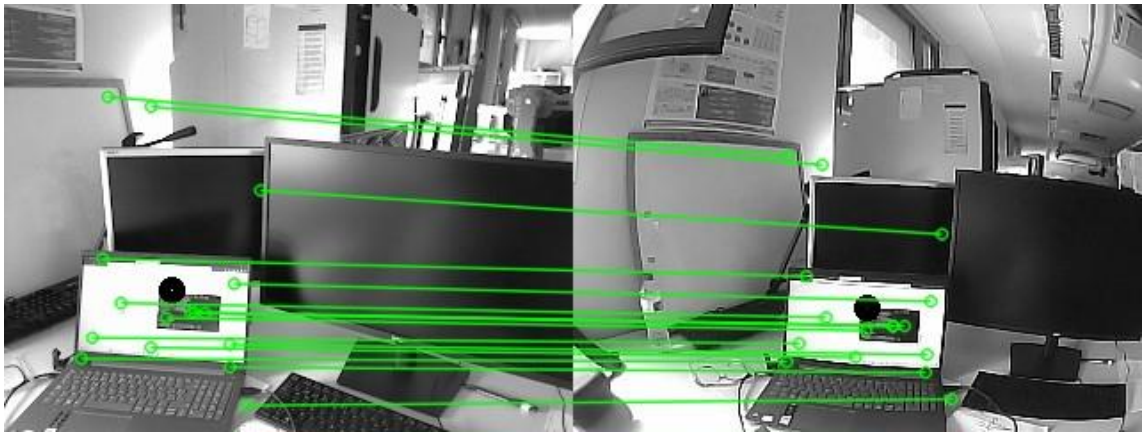

Figure 25. Image matching of front camera Pupil Labs image (left) and Asus Xtion Live Pro image (right).

This mean descriptor point is represented in Figure 26 and Figure 27 with a star and with the combination with some data and after doing trigonometric operations the bearing (*yaw*) and the elevation (*pitch*) of the head can be obtained. For the bank (*roll*) angle the key points of each image are used instead of the mean. The steps to obtain them are:

4.2.4.2.1.1.1 Bearing (*yaw*) angle

The data available before starting the trigonometry operations to obtain the bearing angle of the head are the intrinsic parameters of the Asus Xtion Camera and Pupil Labs front camera in pixels, the distance between the two cameras along the x-axis in millimetres, the location of the mean key point in the image taken by each camera in pixels and the depth distance of the mean descriptor point of the image taken by the Asus Xtion Camera in millimetres. Following Figure 26, the process for determining the bearing angle involves these calculations:

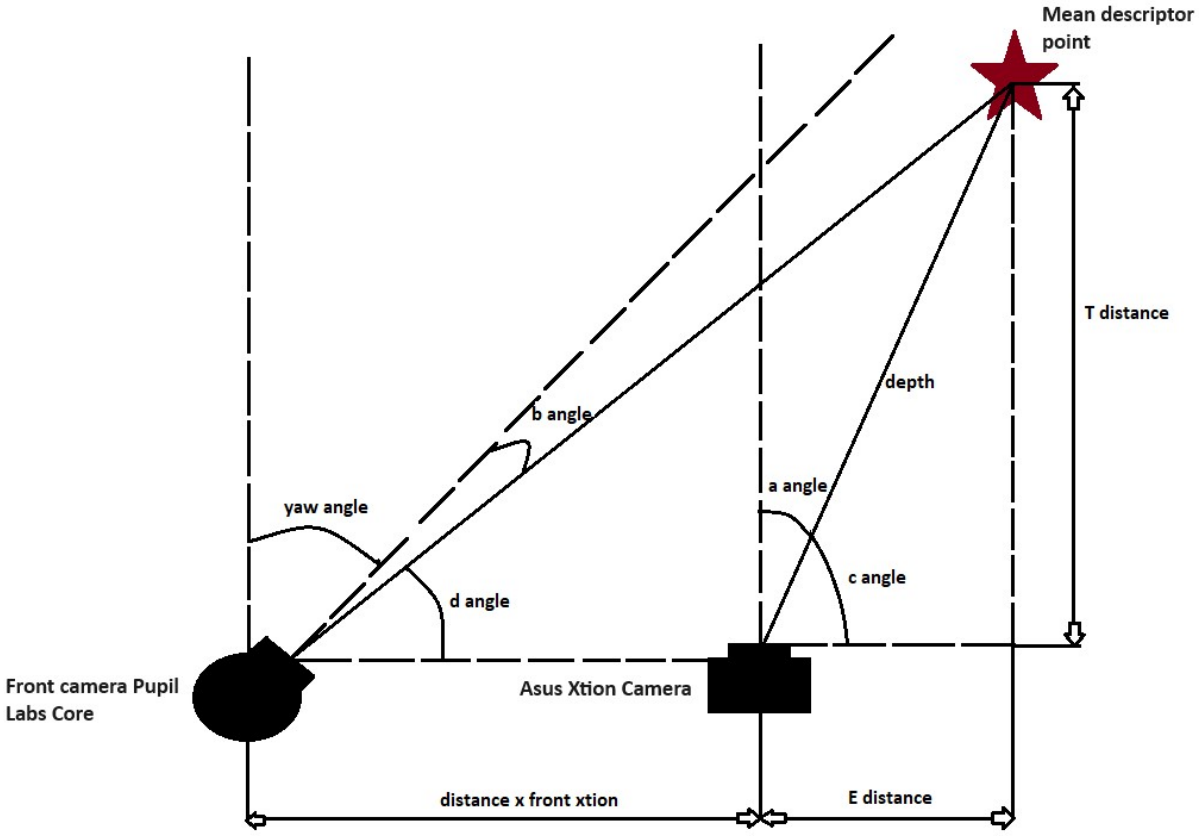

Figure 26. Representation of the angles and distances necessary to obtain the *yaw* angle.

***a* and *b* angle**

The angles *a* and *b* are the angles between the centre of the cameras and the location of the mean descriptor point in the x-axis of the images. Since there is no information about the location of the mean descriptor point in distance units and the front camera of the Pupil Labs does not have depth information, the intrinsic parameters of the cameras are used to obtain the angles. The intrinsic parameters provided by the cameras are the focal length and the principal point on the horizontal x and vertical y axes, see Section 4.2.4.1. Since the bearing angle is a rotation around the vertical y-axis, the values on the vertical y-axis remain constant. Therefore, the intrinsic parameters used for the head bearing are on the horizontal x-axis. Then, *a* angle and *b* angle can be obtained following Equations (3) and (4).

$$a = arctan\left(\frac{x_{xtion} - pp_{x\_xtion}}{f_{x\_xtion}}\right) \quad (3)$$

$$b = arctan\left(\frac{x_{front} - pp_{x\_front}}{f_{x\_front}}\right) \quad (4)$$

Where in Equation (3), x_{xtion} is the mean descriptor point in the x-axis of the Asus Xtion Camera Image, pp_{x_xtion} is the principal point in the x-axis of the Asus Xtion Camera and f_{x_xtion} the focal length in the x-axis of the Asus Xtion Camera.

And in Equation (4), x_{front} is the mean descriptor point in x axis of the front camera Pupil Labs image, pp_{x_front} and f_{x_front} the principal point and the focal length in the x-axis of the Pupil Labs front camera respectively.

_c_ angle

Because *a* angle has been obtained previously, *c* angle can be computed following Equation (5).

$$c = 90° - a \tag{5}$$

_T_ distance

The Asus Xtion camera can provide the depth in distance units of any pixel in the image. Therefore, since the location of the mean descriptor point (mdp) is known, the depth distance in millimetres of this point in the environment will also be known. To obtain *T* distance, Equation (6), the depth in millimetres of the mean descriptor point is needed and the previously calculated angle *c* must be converted to radians.

$$T = cos(c) \cdot depth\ distance_{xtion,mdp} \tag{6}$$

_d_ angle

d angle is computed doing the 2-argument arctangent in order to obtain an angle *d* between -pi and pi radians. The arctangent is not computed because it omits the angles of the II and III quadrants. Equation (7) can be used to calculate *d* angle since it has been previously calculated the depth in millimetres of the mean descriptor point, *a* angle, *T* distance and the distance between the Asus Xtion camera and the front Pupil Labs camera along the x-axis.

$$d = arctan2\left(\frac{depth\ distance_{xtion,mdp} \cdot sin(c)}{distance\ x\ front\ xtion + T}\right) \tag{7}$$

The *d* angle is then transformed from radians to degrees and Equation (8) is applied to obtain

the angles between 0 and 360 degrees.

$$d = (d + 360)\ \%\ 360 \tag{8}$$

***Bearing (yaw)* angle**

After applying the above steps, the yaw angle can be easily calculated by making the difference between 90 degrees and angles *b* and *d*, Equation (9).

$$yaw = 90° - b - d \tag{9}$$

4.2.4.2.1.1.2 Elevation (*pitch*) angle

As it happened with the bearing (*yaw*) angle, the starting point to compute the elevation angle is the data given by the intrinsic parameters of both cameras in pixels, the measurement of the distance between them in millimetres, the coordinate of the mean descriptor point in the image taken by each camera in pixels and the depth distance of the mean descriptor point of the image taken by the Asus Xtion camera in millimetres.

Following Figure 27 as a reference, the steps to precede are the computation of:

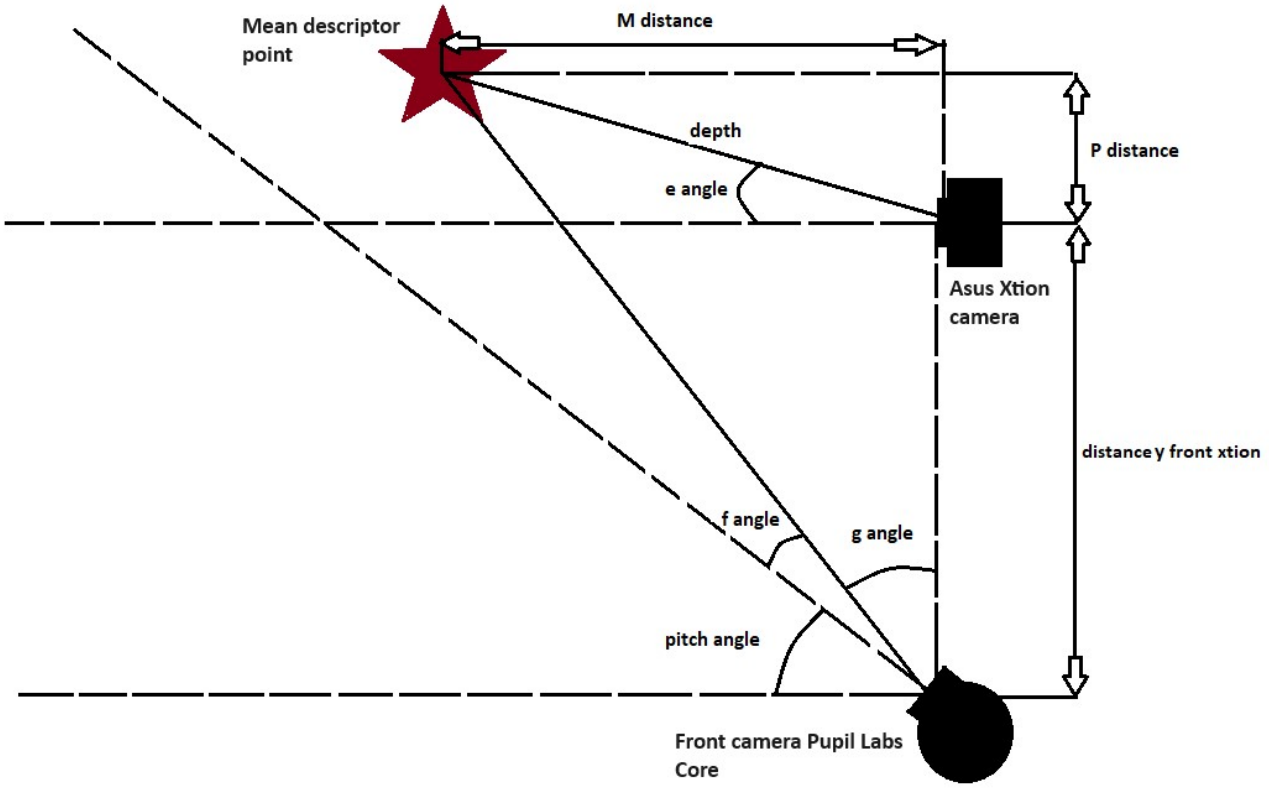

Figure 27. Representation of the angles and distances necessary to obtain the *pitch* angle.

***e* and *f* angle**

e and *f* angles are the angles between the centre of the cameras and the location of the mean descriptor point in the y-axis of the image. As before, the intrinsic parameters are needed but, in this case, the focal length and the principal point of the vertical y-axis. Angles *e* and *f* can be calculated with Equations (10) and (11).

$$e = arctan\left(\frac{y_{xtion} - pp_{y\_xtion}}{f_{y\_xtion}}\right) \quad (10)$$

$$f = arctan\left(\frac{y_{front} - pp_{y\_front}}{f_{y\_front}}\right) \quad (11)$$

Where in Equation (10), y_{xtion} is the mean descriptor point in y axis of the Asus Xtion Camera Image, pp_{y_xtion} is the principal point in y axis of the Asus Xtion Camera and f_{y_xtion} the focal in y axis of the Asus Xtion Camera.

