UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

A NOVEL COMPUTER VISION FRAMEWORK TO ENHANCE ROBOT SKILLS

JAUME MORA LADÀRIA

Thesis supervisor
ANAIS GARRELL ZULUETA (Department of Automatic Control)

Thesis co-supervisor
ISIAH ZAPLANA AGUT (Department of Automatic Control)

Degree
Bachelor's Degree in Artificial Intelligence

Bachelor's thesis

Facultat d'Informàtica de Barcelona (FIB)

Universitat Politècnica de Catalunya (UPC) - BarcelonaTech

Abstract

This work aims to address several challenges in current robotics, particularly the accessibility of mobile robotics for non-expert users and the cost of many teleoperation interfaces. In practice, teleoperating mobile robots such as the TurtleBot3 Waffle Pi typically relies on physical controllers or wearable devices (e.g., watches or gloves). Although these solutions are reliable, they are often expensive and can reduce the naturalness of human–robot interaction, while also adding a barrier to entry for non-expert users. To address this, this work presents a computer-vision-based teleoperation system that enables a TurtleBot3 Waffle Pi to be controlled through hand gestures using only a standard RGB camera, avoiding specialized sensors or wearables and making teleoperation both simpler and more affordable.

The system has been implemented in `ROS 2 Jazzy Jalisco` and follows a distributed setup between a remote PC and the robot. Both the vision pipeline and the control node run on the remote PC, while the robot executes the motion commands and enforces on-board safety constraints.

The architecture follows a modular and efficient design. First, 21 hand landmarks are extracted in real time using `MediaPipe Hands`. These landmarks are then fed into a lightweight deep learning classifier (`EurekaNet`, based on the *EUREKA* approach) to recognize dynamic gestures. Finally, the control node translates the detected gestures into robot actions by publishing timed `TwistStamped` commands and applying safety constraints.

The model was trained on the `IPN Hand` dataset and validated through real experiments. The results show good learning performance (93.83% training accuracy) and consistent behavior in tests conducted by different users. The system runs in real time, and the perceived latency over WiFi is around 200–300 ms, which is suitable for safe teleoperation. Reliability and safety are further improved through confidence thresholds, temporal voting, a gesture-locking mechanism, an emergency-stop gesture, and LiDAR-based checks, which together promote conservative behavior under uncertainty.

Keywords: ROS 2 Jazzy, Computer Vision, TurtleBot3, MediaPipe, EurekaNet, Gesture Recognition, Human–Robot Interaction.

Resum

Aquest treball pretén afrontar diversos problemes de la robòtica actual com són l'accessibilitat per a no-experts de la robòtica mòbil i el cost de molts dispositius de control de robòtica mòbil. De fet, la teleoperació de robots mòbils com el TurtleBot3 Waffle Pi depèn habitualment d'interfícies físiques o dispositius wearables (com rellotges o guants). Tot i ser solucions fiables, solen ser costoses i a més a més limiten la naturalitat de la interacció humà-robot, a més del fet que afegeixen una barrera d'entrada per a usuaris no experts. Aquest treball presenta un sistema de teleoperació basat en visió per computador que permet controlar un TurtleBot3 Waffle Pi mitjançant gestos de la mà utilitzant únicament una càmera RGB estàndard, evitant sensors especialitzats o dispositius wearables, fent així que la teleoperació sigui molt més senzilla i molt més barata.

El sistema s'ha implementat sobre `ROS 2 Jazzy Jalisco`. L'execució del sistema es divideix entre un PC remot i el propi robot. El processament de visió i de control s'executa en el PC remot, mentre que el robot aplica el moviment i la seguretat a bord.

L'arquitectura del sistema segueix un disseny modular i eficient. En primer lloc, el pipeline de visió extreu en temps real els 21 landmarks de la mà amb `MediaPipe Hands` que serveixen com a entrada per a un classificador lleuger de deep learning (`EurekaNet`, basat en l'enfocament *EUREKA*) per reconèixer gestos dinàmics. Llavors, un segon node de control tradueix els gestos detectats en accions del robot publicant comandes temporitzades amb `TwistStamped` i aplicant restriccions de seguretat.

El model s'ha entrenat sobre el dataset `IPN Hand` i s'ha validat amb experiments reals. Els resultats mostren un bon aprenentatge (93.83% d'accuracy en train) i bon rendiment en els assajos realitzats per diferents usuaris. El sistema funciona en temps real i la latència percebuda en teleoperació sobre WiFi se situa al voltant de 200-300 ms, cosa que demostra uns resultats segurs per a la teleoperació. A més a més, la fiabilitat i la seguretat amb alguns aspectes claus en el projecte com són els llindars de confiança, la votació temporal, un mecanisme de locking de gestos, gest d'aturada d'emergència i comprovacions amb LiDAR que tots junts garanteixen un comportament conservador quan hi ha incertesa.

Paraules clau: ROS 2 Jazzy, visió per computador, TurtleBot3, MediaPipe, EurekaNet, reconeixement de gestos, interacció humà-robot.

Resumen

Este trabajo pretende afrontar varios problemas de la robótica actual como son la accesibilidad para no expertos de la robótica móvil y el coste de muchos dispositivos de control de robótica móvil. De hecho, la teleoperación de robots móviles como el TurtleBot3 Waffle Pi depende habitualmente de interfaces físicas o dispositivos wearables (como relojes o guantes). Aunque son soluciones fiables, suelen ser costosas y además limitan la naturalidad de la interacción humano-robot, aparte de que añaden una barrera de entrada para usuarios no expertos. Este trabajo presenta un sistema de teleoperación basado en visión por computador que permite controlar un TurtleBot3 Waffle Pi mediante gestos de la mano utilizando únicamente una cámara RGB estándar, evitando sensores especializados o dispositivos wearables, haciendo que la teleoperación sea mucho más sencilla y mucho más barata.

El sistema se ha implementado sobre `ROS 2 Jazzy Jalisco`. La ejecución del sistema se divide entre un PC remoto y el propio robot. El procesamiento de visión y control se ejecuta en el PC remoto, mientras que el robot aplica el movimiento y la seguridad a bordo.

La arquitectura del sistema sigue un diseño modular y eficiente. En primer lugar, el pipeline de visión extrae en tiempo real los 21 landmarks de la mano con `MediaPipe Hands` que sirven como entrada para un ligero clasificador de deep learning (`EurekaNet`, basado en el enfoque *EUREKA*) para reconocer gestos dinámicos. Entonces, un segundo nodo de control traduce los gestos detectados en acciones del robot publicando comandos temporizados con `TwistStamped` y aplicando restricciones de seguridad.

El modelo se ha entrenado sobre el dataset `IPN Hand` y se ha validado con experimentos reales. Los resultados muestran un buen aprendizaje (93.83% de accuracy en train) y buen rendimiento en los ensayos realizados por distintos usuarios. El sistema funciona en tiempo real y la latencia percibida en teleoperación sobre WiFi se sitúa en torno a 200-300 ms, lo que demuestra unos resultados seguros para la teleoperación. Además, la fiabilidad y la seguridad con algunos aspectos claves en el proyecto como son los umbrales de confianza, la votación temporal, un mecanismo de locking de gestos, gesto de paro de emergencia y comprobaciones con LiDAR que todos juntos garantizan un comportamiento conservador cuando existe incertidumbre.

Palabras clave: ROS 2 Jazzy, visión por computador, TurtleBot3, MediaPipe, reconocimiento de gestos, interacción humano-robot.

Agraïments

En primer lloc, vull expressar el meu més sincer agraïment als meus tutors del TFG, els doctors Anaís Garrell i Isiah Zaplana. Gràcies per la paciència constant, per ajudar-me quan estava encallat, per la confiança dipositada en mi i per facilitar-me el maquinari necessari per fer possible aquest treball. Però, per sobre de tot, gràcies per transmetre'm el vostre coneixement i passió en la visió per computador i robòtica.

A na Clara, gràcies per ser el meu suport dia a dia, per fer-me companyia sempre que ho he necessitat, per l'interès en tot el que faig i per ajudar-me amb totes les meves preocupacions. A la meva família: al meu pare i a la meva mare, en Xavier i na Sara; als meus germans, na Laia i en Joan; a les meves àvies Montse i Aina Maria i a la meva tieta Cristina. Gràcies per ser sempre el meu punt de suport ferm. Heu estat presents en cada etapa, però especialment durant els anys de carrera i en aquests mesos intensos de TFG. Gràcies pels ànims, per la paciència i també per implicar-vos directament en l'experimentació, ajudant-me a gravar-la i a provar el sistema de reconeixement de gestos.

Als meus amics, gràcies per ser-hi sempre que ha fet falta i per recordar-me la importància de desconnectar en els moments oportuns. En especial, a na Blanca i en Carlos, gràcies per ser un suport durant tot l'any i per implicar-vos de ple en l'experimentació, ajudant-me a validar el funcionament del sistema de visió.

Finalment, a l'Institut d'Organització i Control de Sistemes Industrials (IOC) i als companys de l'IOC, gràcies per haver-me obert les portes de les vostres instal·lacions, per haver-me proporcionat l'entorn necessari per desenvolupar aquest treball i per la predisposició constant a ajudar-me sempre que ho he necessitat.

Contents

Chapter 1

Introduction

This chapter provides an overview of the project, establishing the motivation behind the research and defining the context in which it has been developed. It also outlines the main contributions of this work, as well as the scope and the methodology followed to ensure a robust human-robot interaction system.

1.1 Motivation

The selection of the topic was mainly due to my strong interest in Computer Vision, which began with the VO (Visió per Ordinador) subject in the BSc in AI. Computer Vision comprises a wide range of algorithms, pipelines, and frameworks designed to enable machines to acquire, process, analyze, and understand visual information from images and videos. More specifically, this project focuses on how images and videos can be processed and transformed, for example, into control mechanisms for robots, a challenge that I wanted to explore and further develop in this TFG.

Beyond the academic scope, Computer Vision is the fundamental pillar that enables robots to operate in the real world. Nowadays, CV is indispensable across many different environments:

- **Industrial and logistics:** In automated warehouses, vision systems allow robots to navigate narrow aisles and perform complex bin-picking tasks.
- **Search and rescue:** In disaster zones or collapsed buildings, CV helps drones identify survivors through thermal imaging or shape recognition where other sensors fail.
- **Space and underwater exploration:** In environments where GPS is unavailable, robots rely on visual cues for localization and autonomous mapping of unknown terrains.
- **Domestic and healthcare:** Service robots in hospitals or homes use CV to recognize faces, detect falls, and interact safely with non-expert users.

In addition, this motivation is reinforced by the fact that, in a world like ours where society is digitizing very quickly, making systems, robots, and programs as intuitive and interactive as possible is key to not leaving behind older generations who may not be used to them. If we want robots to be part of our daily lives, they must be controllable by any type of user, not only by those with technical knowledge.

This is where my motivation to reduce as many barriers as possible and improve the relationship between humans and robots comes from.

1.2 Project context

This thesis is submitted as part of the Bachelor's Degree in Artificial Intelligence at the Facultat d'Informàtica de Barcelona (FIB), Universitat Politècnica de Catalunya (UPC).

The project lies at the intersection of two key subjects of the degree: Robotics and Computer Vision. Consequently, the development has been supervised by professors from both areas. The experimental phase was carried out within the facilities of the Institute of Industrial and Control Engineering (IOC), which provided the necessary workspace and all the support needed.

1.3 Contributions

The development of this project involved a comprehensive workflow, ranging from low-level hardware configuration to high-level deep learning integration. The main tasks and contributions carried out in this thesis are:

- **Environment setup and system migration:** I performed the complete installation and configuration of `ROS 2 Jazzy Jalisco` on a `Raspberry Pi 4` (Ubuntu 24.04 Server).
- **Incremental vision pipeline development:** I developed two versions of the vision system: an initial prototype based on 2D geometric heuristics and a final robust version based on the `EurekaNet` deep learning architecture.
- **Model training and optimization:** I managed the training process of the neural network using the `IPN Hand` dataset.
- **Control logic implementation:** I programmed the control node responsible for translating semantic gestures into `TwistStamped` motion commands.
- **Safety and reliability systems:** I designed and implemented several software safety layers, including LiDAR-based obstacle detection, an emergency-stop gesture, confidence thresholds for inference, and a temporal voting system to eliminate sensor noise.
- **Experimental validation:** I conducted a rigorous evaluation phase consisting of 600 quantitative tests across 10 gesture classes and qualitative testing with different volunteers.
- **DevOps and Reproducibility:** I implemented containerization practices using `Docker` and `Rocker` to ensure the vision pipeline is portable and can be executed across different development environments without dependency conflicts.

1.4 Scope and limitations

The scope of this project focuses on the software implementation of teleoperating a TurtleBot3 Waffle Pi using hand gestures. The system is designed to work with a Raspberry Pi Camera V2 as the input sensor, avoiding the use of physical controllers, wearable devices, or additional external sensors.

Regarding the software architecture, the development is based on `ROS 2 Jazzy Jalisco`. The solution adopts a distributed approach, where the computational load for computer vision is executed on a remote PC, which connects to the robot via `SSH`.SSH means Secure Shell, a network protocol that provides a secure way to access a remote device, like the TurtleBot3 in this case, over a local wireless network.

The development process followed an incremental approach. Initially, a heuristic-based model was implemented, assigning geometric rules to hand landmarks. Since the obtained performance was not satisfactory and the gesture set was very limited, the system strategy was completely revised. As a result, a Deep Learning–based model (EUREKA architecture) was implemented for the gesture recognition component.

The initial objective of the project also included the use of the OpenManipulator-X robotic arm mounted on the TurtleBot3. However, due to the relatively recent release of `ROS 2 Jazzy`, compatibility issues were encountered between this ROS version and the OpenCR control board firmware. At the time of development, these issues prevented a stable and reliable integration of the robotic arm. Consequently, the final system was implemented without the manipulator, focusing exclusively on mobile base control.

Despite achieving the intended functionality, the system presents some limitations:

- **Network Dependency:** Since the vision processing runs on an external workstation and commands are transmitted through `SSH`/`ROS 2` topics, the system requires a stable WiFi connection to ensure low latency and reliable operation.
- **Lighting Conditions:** As the system relies on RGB data captured by the Raspberry Pi Camera, it performs best in indoor environments with controlled and consistent lighting. Poor illumination or strong direct sunlight can negatively affect landmark detection accuracy.

1.5 Methodology

In order to ensure the reproducibility of the software, the development has followed some modern DevOps practices, such as:

- **Containerization (Docker & Rocker):** To avoid dependency conflicts, OS conflicts and ensure that the code is portable between the development machine and the Remote PC, the whole `ROS 2` environment and the Deep Learning libraries were containerized. For this, the tool `rocker` was used.
- **Remote development:** The development workflow was based on connecting to the robot's Raspberry Pi and the Remote PC via `SSH`. This simulates a realistic teleoperation scenario where the operator is not physically attached to the robot and the robot moves autonomously.

Chapter 2

State of the art

The related work section reviews the state of the art regarding gesture detection and, in addition, describes the works and datasets that have directly influenced the design of the system.

2.1 Evolution of gesture control in robotics

2.1.1 Traditional control interfaces

Throughout history, the control of mobile robots has been based on physical devices (such as controllers, keyboards or mouse) that have gradually evolved to the point where many aspects of control can be performed without the need for a physical device.

Physical controllers tend to be precise, cheap, and reliable and, for this reason, have been widely used in research environments for many years. The main problem, however, is that they force the user to depend on a physical object, which reduces the naturalness of the interaction and also requires prior knowledge of how to operate the device, representing an added difficulty for non-expert users.

In mobile robotics, and especially in human–robot interaction scenarios, this limitation becomes even more evident, since the need to hold a remote control or interact with a keyboard introduces an artificial layer between the user and the robot, making the interaction less natural and more distant.

2.1.2 Gesture control based on wearable devices

The first attempts to improve this interaction experience involved the use of wearable devices, such as gloves with integrated sensors or smartwatches capable of tracking hand movement. These solutions allowed hand positions and movements to be obtained with fairly reliable accuracy, since they were based on direct measurements from inertial or flexible sensors.

Despite their technical effectiveness, wearable-based approaches presented several practical limitations. From a usability point of view, they require the user to carry additional hardware, which is inconvenient and, in many cases, expensive, both in terms of maintenance and overall cost.

2.1.3 Depth cameras and vision-based solutions

With the improvement of computer vision techniques, depth cameras began to be used in gesture recognition tasks. These sensors greatly facilitated the task of isolating the hand from the background, as depth information allows for more robust segmentation regardless of color or texture. This represented a major step forward in vision-based gesture control.

However, depth cameras also introduced new limitations. Their higher power consumption and larger size made them difficult to integrate into small mobile robots like the TurtleBot3 without compromising autonomy or mechanical simplicity. Furthermore, these sensors tend to be more expensive than conventional RGB cameras, and their use is often limited to controlled environments. Again, for everyday use and low-cost robotic platforms, these solutions proved impractical.

2.1.4 RGB cameras and deep learning-based approaches

For all these reasons, the current state of the art involves the use of conventional RGB cameras combined with deep learning models. Instead of relying on specialized hardware or explicit depth information, these approaches allow visual patterns to be learned directly from data. By training neural networks on large gesture datasets, as done in this project, it is possible to achieve robust recognition performance using only standard video.

In addition, this approach greatly reduces the economic and computational cost of the system hardware, making gesture control more accessible and easier to deploy, and allowing further research and faster technological progress. In this context, any standard RGB camera, such as the `Raspberry Pi Camera V2`, can serve as a control sensor.

This data-driven approach fully aligns with the goals of this project, as it prioritizes accessibility, scalability, and usability in real-world environments, while maintaining reliable gesture recognition performance.

2.2 The EUREKA architecture

The main reference for this project, on which the vision node has been based, comes from the research on the `EUREKA` architecture, proposed by Peral, Garrell, and Sanfeliu [23] in the framework of human-robot interaction. The objective of this work was to design a gesture recognition method that was extremely lightweight, allowing service robots to react to human commands in real time without the need for large graphics processing units.

It is important to understand the approach and operation of the model, especially the data processing part. Instead of having a neural network analyze each pixel of the video image, the system first extracts the 21 landmarks of the hand using `MediaPipe`.

Once these *landmarks* are obtained, the coordinates are fed into a network called `EurekaNet`. By working only with geometric coordinates, the system becomes more robust to changes in lighting or background noise, since the model focuses exclusively on the structure and movement of the user's hand, ignoring the rest of the scene.

In addition, the system incorporates a sliding window to observe the trajectory of the points over time. This is key to differentiating gestures that would appear the same in a photo but have opposite meanings depending on the movement, such as the difference between a static stop gesture and a lateral displacement movement.

In this project, the operation of `EUREKA` has been adopted and taken a little further by integrating it into a distributed environment under `ROS 2 Jazzy Jalisco`. This adaptation allows control not only of the mobile base of the TurtleBot3, but also coordination of the movements of the gripper and the OpenManipulator-X arm simultaneously.

2.3 Datasets for gesture recognition

In order to train robust deep learning models, high-quality and diverse data is required. In the field of Human-Robot Interaction, datasets must provide a wide range of lighting conditions and subjects to ensure generalization.

2.3.1 The IPN Hand dataset

In this project, the `IPN Hand` dataset [2] was selected as the primary source for training. This dataset was chosen because it was specifically designed for recognizing gestures in real-life environments.

The size of this dataset is one of the determining factors in its choice, as it has a total of 4,218 gesture samples and more than 800,000 *RGB* frames recorded by 50 different subjects. Having this variety of participants is key in a project of this type, as it ensures that the model can generalize correctly to different hand morphologies and ways of executing the same movement.

A total of 13 gesture classes have been defined, combining static positions and dynamic movements such as sliding, pressing, or rotating. One of the most relevant points that the original *paper* highlights is the inclusion of natural movements that have no control meaning. This is vital for the safety of the robot, since it forces the neural network to learn to differentiate between a real command and the visual noise of a hand moving randomly in front of the camera (see Figure 2.1).

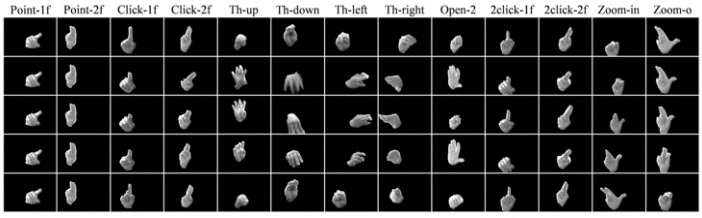


Figure 2.1: Samples of gestures from the `IPN Hand` dataset showing lighting variations and background complexity. Source: [2].

2.4 Technologies for motion, posture, gesture and hand detection

The central and most important component of the project is hand detection. The entire project is based on the extraction of landmarks from live video. Although in this case the system used for this task has been `MediaPipe`, it is crucial to take into account and mention other alternatives that are also considered part of the state of the art.

2.4.1 OpenPose: Multi-person and bottom-up detection

OpenPose is known as one of the first open-source systems capable of performing simultaneous detection of keypoints of the body, hands, and face in a single image [4], as shown in Figure 2.2. Unlike other methods, it uses a bottom-up approach, which means that it first detects all the body parts in the image and then associates them with specific individuals.

Figure 2.2: Estimation of multi-person pose using OpenPose, showing the superposition of the skeletons on the RGB image. Source: [30].

Although it offers extraordinary accuracy in scenes with a lot of people, it presents two major problems specifically related to this project, which are presented below.

First, the computational cost. This model requires intensive use of large libraries and powerful GPUs. In a mobile robotics scenario, with a project that aims to be adaptable to simple mobile robots and PCs without extensive hardware resources, this is a drawback that makes the use of this model for this specific project almost impossible.

Second, latency. Even with very powerful GPUs, latency is much higher than that offered by more optimized models. This is a key point, since introducing delay in the robot's teleoperation is dangerous, as it causes the robot to not respond immediately.

2.4.2 Ultraleap hand tracking sensor

The Leap Motion sensor, from Ultraleap, is a solution based on dedicated hardware that uses infrared light and cameras to track hands with millimeter precision (see Figure 2.3).

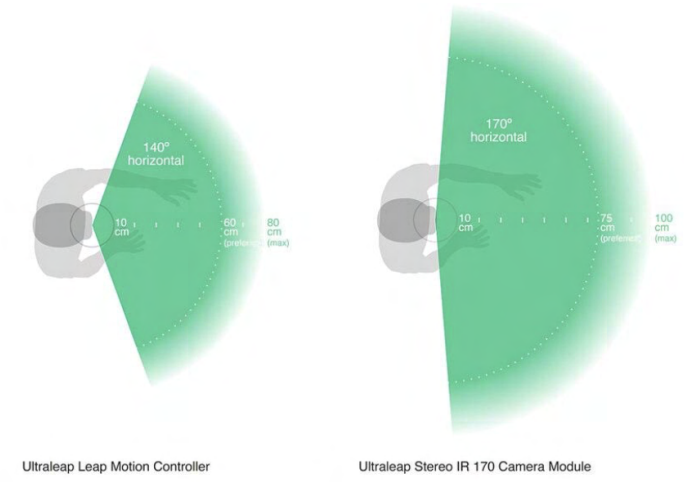


Figure 2.3: Representation of the field of view of the Leap Motion sensor and infrared skeleton tracking. Source: [29].

It is one of the leading technologies for virtual object manipulation and robotic surgery today; however, its architecture presents limitations that make its use in this specific project impractical.

First of all, the reduced field of view. The sensor is designed to be placed on a table or integrated into VR headsets, and the user must keep their hands within a fairly limited distance, approximately 50 cm.

Secondly, it requires the installation of additional hardware on the robot, which breaks one of the basic premises of the project: that it should be accessible and capable of working with any standard camera, thus favoring system adaptability.

Thirdly, the cost, which is above 200€. This makes acquiring such a sensor relatively expensive and favors the choice of alternative methods.

2.4.3 Other vision and deep learning methods

Beyond landmarks-based systems, there are other approaches in the state of the art that address gesture recognition from different architectural perspectives.

Convolutional neural networks (CNNs) have been the mainstay of computer vision for many years. In the context of dynamic gesture recognition, 3D CNNs are commonly used, as they add a temporal dimension to convolutional filters in order to capture movement directly from video [20]. However, despite their power, these networks tend to be very heavy and require a high computational load, which makes their implementation in this work difficult given the limited resources and the use of a

mobile robot.

On the other hand, Vision Transformers (ViTs) represent the most recent paradigm shift, transferring the success of language models to computer vision [6]. Unlike CNNs, which process images locally, Transformers can capture global relationships and long-term temporal dependencies, making them well suited for complex gestures that evolve over several seconds. Although they offer state-of-the-art accuracy, their inference phase is extremely expensive, which once again makes their implementation in this specific case impractical. However, in this project, a similar philosophy has been adopted in terms of temporal analysis (using the EUREKA network), but applied to a reduced data vector, thus avoiding the computational bottleneck of video-based Transformers.

Finally, the YOLO (You Only Look Once) family of models stands out for its ability to perform object detection in a single step, as its name suggests. In addition, variants such as YOLO Pose have recently appeared, extending object detection to real-time body pose estimation [13]. Although YOLO is extremely fast and robust in dynamic environments, its implementation is heavily optimized for graphics cards and tends to focus on the entire body (typically 17 body keypoints). As detailed hand gestures have ultimately been prioritized over general body postures, MediaPipe remains the preferred option. An example of this type of body keypoint detection can be seen in Figure 2.4.

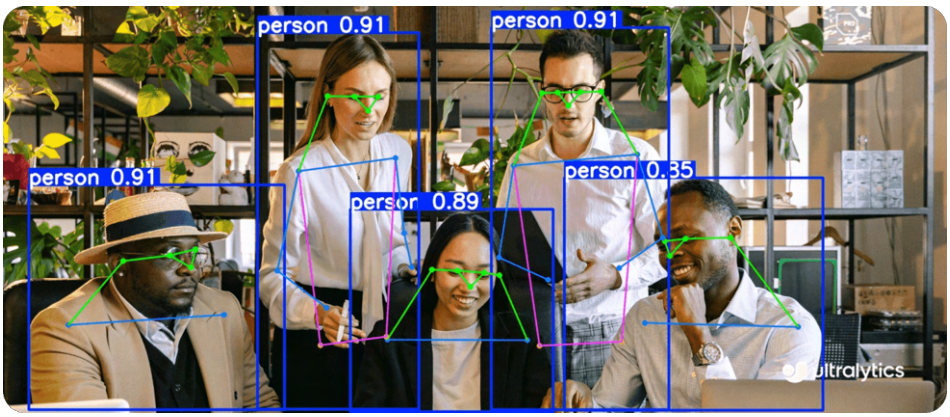


Figure 2.4: Example of body keypoint estimation using YOLO11. Source: [31].

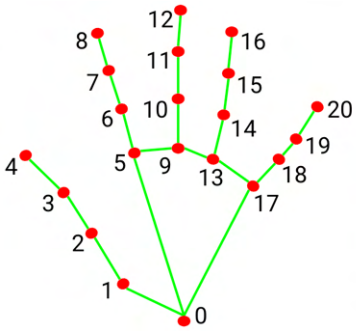
2.5 Justification of MediaPipe Hands

`MediaPipe Hands` [18] has been chosen for gesture extraction. The main advantage of this Google tool is that it offers a pre-trained model capable of identifying 21 3D keypoints from a single RGB image, with minimal latency and without the need for specialized hardware. It is also considerably more robust than more conventional methods, such as color segmentation, which are unstable under changing lighting conditions.

The operation of the system is based on a machine learning model that predicts, in real time, the 3D coordinates of a total of 21 keypoints (landmarks) of the hand. This process is carried out in two separate stages: first, a palm detector called BlazePalm locates the hand, and then a landmark model

predicts the position of the keypoints. This architecture allows the system to be truly portable and to run even on the CPU of a conventional laptop PC or on mobile robotics devices, ensuring that the TurtleBot3 receives gesture commands almost instantaneously.

As shown in Figure 2.5, these points allow the structure of the hand to be modeled geometrically. The main advantage of this approach is that it achieves a drastic reduction in dimensionality: instead of processing thousands of pixels from the raw image, the system operates on a compact vector of landmarks.



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

Figure 2.5: Representation of the hand skeleton with the 21 keypoints detected by `MediaPipe`. Source: [18].

It should be noted that, although using only this model was the initial approach of the project (based on heuristics), these landmarks are currently used as input data for the Deep Learning architecture `EUREKA`, which will be described in detail later.

Chapter 3

System overview

This chapter intends to present a global, schematic and summarized view of the developed system, describing the requirements, structure and architecture of the entire code. The objective is to summarize the operation of the system and how the gesture recognition system is integrated with the control of the mobile robot within a ROS 2 based environment.

3.1 Functional requirements

This section describes the functions and capabilities that the system must achieve for correct operation. They have been divided into two groups: those that affect users and those that directly affect the robot.

3.1.1 User requirements

From the user's point of view, the system must allow a person to control a mobile robot using hand gestures, without the need for additional physical devices. Specifically, the user must be able to:

- execute basic movement commands (forward, backward, turn),
- activate more complex behaviors, such as autonomous navigation based on wall tracking,
- repeat the last executed action using a specific gesture,
- interact with the system in an intuitive manner and with a fast response.

3.1.2 Robot requirements

From the robot's point of view, the system must:

- correctly interpret gesture commands issued by the vision node,
- execute smooth and controlled movements,
- ensure safe operation using proximity sensors (LiDAR),
- cancel or interrupt movements in case of an imminent collision risk.

3.2 Non-functional requirements

Apart from all the previous requirements, the system must meet a number of criteria that guarantee its functionality in a real environment in a safe manner. These are defined below.

3.2.1 Performance and latency

The system must operate in real time. High latency cannot be allowed, since delayed execution of gestures would introduce serious safety issues and increase the risk of robot collisions. Therefore, the total latency has been kept within a range that allows safe teleoperation, approximately between 200 and 300 ms.

3.2.2 Robustness and safety

It is important to note that the system is designed with a conservative behavior that ensures that it:

- avoids false positives in gesture detection,
- prefers false negatives over incorrect actions,
- ensures that no gesture can cause indefinite or uncontrolled movements,
- maintains blocking mechanisms and time limits for all actions.

3.3 High-level architecture

This section describes the high-level architecture that has been designed for the system, which is based on a distributed model that guarantees the immediate functionality of the system. The main idea, as shown below, has been to separate the operation into nodes. Thus, the most computationally intensive tasks have been executed on a remote PC, and the robot has been limited to receiving movement commands and managing the LiDAR.

3.3.1 Main components

The system is composed of the following four main components:

- **Vision node**: processes images from the camera, detects gestures, and publishes the corresponding command.
- **Control node**: subscribes to the command published by the vision node and applies the corresponding movement to the robot.
- **TurtleBot3 Waffle Pi**: physically executes the received movement commands.
- **Remote PC**: executes the vision and control nodes locally and communicates with the robot via SSH.

3.3.2 Data and control flow

The data flow follows a clearly compartmentalized architecture, as shown in Figure 3.1:

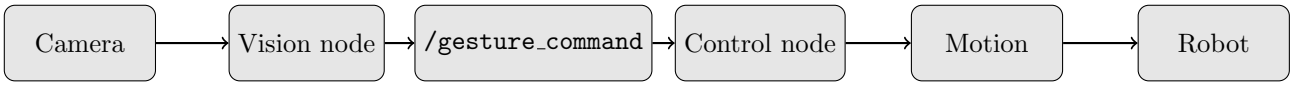


Figure 3.1: Data and control flow from gesture perception to robot motion execution.

3.4 Robotics Middleware: ROS 2 Jazzy Jalisco

This section describes the communication framework that serves as the backbone of the system. To manage the distributed nature of the vision and control tasks, `ROS 2 Jazzy Jalisco` has been selected.

3.4.1 Evolution of robotics middleware: from ROS to ROS 2

The TurtleBot3 Waffle Pi that has been used for this project has always been integrated and widely used within the `ROS 1` ecosystem. However, this project proposes a migration to `ROS 2 Jazzy Jalisco` to leverage its modern capabilities. While the original ROS (Robot Operating System) was based on a centralized master node (`roscore`), ROS 2 uses a communication protocol based on `DDS` (Data Distribution Service) [17].

This point is key in the context of this work because it enables much more efficient communication between the robot and the remote PC that processes the vision data. DDS allows the configuration of Quality of Service (`QoS`) parameters, ensuring that the robot's movement commands arrive with the lowest possible latency, even in suboptimal WiFi networks. In fact, this change is currently considered an essential step in the evolution of robotics research towards more robust commercial, academic, and industrial applications.

In addition to all the above, ROS 2 solves one of the main problems of the previous version: the Single Point of Failure. In ROS 1, if the master node failed, the entire communication network collapsed immediately. In contrast, the DDS protocol implements a dynamic discovery system in which nodes find each other autonomously. This means that if any node (for example, the vision PC) suffers a temporary disconnection, it is isolated, while all other local nodes can continue to function normally without depending on an external master [16].