And in Equation (11), y_{front} is the mean descriptor point in y axis of the front camera Pupil Labs image, pp_{y_front} and f_{y_front} the principal point and the focal length in y-axis of the front camera Pupil Labs respectively.

_P_ distance

P distance can be calculated as the multiplication of the depth distance in millimetres of the mean descriptor point given by the Asus Xtion Camera and the sinus of the *e* angle, see Equation (12).

$$P = sin(e) \cdot depth\ distance_{xtion,mdp} \tag{12}$$

_M_ distance

The computation of *M* distance is the same as *P* distance but with the cosine, Equation (13).

$$M = cos(e) \cdot depth\ distance_{xtion,mdp} \tag{13}$$

_g_ angle

g angle is computed using the 2-argument arctangent to obtain a *g* angle between -pi and pi radians. Equation (14) can be used to calculate the *g* angle since it has previously calculated the *M* distance and *P* distance and it has measured the distance between the front of the Pupil Labs and the Asus Xtion Camera along the y-axis.

$$g = arctan2\left(\frac{M}{distance\ y\ front\ xtion + P}\right) \tag{14}$$

To obtain a *g* angle between 0 and 360 degrees instead of -180 and 180, it needs to be first transformed from radians to degrees and then converted with Equation (15).

$$g = (g + 360)\ \%\ 360 \tag{15}$$

Elevation (pitch) angle

After applying the above steps, the *pitch* angle can be easily calculated by making the difference between 90 degrees and angles *f* and *g*, Equation (16).

$$pitch = 90° - f - g \tag{16}$$

4.2.4.2.1.1.3 Bank (*roll*) angle

To obtain the *roll* angle, the descriptor points are used instead of the mean descriptor point used in the *yaw* and *pitch* angle, as two points are needed to calculate the *roll* angle. The points are taken in pairs and the *roll* angle between the two points is calculated as the 2-argument arctangent of the distance between the two points on the y-axis with respect to the distance between the two points on the x-axis, see Equation (17).

$$roll_{\ xtion,i\_j} = arctan2\frac{point_{xtion,y\_j} - point_{xtion,y\_i}}{point_{xtion,x\_j} - point_{xtion,x\_i}} \tag{17}$$

Then, the same pairs of points at which the *roll* angle is calculated in the image taken by the Asus Xtion camera are used to calculate the *roll* angle in the image taken by the front camera of Pupil Labs, see Equation (18).

$$roll_{\ front,i\_j} = arctan2\frac{point_{front,y\_j} - point_{front,y\_i}}{point_{front,x\_j} - point_{front,x\_i}} \tag{18}$$

The *roll* angles are transformed using Equations (19) and (20), to have ranges between 0 and 360 degrees instead of -180 and 180 degrees and be able to subtract them.

$$roll_{xtion,i\_j} = (roll_{xtion,i\_j} + 360)\%360 \tag{19}$$

$$roll_{front,i\_j} = (roll_{front,i\_j} + 360)\%360 \tag{20}$$

Then, by subtracting the *roll* angles, the final *roll* angle is obtained for each pair of points, Equation (21).

$$roll_{i\_j} = roll_{\ front,i\_j} - roll_{\ xtion,i\_j} \tag{21}$$

It's necessary to have the angles expressed as the shortest rotation direction, Equation (22), to be able to obtain the ultimate *roll* angle as the average of the final roll angles of each pair of

points, Equation (23).

$$roll_{i\_j} = \begin{cases} 360° - |roll_{i\_j}| & if\ roll_{\ i\_j} < -180° \\ -360° - |roll_{i\_j}| & if\ roll_{i\_j} > 180° \end{cases} \quad (22)$$

$$roll\ = \sum roll_{i\_j} \quad (23)$$

4.2.4.2.2 Obtaining head rotation with an inertia sensor

A different proposal was to obtain head rotation with an inertial sensor. The idea was to attach an inertial sensor to the Pupil Labs headset and another one to the wheelchair and then process the data provided by the sensors to obtain the head rotation. The problem for not implementing this proposal was:

The inertial sensor is a bit bulky for the Pupil Labs headset and not comfortable for the user wearing it. As in this project, the finality is to bring functionalities to the user that are comfortable as well as useful, the possibility to put an inappropriate inertial sensor on the head of the user is not a choice to consider. Therefore, the idea was not developed in this project.

4.2.4.2.3 Obtaining head rotation with optical flow

Another proposal at the beginning of the project was to obtain *yaw*, *pitch* and r*oll* angle with the optical flow. Optical flow is a pattern of the apparent movement of the image scene caused by the movement of the camera that is taking the image or the movement of objects, people or things that are within the scene [61]. The motion pattern in this case is expressed as the sum of the motion derived from the rotation of the camera. With optical flow, there is no need for image matching, as it uses consecutive images from the same camera.

The main idea was to calculate the optical flow of the images taken by the Asus Xtion camera attached to the robotic wheelchair and the optical flow of the images taken by the front camera of Pupil Labs that the user carries in his head and then, subtract the two optical flows. If the derived movement is the same for the two cameras, it means that they have not moved relative to each other and therefore the difference is zero, but if it is different, it means that there has been a relative movement between them.

The problem and the reason why this proposal for head rotation was not developed further is that if one camera is looking to one side and the other to the opposite side and there is a

movement of an object in the scene only one of the cameras can capture it would result in a relative movement between the two cameras when this is not the case because they have not moved relative to each other.

4.2.4.3 3D user's gaze

Gaze position with respect to the head camera

The gaze position given by the Pupil Labs camera is with respect to the Pupil Labs front camera (the position and orientation of the reference frame can be seen in Figure 16) and it is expressed in distance coordinates in millimetres, $X = (x, y, z)$. To obtain this information, it is required to subscribe to the topic "gaze_point_3d" in "gaze" events in the Pupil Labs platform.

Gaze position with respect to the camera fixed on the wheelchair.

After obtaining the gaze position with respect to the Pupil Labs front camera $X = (x, y, z)$, it is necessary to do a translation and a rotation to have the gaze position with respect to the Asus Xtion camera attached to the wheelchair $X' = (x', y', z')$,. As the head rotation angles (*yaw* ψ, *pitch* θ, *roll* φ) have been previously calculated, they can be substituted in the rotation matrices of Equations (24), (25) and (26). The rotation matrices are then multiplied by the gaze position with respect to the Pupil Labs front camera and the result, is added to the translation matrix of the distances between the cameras (distance between the center of the two reference frames), Equation (27).

$$R_{z\_roll} = \begin{bmatrix} \cos(\varphi) & -\sin(\varphi) & 0 \\ \sin(\varphi) & cos(\varphi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (24)$$

$$R_{y\_yaw} = \begin{bmatrix} \cos(\psi) & 0 & \sin(\psi) \\ 0 & 1 & 0 \\ -\sin(\psi) & 0 & \cos(\psi) \end{bmatrix} \quad (25)$$

$$R_{x\_pitch} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix} \quad (26)$$

$$X' = [x', y', z'] = R_{z_{roll}} \cdot R_{y_{yaw}} \cdot R_{x_{pitch}} \cdot [x, y, z]^T + \begin{bmatrix} distance\ in\ x\ between\ O\ and\ O' \\ distance\ in\ y\ between\ O\ and\ O' \\ distance\ in\ z\ between\ O\ and\ O' \end{bmatrix} \quad (27)$$

4.2.4.4 Graphical representation.

As mentioned above, module 2 graphically displays the different reference axes that exist between the cameras and the user's head and the movement of the user's head as well as the position of the user's 3D gaze with respect to the Pupil Labs glasses. In Figure 28, it can be seen the scene created in VPython. Different basic 3D objects have been used to create this scene. For example, the user's head is represented by a sphere and several cylinders and the wheelchair with two rectangles and two cylinders. Two reference systems appear on the user's head. In these two systems, the origin is located in the same place, on the user's forehead and would correspond to the centre of the front camera of the Pupil Core Labs Headset. Of the two reference systems on the user's head, one is fixed (x-red, y-green, z-blue) and the other moves with the rotation of the user's head (x-violet, y-magenta, z-orange). In addition, an additional black line appears on the user's forehead This line corresponds to the user's gaze position with respect to the front camera. Finally, in the VPython scene (top left), the camera attached to the wheelchair appears and corresponds to the Asus Xtion camera. The camera's reference system (x-red, y-green, z-blue) is represented.

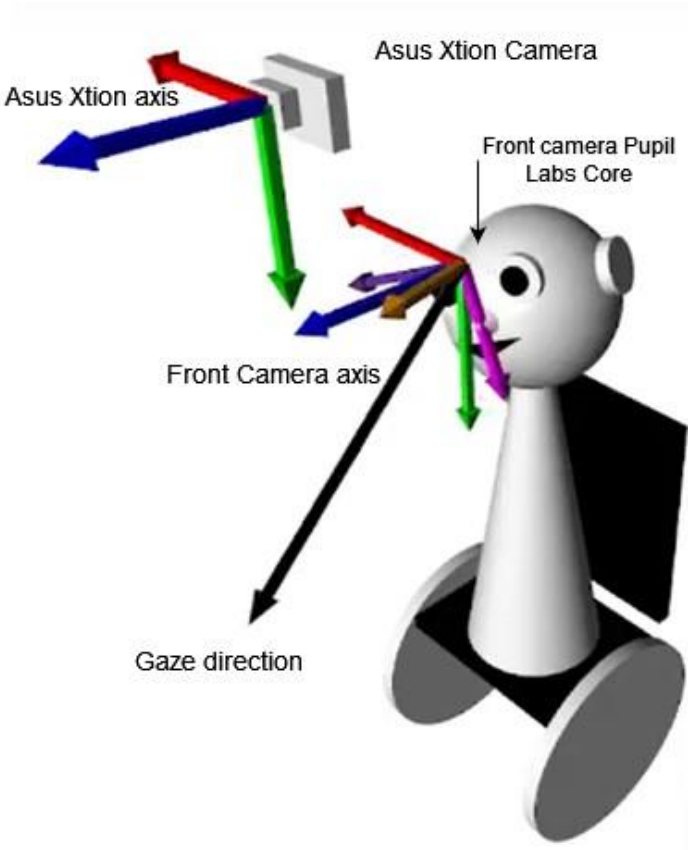

Figure 28. Scene created in VPython.

4.3 Object detection (M3)

4.3.1 Hardware

Pupil Labs Core headset

For this module, only the Pupil Labs Core headset eye-tracking glasses were needed, whose specifications have already been seen in a previous module, Section 4.1.1. In this case, the resolution chosen is 1280 x 720 px for the front camera of the headset and 192 x 192 for the eye cameras.

4.3.2 Software

Pupil Labs software

The software used is the same as in the previous modules, Pupil Labs software. This software in its execution shows and executes two processes "world" and "eye". In this module, a new plugin has been created that accesses the variable events to obtain information about the frame of the scene and the fixation of the user's gaze. In addition, it displays on the "world" screen the detection of the different entities as well as the entity on which the user is gazing.

4.3.3 Algorithms

YOLOv5 is the algorithm used for object detection. You Only Look Once (YOLO) is a fast and simple deep convolutional neural network which performs all its predictions with only a single fully connected layer allowing it to process the images in real-time. Among different detection algorithms, this one has been chosen because it can be used in real-time. For this project version, Yolov5 is used although, there are different versions of YOLO. Yolov5 was created as an open-source project in 2020 by Ultralytics. Yolov5 uses the EfficientDet architecture that it is based on the EfficientNet network achieving higher accuracy with fewer parameters, faster processing on GPU/CPU and lower FLOPs (Floating point Operations Per Second) than its previous versions [64], [65]. As can be seen in Figure 29, the EfficientDet network architecture is composed of an EfficientNet as the backbone network, the BiFPN [65] as the feature network

and a detection head with a class/box prediction net.

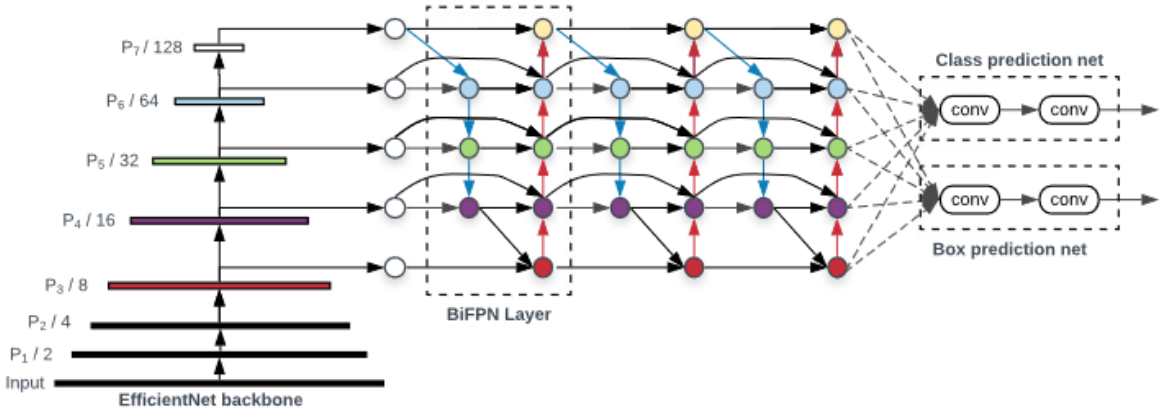

Figure 29. EfficientDet architecture [65].