One of the most determining factors in this transition is the granular management of data through QoS policies. In this TFG, two types of critical data flows have been differentiated:

- **Telemetry and video flow:** First, it is configured as Best Effort to prioritize immediacy; if an image packet is lost, it is preferable to discard it and move on to the next one rather than attempting retransmission and generating latency.
- **Control flow and safety:** Second, it is configured as Reliable, ensuring that critical commands such as the emergency stop reach the TurtleBot3 correctly [17].

Finally, the specific choice of the `Jazzy Jalisco` distribution (released in May 2024) is motivated by the need to use a long-term support (LTS) environment on Ubuntu 24.04. In addition, this version

incorporates significant improvements in the performance of the Python client and notable optimizations in resource consumption on the robot's Raspberry Pi 4 [21].

3.4.2 ROS 2 nodes and topics

This section describes the main nodes of the system and specifies which topics are published and subscribed to by each of them. The objective is to clearly explain how information flows within the system and to justify this design in order to optimize the data exchange, avoiding unnecessary subscriptions or publications that could introduce additional computational overhead.

3.4.2.1 Vision node

The vision node is responsible for processing the gesture observed through the camera and translating it into a numerical command (an integer).

Its main functions are:

- subscribe to the camera image stream,
- extract hand landmarks using `MediaPipe`,
- build a temporal representation of the gesture,
- infer the associated command using the `EurekaNet` model,
- publish the detected gesture command (an integer depending on the gesture).

The node interacts with the ROS 2 system through a small and well-defined set of topics, clearly differentiated according to whether they are subscribed to or published. This relationship is summarized schematically in Table 3.4.2.1.

Subscribe	Publish
`/image_raw/compressed`	`/gesture_command` (`std_msgs/Int8`)

Table 3.1: ROS 2 vision node topics

3.4.2.2 Control node

The control node is responsible for translating the command published by the vision node into physical robot motion commands, while ensuring both safety and temporal coherence of the actions.
Its main responsibilities are:

- receive and interpret the gesture command,
- map it to a specific robot action,
- manage timed actions and blocking mechanisms,
- implement autonomous behaviors based on internal states,
- monitor the environment using LiDAR to avoid collisions.

As in the case of the vision node, the number of topics is deliberately kept to a minimum in order to optimize the overall system. The complete topic interface of the control node is summarized in Table 3.4.2.2.

Subscribe	Publish
`/gesture_command` (`std_msgs/Int8`) `/scan`	`/cmd_vel` (`geometry_msgs/TwistStamped`)

Table 3.2: ROS 2 control node topics

3.4.2.3 Overview of ROS 2 communication flow

Figure 3.2 provides a global and visual overview of the ROS 2 communication architecture of the system, showing how data and commands flow between the robot, the vision node, and the control node.

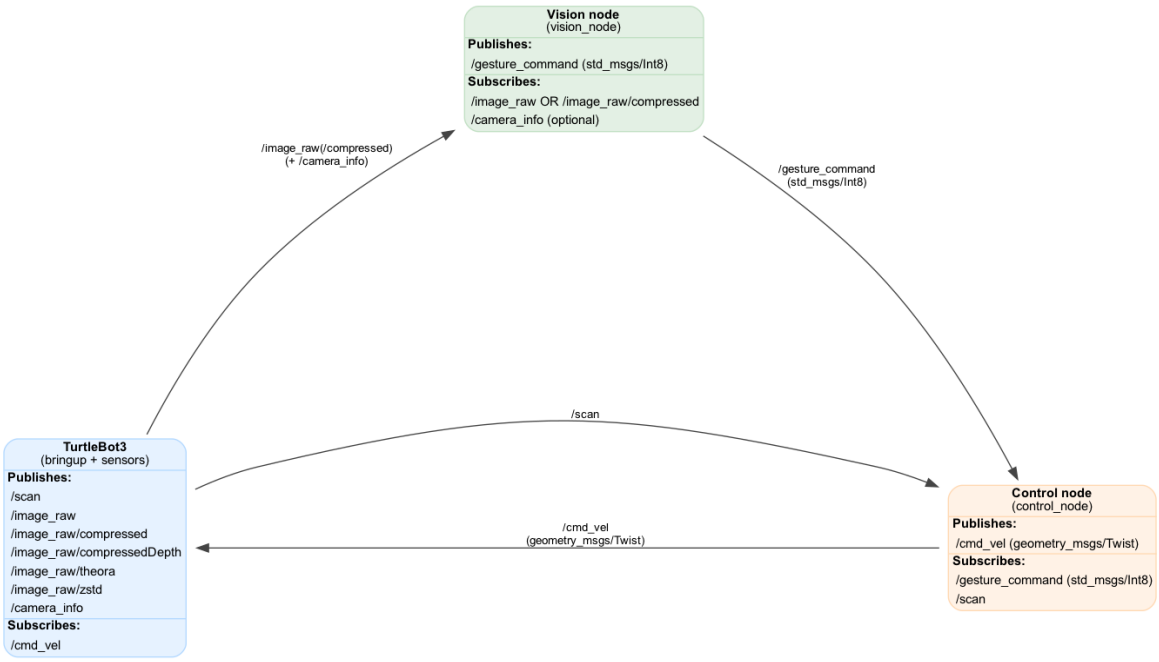


Figure 3.2: Global overview of the ROS 2 nodes and topics.

3.4.2.4 External tools and visualisation

In addition to the topics directly used by the developed nodes, several auxiliary tools were employed during system deployment and testing to ensure correct operation. ROS 2 terminals enabled continuous monitoring of node execution and real-time error detection and analysis. Complementarily, inspecting selected topics made it possible to verify connectivity between the robot and the PC whenever required. Likewise, visualization of the camera stream and inspection of terminal outputs helped ensure that the system behaved correctly at all times. Finally, video recordings were essential for

documenting the experiments and validating the obtained results.

Furthermore, it is important to note that the system also relies on internal ROS 2 topics for management purposes:

- `/rosout`,
- `/parameter_events`.

These topics are not directly part of the functional logic of the system, but they are essential for supervision, debugging, and maintenance.

3.5 Project structure

This section presents the structure of the project and the organization of its main components. The repository is structured into several directories, each corresponding to a specific stage or functionality of the system.

The computer_vision directory contains the ROS 2 workspace, including the build, install, and log folders, as well as the src directory with the developed packages. Within src, the gesture_recognition_pkg implements the computer vision pipeline for gesture recognition, while the turtlebot_control_pkg is responsible for robot motion control and LiDAR management.

The heuristic_version directory includes an alternative implementation based on heuristic methods. The train_model directory contains the resources related to model training, including the training notebook and the trained neural network weights. Experimental scripts and recorded videos are organized under the experiments directory, which is further divided into experiments conducted with and without the robot.

Finally, the repository also includes a README file with general project information and a PDF report describing the work in detail. All project code, as well as associated documentation, is available in a public GitHub repository:

`https://github.com/JaumeMiL/computer_vision_robot_skills_tfg`

```
computer_vision_robot_skills_tfg/

 computer_vision/
    build/
    install/
    log/
    src/
        gesture_recognition_pkg/
           gesture_recognition_pkg/
              vision_node.py
           package.xml
           setup.py

        turtlebot_control_pkg/
```

```
            turtlebot_control_pkg/
               control_node.py
            package.xml
            setup.py

heuristic_version/

train_model/
   Train.ipynb
   eureka_model_100_epochs.pth

experiments/
   Experiments.py
   videos/
       with_robot/
       without_robot/

README.md
Report.pdf
```

Chapter 4

Vision Node: evolution from heuristics to deep learning

This chapter describes the core of the project, the vision node. It explains the evolution of the vision node from an initial and very basic version based on geometric relationships between landmarks extracted by MediaPipe. This first version relied mainly on static heuristics, with only one dynamic gesture that proved to be unstable in practice. In addition, it was limited in terms of the number of gestures and highly dependent on the exact position and orientation of the hand. For these reasons, it was decided to implement a more robust deep learning–based solution, in which all gestures are treated as dynamic patterns, properly supporting variability between users, as well as hand inclinations and translations within the image plane.

4.1 Design goals and real-time constraints

To understand the overall operation of the vision node, it is important to clarify that it acts as the perception subsystem of the architecture. Its responsibility is to transform hand gestures captured by an RGB camera into a discrete numerical command, which is then published on the `/gesture_command` topic.

In both versions of the vision node, the design has aimed to satisfy three key goals that are central to this project:

1. usability for non-expert users,
2. low latency to ensure safe teleoperation,
3. conservative behavior to avoid unintended activations.

4.1.1 Usability for non-expert users

The interaction with the user should be intuitive, highly explanatory, and require very little preparation. The user must be able to learn the gestures quickly; therefore, they should be simple and reasonably common gestures in everyday life.

In addition, the user must be able to see at all times what the system is recognizing, as well as the image of everything that the system is perceiving.

It is also essential that the system is able to tolerate differences in hand shape and size, variations in the execution style of each gesture (which can differ significantly between users), and differences in gesture speed.

Finally, it is necessary to avoid continuous and sudden changes of state, since small posture variations should not affect or destabilize the system.

4.1.2 Latency requirements

In teleoperation, low latency is an essential requirement to guarantee safe robot control. For this reason, the complete processing pipeline has been designed to maintain a latency that allows safe operation at all times.

In the final integrated experiments, the typical end-to-end latency is usually kept within 200–300 ms under normal operating conditions, which provides sufficiently fast response to ensure reliable control of the robot.

4.1.3 Conservative behavior of the system

One of the fundamental design criteria of the vision node is the adoption of a clearly conservative behavior. In the context of mobile robot teleoperation, a misclassification can have physical consequences, either in the form of unexpected movements or collision situations. For this reason, the system is designed to prioritize safety over immediate reactivity.

In this sense, it is explicitly preferred to generate false negatives rather than false positives. This means that, when facing ambiguity or low confidence in gesture detection, the system chooses not to execute any action instead of risking the activation of an incorrect command. From the user's point of view, this may occasionally require repeating a gesture, but it effectively prevents unwanted activations that could lead to unexpected robot movements.

This approach is reinforced through several mechanisms, such as high confidence thresholds, temporal voting across consecutive predictions, and blocking mechanisms once an action has been validated. All together, these decisions ensure a more stable operation of the system.

4.2 Image acquisition pipeline

This section describes the entire image acquisition process, starting with the capture and reception of images from the robot's camera until they reach the remote PC.

4.2.1 Compressed image topics in ROS 2

As explained previously in the System Overview chapter, in order to improve WiFi latency and facilitate efficient image preprocessing, the system mainly relies on the compressed image stream. More specifically, the vision node subscribes to the topic `/image_raw/compressed` (`sensor_msgs/CompressedImage`).

Although there are several other topics that publish image data using different formats and quality levels, this option was selected because it provides the best compromise between visual quality, CPU usage, and end-to-end latency.

4.2.2 Preprocessing with `OpenCV` and `CVBridge`

All messages received in the `CompressedImage` format are decoded into OpenCV frames using `CVBridge`. The images are then converted to RGB and processed by `MediaPipe Hands`, which extracts the 2D hand landmarks. For visualization and debugging purposes, these landmarks are superimposed on the original image in real time.

Internally, however, the gesture classification pipeline relies exclusively on the numerical coordinates of the detected landmarks. This design choice keeps the processing lightweight and allows the vision node to run entirely on CPU, while still maintaining real-time performance. Figure 4.1 shows the visual result of this process.

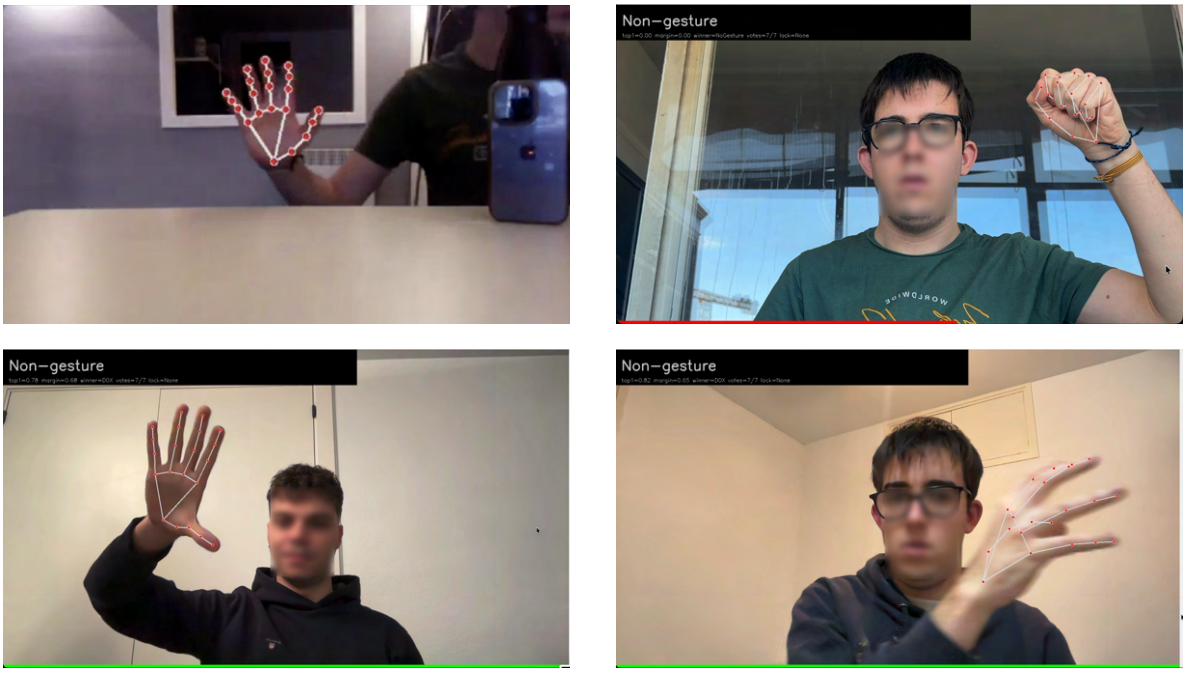


Figure 4.1: Collage of frames showing the real-time hand landmark extraction process.

4.3 Phase I: Geometrical approximation (baseline)

This initial phase establishes a very basic and deterministic approach to gesture recognition by applying handcrafted rules directly to the 21 MediaPipe landmarks. This heuristic approximation allowed a first contact with the problem, an initial quick testing phase, and a validation of the ROS 2 data flow and topics in real time, with a very low computational cost. However, its high sensitivity to hand rotations and the limited number of supported gestures motivated the transition to a deep learning–based solution, which is described in Phase II.

4.3.1 Normalization and scale reference

First of all, in order to reduce the problems caused by different hand sizes and user–camera distances, the wrist (landmark 0) is defined as the local coordinate origin, and a reference metric d_{ref} is computed based on the hand geometry.

$$d_{\text{ref}} = \sqrt{(L_{5,x} - L_{0,x})^2 + (L_{5,y} - L_{0,y})^2} \tag{4.1}$$

where:

- L_0 corresponds to the wrist landmark of the hand,
- L_5 corresponds to the landmark of the base of the index finger,
- $L_{i,x}$ and $L_{i,y}$ represent the horizontal and vertical coordinates of landmark i in the 2D image plane,
- d_{ref} defines a reference metric proportional to the size of the user's hand.

4.3.2 Heuristics for static classification

As explained above, in this case the classification is based on geometric relationships between the hand landmarks. By comparing the relative positions of these landmarks, the system determines the configuration of the fingers and the overall posture of the hand. The logic behind the four implemented static gestures is presented in detail below.

4.3.2.1 Closed fist

The *closed fist* gesture is characterized by the fact that all the fingertips are close to the wrist. For this reason, and in order to detect this gesture, a threshold based on a reference distance d_{ref} is defined, which adapts to the scale of the hand:

$$\theta_{\text{fist}} = 1.2 \cdot d_{\text{ref}} \tag{4.2}$$

where d_{ref} is the previously defined distance corresponding to the open hand, used to normalize the measured distances.

The *fist* gesture is detected if all the following conditions are satisfied:

$$\begin{aligned} d(L_0, L_8) &< \theta_{\text{fist}} \\ d(L_0, L_{12}) &< \theta_{\text{fist}} \\ d(L_0, L_{16}) &< \theta_{\text{fist}} \\ d(L_0, L_{20}) &< \theta_{\text{fist}} \end{aligned} \tag{4.3}$$

Let L_0 denote the point corresponding to the wrist, and L_8, L_{12}, L_{16}, and L_{20} the tips of the index, middle, ring, and little fingers, respectively. This configuration is illustrated in Figure 4.2.

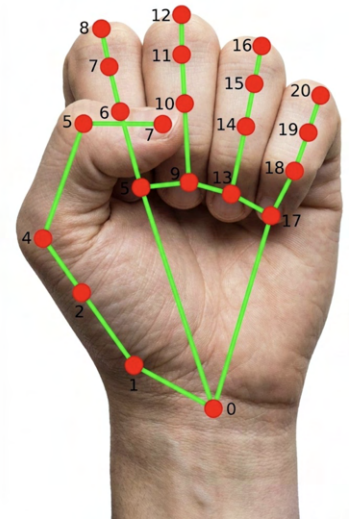


Figure 4.2: Example of the *closed fist* gesture (Self-generated illustration).

4.3.2.2 Open hand

The *open hand* gesture is characterized by an extended configuration of the fingers, where all the fingertips are located at a relatively large distance from the wrist. Again, the reference distance has been used, in order to manage variations in hand scale or camera distance:

$$\theta_{\text{open}} = 1.8 \cdot d_{\text{ref}} \tag{4.4}$$

The *open hand* gesture is detected if all the following conditions are satisfied:

$$\begin{aligned} d(L_0, L_8) &> \theta_{\text{open}} \\ d(L_0, L_{12}) &> \theta_{\text{open}} \\ d(L_0, L_{16}) &> \theta_{\text{open}} \\ d(L_0, L_{20}) &> \theta_{\text{open}} \end{aligned} \tag{4.5}$$

Let L_0 denote the point corresponding to the wrist, and L_8, L_{12}, L_{16}, and L_{20} the tips of the index, middle, ring, and little fingers, respectively (see Figure 4.3).

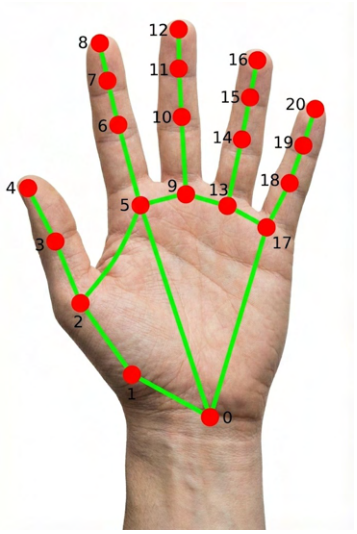


Figure 4.3: Example of the *open hand* gesture (Self-generated illustration).

4.3.2.3 Thumb up

The *thumb up* gesture is characterized by an extended thumb while the other fingers remain close to the wrist. In contrast to distance-based gestures, this configuration is detected by analyzing the relative vertical positions of the finger landmarks in image coordinates.

The fingers are considered flexed if their tips are located below their intermediate joints in the image:

$$\begin{aligned} L_8 &> L_6 \\ L_{12} &> L_{10} \\ L_{16} &> L_{14} \\ L_{20} &> L_{18} \end{aligned} \tag{4.6}$$

L_4 denotes the tip of the thumb and L_5 its proximal joint. L_8, L_{12}, L_{16}, and L_{20} represent the tips of the index, middle, ring, and little fingers, while L_6, L_{10}, L_{14}, and L_{18} correspond to their respective intermediate joints. The fingers are considered closed if their tips are located below their intermediate joints in the image.

The thumb is considered extended if its tip is located above its proximal joint:

$$L_4 < L_5 \tag{4.7}$$

If all these conditions are satisfied, the gesture is classified as a *thumb up*, as shown in Figure 4.4.

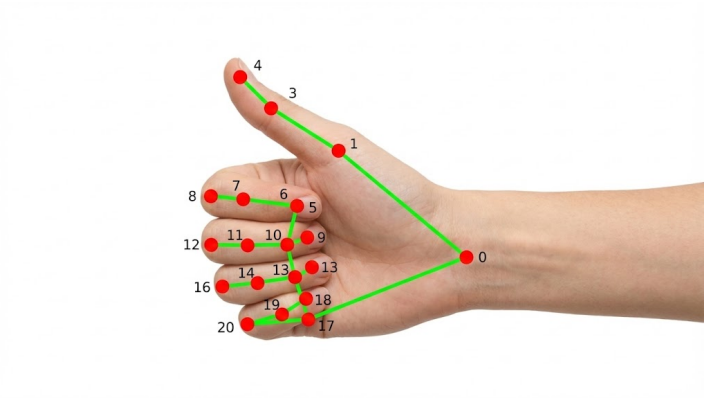


Figure 4.4: Example of the *thumb up* gesture (Self-generated illustration).

4.3.2.4 Two fingers

The *two fingers* gesture is characterized by the extension of the index and middle fingers, all the other ones remain closed. As in the previous gesture, this configuration is detected by analyzing the positions of the finger landmarks rather than distance.

$$\begin{aligned} L_8 &< L_6 \\ L_{12} &< L_{10} \\ L_{16} &> L_{14} \\ L_{20} &> L_{18} \end{aligned} \tag{4.8}$$

L_8 and L_{12} denote the tips of the index and middle fingers, with L_6 and L_{10} their respective intermediate joints. L_{16} and L_{20} represent the tips of the ring and little fingers, and L_{14} and L_{18} their corresponding intermediate joints. Figure 4.5 shows an example of this posture.

The index and middle fingers are considered extended if their tips are located above their intermediate joints, while the ring and little fingers are considered closed if their tips are located below their respective joints.

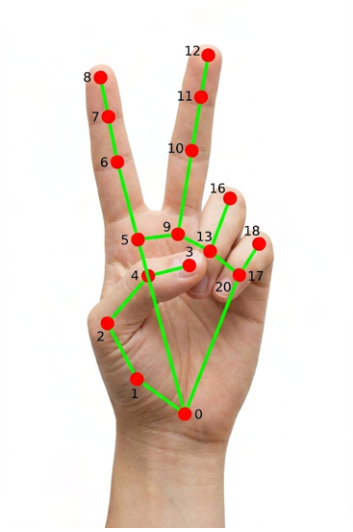


Figure 4.5: Example of the *two fingers* gesture (Self-generated illustration).

4.3.3 Dynamic gesture: circular trajectory detection

To make the system slightly more complete, a dynamic gesture was implemented in a heuristic way to allow the repetition of the last gesture, as will be described later in the deep learning solution. For this detection, the procedure is relatively simple. The position of the tip of the index finger is analyzed over a short and finite period of time.

To find the center of the gesture, the trajectory is averaged as follows:

$$c_x = \frac{1}{N}\sum_{i=1}^{N} x_i, \qquad c_y = \frac{1}{N}\sum_{i=1}^{N} y_i \tag{4.9}$$

Once the center is obtained, the distance of each point with respect to it is computed, and the standard deviation of these distances is used to check whether the trajectory is approximately circular. An approximation is required, since it is very difficult for a human to perform a perfectly circular gesture.

$$r_i = \sqrt{(x_i - c_x)^2 + (y_i - c_y)^2} \tag{4.10}$$

$$\sigma_r < 0.4 \cdot \mu_r \tag{4.11}$$

where σ_r is the standard deviation of the radii and μ_r is their mean value.

Once this condition is verified, the rotation of the movement is analyzed by computing the accumulated angular variation, obtained by summing the angular differences between consecutive points.

$$\theta_i = \arctan 2(y_i - c_y,\ x_i - c_x) \tag{4.12}$$

$$\Theta = \sum_{i=2}^{N} |\theta_i - \theta_{i-1}| \tag{4.13}$$

Finally, if the accumulated rotation is greater than one full turn, the gesture is classified as a circular gesture:

$$\Theta > 2\pi \tag{4.14}$$

4.3.4 Limitations of the geometric approximation

The heuristic approach is efficient, but it relies on geometric relationships that are too simple. When analyzing the previous rules, it can be observed that they are based on 2D coordinates, which are no longer reliable when the hand tilts or changes its orientation.

The dynamic gesture detection also presents several limitations. Although a margin is allowed when checking the circular trajectory, it is still very difficult for a user to perform a perfect or near-perfect gesture, which makes detection unreliable. In addition, the system had serious problems to determine when the gesture started and ended.

For these reasons, and with the goal of going further and supporting a larger set of dynamic gestures, it was decided to start from scratch and implement a solution based on deep learning. This new approach uses a model capable of better generalizing across different users, hand orientations, and dynamic gestures.

4.4 Phase II: Approach using deep learning (EurekaNet)

As explained in the previous section, the limitations of the heuristic approach due to the low variability of gestures, the limited ability to generalize, and the direct dependence on hand position—created the need for a new detection model, this time based on deep learning.

In this approach, the system no longer compares landmark positions using geometric rules on a plane, but instead uses `MediaPipe Hands` to extract landmarks from videos of the `IPN Hands` dataset. From these landmarks, the model learns complex patterns over complete temporal windows of hand movement.

4.4.1 Justification of the EUREKA approach

The selection of the EUREKA architecture over other deep learning methods is based on three key factors that were central objectives of this project: low computational cost, robustness to noise, and low latency, all of which are guaranteed by this architecture.

First, computational efficiency is essential in a teleoperation scenario. During the inference phase, highly powerful machines are not always available, and access to GPUs cannot be assumed. Therefore, it is necessary to rely on models that are not overly complex but still provide reliable performance. In this context, EurekaNet, being a Multilayer Perceptron (MLP) that operates on landmark coordinates rather than raw image pixels, enables extremely fast inference (below 10 ms) on a standard CPU of a conventional computer.

Second, this approach is highly robust to environmental changes. By learning from MediaPipe landmarks instead of raw images, variations in background, lighting, or scene composition have little impact on the system, as long as the hand is correctly detected. It is important to note that the network receives only the geometric information of the hand and not the full RGB data captured by the camera.

Finally, the use of the DistTime vector allows the system to effectively capture the temporal structure of gestures, enabling the correct classification of dynamic gestures by understanding their beginning and end in each execution.

4.4.2 `EurekaNet` architecture

The architecture is based on the `EurekaNet` model, a Multi-Layer Perceptron neural network designed to be simple and light, allowing it to run without a GPU during the inference phase while ensuring good real-time performance for robot teleoperation.

More specifically, the model uses three fully connected layers with regularization mechanisms to avoid overfitting and improve convergence. The final network architecture is shown in Table 4.1:

Layer	Neurons / Type	Activation	Regularization
Input	12,348 (Vector DistTime)	-	-
Hidden 1	128 neurons	ReLU	BatchNorm1d + Dropout (0.25)
Hidden 2	64 neurons	ReLU	BatchNorm1d + Dropout (0.25)
Output	14 classes (IPN Gestures)	Softmax	-

Table 4.1: EurekaNet architecture specifications.

4.4.3 Data representation: the DistTime vector

The performance of this node is largely driven by the use of a feature vector that encodes the coordinate data extracted with MediaPipe as frame-by-frame temporal differences, referred to as the *DistTime* vector. This representation is preferred over the use of absolute coordinates, since absolute positions would be computationally more expensive and would degrade the temporal relationship between frames.

For each gesture, a temporal *buffer* of $K = 15$ consecutive frames is maintained. This buffer is used to compute the vector differences of the landmarks between all consecutive pairs of frames (i.e., t and $t-1$). As a result, the representation is invariant to global translations of the hand:

$$\Delta L_t = L_t^{(i)} - L_{t-1}^{(j)} \quad \forall i, j \in \{0, \ldots, 21\} \tag{4.15}$$

Finally, all these differences are concatenated and flattened into a single input vector, which is fed into the first layer of the network and has a fixed dimension of $D = 12{,}348$.

4.4.4 Vision node data flow and inference

It is important to mention that the vision node is not only responsible for executing the EurekaNet model, but also manages the entire data flow asynchronously. This allows maintaining low latency and guarantees the correct real-time operation of the system.

The operation follows a simple flow. First, each image received by the topic `image_raw/Compressed` is decoded and processed with `MediaPipe Hands` to extract the hand landmarks. As explained in the previous section, these landmarks are stored in a temporary buffer from which the DistTime vector is constructed.

This entire process is executed inside the node's callback function, allowing the node to continue receiving new images in real time, without stopping the system or increasing latency.

Latency is one of the most critical aspects of the system, so the node has been configured with a real-time–oriented QoS profile (reliability *BEST_EFFORT* and *depth*=1), which always prioritizes processing the most recent frame instead of increasing the queue. In this way, delays are avoided and the control remains fully real-time.

For the inference stage, the output of the network is transformed into probabilities and the two best classes are considered in each case. As explained in more detail later, a gesture is only accepted if it exceeds a confidence threshold and presents a minimum confidence margin over the second option.

Finally, when no hand is detected, the system publishes the *NoGesture* state and the buffer is progressively updated to prevent accumulation and excessive latency. As will also be explained later, the final classification decision is based on temporal voting.

If a gesture is detected, the node publishes a discrete command (`std_msgs/Int8`) to the `/gesture_command` topic.

4.4.5 Stability and safety mechanisms

Since the TurtleBot3 is a physical system, an incorrect gesture detection could cause a sudden movement or even a collision of the robot. To avoid this, three safety measures have been applied. These mechanisms were not included in the original system and make it slightly more robust and conservative:

- **Adaptive confidence thresholds:** First, this solution was chosen to allow a more cautious gesture classification. No gesture is accepted if the probability of the winning class does not exceed a strict threshold (0.75 for finger-pointing gestures and 0.55 for the others). In addition, a minimum confidence margin with respect to the second-best option is required.
- **Sliding window:** In this case, the mechanism keeps a record of the last detections. A gesture is only classified if it is the majority (at least 4 out of 7 votes) among the last 7 frames. This avoids noise and frequent gesture changes that should not be classified, as they usually correspond to momentary detections or transitions between gestures.
- **Locking mechanism:** Finally, to avoid requiring the user to continuously perform a gesture while the robot is executing an action, a lock state is implemented. Once a gesture is validated and the action begins, it remains active until the system consistently detects, with 5 consecutive votes, that the hand has returned to a neutral position or that a different gesture is being performed.

Chapter 5

Control Node and Motion Logic

The control node is the unit of the system that directly executes actions on the robot. Its main function is to translate the integer-based commands received from the vision node and convert them into actual robot movement (such as linear and angular velocities). This function is simultaneously coupled with the data received from the LiDAR sensor, which allows for safe and robust navigation.

5.1 Motion logic: mapping gestures to robotic actions

Each identifier received by the vision node is mapped to a different action with defined movement parameters, usually the linear velocity v and the angular velocity ω of the robot.

The control node incorporates a mechanism that prevents the immediate repetition of gestures, so that the user, in order to repeat the last successfully executed gesture, must perform another specific gesture that allows this action. This mechanism also helps to guarantee that each gesture activates an action only once.

The following section describes all the actions triggered by the different gestures recognized by the system and explains in detail the behavior associated with each gesture.

5.1.1 *Throw up* gesture: forward (`ID_FORWARD`, 5)

This gesture is quite simple, as it activates a straight forward movement with a constant linear speed of +0.2 m/s for an interval of 5.0 seconds.

In addition, during the entire execution, the node continuously monitors the front distance using the LiDAR, so that if an obstacle is detected at a distance below the safety threshold, the robot stops immediately.

```
if act == self.ID_FORWARD:
    self._latch_execute(act)
    self._start_timed_action(+self.lin_speed, 0.0,
                             self.t_fwd, "FORWARD")
```

Listing 5.1: Forward motion execution.

5.1.2 *Throw down* gesture: backward (ID_BACKWARD, 3)

The backward gesture is identical to the previous one but in the opposite direction. It activates a straight backward movement with a constant linear speed of -0.2 m/s for an interval of 5.0 seconds.

```
if act == self.ID_BACKWARD:
    self._latch_execute(act)
    self._start_timed_action(-self.lin_speed, 0.0,
                             self.t_bwd, "BACKWARD")
```

Listing 5.2: Backward motion execution.

5.1.3 *Throw left* gesture: turn left (ID_TURN_LEFT, 4)

This gesture activates a rotation on the robot's vertical axis to the left with an angular velocity of -0.8 rad/s for 2.0 seconds, thus achieving a 90-degree rotation without changing its position.

```
if act == self.ID_TURN_LEFT:
    self._latch_execute(act)
    self._start_timed_action(0.0, -self.ang_speed,
                             self.t_turn_90, "TURN_LEFT")
```

Listing 5.3: Left turn execution.

5.1.4 *Throw right* gesture: turn right (ID_TURN_RIGHT, 10)

Once again, this gesture is functionally equivalent to turning left, but in the opposite direction. It activates a rotation on the robot's vertical axis to the right with an angular velocity of 0.8 rad/s for 2.0 seconds, thus achieving a 90-degree rotation without changing its position.

```
if act == self.ID_TURN_RIGHT:
    self._latch_execute(act)
    self._start_timed_action(0.0, +self.ang_speed,
                             self.t_turn_90, "TURN_RIGHT")
```

Listing 5.4: Right turn execution.