Backbone network

The EfficientNet is a convolutional neural network (CNN) and scaling method in charge of the feature extraction of the input image. The first layers of a backbone network extract low-level features (low-quality features) such as lines, dots or curves while the layers on the top extract high-level features that have more information about the input image based on the features of previous layers [66].

The effectiveness of the EfficientNet depends on the baseline network and the scaling method. The baseline network is similar to MobileNetV2 and MnasNet since it uses inverted bottleneck convolutions (MBConv). The only difference is that EfficientNet is larger than mobile networks due to an increase in Floating-Point Operations [67]. The architecture of the baseline-B0 of the EfficientNet can be observed in Table 4.

Table 4. EfficientNet-B0 baseline architecture [68].

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224×224	32	1
2	MBConv1, k3x3	112×112	16	1
3	MBConv6, k3x3	112×112	24	2
4	MBConv6, k5x5	56×56	40	2
5	MBConv6, k3x3	28×28	80	3
6	MBConv6, k5x5	14×14	112	3
7	MBConv6, k5x5	14×14	192	4
8	MBConv6, k3x3	7×7	320	1
9	Conv1x1 & Pooling & FC	7×7	1280	1

The compounding scaling method of an EfficientNet is that the dimensions of the network, the width, the depth, and the resolution are uniformly scaled in all three dimensions by a fixed ratio. Other methods scaled only one dimension of the network in improving accuracy, but the accuracy gain diminishes for bigger models [68]. The use of a mobile-size EfficientNet model that is easy to scale improves accuracy and efficiency with fewer parameters and FLOPS. The dimensions of the EfficientNet are listed in a family of models with B0 being the smallest dimension model to B7 with the biggest.

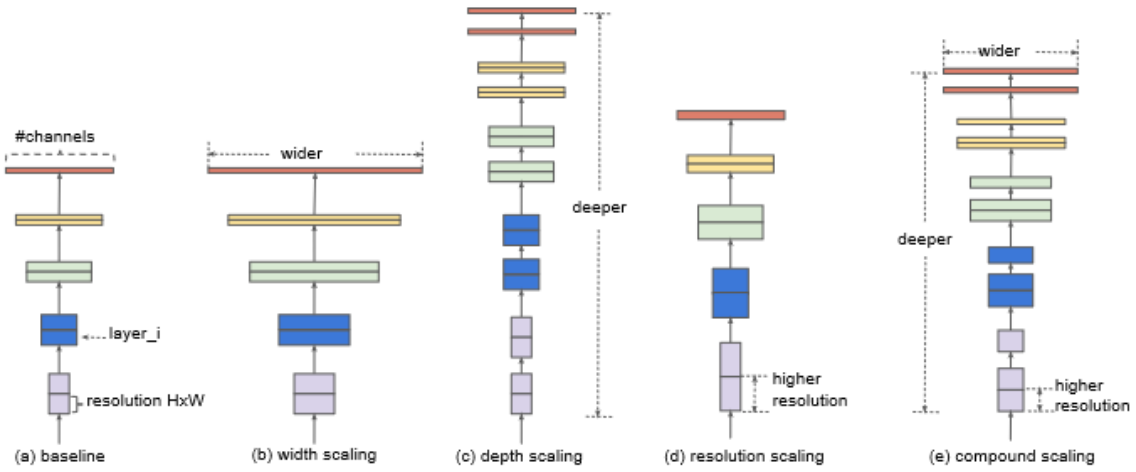

Figure 30. Model Scaling (a) baseline network. (b)-(d) conventional scaling only increases one dimension of the network and (e) compound scaling for all three dimensions with a fixed ratio [68].

Feature network

With the feature network it is possible to capture characteristics of different resolutions and combine all features. In other studies, they used Feature Pyramid Networks (FPN) as feature networks to multi-scale features however, they sum up different input resolutions without distinction. In EfficientDet, the feature network is a weighted Bi-directional Feature Pyramid Network (BiFPN) where weights are applied to the inputs to equally contribute to the output and at the same time, a top-down and bottom-up scale future fusion [65]. As can be seen in Figure 29, the BiFPN is a bi-directional network with more than one input edge in each node, there is an edge between the input and the output nodes of the same level and the bidirectional path is treated as a one-feature network layer and repeated multiple times to allow higher-level feature fusion. Furthermore, the feature network is also scalable but, instead of using a fixed ratio as the EfficientNet network, the authors [65] of the BiFPN use a simple compound coefficient to scale up the backbone, the feature and the class/box network at the same time using a heuristic-based scaling approach. It can be seen in Table 5, how the compound coefficient (ø) determines the input size, the model of the backbone network, the number of channels and layers of the BiFPN network and the number of layers of the box/class prediction network.

Table 5. Scaling configurations for different models of EfficientDet [65].

	Input size R_{input}	Backbone Network	BiFPN #channels W_{bifpn}	BiFPN #layers D_{bifpn}	Box/class #layers D_{class}
D0 ($\phi = 0$)	512	B0	64	3	3
D1 ($\phi = 1$)	640	B1	88	4	3
D2 ($\phi = 2$)	768	B2	112	5	3
D3 ($\phi = 3$)	896	B3	160	6	4
D4 ($\phi = 4$)	1024	B4	224	7	4
D5 ($\phi = 5$)	1280	B5	288	7	4
D6 ($\phi = 6$)	1280	B6	384	8	5
D7 ($\phi = 7$)	1536	B6	384	8	5
D7x	1536	B7	384	8	5

Detection head

After combining the features, the box/class prediction network is applied as a detection head. The box/class predicts the bounding boxes and the class of each object.

In addition to the EfficientDet architecture that YOLOv5 has, there are two more features compared to older versions of YOLO [64]. The first thing is that the bounding boxes are better aligned to the object's size and shape since it uses a dynamic generator of predefined

bounding boxes. And the second is that YOLOv5 uses a spatial pyramid pooling (SPP), a pooling layer that allows using different sizes of input images instead of a fixed image size [69].

4.3.4 Implementation steps

For the implementation of module 3, only a Python programming language file is needed. This file is a plugin created specifically for this module and it follows the Pupil Labs structure. The first step in this file is to load the convolutional network model used for object detection, YOLOv5. In this case, instead of training a model with a dataset using the YOLOv5 convolutional network, a pre-trained model is used. YOLOv5 has different pre-trained models that can be loaded using PyTorch Hub and used directly for inference: yolov5n, yolov5s, yolov5m, yolov5l, etc. The pre-trained model chosen is yolov5m because it has a good trade-off between detection capability and speed. The dataset used to pre-train the models is the MS COCO dataset and these models can classify by default 80 classes [62]. These 80 classes are of different categories and can be found indoors as well as outdoors: spoon, knife, cup, laptop, keyboard, stop sign, person, etc. In [63] can be found all classes that can be detected by YOLOv5. In the results section of this module, the accuracy of the model will be measured with respect to the images taken by the eye-tracking device camera, see Section 5.3.

After loading the model, the next step is to obtain the image of the scene so that the model can perform the inference. To access this image provided by the front camera of the Pupil Labs Core headset, it is necessary in the recent_events part of the class created for the plugin to access the variable "events['frame']". The resolution of this image is 1280 x 720 px. Once the image is obtained, it is provided to the model every time a voluntary fixation of the user's gaze occurs. To obtain the fixation, the variable "events["fixations"]" is accessed, which provides information about the normalised position of the gaze with respect to the Pupil Labs front camera, the fixation index, which is equal to the number of fixations so far and the timestamp. In this project, two types of fixations have been distinguished: voluntary and involuntary. Voluntary fixations are those in which the fixation lasts for a certain time and, therefore, during an interval of time the fixation is the same, the fixation index does not change. In this case, the time to discard an involuntary fixation is 1 second and the timestamp, fixation confidence and fixation index are used to filter them out. Once a voluntary fixation is detected, the model is provided with the image so that it can make the inference.

The result of the inference is 7 columns for each detected entity. The first four are the xmin, ymin, xmax and ymax coordinates within the pixel image of the bounding box created for the entity, the fifth is the confidence of the detection followed by the class and the last column, the name of the class. All detected entities whose confidence is high are displayed on the Pupil Labs "world" process screen, see Figure 31. To do this, bounding boxes are created by drawing lines through the coordinates provided in the inference results in the process callback "gl_display" created within the class.

The purpose of this module, apart from detecting all the entities in the scene, is to obtain the label or name of the entity on which the user is gazing. To obtain the position of the fixation of the gaze within the scene image, the normalised position of the gaze with respect to the Pupil Labs front camera obtained by accessing "events["fixations"]" is multiplied by the size of the image in pixels. The result is the position of the fixation in pixels within the image. If this position is inside the bounding box of one of the detected entities, a match is produced and as a result, the name of the entity is obtained. When there are several detected objects and the gaze fixation is within several bounding boxes, the one whose centre is closest to the fixation position is taken as the matching object. In addition, every time there is a match, the "world" process of the Pupil Labs app displays an "x" cross inside the bounding box of the entity the user is looking at and the name of the detected class. To display the match, the process callback "gl_display" is used once again. In Figure 31, the gaze fixation is on the keyboard, although the algorithm also detects the cup and the screen. As a result, when a match is made, a cross in "X" is shown inside the bounding box of the entity of the match and in the upper left part it is indicated with which entity the match has been made, in the figure "Match with: keyboard" is shown.

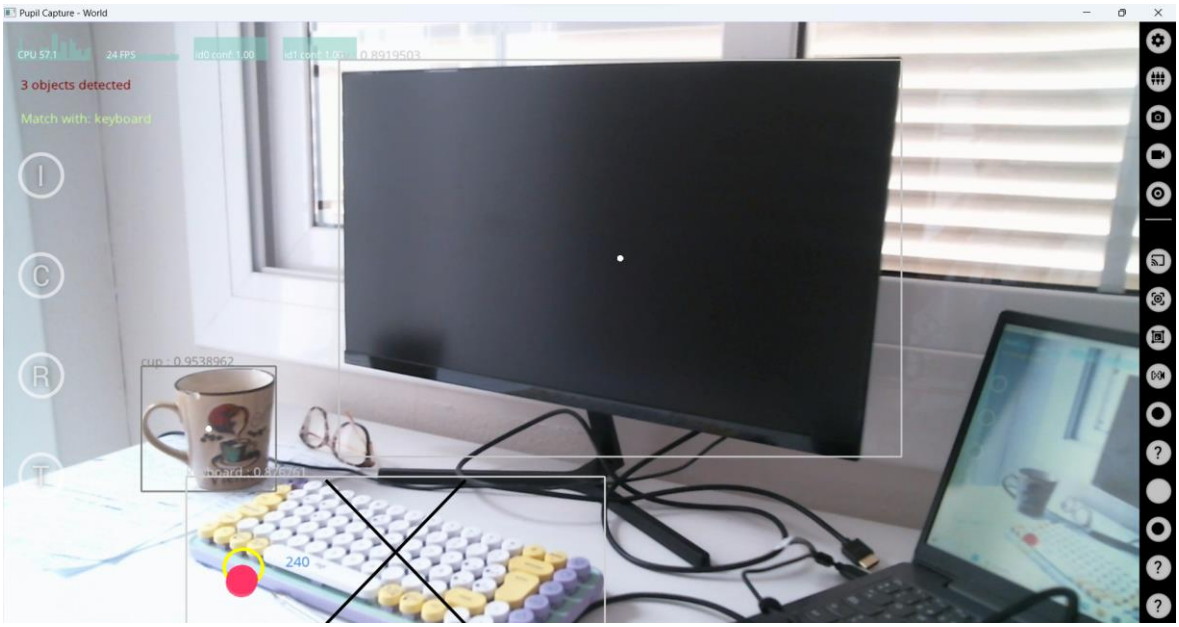

Figure 31. Module 3 matching.

4.4 2D Gaze estimation (M4).

4.4.1 Hardware

Pupil Labs Core headset

To obtain the gaze, the Pupil Labs eye-tracking device has been used, whose specifications are the same as in Section 4.1.2 except that the resolution of the two eye cameras is 192 x 192 px and the scene camera is 1280 x 720 px.