5.1.5 *Open twice* gesture: full rotation (ID_ROTATE, 9)

This gesture activates a rotation of the robot on itself until it completes a full turn. This action is designed for tasks related to environment exploration. This is achieved with an angular velocity of 0.8 rad/s for 8.0 seconds.

```
if act == self.ID_ROTATE:
    self._latch_execute(act)
    self._start_timed_action(0.0, +self.ang_speed,
                             self.t_rotate, "ROTATE")
```

Listing 5.5: Full rotation execution.

5.1.6 *Zoom in* gesture: right wall following (ID_FOLLOW_RIGHT, 6)

This gesture activates an autonomous wall-following behavior using LiDAR, taking the wall on the right side of the robot as a lateral reference. If the wall is not detected, the robot rotates to scan the environment until it is found. The motion control is based on a finite state machine that allows the robot to move parallel to the wall. The follow mode includes a 30-second safety *timeout*, after which the robot stops.

```
if act == self.ID_FOLLOW_RIGHT:
    self._latch_execute(act)
    self._start_follow("right")
```

Listing 5.6: Right wall following activation.

5.1.7 *Zoom out* gesture: left wall following (ID_FOLLOW_LEFT, 7)

This gesture implements exactly the same behavior as the right wall tracking, but in a symmetrical way. It also activates an autonomous wall-following behavior using the LiDAR, taking, in this case, the wall located on the left side of the robot as a lateral reference. If the wall is not detected, the robot scans until it is found. The motion control is based on a finite state machine that allows the robot to move parallel to the wall. The tracking mode includes a 30-second safety *timeout*, after which the robot stops.

```
if act == self.ID_FOLLOW_LEFT:
    self._latch_execute(act)
    self._start_follow("left")
```

Listing 5.7: Left wall following activation.

5.1.8 *Pointing with one finger* gesture: repeat last action (ID_REPEAT, 1)

This action is interesting because it allows the user to repeat the last valid action (`last_action_id`) saved by the system. In this way, it is possible to re-execute the immediately previous action without having to repeat the original gesture.

```
def _repeat_last(self):
    if self.last_action_id is None:
        return
    act = self.last_action_id
    self.rearmed = True
    self._handle_action(act, from_repeat=True)
```

Listing 5.8: Repeat last action logic.

5.1.9 *Pointing with two fingers* gesture: emergency stop

Finally, one of the most important gestures is the one that triggers an emergency stop and has absolute priority over any other gesture. When detected, the node immediately cancels all active actions and publishes all speeds as 0.

This allows the user to stop the system quickly in the event of a risky situation.

```
if act in self.STOP_IDS:
    self._cancel_all("EMERGENCY STOP")
    self._arm_again()
    return
```

Listing 5.9: Emergency stop handling.

5.2 Implementing complex behaviors

This section describes the control logic that allows the robot to perform navigation tasks more advanced than simple movements, as in the case of wall-following actions, integrating LiDAR sensor data with a finite state machine (FSM).

5.2.1 LiDAR environment monitoring

Safety is implemented by analyzing different sectors of the LiDAR sensor. The node processes the array of range measurements to extract the minimum distance in critical areas:

- **Front sector**: Defined by an angle of $\pm 20°$ with respect to the robot's x axis. If the minimum distance in this sector is lower than 0.35 m, the robot activates an emergency stop protocol and immediately sets all speeds to 0.
- **Side sectors**: Defined around angles of $\pm 90°$ (with a margin of $15°$), and used to detect the presence of walls at a distance of up to **0.50 m** during wall-following modes.

5.2.2 Finite state machine (FSM) for wall tracking

Wall tracking is managed by a finite state machine that guarantees robust behavior in dynamic environments:

1. **IDLE**: Initial state in which the robot moves in a straight line until it detects either a frontal obstacle or a nearby wall.
2. **DECIDE**: When an obstacle is detected, the robot evaluates the available side space to select the preferred direction of rotation.
3. **TURNING**: The robot performs a pure rotation around its axis until the front sector is free of obstacles.
4. **FOLLOW**: Parallel navigation state in which the robot maintains a constant linear velocity while monitoring the lateral distance to the wall.

5. **RECOVER**: Activated when the robot loses contact with the wall, causing it to initiate a smooth corrective turn.

Chapter 6

Experiments and evaluation

This chapter describes several experiments that demonstrate the functionality and robustness of the system. It also explains the installation of the entire system and the update of the TurtleBot3. In addition, an exhaustive error analysis is performed to detect possible errors and weak points in the system.

6.1 Phase I: Gesture recognition vision model validation

Before testing the model and the system with a specific robot, it was considered preferable to perform an initial validation and error analysis of the gesture detection system based on computer vision. The two main objectives of this phase are: first, to discover how well the `EurekaNet` network described above has learned, and second, to evaluate its robustness in various real-life environments through different tests conducted by different users.

6.1.1 Model training and performance analysis

As mentioned above, the model was trained on the `IPN Hand` [2] dataset, processing a total of over 4,000 gesture samples collected across approximately 200 videos. To optimize computation time, a local environment based on the Apple Silicon architecture was used, taking advantage of the GPU acceleration provided by the Apple M2 chip through Metal Performance Shaders (MPS) on a MacBook Air.

6.1.1.1 Pre-processing and Feature Extraction

The model does not process raw video directly, as doing so would be computationally expensive and would make training under these conditions practically unfeasible, in addition to significantly reducing the final performance of the model. Instead, the model is trained using differential representations of hand geometry. For each sample, 21 3D hand landmarks are extracted using `MediaPipe`. The innovation of this process lies in the selection of 15 key frames for each sequence, generating a highly specialized input vector:

$$D = L^2 \cdot C \cdot (K - 1) \tag{6.1}$$

Where:

- D is the input vector dimensionality,
- L is the number of hand landmarks ($L = 21$),
- C is the number of spatial components per landmark,
- K is the number of selected key frames per gesture sequence.

With $L = 21$, $K = 15$, and $C = 2$, this configuration results in a 12,348-dimensional input vector. This representation captures both the geometric configuration of the user's hand and how it evolves over time, being invariant to the hand's position in the video and enabling the model to focus on dynamic gestures rather than static ones.

6.1.1.2 Training analysis

The training was executed following the recommendations of previous work presented in the related work section [23], for 100 epochs with a batch size of 32, using the Adam optimizer with a learning rate of $lr = 0.001$.

The evolution of the metrics, detailed in Plot 6.1, shows robust convergence and a steady reduction of the error:

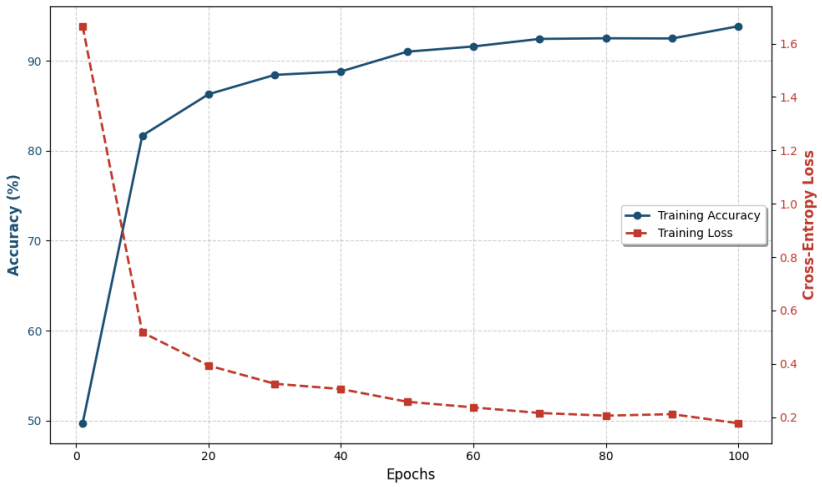


Figure 6.1: Evolution of the accuracy and loss of the EurekaNet training.

From the observation of these results, the following conclusions can be drawn about the learning process:

- **Convergence:** Thanks to the `BatchNorm1d` layers, the model reaches an accuracy higher than 80% before epoch 30.
- **Loss stability:** The loss function steadily decreases from 1.6651 to a final value of 0.1774.
- **Regularization:** The `Dropout(0.25)` layers in the two hidden layers (128 and 64 neurons) help to avoid overfitting and maintain stable training.

- **Final result:** The final saved model (`eureka_model_100_epochs.pth`) achieves a training accuracy of 93.83%.

6.1.2 Quantitative evaluation: confusion matrix

One of the most important points of the project is the ability to validate the gesture detection system in real environments and with several people: users accustomed to the detection system, such as myself, as well as people with no previous knowledge of either the vision system or robotics in general.

This allows an external analysis of the different situations in which the system may fail, either because a user performs a gesture in a significantly different way from other users, or because there is an erroneous classification of the gesture. As will be discussed later, several factors can directly affect the system and lead to recognition errors.

In order to validate the reliability and accuracy of the model, a battery of **600 real tests** was performed (60 experiments for each of the 10 defined gesture classes). These tests involved different volunteers to introduce variability in hand morphology, gesture execution style, and gesture execution speed.

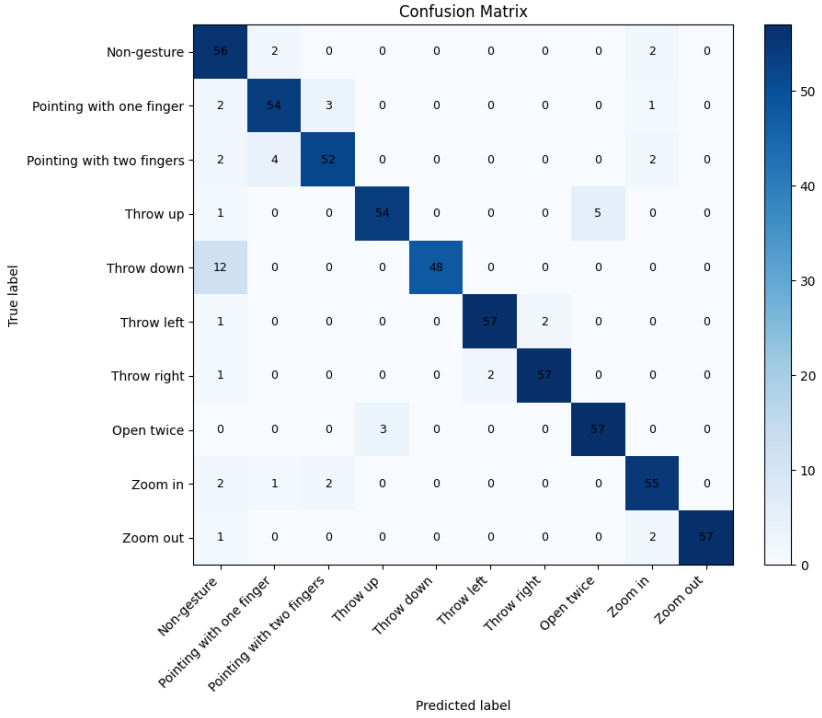


Figure 6.2: Confusion matrix for the gesture recognition system over 600 test.

The analysis of the confusion matrix demonstrates a generally accurate and robust behavior of the gesture recognition system in real scenarios. In particular, several dynamic gestures with a very clear and differentiated trajectory, such as *Throw left*, *Throw right*, and *Open twice*, present excellent recognition rates, above 95%. These results indicate that the model is capable of effectively capturing well-defined

and repetitive temporal patterns.

Regarding the *Pointing* gestures, both with one and two fingers, a high success rate is observed (around 88-90%). However, occasional confusions appear between these two classes and with the *Zoom in* gesture, which is performed with three extended fingers. This behavior is consistent with the geometric similarity between these hand postures: pointing with one and two fingers differs only slightly from zooming with three fingers, making partial confusion expected.

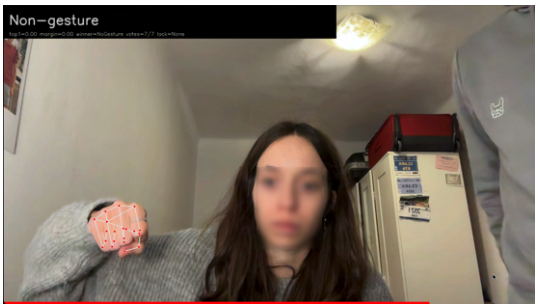
The most notable negative case corresponds to the *Throw down* gesture, which presents a slightly lower success rate compared to the other gestures. Approximately 20% of the samples are classified as *Non-gesture*. This behavior does not mainly correspond to confusion with another active gesture, but rather to an absence of prediction caused by the system's safety mechanisms (confidence thresholds and temporal voting), combined with the more limited visible trajectory when the hand moves downward and exits the camera field of view.

Overall, the matrix reflects a system designed to prioritize safety over uncertainty, where most errors do not derive from incorrect robot actions but from an explicit decision not to act when confidence is insufficient. In the following sections on qualitative evaluation and error analysis, a more detailed study of specific gestures will be presented in order to identify the exact causes of the observed failures and evaluate possible improvements in the system design.

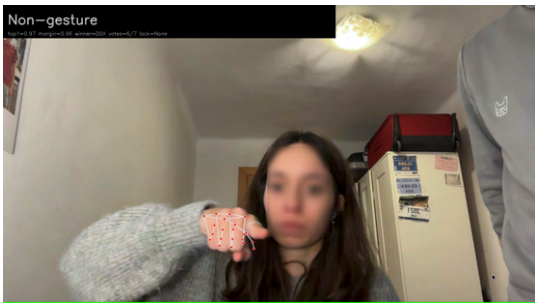
6.1.3 Qualitative evaluation: cases of successful detection

Quantitative evaluation is as important as qualitative evaluation. Most tests, conducted with different volunteers, resulted in accurate and immediate detections. Below, a series of examples is presented, divided into four frames taken consecutively but spaced according to each gesture, in which a correct and immediate detection of the gesture can be observed. These examples represent only a selection of all those collected in the different videos; however, the most representative and explanatory ones have been chosen.

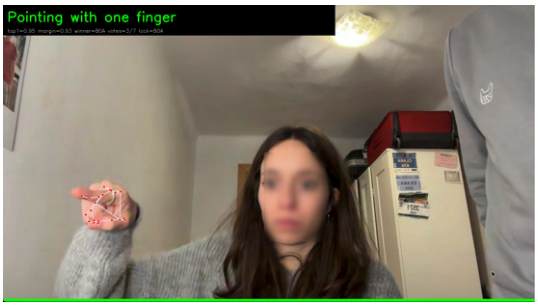
6.1.3.1 Example 1: *Pointing with one finger* gesture (Clara)



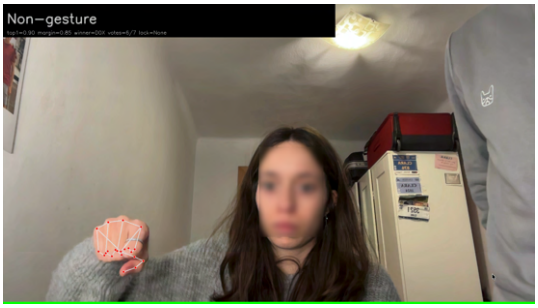
Frame 1



Frame 2



Frame 3



Frame 4

Figure 6.3: Sequence of a successful case in the detection of the gesture *Pointing with one finger*.

The analysis of Figure 6.3 shows a case of success in detecting the gesture *Pointing with one finger*.

Starting with **frame 1**, the user has not yet started the gesture. The hand is in a closed fist position, corresponding to a configuration that the system correctly classifies as *Non-gesture*. This first instant is important, since it shows that the system does not generate false positives for postures that do not really intend to control the robot.

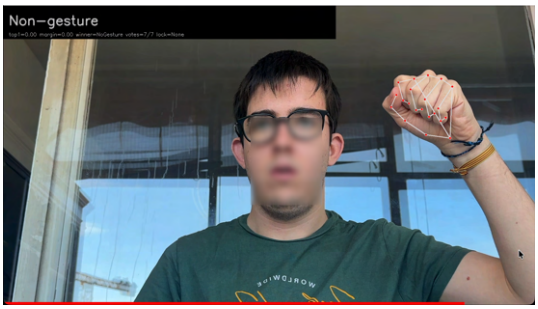
In **frame 2**, the index finger begins to extend, but the system continues to classify the scene as *Non-gesture*. This behavior is expected and desired. We consider that there is a level of confidence that must be met and, in this case, since the gesture has only recently started and we recall that it is dynamic, it is normal that in the first frame in which the extended finger appears, it is not yet detected.

In fact, it is in **frame 3** when the gesture is finally correctly detected as *Pointing with one finger*. At this point, the index finger has already traveled a sufficient trajectory and has been moving for a longer period of time. The temporal window used by the model allows this evolution of the movement to be identified and the established confidence thresholds to be exceeded. It should be noted that the entire gesture takes place in less than a second, which demonstrates the speed of the system.

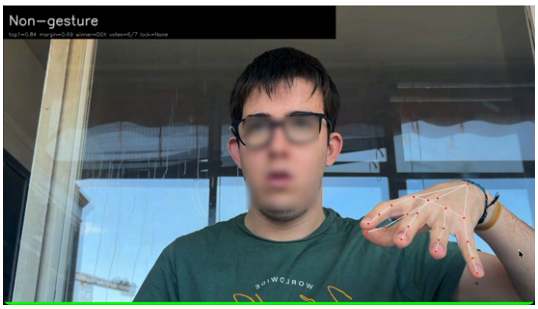
Finally, in **frame 4**, the user closes her hand again into a fist, and the system reacts immediately by returning to *Non-gesture*, thus correctly completing the action.

Overall, this sequence exemplifies a robust and stable operation of the system: it is not activated prematurely, it detects the gesture when sufficient time has elapsed, and finally returns to the initial state when the action is finished.

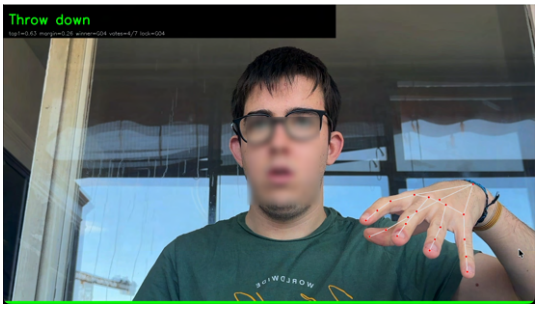
6.1.3.2 Example 2: *Throw down* gesture (Jaume)



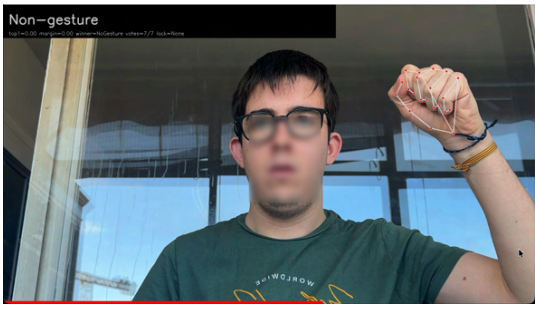
Frame 1



Frame 2



Frame 3



Frame 4

Figure 6.4: Sequence of a successful case in detecting the *Throw down* gesture.

Figure 6.4 shows a second successful case corresponding to the dynamic gesture *Throw down*. It is important to highlight the relevance of including this gesture, since it is one of the most difficult to detect due to the short vertical path and the possible exit of the hand from the field of vision. Later, in the error analysis, an error case with this gesture is described, so it is important to clearly distinguish when it is detected correctly and when it is not.

In **frame 1**, the user starts the sequence with his arm raised and his hand in a fist position. It can be observed that he starts with his arm quite high so that it can have the maximum possible vertical path. In any case, it is still detected as *Non-gesture*, since the objective is to avoid premature detection.

In **frame 2**, the hand has already started moving downwards, but this is the initial moment of the movement and it is still not detected as a gesture because the action has not yet been sufficiently completed.

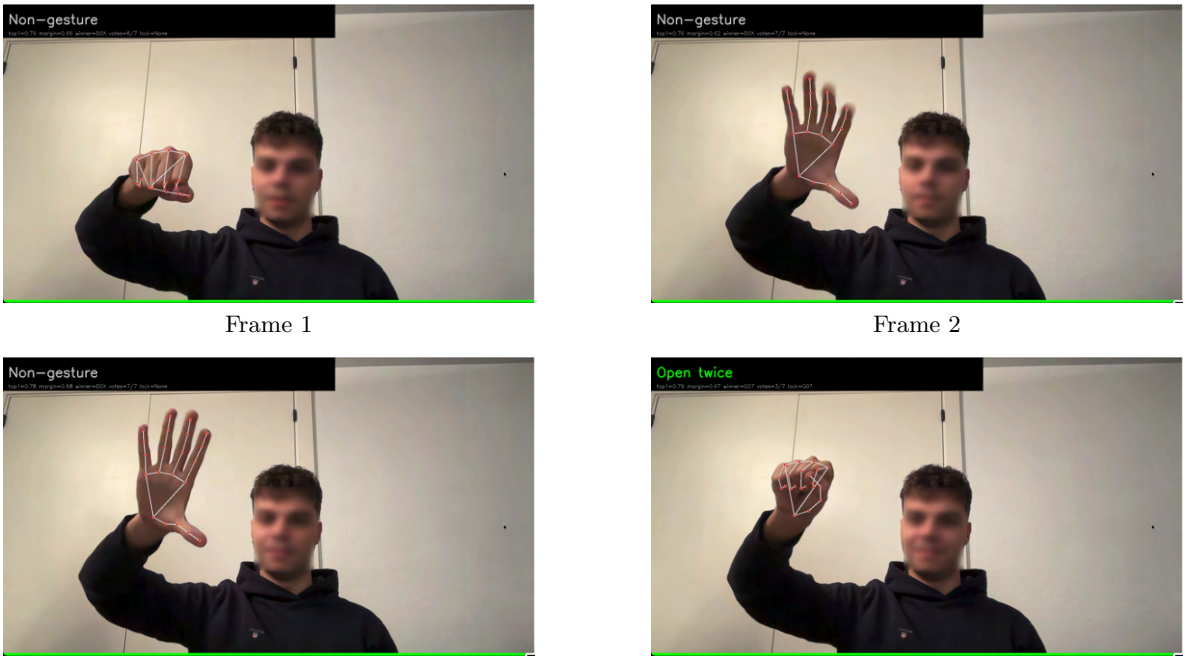
Once again, it is in **frame 3** when the *Throw down* gesture is correctly detected, since at this moment the hand has already traveled a sufficient vertical trajectory and matches the expected gesture pattern,

allowing the model to predict the gesture correctly.

Finally, in **frame 4**, the user raises his arm again and adopts the initial fist position once more. As a result, the system responds appropriately to this change, returning to the *Non-gesture* state and indicating that the action has ended.

In addition, this example is interesting due to the environment, since there is glass in the background, which generates reflections and changes in lighting. Nevertheless, the system remains invariant to these potentially adverse conditions, demonstrating the robustness of the model even in somewhat unfavorable environments.

6.1.3.3 Example 3: *Open twice* gesture (Carlos)



Frame 1 Frame 2

Frame 3 Frame 4

Figure 6.5: Sequence of a successful case in detecting the *Open twice* gesture (Carlos).

In this case, we have one of the gestures that, if we consider the previous confusion matrix, Figure 6.2, performs best. In Figure 6.5, four frames of the correctly classified gesture *Open twice* are shown, which is based on opening the hand twice in a row.

Once again, we can observe that in **frame 1**, the user has not yet started the gesture and has a closed fist, which is classified as *Non-gesture*.

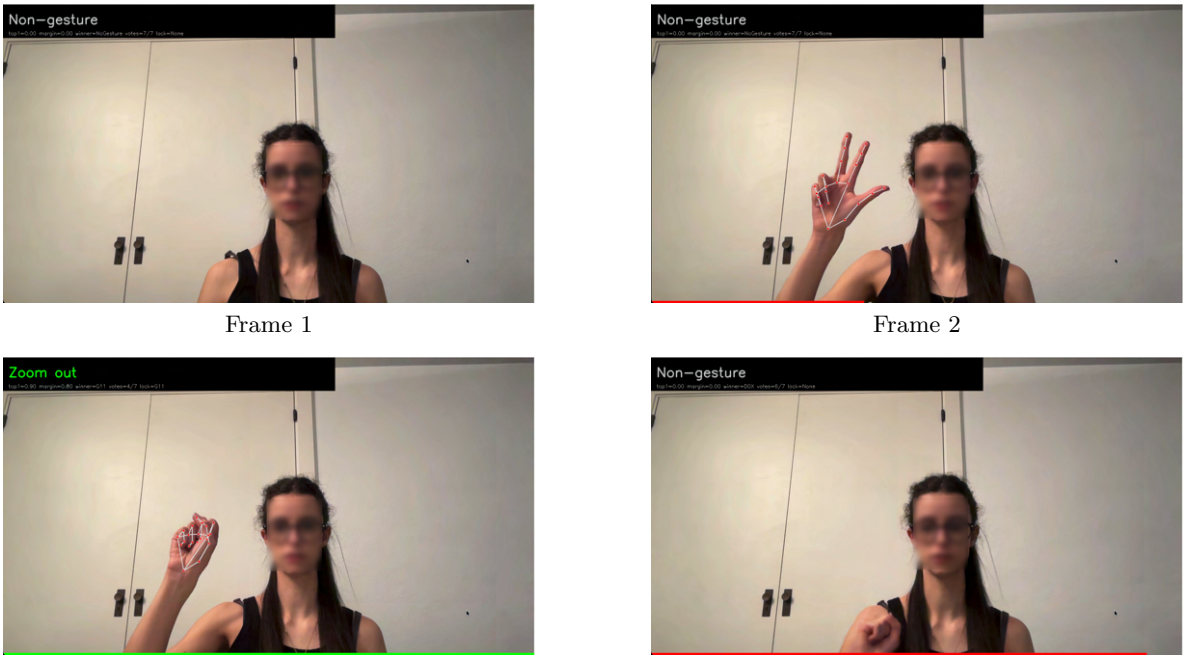
In **frame 2**, the hand is opened for the first time, but the detection continues to be *Non-gesture*, since the gesture *Open twice* is not completed until the second opening and closing of the hand occurs. The

opening gesture is repeated, as can be seen in **frame 3**.

It is in **frame 4** when, once the complete open–close–open–close cycle has been completed, the system correctly classifies the gesture *Open twice*.

This demonstrates, once again, the model's ability to correctly understand when an action begins and ends, and to avoid rare premature detections that could have fatal consequences for the robot.

6.1.3.4 Example 4: *Zoom out* gesture (Blanca)



Frame 1 Frame 2

Frame 3 Frame 4

Figure 6.6: Sequence of a successful case in the detection of the gesture *Zoom out* (Blanca).

Figure 6.6 presents a successful case of the *Zoom out* gesture. In this case, the aim is to highlight the correct functioning of a gesture in which the hand starts outside the field of vision.

In fact, this is why in **frame 1**, the hand does not appear in the scene. This results in correct behavior of the model, which classifies the absence of the hand as *Non-gesture*, demonstrating that the lack of a visible hand does not generate false positives.

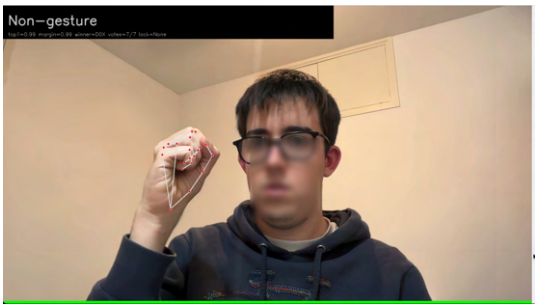
In **frame 2**, the hand appears with three open fingers. Although the gesture has started, the system continues to indicate *Non-gesture*, since it is not defined by a static posture, but rather by a dynamic change to a closed hand.

It is in **frame 3** when the user quickly closes her fingers, completing the corresponding dynamic gesture, and the system immediately classifies the gesture as *Zoom out*.

Finally, in **frame 4**, the hand disappears again and the system reacts correctly once more, ending the action.

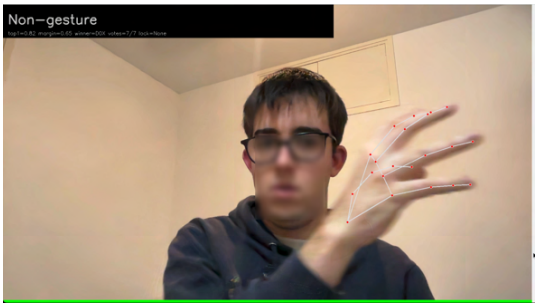
This example demonstrates the robustness of the system both in detecting dynamic gestures and in managing the entry and exit of the hand from the camera view.

6.1.3.5 Example 5: *Throw left* gesture (Jaume)

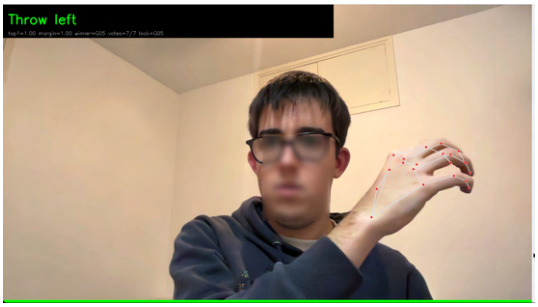


Frame 1

Frame 2



Frame 3



Frame 4

Figure 6.7: Sequence of a successful case in the detection of the gesture *Throw left* (Jaume).

Finally, to close the section of positive examples, the case of *Throw left* has been chosen, which can be seen represented in Figure 6.7. In this example, the entire fluid movement from the middle to the side of the gesture can be correctly observed.

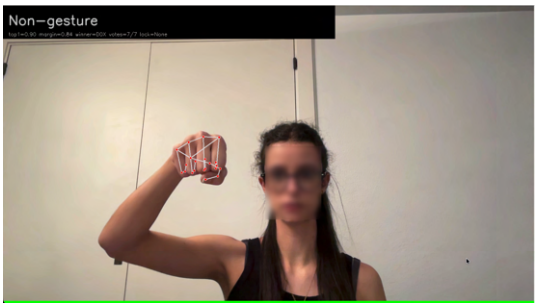
We can see how the hand starts in **frame 1**, in such a way that it is in a centered position and is detected as a *Non-gesture*. In **frames 2 and 3**, the action progresses by moving to the side, but although the movement begins to be visible, it is not yet detected as a *Throw left*.

It is in **frame 4** when the lateral movement has ended, the hand is clearly at the leftmost point (from the camera's point of view), and the gesture is already correctly detected as a *Throw left*.

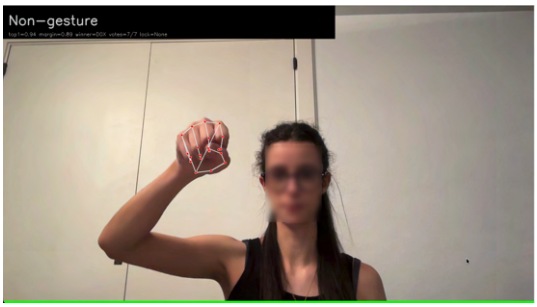
This example is interesting because it allows us to observe the fluidity in detecting the gesture and the accuracy of the system in detecting the gesture once it has finished.

6.1.4 Error analysis and failure cases

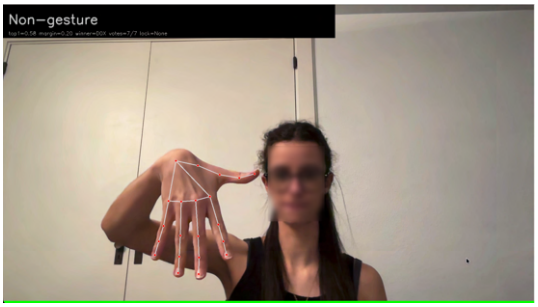
6.1.4.1 Error 1: *Throw down gesture* (Blanca)



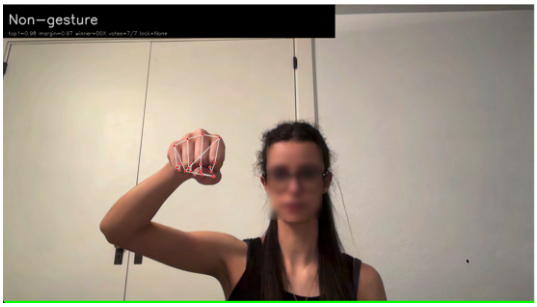
Frame 1



Frame 2



Frame 3



Frame 4

Figure 6.8: Sequence of a case of error in the detection of the gesture *Throw down* (Blanca).

Let's start the error cases with Figure 6.8, which shows the gesture *Throw down*. It should be remembered that this example is very relevant, since, as the confusion matrix of Figure 6.2 demonstrates, it has been identified as the most problematic. Here, it is shown that this is mainly due to the need for a sufficiently long and marked vertical trajectory.