4.4.2 Software

Pupil Labs software

Although the Pupil Labs software is not used to obtain the 2D gaze in real-time, it is used to perform the calibration with the marker choreography. Calibration using marker choreography with the Pupil Labs software is very easy. When the app is launched, two processes/displays are executed "world" and "eye". In the "world" tab in the middle left, different markers appear, one of them being the calibration marker. Once the calibration is executed, the user must look at the different markers and once the calibration is finished, the software saves the data. The software has the advantage that it automatically saves all calibration data in files with a "pldata" extension that can be easily accessed using python.

libuvc

To use the Pupil Labs eye-tracking device without using the Pupil Labs software for real-time gaze, it is first necessary to download the open-source libuvc library. This library allows to interact with USB video capture devices using different operating systems, Linux and Windows. It recognises headset cameras without the need to install specific drivers. More information about this library is available at [36].

pyuvc

The second library that needs to be downloaded is pyuvc. This library allows to configure the UVC video cameras such as resolution, frames per second or to list all detected uvc devices. Also, it allows to capture the images taken by the cameras in mjpeg compressed format. More information in [37].

pupil_detectors

Once the images of the eyes have been obtained, it is necessary to extract the position of the pupils from them. To do this, the pupil-detectors library is used, which performs a 2D detection

of the pupils. This library receives the image of the eyes and, as a result, obtains for each eye the location of the pupil in the image space, the confidence of the detection, the diameter of the pupil, as well as the exact parameters of the pupil ellipse, such as the angle and the axes, see [38].

sklearn

sklearn, also known as scikit-learn, is an open-source machine learning library. The model that is trained and used to predict the 2D gaze is extracted from this library, a linear regression model, see [39].

4.4.3 Implementation steps

This module 4 has been implemented to obtain 2D gaze using the intervention of both eyes (binocular vision), and therefore, if only one eye is used this module does not work. All the steps mentioned below can be found in the Pupil Labs app only the 2D mapping within the app is done with the user interface.

2D gaze

Obtaining the 2D gaze of the user outside the Pupil Labs software using the mathematical model has two differentiated processes. The first is to train a model that does not have a dependency relationship between variables. The second process is the prediction of new gaze data from the previously constructed model. Since these are two different processes, they are also divided when it comes to explaining the implementation:

Training the model

The first step in training the model is to perform and run the calibration routine provided by the Pupil Labs app that uses the user interface. Once the calibration is done, the app automatically saves a file called "notify.pldata" with all the calibration information.

The next step is to load the file and extract from "notify.pldata" all the calibration information into the new Python file created for this module 4 and named "module4_tfm.py". In particular, the calibration information needed to train the model is the gaze position in normalised coordinates with respect the image taken by the scene camera, front camera of Pupil Labs headset and the corresponding normalised pupil position of the left eye and right with respect the images of the eye cameras of the Pupil Labs headset. Once the calibration data is obtained, it is filtered by choosing only those data whose confidence is higher than a threshold to avoid bad fitting. Since the normalised position of the left pupil and the right pupil with respect the eye camera images and the normalised gaze reference data with respect the scene camera image are obtained at different times, with different time stamps, the closest time

stamps between the three data need to be matched. If there is a large time dispersion between the three, they should be rejected.

Once the matches are obtained, the features to train the model are the independent variables: the normalised position of the left and right pupil, and as a dependent variable the normalised position of the gaze. The problem is that if the model is trained with these features, the dimensionality of the feature space is very small and therefore the model is not able to capture complex dependencies between variables. To increase the dimensionality of the feature space for a good fit and to capture the non-linear relationships between variables, the independent variables are transformed to be of polynomial type. For this purpose, the number of independent variables is increased from two to six for each eye. The first two variables are the normalised position in the x-coordinate and in the y-coordinate. The third variable is the multiplication of the normalised x-coordinate and the normalised y-coordinate. The fourth and fifth variables are the normalised x and y coordinates squared. And the last variable is the multiplication of the fourth and fifth variables. Thus, there are 12 independent variables, 6 for each eye.

After increasing the dimensionality of the feature space, the dependent and independent variables are fitted into the model. Since polynomial features are used, the mathematical model used is a linear model that can capture the relationships of the polynomial terms. Once the model has been fitted for the first time, outliers need to be iteratively removed and therefore it is necessary to iteratively refit the model on a subset of the data until a good fit is obtained. To identify outliers, the difference between the dependent variable obtained in the calibration and the dependent variable obtained by predicting the fitted model with the same independent variable used to train it is calculated. The result is the difference in the normalised gaze position of the two variables with respect to the screen and to obtain this difference in pixels, the result is multiplied by the screen resolution. To obtain the error, the norm 2, the Euclidean distance, is performed along the x and y coordinates, for each row. If this error is greater than a threshold, the data are considered outliers and should therefore be discarded. All data whose error value is below the threshold are stored as a subset of the data and are used to refit the model. This process can be seen in Figure 32. After obtaining a good fitting of the model, the next step, the prediction of the 2D gaze, can be taken.

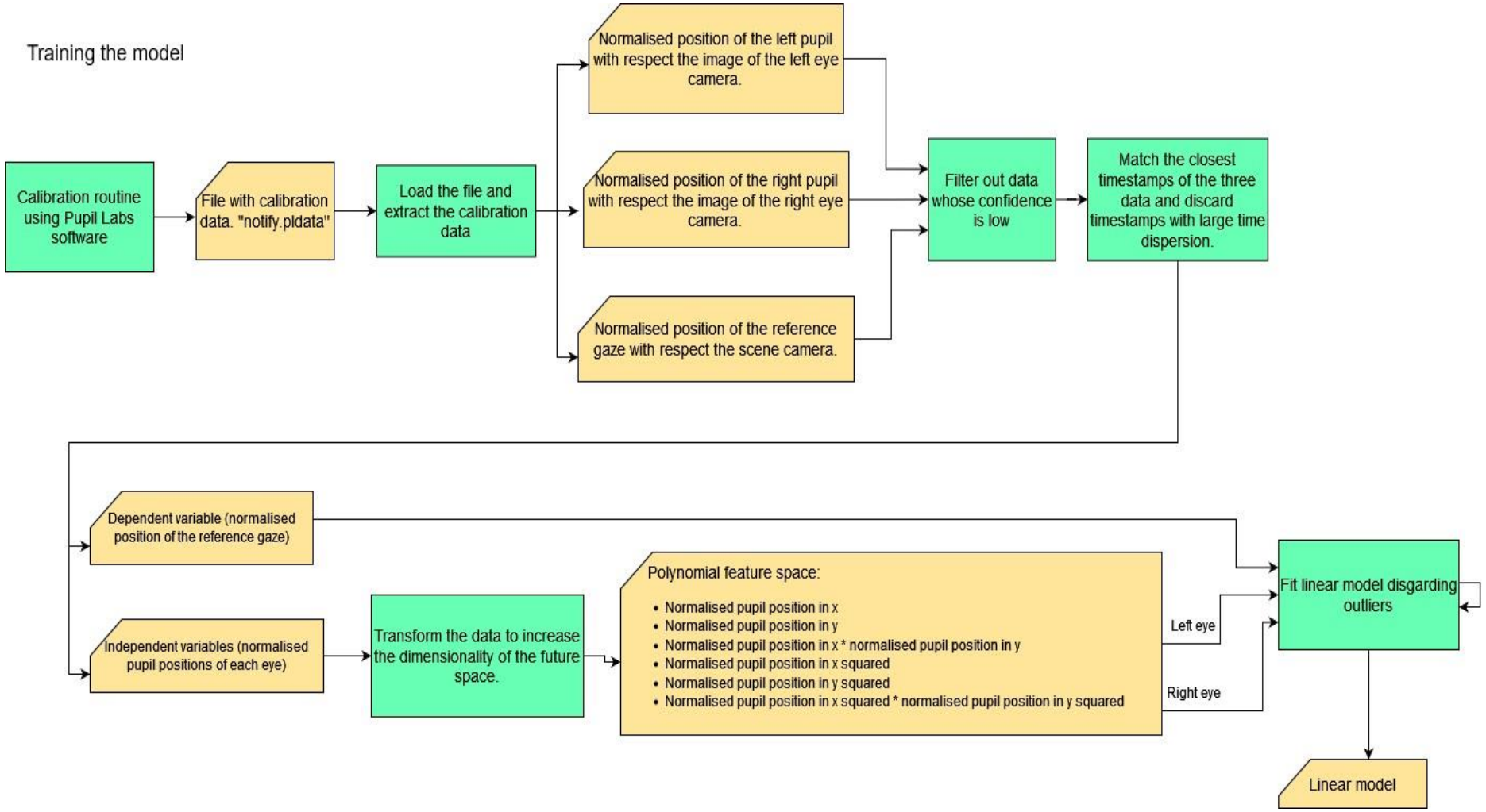

Figure 32. Model training for 2D mapping.

Predicting the 2D gaze

To make the prediction, the calibration data is no longer used, only the previously trained model. To make the prediction, it is necessary to provide to this trained model with input variables, the independent variables. These variables are of the same nature as those used to train the model, the normalised position with respect to the image size of the right and left pupil. To detect these two positions, it is first necessary to access the Pupil Labs headset cameras using the "libuvc" library, set the resolution and then, from these cameras extract the images of the eyes using the "pyuvc" library. To detect the 2D position of the pupils in the left and right eye images, the "pupil-detectors" library is used. If the confidence in the detection is high, the prediction proceeds but otherwise the values are discarded. To perform the prediction, it is necessary, as previously done in the model fitting, to increase the dimensionality of the independent variables, the normalised position with respect to the image size of the right and left pupil. To do this, it's increased from two to six independent variables for each eye: the normalised position in x-coordinate, the normalised position in y-coordinate, the normalised position in x-coordinate multiplied by the y-coordinate, the normalised position in x-coordinate squared, the normalised position in y-coordinate squared, and the multiplication of the squared normalised positions in x- and y-coordinate. The result is 12 independent variables. Once the variables have been transformed, the dependent variable can be predicted. This variable is the 2D normalised position of the user's gaze relative to the size of the front camera image of the Pupil Labs headset. To obtain this position in pixels, it is only necessary to multiply the normalised position by the size of the image. This process explained above to obtain the 2D mapping can be seen in Figure 33.

Two things must be highlighted in the 2D gaze acquisition. The first is that the 2D gaze is only obtained in pixels, as the data obtained in the calibration and used to train the model are in pixel units. The second is that the 2D mapping is sensitive to slippage of the eye-tracking device when the wearer wears the glasses for a long period of time because the headset can move. Since the polynomial regressor is trained by giving as input variable the position of the pupil eyes and the coordinates on the gaze screen, if a slippage of the eye-tracking headset occurs, the correlation changes and therefore, the trained mathematical model cannot be used again.

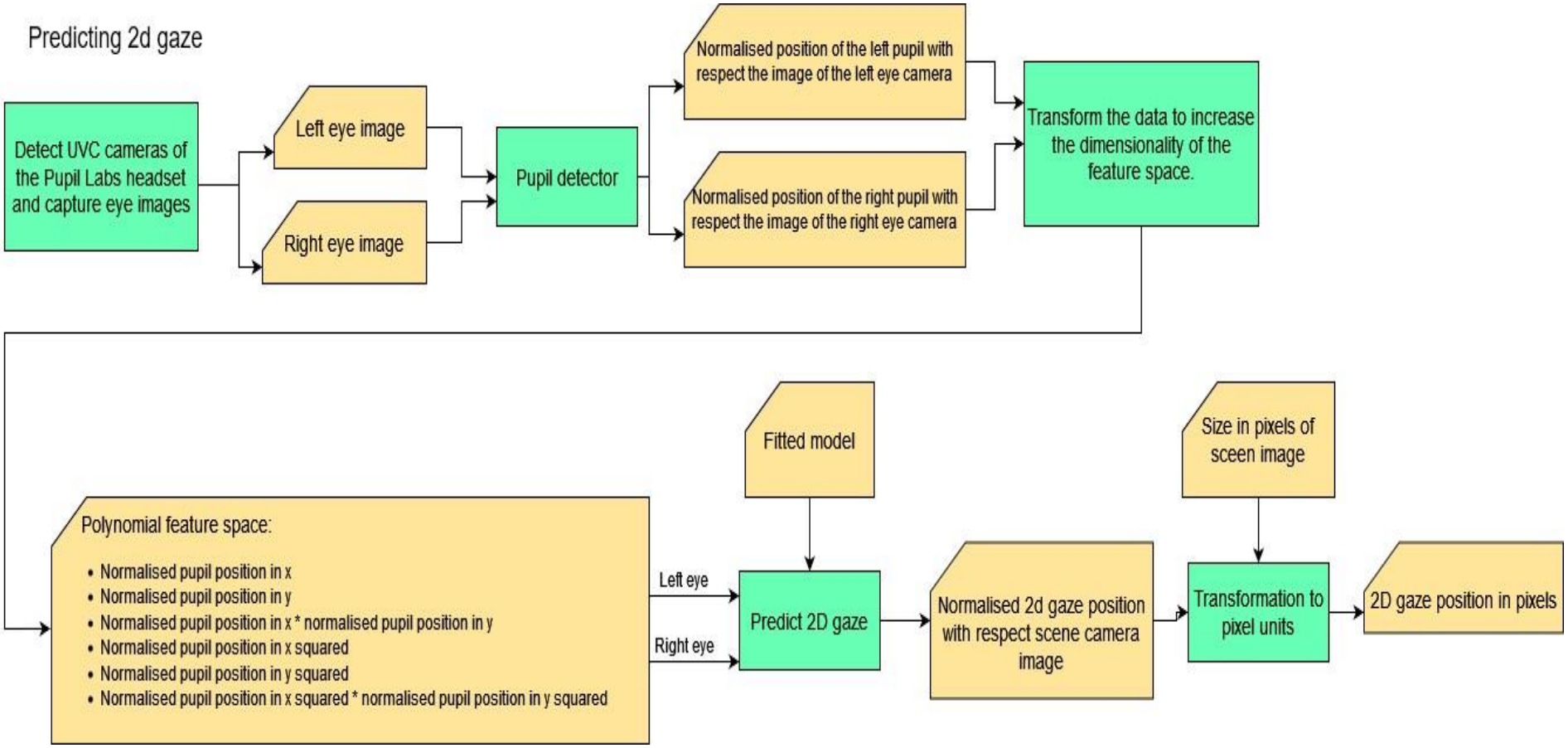

Figure 33. Model prediction for 2D mapping.