In **frame 1**, the user starts the sequence with her hand closed and her arm in a centered position, and it is classified as *Non-gesture*. Also, in **frame 2**, the hand does not move at all, neither up to gain momentum nor downwards, and the classification remains unchanged.

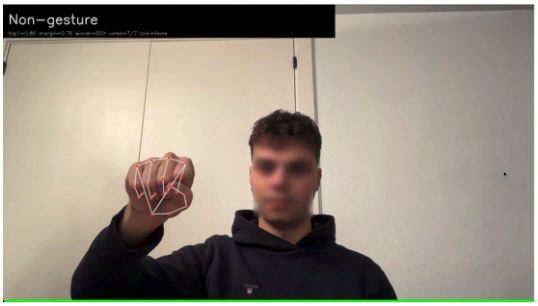
It is in **frame 3** when the hand moves slightly to a lower position, but it is clear to see that the difference in the position of the hand with respect to the previous frames is minimal and insufficient to detect the *Throw down*. In this case, the movement does not reflect an action of "pulling down", but rather of releasing the arm.

Then, in **frame 4**, the user returns to the initial position without the gesture having been correctly detected.

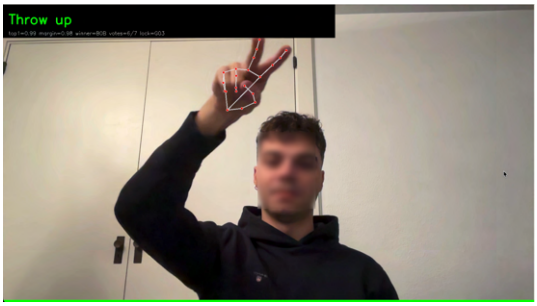
In conclusion, this case demonstrates an expected limitation of the system, namely that the detection of dynamic gestures requires marked, fast, and sufficiently broad movements to be classified correctly.

This is not the case in this example, since the *Throw down* gesture is too brief and, above all, too static.

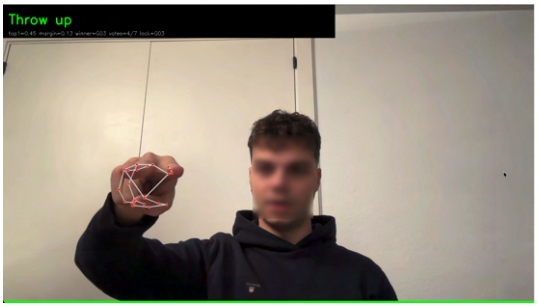
6.1.4.2 Error 2: *Pointing with two fingers* gesture (Carlos)



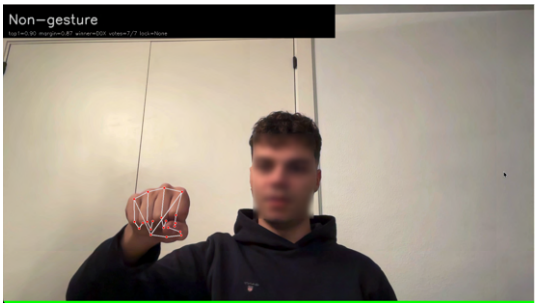
Frame 1



Frame 2



Frame 3



Frame 4

Figure 6.9: Sequence of a case of error in the detection of the gesture *Two fingers* (Carlos).

In Figure 6.9, an error case associated with the gesture *Pointing with two fingers* is highlighted. However, unlike the previous error, this is not due to a lack of forcefulness when making the gesture, but rather to confusion with another gesture caused by the user's poor execution.

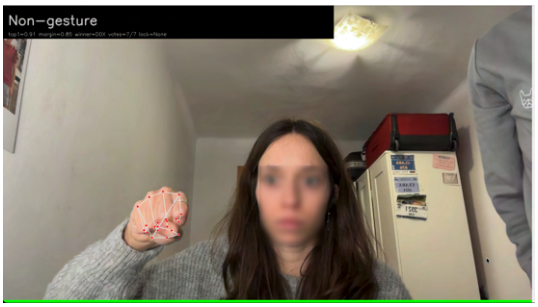
In this case, the flow of the gesture is not entirely correct. In **frame 1**, nothing is detected (*Non-gesture*). The problem begins when, in **frame 2**, the user quickly stretches two fingers and makes a fast upward movement. This movement clearly confuses the model as a *Throw up* gesture, since the movement, despite being made with only two fingers, is clearly upward.

Then, in **frame 3**, the *Throw up* gesture continues to be detected, since the user is moving very fast, does not clearly show two fingers, is very frontal to the camera, and does not make the usual rectangular movement.

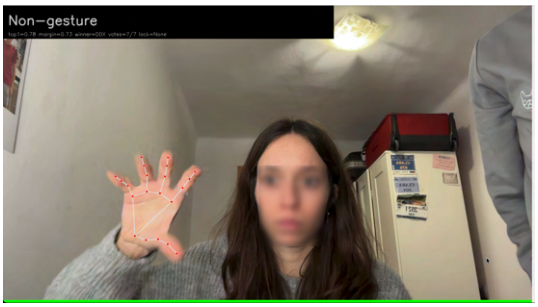
Finally, in **frame 4**, the action ends without the gesture having been correctly detected at any time.

In this case, the error demonstrates that gestures require a somewhat structured dynamic and that incorrectly combining more than one gesture can lead to a wrong classification, as is the case here.

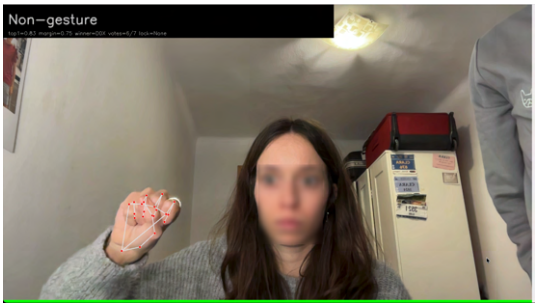
6.1.4.3 Error 3: *Open twice* gesture (Clara)



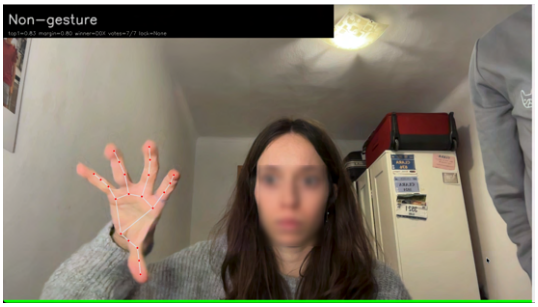
Frame 1



Frame 2



Frame 3



Frame 4

Figure 6.10: Sequence of a case of error in the detection of the gesture *Open twice* (Clara).

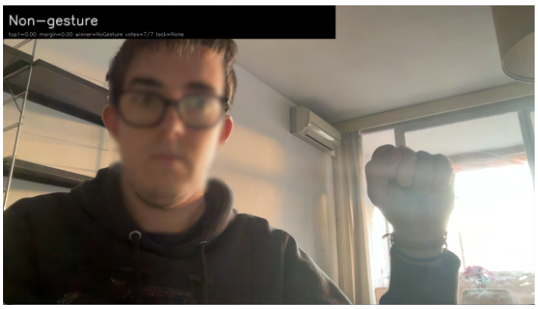
This error case clearly shows an error in detecting the *Open twice* gesture, mainly caused by a poorly marked execution of the gesture, as can be seen in Figure 6.10.

The flow of the action is correct. As can be observed, the open–close–open–close gesture is performed correctly, but the problem appears in **frames 2 and 4**, when the hand is theoretically open but not open enough, which causes the gesture not to be correctly classified as *Open twice*.

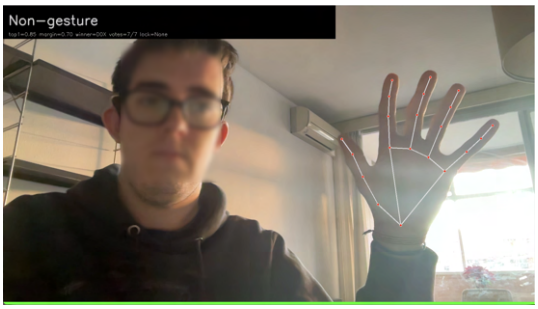
This is because the opening of the hand is not pronounced enough for the system to clearly differentiate it from a partially closed hand. As a result, due to the conservative behavior of the system, the confidence limits are not exceeded and the gesture is not classified. The complete sequence does not exceed the confidence thresholds, and the gesture is not classified.

As explained previously, this error demonstrates one of the strengths of the system, which is its conservative behavior. Although the *Open twice* gesture is one of the best classified, if the system does not perceive it clearly enough, it does not classify it, thus avoiding false positives that could be critical for robot control. In this context, it is preferable to generate a false negative and require the user to repeat the gesture rather than produce a false positive that could result in unintended robot motion.

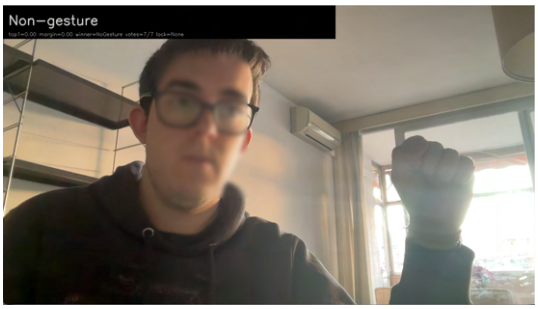
6.1.4.4 Error 4: *Open twice* gesture (Jaume)



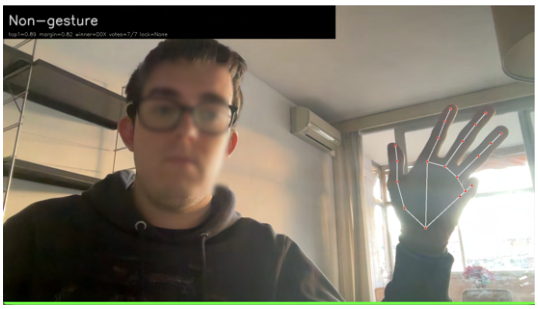
Frame 1



Frame 2



Frame 3



Frame 4

Figure 6.11: Sequence of a case of error in the detection of the gesture *Open twice* (Jaume).

In the case of Figure 6.11, the error has been caused intentionally in order to evaluate the system under adverse lighting conditions.

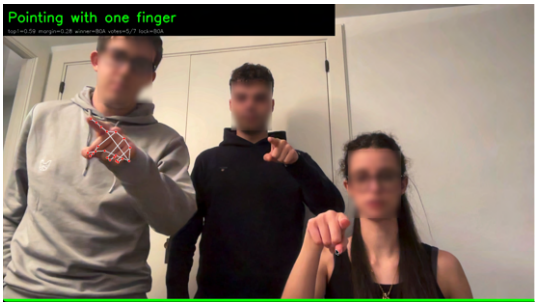
In this sequence, the camera used is a basic RGB camera and has been placed in a backlight situation. This has caused that, as can be observed in **frames 1 and 3**, when the hand is strongly illuminated from behind, it appears as a uniform shape in which the fingers are not distinguishable, and the system is not able to extract the landmarks.

On the other hand, surprisingly, in **frames 2 and 4**, the hand is presented with a clear opening, which allows the system to correctly place the landmarks.

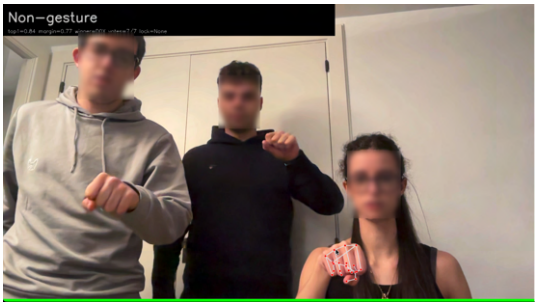
However, the interruptions in the detection prevent the system from classifying the gesture correctly as *Open twice*. This demonstrates a limitation of the system, in particular that under conditions such as these, when using basic cameras, sensitivity to external lighting is higher than with higher-quality cameras, leading to detection difficulties such as the one shown in this example.

6.1.5 Special case: detection with three people

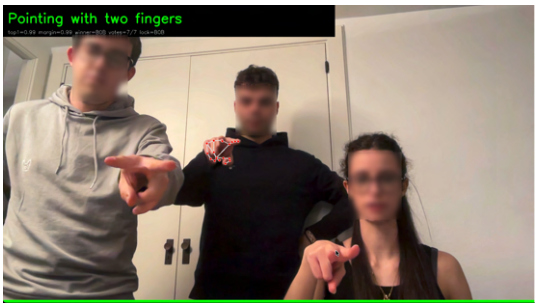
As this case cannot be clearly classified as either a success or a failure, it has been presented separately.



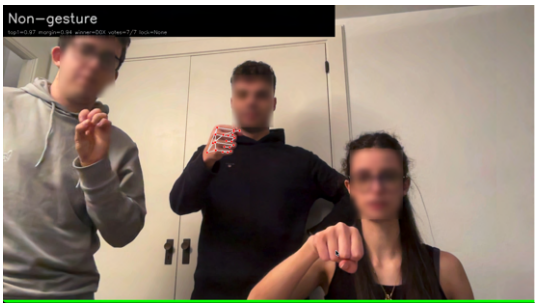
Frame 1



Frame 2



Frame 3



Frame 4

Figure 6.12: Sequence of a special case of detection with three people in the scene.

Figure 6.12 presents a special case that was tested to analyze how the system reacts. It was tested because there are situations in which more than one person may appear in the field of vision, and it was necessary to study how the system behaves in such cases. In any case, it should be noted that this situation does not correspond to an error as such, but neither to a case of complete success.

By observing Figure 6.12, it can be seen that the system continues to function correctly, and the gestures are properly detected, since the three users perform gestures simultaneously with their hands. The behavior observed, however, is that in each frame only one hand is detected, specifically the one with the highest level of confidence.

Even so, the gestures continue to be detected correctly. In **frame 1**, for example, the gesture *Pointing with one finger* is correctly detected, and in **frame 3**, the gesture *Pointing with two fingers*.

This case demonstrates that, in scenarios with multiple people, the system does not discriminate between users, but instead selects the hand that maximizes detection confidence at each instant and is therefore able to continue functioning. However, to ensure optimal operation, it is recommendable to have only one user within the field of view.

6.1.6 Conclusions of Phase I

Overall, the results of this first validation phase show that the gesture recognition system works reliably in real environments and with different users, even when the execution style, speed, or hand

morphology varies. The quantitative evaluation, based on 600 real tests and summarized in the confusion matrix, indicates that most gesture classes achieve high recognition rates, especially those with a clear movement.

From the qualitative examples, it can be concluded that the system behaves in a conservative and consistent way: it does not activate prematurely, it detects the gesture once enough temporal evidence has been accumulated, and it returns to the *Non-gesture* state quickly when the action ends. This behavior is positive for robot control, since it reduces the risk of unintended activations.

At the same time, the error analysis highlights the main limitations of the approach. First, some gestures require a sufficiently marked trajectory to be recognized, as is the case for *Throw down*, where short or weak vertical movements tend to be rejected as *Non-gesture*. Second, there are failure cases caused by user execution, especially when the motion direction resembles another gesture class, producing confusion (for example, *Pointing with two fingers* being interpreted as *Throw up*). Finally, external conditions can directly affect landmark extraction: in backlight situations and with basic RGB cameras, the hand can lose definition and the system may not be able to extract reliable landmarks, interrupting temporal continuity and preventing correct classification.

The special case with three people in the scene also provides an important insight: the system can keep working, but it is not designed to handle multiple users simultaneously, since it effectively selects a single hand per frame based on the highest confidence. Therefore, for practical use, it is preferable to operate with only one user within the field of view.

In conclusion, Phase I validates the gesture recognition model as a solid basis for robot interaction, with good general performance and a safety-oriented behavior. The observed limitations are mainly linked to subtle gesture execution, ambiguous trajectories between classes, and challenging lighting conditions, which will be important to consider in the next phase when integrating the system with the robot.

6.2 Phase II: Robot setup and tuning

Although the vision system based on `EUREKA` and `MediaPipe` could be applied to any robotic platform, for this work the TurtleBot3 Waffle Pi has been selected as a test environment. This section details the configuration of this hardware to convert it into a gesture-based command receiver.

6.2.1 Selection of the TurtleBot3 Waffle Pi

The choice of this platform responds to my use in undergraduate robotics subjects, in addition to its popularity in the academic research community in mobile robotics. The Waffle Pi offers us a stable mobile base to test how artificial vision can replace physical controls. In addition, the fact of incorporating a Raspberry Pi 4 and a LiDAR allows us to validate not only the reception of gestures, but also the physical safety of the robot during teleoperation.