Chapter 5

5 Results and Analysis

5.1 Movement and seat position control of the robotic wheelchair (M1)

The following video link shows the results of the implementation:

https://drive.google.com/file/d/1-4qMHfrDtcNh1PMDGVz861cNmDs9r5Ec/view?usp=sharing

This video shows three windows of the Pupil Labs software. On the right of all, the image of the eyes in real-time (left eye up and right eye down). In the middle left, the image of the environment is obtained in real-time with the Pupil Labs front camera. The position of the gaze is shown in the environment image with a large red dot. At the bottom left of the image are three lines of text, the top one in blue indicates the current configuration or mode the wheelchair is in. The other two indicate the commands for controlling the wheelchair's movement.

The user starts with configuration "0" of the wheelchair (the current configuration can be seen at the bottom left of the video under "mode") which corresponds with the movement of the wheels of the wheelchair. The commands given before moving to the next configuration are the commands to move the wheelchair forward left as the gaze is in the upper left corner (the commands can be seen in the lower-left corner of the video where F for Forward and L for Left appear). Afterwards, the user fixes his gaze in the centre to give commands to the wheelchair to stop the movement (two N's corresponding to neutral appear in the lower left corner of the video). Later, the user changes the wheelchair setting to "1" through a voluntary blink. And finally, he/she indicates that he/she wants to move the upper backrest backwards by placing the gaze downwards to the right (in the lower left corner of the video an R for Right and B for Backwards appear).

Two types of information are extracted in this module, the blinking of the eyes to change the configuration of the wheelchair and the position of the gaze to control the movement of the wheelchair in the environment or the different seat positions. The analysis of this module is therefore carried out in two parts: the first is to check the blink detection system and the second the control of the wheelchair with the gaze position.

As discussed in the implementation part of this module, blinks are filtered into voluntary and involuntary blinks. Involuntary blinks are those that are made unconsciously by the user and are usually 4 per minute. To filter them, the user must blink twice in a row in less than 0.5 seconds. If only one or two blinks are made but in a time interval longer than 0.5 seconds, it is not considered a blink that the user makes expressly to change the wheelchair configuration. To test the blink detection system, 110 voluntary blinks were analysed. As a result of the analysis, 100 voluntary blinks were well detected while 10 were not, 90,9 % voluntary blinks detected. At no time was involuntary blinking detected as voluntary. The detection system can therefore be considered acceptable.

Two observations can be drawn from the analysis of the wheelchair control via the gaze position. The first relates to the user's gaze and the second to the user's experience of controlling the wheelchair.

To obtain a good gaze accuracy, a calibration must be carried out beforehand. In this module, as in all the others, calibration is carried out with a computer monitor where the 5 markers are shown on the screen at a fixed distance from the user. This type of calibration is a problem when the user moves with the wheelchair, because to obtain a good gaze accuracy the user must look at the objects in the environment at a similar distance to the one used in the calibration. If the user moves, then the distance from the user to the objects changes and therefore the accuracy is not good. This observation was obtained once this module was finished being implemented, so as a correction and future improvement work on this module would be to calibrate the system with different markers at different distances. For this, it is not possible to use the calibration routine provided by Pupil Labs for 5 markers or single marker, instead a new file would have to be created with the new specific calibration routine.

Although it has been concluded that the accuracy of the gaze is not good, the video shows that the user can still obtain the different commands with the gaze (forward left, backward right...) because, for the wheelchair control the user only must focus the gaze on the region where he/she wants to obtain the command and stay there. These regions can be seen delimited in Figure 11.

It has also been observed that the control of the wheelchair is uncomfortable for the user because the module constantly captures the user's gaze. This means that the user can never relax their gaze or look in any direction other than the direction in which the user wants to move the wheelchair.

5.2 Gaze estimation relative to the wheelchair (M2)

The results of the implementation of module 2 can be seen in the following video:

https://drive.google.com/file/d/1uaqb3mMKnBJULtZIXURzSNlD8b7vjfpB/view?usp=sharing

In the video, on the left, the real-time images of the front camera and the eyes are shown with the Pupil Labs software. In the middle, the scene built with VPython (see Section 4.2.4.4 to see what is presented) and on the right, at the top, the terminal window with all the information that has been measured from the module: the head displacement (yaw, pitch, roll), the position of the 3D gaze relative to the front camera in millimetres, the distance between cameras (translation matrix), the position of the 3D gaze rotated to the wheelchair reference system, the position of the 3D gaze relative to the wheelchair and finally, the code processing time (time between collecting the images from the cameras and obtaining the results). On the bottom right, the real wheelchair user is shown with the eye-tracking device and the environment camera attached to the wheelchair, the Asus Xtion. At the beginning of the video the user keeps his head fixed and only moves his gaze, places his gaze in different positions in the environment and as a result, in the VPython scene, the black arrow corresponding to the 3D gaze position with respect to the front camera moves. The results obtained in the terminal window for bearing, bank and roll angles in this case are close to zero because the user has not yet rotated his head. Next, the video shows how the user starts to move the head, first by doing a bearing clockwise and then counterclockwise. Later, the user performs the head lift clockwise and then counterclockwise and finally, a head roll first clockwise and then counterclockwise. All the values obtained of rotation and gaze are shown on the right side of the video, terminal window.

To analyse this module and to be able to decide whether the values of the gaze position with respect to the wheelchair are good, it is necessary to analyse the user's head rotation (bearing, elevation and bank angles) that are used for the rotation matrix, the distance between the front camera and the camera attached to the wheelchair (translation matrix) and finally, the accuracy of the user's gaze position with respect to the front camera.

After testing and as shown in the video the head rotation angles are not exact but are close to the actual rotation of the user. When the head is straight and looking straight ahead, the values for all head rotation angles are very close to zero as there has been no rotation as can be seen in the video. When the user moves the head and does a clockwise bearing rotation approximately if it moves 20 degrees, the code approaches those values but with some offset and the same happens with the rotation in elevation and bank. As can be seen in the video, because the rotation is obtained through image matching, when the two cameras do not capture similar images, no angle is obtained and, in the VPython scene, where the user with

the wheelchair is shown, it stops. In this case, although nothing is shown in the scene, the programme still analyses the images and when it finds two matching images, it shows the result of the rotation of the head in the scene and in the terminal window. The impossibility to obtain the rotation of the head and therefore the gaze with respect to the wheelchair at certain times is a serious problem if the module were to be used in the autonomous control of the wheelchair. Therefore, to always obtain rotation it is necessary to add other devices or techniques. It is worth mentioning that, as can be seen in the video, at some moments the calculation of the rotations is erroneous due to a failure in the matching of the images, for example, it could be that false descriptor points have been obtained in the matching of the images or that the value given by the Xtion camera for the depth of the middle descriptor point is erroneous. This error causes the displayed user's head to make some unintended head movement in the video.

The distances between cameras (translation matrix) are considered constant in this module and in reality, the distance between cameras is not fixed. As soon as the front camera on the user's head moves relative to the Asus Xtion camera attached to the wheelchair, the translation matrix can no longer be considered constant. The movement of the user's head can be due to different causes. For example, the user may move around in the wheelchair by changing position because he/she does not find it comfortable or by changing the different inclinations of the seat. It should also be considered that when the user rotates their head, whether they want to or not, and even if they try to do a small movement, there will be a change in the distance between the cameras. In this module it has been considered constant, although for greater precision of the user's gaze with respect to the wheelchair it would be necessary to consider implementing a system for measuring and monitoring the distance between the two cameras in real time.

Finally, the accuracy of the gaze position with respect to the wheelchair will also be determined by the accuracy of the gaze position obtained with the Pupil Labs Core eye-tracking device. In order to obtain a good gaze position in this device, so that the gaze distance obtained from the user is the same distance as where the user is actually looking, a good calibration has to be carried out. In this case, as in the other modules, the calibration is carried out by means of 5 markers whose positions in the scene camera coordinates are related to certain parameters of the eye such as the position of the centre of the pupils. This calibration has been performed at a certain distance and through a computer monitor. Therefore, the accuracy of the gaze is higher when the objects the user is looking at are at the same distance as the distance at which the calibration was done. If the user, for example, chooses to look at distances further away than the calibration distance, the accuracy of the gaze given by the device is lower. Since the user's head rotates and when rotating the distances between the user and the objects can be different, and if, in addition, the user moves around with the wheelchair in the environment, the calibration that has been done so far using the computer monitor is not adequate.

It can be concluded that obtaining the user's gaze with respect to the wheelchair does not give a satisfactory value due to the influence of the multiple variables and factors discussed above. However, it is important to note that the steps followed in the implementation, including the mathematical equations applied, have been consistent with the established approach.

In addition, there are times when gaze information cannot be acquired due to cameras capturing different images of the environment. This poses a significant challenge since it is imperative to consistently obtain gaze data when implementing gaze tracking within an autonomous wheelchair control system.

5.3 Object detection (M3)

The results of the implementation of module 3 can be viewed at the following link:

https://drive.google.com/file/d/1URN0PMbOot8aVww69qkLMU_6fqgH7OTB/view?usp=sharing

In the video a large image of the environment taken by the front camera of Pupil Labs can be seen and in the lower left corner the images of the eyes. The user's gaze is reflected in the image of the environment with a large red circle. In addition, this image also shows in the upper left corner the position of the gaze relative to the front camera in millimetres, the number of objects detected and if a match has occurred, the name of the class of the object on which the user's gaze was fixed. At the beginning of the video, the user does not fix his gaze on any object until the user fixes his gaze on one of the cups. When the user fixes his gaze on the cup for a second, several objects are detected, although the cup on which he fixes his gaze is not detected. The user then decides to look at another cup, the cup whose design is all white but is very close to another cup. Being very close, the algorithm detects both but matches the other because the centre of its bounding box is closer to the gaze position. Later, it fixes its gaze on a keyboard, a chair, a mouse and a computer in which they are well detected by the algorithm and a match is made, with the name of the class in which it occurred being displayed at the top left of the image.

First, as an analysis in this module, it is important to check whether the detection of entities in the environment using the YOLOv5 algorithm is acceptable or not. As seen previously, YOLOv5 can detect eighty classes of entities. These classes are objects that can be found both indoors and outdoors. Since analysing the detection of all eighty classes would be a very extensive analysis, it is decided to analyse the detection of two classes of objects that can be very frequent to see when the user uses the wheelchair. The first class of object analysed is chairs, and it has been chosen because when the user is in a wheelchair and must move around, one of the most common obstacles he/she will encounter is chairs.

For the analysis, 46 images were taken with a total of 57 chairs, some images had more than one chair. In Figure 34, four of the images used for the detection are shown. In the top left of these images, a red number corresponding to the class number can be seen in some of them. In the case of the chairs, the class number is 56. If no number appears, no object has been detected. In the case of the upper and lower right image, the chairs have been detected because they have the class number 56 in the image. On the other hand, in the upper left image, another object class has been detected, whose number, 41, corresponds to the class number of the cups. In the lower left part, no object has been detected because there is no number.

The algorithm only detected 20 chairs out of 57 chairs, 35% of all chairs. This result is very low. One positive thing is that at no time has any other kind of object been detected on the chairs shown in the image. This result may be because not enough chair models with different geometric shapes have been considered when training the neural network. In addition, in some of the images used for the analysis, the chair was occluded by an object, or the full shape of the chair could not be seen.

Figure 34. Four images with chairs used for YOLOv5 analysis.

To check that the detection is not only bad with one of the classes, another object has been used, in this case, cups. This object was chosen because if the robotic arm must pick up objects, cups are very common. Fifty-four images of cups are analysed with a total of 96 cups, some images have more than one cup. In Figure 35, four images of the 54 used are shown. As can be seen in the figure, in the upper and lower right images no cup has been detected

as there is no number. In the upper left image, the two cups have been detected because there are two numbers 41. In the lower left image, the cup has not been detected but another object, which corresponds to the class number of the computer mouse.

The result of all images is 33 cups detected, 34.4% of all cups. In none of the images were the cups detected as another object. This result is still low and the reasons for it being so low are the same as those due to the chair class.

Figure 35. Four images with mugs used for YOLOv5 analysis.

The proposal to improve this detection data is to train the neural network using a new data set instead of using an already trained YOLOv5 model by default. In this new data set, I would add images of the objects that I would be most interested in detecting and reduce the detection problem by reducing the number of object classes to be detected.

Once the algorithm that detects the objects has been tested, a qualitative analysis of the module is carried out. As explained above in the implementation part, if the user's gaze is fixed for more than one second, the objects in the environment are detected. If the user's gaze is on one of the detected objects, a match is made and the class name of the object on which the user is gazing is obtained. If there are several objects whose bounding boxes have the gaze position inside, the match is made with the object whose bounding box centre is closest to the gaze position. This match is shown on the screen with a cross in "X". As can be seen in Figure 36, the user wants to fix his gaze on the computer but obtains as a match the cup. This is because the gaze position was closer to the centre of the cup's bounding box than to the computer. This can be considered a problem because the user is detecting another object that he does not want to detect with his gaze. As can be noticed, when the detected objects are

separated from each other, there is no problem with the match but if the objects are close together and in the field of view some are occluded by others, the matching can be wrong.

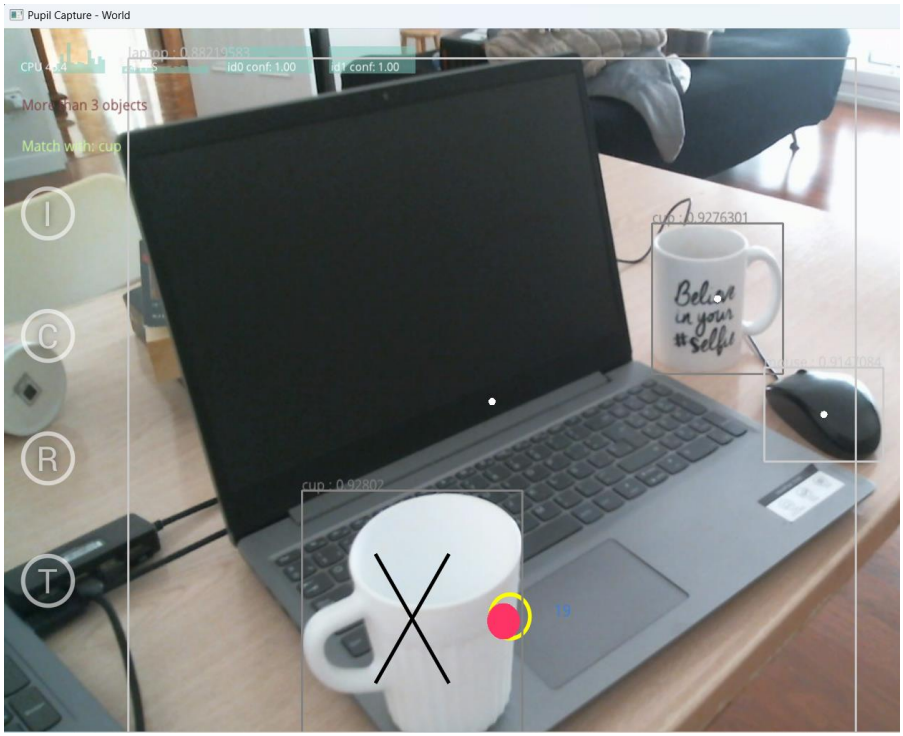

Figure 36. Object detection and fixation matching using module 3 code.

5.4 2D Gaze estimation (M4)

The results of the module 4 can be viewed in the following link:

https://drive.google.com/file/d/1SFKuBfG5VkXiopatNGKaaQCNwLNB4X_Z/view?usp=sharing

This video starts with the calibration of the Pupil Labs eye-tracking device using the Pupil Labs software. The calibration consists of a choreography of 5 markers where the user has to fix the gaze on each of the markers without moving the head. Once the glasses are calibrated, the user's gaze position is displayed using Pupil Labs software. The position is shown with a red circle and it can be seen how the user looks at different objects: first a cup, then a book, then a bottle and finally a bottle of water. Specifically, the user focuses his gaze on the centre of the crosses on each of the objects. This is intended to show the accuracy of 2D gaze acquisition using the Pupil Labs software. As can be seen and given that the calibration has been done at the same distance at which the user looks at the objects, the accuracy is very high, as the user's gaze calculated by the Pupil Labs software remains in the centre of the crosses of all the objects.

The Pupil Labs program is then closed, and the accuracy of the gaze is compared with the program that has been created for this module. To do this, the user performs the same action, which is to look at the centre of the crosses of the different objects. As can be seen, the 2D gaze obtained with the program created also provides good accuracy since the calculated 2D gaze also remains in the centre of the crosses of each of the objects. Although the video shows the image of the environment, the image of the eyes and the position of the gaze, the program created has also the option of not showing any image and only obtaining the data of the 2D gaze when the program is executed, which is the purpose of the creation of this module.

Next, a more detailed analysis of how accurate the 2D gaze is with the created program is carried out. For this, the four objects shown in the video are used again: the two books, the bottle and the cup. To perform the analysis, the error is obtained by comparing the position in coordinates of the object's crosshairs in the environment image with the 2D position calculated by the program when the user looks at the centre of the object's crosshairs.

As indicated above, the program created uses a polynomial model to predict the 2D gaze that is trained with the data from the previous calibration performed with the Pupil Labs software. For a rigorous analysis, the calibration of the device must be performed several times to check whether the error obtained is due to the calibration or to the polynomial model used.

Table 6 shows all the measurements taken. It shows for each calibration and object, the position of the object in the coordinates of the image (real position) and the position of the user's 2D gaze calculated by the program in the same image (gaze position). The data is displayed by first providing the x-coordinate of the position in the image (horizontal coordinate) and then the y-coordinate (vertical coordinate) in pixels. As can be seen, the position of the objects in the image (position of the centre of the crosses) changes because the image is captured from a different perspective. This table also shows the error in each of the measurements, the mean absolute error of the 2D gaze position for each object and the average error obtained taking into account all the measurements. This last error was 12 pixels of error in the x-coordinate and 14.5 pixels of error in the y-coordinate (12 x 14.5). A very small error compared to the image size of 1280 x 720 pixels.

The results are therefore very satisfactory as good accuracy of the user's 2D gaze is achieved as in the Pupil Labs software, but with the improvement that it is not necessary to have the graphical user interface open.

Table 6. 2D gaze error.

Calibration number		**Bottle**	**Book**	**Computer mouse**	**Cup**
Nº1	Real position (pixels)	693 x 419	640 x 363	642 x 357	625 x 370
	Gaze position (pixels)	670 x 420	632 x 370	630 x 350	610 x 360
	Error (pixels)	**23 x 1**	**8 x 7**	**12 x 7**	**15 x 10**
Nº2	Real position	678 x 384	735 x 370	747 x 348	704 x 362
	Gaze position	680 x 380	730 x 370	740 x 320	690 x 336
	Error	**2 x 4**	**5 x 0**	**7 x 28**	**86 x 26**
Nº3	Real position	738 x 389	738 x 389	687 x 364	672 x 395
	Gaze position	750 x 400	720 x 380	680 x 360	670 x 390
	Error	**12 x 11**	**18 x 9**	**7 x 4**	**2 x 5**
Nº4	Real position	734 x 402	715 x 402	648 x 361	646 x 390
	Gaze position	740 x 400	690 x 380	630 x 380	620 x 410
	Error	**6 x 2**	**25 x 18**	**18 x 19**	**26 x 20**
Nº5	Real position	728 x 383	714 x 349	666 x 347	654 x 380
	Gaze position	740 x 340	720 x 320	670 x 320	660 x 360
	Error	**12 x 43**	**6 x 29**	**4 x 27**	**6 x 20**
Mean absolute error		11 x 12,2	12,4 x 12,6	9,6 x 17	27 x 16,2
Total mean absolute error					12 x 14,5

Chapter 6

6 Budget and time planning

6.1 Budget

In this chapter, the cost incurred in the life of this project is split into four categories: rental, material, energy and staff costs. Furthermore, the final total cost of the project is calculated. It must be mentioned that the calculation is based on an estimation of the real cost.

Rental costs

The project has been developed in the Robotics lab which belongs to the Automatic Control and Computer Engineering department in Campus Nord. The estimated cost of the work area used is shown in Table 7.

Table 7. Rental costs final project.

Type	Rental Cost (€/month)	Usage Time (months)	Total Cost (€)
Area of work	200	7	1.400

Material costs

The equipment used for the project involves a computer and two cameras. The laptop is a Lenovo IdeaPad 3 [70] and the cameras are an Asus Xtion Pro Live [71] and a Pupil Labs Core [72]. In addition, the price of the robotic wheelchair is also included; although the exact price is not known, an average price is chosen for wheelchairs with the same features: chair batteries, motors, and seats. Because none of the cameras, the computer or the robotic wheelchair has not been bought, especially for the project, the material costs are only the depreciation costs, the costs incurred during the project for the use of the material. According to Law 27/2014 on Corporate Income Tax defined by the Spanish Government [73], the depreciation of cameras considered "Electronic equipment" has a maximum linear coefficient of 20%. The computer considered "Information processing equipment" has a maximum linear coefficient of 25%. In the case of the wheelchair, it has been considered an "internal transport

element" with a coefficient of 10%. The depreciation costs are calculated based on a worst-case scenario, considering the maximum linear coefficient.

The depreciation costs can be calculated following Equation (28).

$$Depreciation\ Cost = Maximum\ linear\ coefficient \cdot Purchase\ Cost \cdot \frac{Usage\ Time}{12} \quad (28)$$

The material cost are shown in Table 8.

Table 8. Material costs final project.

Device	Purchase Cost (€)	Usage Time (months)	Maximum linear coefficient (Percentage)	Depreciation Cost (€)
Laptop	649	7	25	94,65
Asus Xtion Pro Live Camera	300	7	20	35
Pupil Labs Core Camera	2850	7	20	332,5
Robotic Wheelchair	3100	2	10	51,67

Energy costs

The energy costs of this project are the costs of consuming electrical energy to charge the batteries of the wheelchair as well as the energy consumed by the computer and the lighting of the work area. As the robotic wheelchair was only used for the last two months before the end of the project, the energy costs were broken down as the computer and workstation were used for seven months. Completing this master's thesis required 20 working days per month with four hours of work per day. In this case, the electricity consumption of the cameras is not considered because the computer battery powers them. As the electricity is consumed at specific times of the day, the use time has been counted in hours.

According to the technical specifications of the laptop, the laptop consumes 38 W [70]. The battery of the wheelchair, considering a similar wheelchair on the market, is estimated to consume 320 W for each motor, considering two motors [74]. Finally, the lighting of the workstation is provided by fluorescent lamps consuming 30 W each, with six lamps in the working area.

The evolution of the PVPC tariff, Voluntary Price for Small Consumers, which follows the regulated electricity tariff is 0.1105 € per kWh in 2023.

Therefore, considering all the above, the energy costs are shown in Table *9*.

Table 9. Energy costs final project.

	Power Usage (W)	**Months (month)**	**Time of Use (hours)**	**Energy (kWh)**	**Total Cost (€)**
Laptop	38	7	560	21,28	2,35
Working area	180	7	560	100,8	11,14
Robotic wheelchair	640	2	160	102,4	11,32

Staff costs

Three people took part in the master's degree project. The person in charge of implementing the interface to move the wheelchair worked for 7 months, 20 days a month and 4 hours a day, in total about 560 hours. The average salary of a researcher at the Polytechnic University of Catalonia[75] is €1,280.1 per month full-time, which is €8 per hour.

There were two lecturers in charge of tutoring this master's thesis. Assuming a dedication of 10% of this master's thesis, this would correspond to 56 hours each. The average salary of a lecturer at the UPC is €21.47 per hour [75].

The staff costs are shown in Table *10*.

Table 10. Staff costs final project.

Staff	Dedication (hours)	Average Salary (€/h)	Total Cost (€)
Engineer	560	8	4.480
Professors	112	21,47	2.404,64

Final total costs

The total costs are a sum of all the costs described above. Since they are an estimate of the real cost, a deviation of 25% has been considered, taking as the final cost the deviation that would give the highest cost. Therefore, the final total cost of the project is shown in Table 11.

Table 11. Final total costs of the final project.

Type of cost	Cost (€)
Rental costs	1400
Material costs	513,82
Energy costs	24,81
Staff costs	6.884,64
Total Cost	**8.823,27**
Cost deviation (percentage)	25%
Final total cost	**11.029,09**

6.2 Time planning

Since a master's thesis takes months to complete, it is very important to plan and organise it well. For the organisation of this project, the Gantt chart has been used, a tool widely used by professionals who manage projects. A Gantt chart is characterised by visually showing the division and relationship of the tasks to be carried out and the dedication needed for each task to be completed. This project has been divided into three phases that include different tasks: research, project development, and project draft. In the research phase, the best algorithms, convolutional networks, cameras for the project, etc. are investigated. In the development phase, the project is designed, created, programmed, and implemented and the last phase is the documentation of the project, which is what can be seen in this report, and which is written during the life of the project. Figure 37 shows the project phases and tasks.

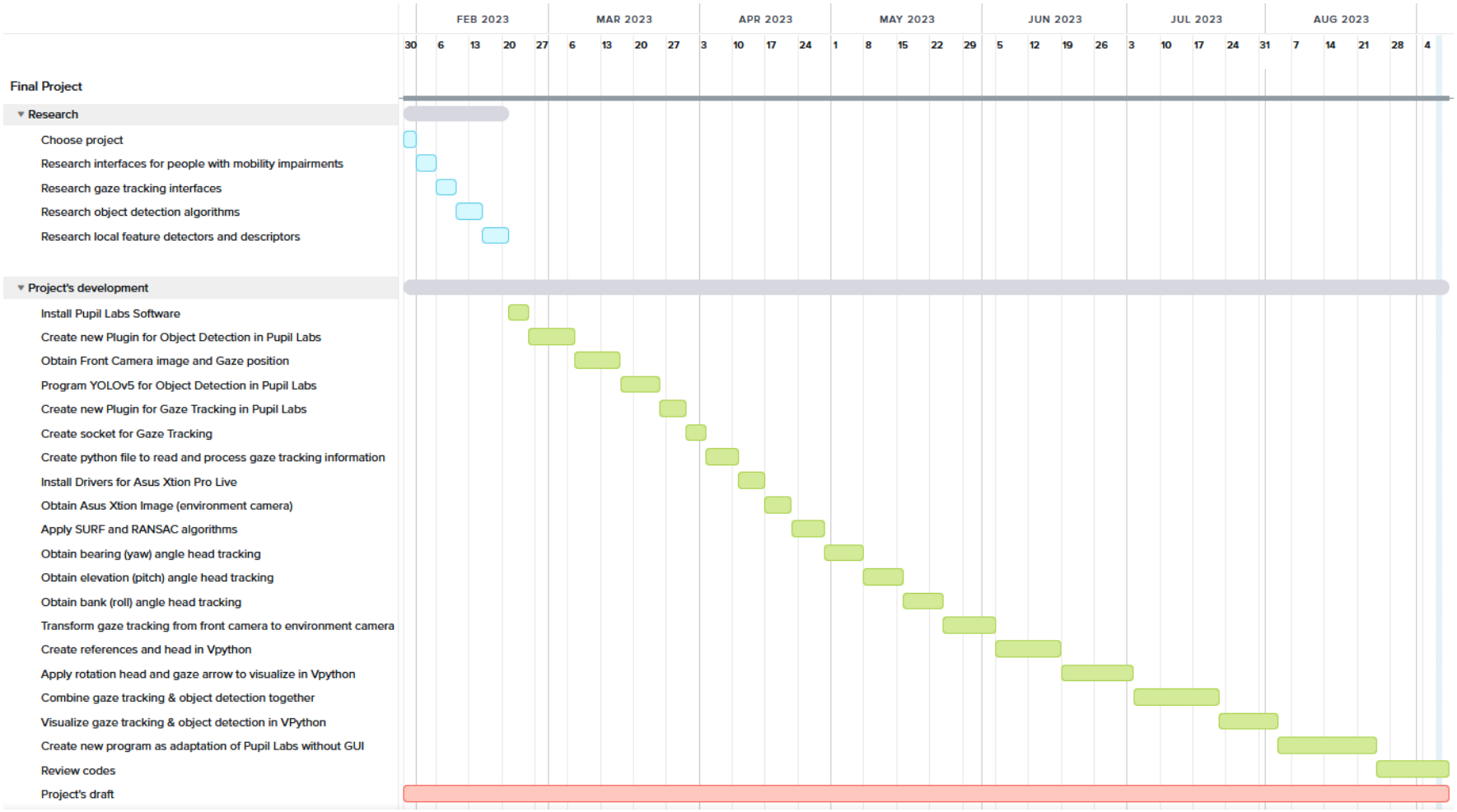

Figure 37. Project's Gantt Diagram.

Chapter 7

7 Social, Economic and Environmental Impact

The social (Section 7.1), economic (Section 7.2), and environmental impact (Section 7.3) can be viewed from two perspectives: the impact that the project has already made and the future impact. Since it is not possible to test the project in a real scenario with people with physical impairments since it is part of a research project that has not been finished yet, the present and future impact can only be analysed in a general way without any quantification. Furthermore, the future impact has been made by predicting what it is believed that it might be.

7.1 Social impact

The social impact that this project generated was the opportunity for a master's student who had no experience in working on a research project to work on it. This involvement in a research environment has contributed to the student developing skill competencies and knowledge in installing and programming software and in computer vision and deep learning techniques. Furthermore, this project has helped me develop a research mindset and empowered me towards the possibility of a research career.

In the future, this project will have a great impact on physically disabled people. It will bring the opportunity for disabled people to be more independent when using the wheelchair, boosting self-esteem and empowerment. Furthermore, it will help them in their day to day minimising the emotional and physical burden they have.

7.2 Economic impact

The economic impact of the project is described in Chapter 6.

In the future, the economic impact of this master's thesis and the research project to which it belongs is a saving for people with mobility problems in personal care costs, as the wheelchair will offer greater autonomy to users, requiring less assistance from other people or

professionals. Furthermore, if this type of robotic wheelchair is developed safely and reliably for all users, it will allow the creation of a new assistive technology market with economic opportunities for all types of professionals, especially those specialising in this type of technology.

7.3 Environmental impact

The main environmental impact that this project has generated is related to the power consumption of the equipment and the working area that this project has used for its development. To quantify the impact of the project activity, the carbon footprint, an environmental indicator that reflects the total amount of greenhouse gases emitted into the atmosphere by the activity of a company or individual, is used as a measure. In this case, it is an approximation of the carbon footprint generated by the consumption of electricity to charge the batteries of the wheelchair as well as the consumption by the computer and the lighting of the work area.

As it has been shown in Chapter 6 in the section of “Energy costs”, the energy consumption was: 21,28 kWh for the laptop, 100,8 kWh for the fluorescent lamps and 102,4 kWh for the robotic wheelchair batteries, a total of 224.48 kWh. Considering the electricity supply is from non-renewable energy sources and using the Ecodes calculator [76] the estimated carbon footprint is 92,045 kg of CO2.

The environmental impact this project may have in the future is the carbon footprint emitted due to the electricity consumed to charge the wheelchair batteries.

Chapter 8

8 Conclusions and Future Work

Globally, it can be concluded that the objectives of the work have been adequately met.

The first objective was to provide the user with a simple control to control easily the movement of the wheelchair and the position of the different parts of the seat. This objective has been achieved, although it has been observed that the accuracy of the gaze was not good. However, the user could still move the wheelchair as he/she only had to look in different regions to obtain the different commands to move the wheelchair. Furthermore, from the user experience, it has been observed that the application was not very convenient to use by the user, as the user had to always give instructions on where he/she wanted to move with his/her eyes and could not be distracted.

The second objective was to take the first steps for the autonomous control of the wheelchair by detecting the objects in the environment, the object on which the user was focusing the gaze and obtaining the user's gaze with respect to the wheelchair to know the approximate distance at which the user wanted to move. In terms of detection, it has been observed that the pre-trained convolutional network used, YOLOv5, was not the most suitable because the detection rate was very low. In addition, the matching of objects with the user's gaze gave errors when the objects in the field of vision were very close and covered by each other.

In obtaining the gaze with respect to the wheelchair, it was observed that the result was conditioned by three factors: the rotation of the user's head, the distance between cameras and the 3D gaze with respect to the front camera of the headset. In the first of these, obtaining the user's head rotation angles, it was noticed that the image matching technique used presented a major challenge, if the cameras were looking at different areas of the environment, obtaining the rotation angles was impossible. Secondly, it was found that taking the distance between the cameras as constant could lead to errors. Thirdly, obtaining the 3D gaze with respect to the front camera was assessed that to achieve good accuracy, the calibration, as in the previous case, had to be carried out at different distances.

The last objective was to try to improve the Pupil Labs software to obtain 2D gaze data without the need for a user interface when the program was running. To do this, the program created took as input data the calibration performed with the Pupil Labs software and the images of the eyes in real-time. With the calibration data, it trained a polynomial regression model and, using software libraries, detected the position of the pupils. After training the model, the 2D

position of the gaze was predicted. The results of the implementation of the improved software were very satisfactory because the error obtained between the real position where the user looks, and the calculated position was small.

Future work

As future work, the following ideas are mentioned to try to solve all the problems observed in the implementation of the modules.

- Create a calibration routine in which the markers are at different distances from the user. The aim of this improvement is to obtain greater accuracy of gaze with the Pupil Core Labs device when the user is moving around with the wheelchair.

- Improve the user's experience in controlling the movement of the wheelchair and the position of the different parts of the seat through gaze and blinking. To do this, instead of continuously asking for the user's gaze, only obtain it when the user gives the order.

- For future autonomous control of the wheelchair and to correctly obtain the gaze with respect to the wheelchair, a device should be added to know the distance between the cameras in real-time. In addition, a technique other than image matching should be applied to always obtain the rotation of the user’s head.

- Create a neural network with its own data set to increase the number of object detections. In addition, apply other object detection techniques that do not use bounding boxes such as semantic segmentation to improve, when there is occlusion between objects, the match between the object that the user wants to detect and the one detected by the module.

- Continue to improve the adaptation of the Pupil Labs software for gaze acquisition with the Pupil Labs Core headset without the use of the user interface. For this, the next step would be to obtain the gaze position in 3D.

Personal conclusions

During my master's thesis, I was faced with several challenges that demanded significant effort. My research focused on the use of an eye-tracking device that was initially completely unknown to me in terms of how it worked and how I could modify the device's software code to suit the objectives of my study.

One of the first barriers I had to overcome was the installation of the camera drivers and associated libraries. This task was complicated by incompatibilities between the different

hardware and software elements. In addition, I had to find a way to provide a solution on how to fix and replace one of the cameras of the Pupil Labs Core headset that had broken down. However, I feel that this experience provided me with valuable knowledge in the field of software installation and technical troubleshooting, skills that I consider fundamental in my training.

In addition, during my research, I came across different techniques and algorithms that were completely new to me, which I had never had the opportunity to use before. This allowed me to expand my knowledge in computer vision, Python programming and artificial intelligence.

One of the most interesting aspects of my research was to learn how to go from obtaining raw pupil position data to obtaining information about the user's gaze. This process led me to delve into the inner workings of the eye-tracking device, an aspect we often take for granted.

Despite the many challenges I faced, and the effort invested, the master’s thesis has been an extremely enriching experience as I have not only managed to broaden my technical skills and knowledge but also gained greater confidence in my ability to tackle complex projects and solve technical problems.

Bibliography

[1] "History of the Wheelchair - Science Museum Blog." https://blog.sciencemuseum.org.uk/history-of-the-wheelchair/ (accessed Jul. 05, 2023).

[2] "The history of electric wheelchairs - dietz-mobility.co.uk." https://dietz-mobility.co.uk/the-history-of-electric-wheelchairs/ (accessed Jul. 07, 2023).

[3] Farnell. An avnet company., "Robotic Wheelchair." https://uk.farnell.com/robotic-wheelchair-applications (accessed Jul. 07, 2023).

[4] "Disability." https://www.who.int/news-room/fact-sheets/detail/disability-and-health (accessed May 10, 2023).

[5] "La movilidad reducida, primera causa de discapacidad en España - Comunicación ASPAYM." https://comunica.aspaym.org/movilidad-reducida-discapacidad-espana/ (accessed May 10, 2023).

[6] U. Nations, "Envejecimiento | Naciones Unidas", Accessed: Jul. 18, 2023. [Online]. Available: https://www.un.org/es/global-issues/ageing

[7] A. Ruíz-Serrano, M. C. Reyes-Fernández, R. Posada-Gómez, A. Martínez-Sibaja, and A. A. Aguilar-Lasserre, "Obstacle avoidance embedded system for a smart wheelchair with a multimodal navigation interface," in *2014 11th International Conference on Electrical Engineering, Computing Science and Automatic Control, CCE 2014*, Institute of Electrical and Electronics Engineers Inc., 2014. doi: 10.1109/ICEEE.2014.6978290.

[8] L. Montesano, M. Díaz, S. Bhaskar, and J. Minguez, "Towards an intelligent wheelchair system for users with cerebral palsy," *IEEE Trans Neural Syst Rehabil Eng*, vol. 18, no. 2, pp. 193–202, Apr. 2010, doi: 10.1109/TNSRE.2009.2039592.

[9] H. A. Yanco, "Integrating Robotic Research: A Survey of Robotic Wheelchair Development".

[10] F. Peralta, M. Arzamendia, D. Gregor, D. G. Reina, and S. Toral, "A comparison of local path planning techniques of autonomous surface vehicles for monitoring applications: The Ypacarai lake case-study," *Sensors (Switzerland)*, vol. 20, no. 5, Mar. 2020, doi: 10.3390/s20051488.

[11] "What is a Joystick? - Definition from Techopedia."

https://www.techopedia.com/definition/31108/joystick (accessed May 23, 2023).

[12] “Wheelchair control - newVSI - CURTISS-WRIGHT - integrated.” https://www.directindustry.com/prod/curtiss-wright/product-4591-1691689.html (accessed May 11, 2023).

[13] S. Mitra, S. Member, and T. Acharya, “Gesture Recognition: A Survey,” *APPLICATIONS AND REVIEWS*, vol. 37, no. 3, 2007, doi: 10.1109/TSMCC.2007.893280.

[14] A. Menache, “Motion Capture Primer,” *Understanding Motion Capture for Computer Animation*, pp. 1–46, 2011, doi: 10.1016/B978-0-12-381496-8.00001-9.

[15] “Switch Interfaces | Assistive Technology | eSpecial Needs.” https://www.especialneeds.com/shop/assistive-technology/switches/switch-interfaces.html (accessed May 23, 2023).

[16] “Alternative Drive Controls: Head Array, Sip & Puff, and Switch Arrays & Multiple-Switch Control.” https://hub.permobil.com/blog/alternative-drive-controls-head-array-sip-puff-switch-arrays (accessed May 23, 2023).

[17] “Voice User Interface: Key Benefits and Challenges | Vilmate.” https://vilmate.com/blog/voice-user-interface-and-speech-recognition/ (accessed May 11, 2023).

[18] “Sip & Puff Switches & Accessories | Performance Health.” https://www.performancehealth.ca/sip-puff-switch-accessories (accessed May 11, 2023).

[19] “What are Voice User Interfaces? | IxDF.” https://www.interaction-design.org/literature/topics/voice-user-interfaces (accessed May 11, 2023).

[20] “Advantages of Voice User Interfaces | Speechly.” https://www.speechly.com/blog/advantages-of-voice-user-interfaces/ (accessed May 11, 2023).

[21] “Tips for Designing Accessibility in Voice User Interfaces | by Bo Campbell | UX Collective.” https://uxdesign.cc/tips-for-accessibility-in-conversational-interfaces-8e11c58b31f6 (accessed May 11, 2023).

[22] A. F. Klaib, N. O. Alsrehin, W. Y. Melhem, H. O. Bashtawi, and A. A. Magableh, “Eye tracking algorithms, techniques, tools, and applications with an emphasis on machine learning and Internet of Things technologies,” *Expert Syst Appl*, vol. 166, Mar. 2021,

doi: 10.1016/J.ESWA.2020.114037.

[23] D. A. Robinson, “A Method of Measuring Eye Movement Using a Scleral Search Coil in a Magnetic Field,” *IEEE Transactions on Bio-medical Electronics*, vol. 10, no. 4, pp. 137–145, 1963, doi: 10.1109/TBMEL.1963.4322822.

[24] “What is BCI? | Cumming School of Medicine | University of Calgary.” https://cumming.ucalgary.ca/research/pediatric-bci/bci-program/what-bci (accessed May 10, 2023).

[25] J. J. Shih, D. J. Krusienski, and J. R. Wolpaw, “Brain-Computer Interfaces in Medicine,” *Medical Education and Research Mayo Clin Proc*, vol. 87, no. 3, pp. 268–279, 2012, doi: 10.1016/j.mayocp.2011.12.008.

[26] J. D. R. Millán *et al.*, “Combining brain-computer interfaces and assistive technologies: State-of-the-art and challenges,” *Front Neurosci*, vol. 4, no. SEP, p. 161, Sep. 2010, doi: 10.3389/FNINS.2010.00161/BIBTEX.

[27] S. Saha *et al.*, “Progress in Brain Computer Interface: Challenges and Opportunities,” *Frontiers in Systems Neuroscience*, vol. 15. Frontiers Media S.A., Feb. 25, 2021. doi: 10.3389/fnsys.2021.578875.

[28] “Pupil Core - Eye tracking platform technical specifications - Pupil Labs.” https://pupil-labs.com/products/core/tech-specs/ (accessed Jul. 24, 2023).

[29] “Plugin API | Pupil Labs.” https://docs.pupil-labs.com/developer/core/plugin-api/ (accessed Jul. 24, 2023).

[30] “GitHub - pupil-labs/pupil: Open source eye tracking.” https://github.com/pupil-labs/pupil#installing-dependencies (accessed Aug. 18, 2023).

[31] “pupil/pupil_src/shared_modules/plugin.py at master · pupil-labs/pupil · GitHub.” https://github.com/pupil-labs/pupil/blob/master/pupil_src/shared_modules/plugin.py (accessed Jul. 25, 2023).

[32] “Getting Started | Pupil Labs.” https://docs.pupil-labs.com/core/ (accessed Jul. 27, 2023).

[33] “OpenNI 2 Downloads and Documentation | The Structure Sensor.” https://structure.io/openni (accessed Aug. 18, 2023).

[34] “VPython Help.” https://www.glowscript.org/docs/VPythonDocs/index.html (accessed Aug. 18, 2023).

[35] “No installation.” https://vpython.org/presentation2018/noinstall.html (accessed Aug. 18, 2023).

[36] “GitHub - libuvc/libuvc: a cross-platform library for USB video devices.” https://github.com/libuvc/libuvc (accessed Aug. 30, 2023).

[37] “GitHub - pupil-labs/pyuvc: python binding to libuvc.” https://github.com/pupil-labs/pyuvc (accessed Aug. 30, 2023).

[38] “GitHub - pupil-labs/pupil-detectors.” https://github.com/pupil-labs/pupil-detectors/tree/master (accessed Aug. 30, 2023).

[39] “sklearn.linear_model.LinearRegression — scikit-learn 1.3.0 documentation.” https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html (accessed Aug. 31, 2023).

[40] “¿Qué es el User Datagram Protocol (UDP)? | Cloudflare.” https://www.cloudflare.com/es-es/learning/ddos/glossary/user-datagram-protocol-udp/ (accessed Aug. 16, 2023).

[41] “What Is Focal Length and How to Use It - 42West.” https://www.adorama.com/alc/understanding-focal-length-and-how-to-use-it/ (accessed Aug. 17, 2023).

[42] “Defining the principal point.” https://catalyst.earth/catalyst-system-files/help/concepts/orthoengine_c/Chapter_45.html (accessed Aug. 17, 2023).

[43] “Camera Calibration and 3D Reconstruction — OpenCV 2.4.13.7 documentation.” https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html (accessed May 09, 2023).

[44] “What Is Camera Calibration? - MATLAB & Simulink.” https://www.mathworks.com/help/vision/ug/camera-calibration.html (accessed Aug. 19, 2023).

[45] “Introduction To Feature Detection And Matching | by Deepanshu Tyagi | Data Breach | Medium.” https://medium.com/data-breach/introduction-to-feature-detection-and-matching-65e27179885d (accessed May 09, 2023).

[46] M. Hassaballah, A. A. Abdelmgeid, and H. A. Alshazly, “Image features detection, description and matching,” *Studies in Computational Intelligence*, vol. 630, pp. 11–45, 2016, doi: 10.1007/978-3-319-28854-3_2.

[47] E. Oyallon and J. Rabin, “Submitted on 2013-02-12, accepted on 2015-05-30,” *Int J Comput Vis*, vol. 60, pp. 91–110, 2009, doi: 10.5201/ipol.2015.69.

[48] “Introduction to SURF (Speeded-Up Robust Features) | by Deepanshu Tyagi | Data Breach | Medium.” https://medium.com/data-breach/introduction-to-surf-speeded-up-robust-features-c7396d6e7c4e (accessed May 09, 2023).

[49] H. Bay, T. Tuytelaars, and L. Van Gool, “SURF: Speeded Up Robust Features”.

[50] “Hessian Matrix | Brilliant Math & Science Wiki.” https://brilliant.org/wiki/hessian-matrix/ (accessed May 09, 2023).

[51] M. Brown and D. Lowe, “Invariant Features from Interest Point Groups”.

[52] “Introduction to SURF (Speeded-Up Robust Features) | by Deepanshu Tyagi | Data Breach | Medium.” https://medium.com/data-breach/introduction-to-surf-speeded-up-robust-features-c7396d6e7c4e (accessed May 17, 2023).

[53] “Speeded up robust features - Wikipedia.” https://en.wikipedia.org/wiki/Speeded_up_robust_features (accessed May 17, 2023).

[54] “OpenCV: Feature Matching + Homography to find Objects.” https://docs.opencv.org/3.4/d1/de0/tutorial_py_feature_homography.html (accessed May 18, 2023).

[55] “OpenCV: Feature Matching.” https://docs.opencv.org/4.x/dc/dc3/tutorial_py_matcher.html (accessed May 18, 2023).

[56] J. Xingteng, W. Xuan, and D. Zhe, “Image matching method based on improved SURF algorithm,” *Proceedings of 2015 IEEE International Conference on Computer and Communications, ICCC 2015*, pp. 142–145, Jan. 2016, doi: 10.1109/COMPCOMM.2015.7387556.

[57] Z.-P. Cai and D.-G. Xiao, “Feature matching algorithm based on KAZE and fast approximate nearest neighbor search,” 2014.

[58] D. G. Lowe, “Distinctive Image Features from Scale-Invariant Keypoints,” *Int J Comput Vis*, 2004.

[59] “Dealing with Outliers: RANSAC | Image Stitching - YouTube.” https://www.youtube.com/watch?v=EkYXjmiolBg (accessed May 19, 2023).

[60] J. D. Foley, M. A. Fischler, and R. C. Bolles, “Graphics and Image Processing Random

Sample Consensus: A Paradigm for Model Fitting with Apphcatlons to Image Analysis and Automated Cartography," 1981.

[61] "OpenCV: Optical Flow." https://docs.opencv.org/3.4/d4/dee/tutorial_optical_flow.html (accessed May 08, 2023).

[62] "Training YOLOv5 custom dataset with ease | by Luiz doleron | MLearning.ai | Medium." https://medium.com/mlearning-ai/training-yolov5-custom-dataset-with-ease-e4f6272148ad (accessed Aug. 21, 2023).

[63] "yolov5/data/coco.yaml at master · ultralytics/yolov5 · GitHub." https://github.com/ultralytics/yolov5/blob/master/data/coco.yaml (accessed Sep. 01, 2023).

[64] "YOLO Algorithm for Object Detection Explained [+Examples]." https://www.v7labs.com/blog/yolo-object-detection (accessed May 11, 2023).

[65] "EfficientDet: Scalable and Efficient Object Detection", Accessed: May 11, 2023. [Online]. Available: https://github.com/google/automl/tree/

[66] "What Does Backbone Mean in Neural Networks? | Baeldung on Computer Science." https://www.baeldung.com/cs/neural-network-backbone (accessed May 11, 2023).

[67] "EfficientNet: Improving Accuracy and Efficiency through AutoML and Model Scaling – Google AI Blog." https://ai.googleblog.com/2019/05/efficientnet-improving-accuracy-and.html (accessed May 11, 2023).

[68] M. Tan and Q. V Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks".

[69] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition", Accessed: May 11, 2023. [Online]. Available: http://research.microsoft.com/en-us/um/people/kahe/

[70] "Portátil | Lenovo IdeaPad 3 15ITL6, 15.6" Full HD, Intel® Core™ i7-1165G7, 16GB RAM, 512GB SSD, Iris® Xe Graphics, Sin sistema operativo." https://www.mediamarkt.es/es/product/_portatil-lenovo-ideapad-3-15itl6-156-full-hd-intelr-coretm-i7-1165g7-16gb-ram-512gb-ssd-irisr-xe-graphics-sin-sistema-operativo-1552804.html?utm_source=google&utm_medium=cpc&utm_campaign=rt_shopping_generic_nsp_na_MM-ES-S-G-CAT-PLA-PMAX.PH-PROMO-ALL-ALL&gclid=CjwKCAjw04yjBhApEiwAJcvNoS1rwQ3KPYORFD_4IuIWMY7iunv2J000alEqL5NY79frsHKsvvzMchoCN0YQAvD_BwE&gclsrc=aw.ds (accessed May 16, 2023).

[71] “Xtion PRO LIVE.” http://xtionprolive.com/asus-3d-depth-camera/asus-xtion-pro-live (accessed May 16, 2023).

[72] “Pupil Core - Open source eye tracking platform - Pupil Labs.” https://pupil-labs.com/products/core/ (accessed May 16, 2023).

[73] “BOE-A-2014-12328 Ley 27/2014, de 27 de noviembre, del Impuesto sobre Sociedades.” https://www.boe.es/buscar/act.php?id=BOE-A-2014-12328&p=20230629&tn=1 (accessed Jul. 08, 2023).

[74] “Silla de Ruedas Eléctrica TODOTERRENO Latitude | Dortomedical.” https://dortomedical.com/sillas-de-ruedas-electricas-no-plegables/4102-silla-ruedas-electrica-todoterreno-luces.html?gclid=Cj0KCQjwkqSlBhDaARIsAFJANkiqSnC6sYq6CmUakZJw2O4KPd2hI3L93p9FbPqyfkxCxm8HeQRs-W4aAmj-EALw_wcB (accessed Jul. 08, 2023).

[75] “Taules retributives del personal docent i investigador Any 2022”.

[76] “Huella de carbono consumo eléctrico.” https://ecodes.org/tiempo-de-actuar/hogares-sostenibles/ahorro-energetico/calculadora-electricidad (accessed Jul. 08, 2023).

Annexes

List of Figures

List of Tables