6.2.2 MicroSD preparation and OS selection

The entire OS of the robot is installed on a MicroSD card (a 32GB MicroSD card was used in this case). The first step of the setup was to prepare this storage media to ensure a clean installation, by fully formatting the card.

6.2.2.1 Operating System selection

The installation on the MicroSD card was carried out using the Raspberry Pi Imager software. To ensure a correct and compatible setup, the following parameters were selected:

1. **Device: Raspberry Pi 4**.
2. **Operating System:** *"Ubuntu" → "Ubuntu Server 24.04 LTS (64-bit)"*.
3. **Storage:** 32GB MicroSD card.

Ubuntu Server 24.04 was selected because `ROS 2 Jazzy Jalisco` specifically requires this version (see Figure 6.13). The Server edition was preferred to reduce RAM usage.

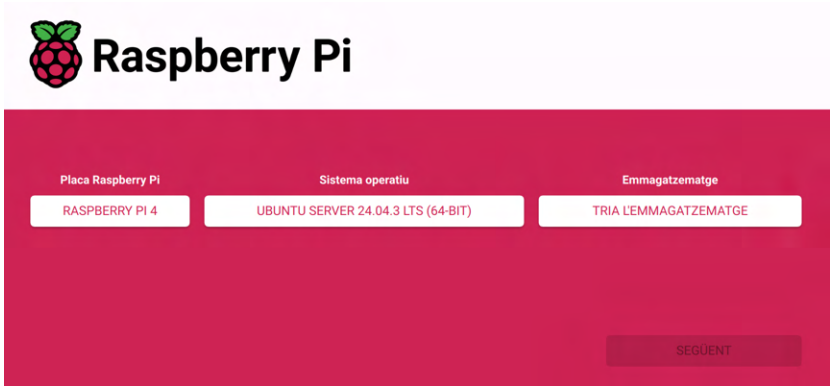


Figure 6.13: Selection of the hardware, OS, and storage in Raspberry Pi Imager.

6.2.3 Network configuration and remote access

Inside the Raspberry Pi Imager, the network was configured onto the MicroSD card before the first boot, enabling a true "headless" setup. This is a very important step as `ROS 2` needs a stable network in order to transmit data and packages between the robot and all the remote PCs.

6.2.3.1 Pre-configuration settings

The customization menu was used to define the following:

- **General:** Hostname set to `turtlebot`.
- **Wireless LAN:** Local WiFi credentials entered.
- **Services:** `SSH` interface enabled with password authentication.

6.2.3.2 Connection verification

Once powered on, the robot connected automatically. The assigned IP address was 10.2.24.214. Connection was verified from the PC:

```
# Ping check
ping 10.2.24.214

# SSH Login
ssh ubuntu@10.2.24.214
```

Listing 6.1: Network connectivity and SSH access verification

6.2.4 First boot and hardware verification

The MicroSD card was inserted into the Raspberry Pi. For the initial configuration, a VGA monitor and a USB keyboard were connected directly to the board.

6.2.4.1 Power supply considerations

It is **very important** to mention that the entire installation of `ROS 2` is a very long and energy-consuming process. Connecting the Raspberry Pi to electricity with a USB-C cable is recommended to avoid system shutdown during installation.

6.2.4.2 Camera replacement and testing

The original camera was replaced with a new **Raspberry Pi Camera Module 2**. To verify it, the `v4l2` utilities were used:

```
sudo apt update && sudo apt install v4l2-utils
v4l2-ctl --list-devices
```

Listing 6.2: Camera device detection using v4l2 utilities

The camera was recognized as `/dev/video0`. The camera module used is shown in Figure 6.14.

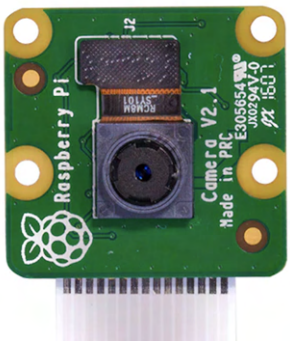


Figure 6.14: The Raspberry Pi Camera Module V2

6.2.5 Installation of `ROS 2 Jazzy Jalisco` and dependencies

`ROS 2 Jazzy Jalisco` was installed from the official repository. The `ros-base` version was selected to minimize resource consumption on the Raspberry Pi 4.

```
# Installation of build dependencies and TB3 drivers
sudo apt update && sudo apt install -y \
python3-colcon-common-extensions \
libboost-system-dev \
libudev-dev \
ros-jazzy-dynamixel-sdk \
ros-jazzy-turtlebot3-msgs
```

Listing 6.3: Installation of ROS 2 dependencies and TurtleBot3 drivers

6.2.6 Configuring the TurtleBot3 workspace (`/turtlebot3_ws`)

Within this workspace, the `jazzy` branches of the official `ROBOTIS` repositories were cloned. `navigation2` and `cartographer` were removed to speed up compilation. This process took almost 14 hours on the robot.

```
# Incremental compilation on the robot (SBC)
colcon build --symlink-install --parallel-workers 1
```

Listing 6.4: Incremental compilation of the TurtleBot3 workspace

6.2.6.1 PC–Robot communication configuration

To ensure that the PC and robot nodes can find each other, not only must the two devices be connected to the same network, but it is also important to share a common configuration based on `DDS`. In addition, the relationship between the IP addresses of the two devices and the ROS domain must be explicitly defined, as shown below.

```
export ROS_DOMAIN_ID=30
export TURTLEBOT3_MODEL=waffle_pi
export RMW_IMPLEMENTATION=rmw_fastrtps_cpp
```

Listing 6.5: Common DDS and TurtleBot3 environment configuration

This specific configuration avoids interference with other ROS 2 robots that may be running on the same network, which is an important consideration in environments such as a robotics laboratory, where more than one device may be operating simultaneously.

Since, during testing in real environments, it has been observed that automatic discovery between the robot and the PC does not always work correctly, it was decided to force the discovery process and explicitly define the known devices using static IP addresses.

Therefore, the configurations on the PC and the robot are as follows. On the lab PC, the initial verification is performed with:

```
export ROS_DOMAIN_ID=30
export ROS_AUTOMATIC_DISCOVERY_RANGE=SUBNET
export ROS_STATIC_PEERS="IP_DEL_ROBOT"
ros2 daemon stop
ros2 daemon start
sleep 2
ros2 topic list
```

Listing 6.6: Forced DDS discovery and topic verification on the PC

Equivalently, on the robot:

```
export ROS_DOMAIN_ID=30
export ROS_AUTOMATIC_DISCOVERY_RANGE=SUBNET
export ROS_STATIC_PEERS="IP_DEL_PC"
ros2 daemon stop
ros2 daemon start
sleep 2
ros2 topic list
```

Listing 6.7: Forced DDS discovery and topic verification on the robot

The correct, and above all identical, topic list on both devices confirms that the communication between the PC and the robot is working properly.

6.2.7 Startup and final verification of the system

Once communication has been established, it is time to start up the robot hardware. The boot process is defined in a modular way to facilitate error detection and, therefore, each command is executed in a different terminal until the complete start-up of the system on the robot is achieved.

6.2.7.1 Robot bringup

First, in a terminal on the robot (after accessing it via SSH), the TurtleBot3 base system is launched:

```
ros2 launch turtlebot3_bringup robot.launch.py
```

Listing 6.8: Launching the TurtleBot3 base system

```
ubuntu@turtlebot:~$ ros2 launch turtlebot3_bringup robot.launch.py
[INFO] [launch]: All log files can be found below /home/ubuntu/.ros/log/2026-01-08-17-01-01-541437-turtlebot-1304
[INFO] [launch]: Default logging verbosity is set to INFO
urdf_file_name : turtlebot3_waffle_pi.urdf
[INFO] [robot_state_publisher-1]: process started with pid [1308]
[INFO] [ld08_driver-2]: process started with pid [1309]
[INFO] [turtlebot3_ros-3]: process started with pid [1310]
[robot_state_publisher-1] [WARN] [1767888062.614225964] [rcl]: ROS_LOCALHOST_ONLY is deprecated but still honored if it is enabl
ed. Use ROS_AUTOMATIC_DISCOVERY_RANGE and ROS_STATIC_PEERS instead.
[robot_state_publisher-1] [WARN] [1767888062.614293714] [rcl]: 'localhost_only' is disabled, 'automatic_discovery_range' and 'st
atic_peers' will be used.
[ld08_driver-2] [WARN] [1767888062.626255330] [rcl]: ROS_LOCALHOST_ONLY is deprecated but still honored if it is enabled. Use RO
S_AUTOMATIC_DISCOVERY_RANGE and ROS_STATIC_PEERS instead.
[ld08_driver-2] [WARN] [1767888062.626335056] [rcl]: 'localhost_only' is disabled, 'automatic_discovery_range' and 'static_peers
' will be used.
[turtlebot3_ros-3] [WARN] [1767888062.646514302] [rcl]: ROS_LOCALHOST_ONLY is deprecated but still honored if it is enabled. Use
 ROS_AUTOMATIC_DISCOVERY_RANGE and ROS_STATIC_PEERS instead.
[turtlebot3_ros-3] [WARN] [1767888062.646632975] [rcl]: 'localhost_only' is disabled, 'automatic_discovery_range' and 'static_pe
ers' will be used.
[turtlebot3_ros-3] [INFO] [1767888062.738474743] [turtlebot3_node]: Init TurtleBot3 Node Main
[turtlebot3_ros-3] [INFO] [1767888062.741246230] [turtlebot3_node]: Init DynamixelSDKWrapper
[turtlebot3_ros-3] [INFO] [1767888062.748297367] [DynamixelSDKWrapper]: Succeeded to open the port(/dev/ttyACM0)!
[turtlebot3_ros-3] [INFO] [1767888062.761192333] [DynamixelSDKWrapper]: Succeeded to change the baudrate!
[turtlebot3_ros-3] [INFO] [1767888062.798318977] [turtlebot3_node]: Start Calibration of Gyro
[robot_state_publisher-1] [INFO] [1767888062.822875186] [robot_state_publisher]: Robot initialized
[ld08_driver-2] /dev/ttyACM0    OpenCR Virtual ComPort in FS Mode
[ld08_driver-2] /dev/ttyUSB0    CP2102 USB to UART Bridge Controller
[ld08_driver-2] FOUND LDS-02
[ld08_driver-2] LDS-02 started successfully
```

Figure 6.15: Terminal output showing the launch of the TurtleBot3 base system using `turtlebot3_bringup`.

Once started without errors, this terminal is left running for the entire time the robot is in use (see Figure 6.15).

6.2.7.2 Camera bringup

Then, in another terminal, also on the robot, the camera node is started. This terminal is likewise left running, as shown in the terminal output in Figure 6.16.

```
ros2 run v4l2_camera v4l2_camera_node
```

Listing 6.9: Starting the camera node

```
ubuntu@turtlebot:~$ ros2 run v4l2_camera v4l2_camera_node
[WARN] [1767887828.930318728] [rcl]: ROS_LOCALHOST_ONLY is deprecated but still honored if it is enabled. Use ROS_AUTOMATIC_DISC
OVERY_RANGE and ROS_STATIC_PEERS instead.
[WARN] [1767887828.930405855] [rcl]: 'localhost_only' is disabled, 'automatic_discovery_range' and 'static_peers' will be used.
[INFO] [1767887829.397637725] [v4l2_camera]: Driver: bcm2835 mmal
[INFO] [1767887829.397962437] [v4l2_camera]: Version: 395276
[INFO] [1767887829.398085582] [v4l2_camera]: Device: mmal service 16.1
[INFO] [1767887829.398172042] [v4l2_camera]: Location: platform:bcm2835_v4l2-0
[INFO] [1767887829.398234800] [v4l2_camera]: Capabilities:
[INFO] [1767887829.398292465] [v4l2_camera]:   Read/write: YES
[INFO] [1767887829.398347278] [v4l2_camera]:   Streaming: YES
[INFO] [1767887829.398423498] [v4l2_camera]: Current pixel format: JPEG @ 1024x768
[INFO] [1767887829.399741292] [v4l2_camera]: Available pixel formats:
[INFO] [1767887829.399820882] [v4l2_camera]:   YU12 - Planar YUV 4:2:0
[INFO] [1767887829.399880658] [v4l2_camera]:   YUYV - YUYV 4:2:2
[INFO] [1767887829.399936064] [v4l2_camera]:   RGB3 - 24-bit RGB 8-8-8
[INFO] [1767887829.400013099] [v4l2_camera]:   JPEG - JFIF JPEG
[INFO] [1767887829.400075208] [v4l2_camera]:   H264 - H.264
[INFO] [1767887829.400133243] [v4l2_camera]:   MJPG - Motion-JPEG
[INFO] [1767887829.400187408] [v4l2_camera]:   YVYU - YVYU 4:2:2
[INFO] [1767887829.400240388] [v4l2_camera]:   VYUY - VYUY 4:2:2
[INFO] [1767887829.400293275] [v4l2_camera]:   UYVY - UYVY 4:2:2
[INFO] [1767887829.400346829] [v4l2_camera]:   NV12 - Y/UV 4:2:0
[INFO] [1767887829.400400717] [v4l2_camera]:   BGR3 - 24-bit BGR 8-8-8
[INFO] [1767887829.400454104] [v4l2_camera]:   YV12 - Planar YVU 4:2:0
[INFO] [1767887829.400507602] [v4l2_camera]:   NV21 - Y/VU 4:2:0
[INFO] [1767887829.400561045] [v4l2_camera]:   RX24 - 32-bit XBGR 8-8-8-8
[INFO] [1767887829.400614359] [v4l2_camera]: Available controls:
[INFO] [1767887829.400685393] [v4l2_camera]:   Brightness (1) = 50
```

Figure 6.16: Terminal output corresponding to the execution of the camera node using `v4l2_camera`.

6.2.7.3 Vision node execution

Once the publication of the images is confirmed, the vision node responsible for gesture recognition is executed in a terminal on the PC where the corresponding ROS package is installed. It is important to remember to compile the package before execution (see Figure 6.17).

```
cd ~/ws
source install/setup.bash
ros2 run gesture_recognition_pkg vision_node
```

Listing 6.10: Launching the gesture recognition vision node

```
jaumemil@jaumemil:~/tfg_jaume_v5$ ros2 run gesture_recognition_pkg vision_node
/home/jaumemil/.local/lib/python3.12/site-packages/matplotlib/projections/__init__.py:63: UserWarning: Unable to import Axes3D.
This may be due to multiple versions of Matplotlib being installed (e.g. as a system package and as a pip package). As a result,
 the 3D projection is not available.
  warnings.warn("Unable to import Axes3D. This may be due to multiple versions of "
[WARN] [1767887881.302809471] [rcl]: ROS_LOCALHOST_ONLY is deprecated but still honored if it is enabled. Use ROS_AUTOMATIC_DISC
OVERY_RANGE and ROS_STATIC_PEERS instead.
[WARN] [1767887881.302836904] [rcl]: 'localhost_only' is disabled, 'automatic_discovery_range' and 'static_peers' will be used.
[INFO] [1767887881.407356390] [node_de_visio]: Carregant IA: /home/jaumemil/tfg_jaume_v5/install/gesture_recognition_pkg/share/g
esture_recognition_pkg/models/eureka_model.pth
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
I0000 00:00:1767887881.510289   19176 gl_context_egl.cc:85] Successfully initialized EGL. Major : 1 Minor: 5
I0000 00:00:1767887881.517849   19216 gl_context.cc:357] GL version: 3.2 (OpenGL ES 3.2 Mesa 25.0.7-0ubuntu0.24.04.2), renderer:
 Mesa Intel(R) HD Graphics 5500 (BDW GT2)
Warning: Ignoring XDG_SESSION_TYPE=wayland on Gnome. Use QT_QPA_PLATFORM=wayland to run on Wayland anyway.
INFO: Created TensorFlow Lite XNNPACK delegate for CPU.
W0000 00:00:1767887881.539807   19209 inference_feedback_manager.cc:114] Feedback manager requires a model with a single signatu
re inference. Disabling support for feedback tensors.
W0000 00:00:1767887881.560246   19210 inference_feedback_manager.cc:114] Feedback manager requires a model with a single signatu
re inference. Disabling support for feedback tensors.
[INFO] [1767887881.780009607] [node_de_visio]: Finestra GUI iniciada. Prem 'q' o Ctrl+C per sortir.
```

Figure 6.17: Terminal output showing the execution of the gesture recognition vision node on the laboratory PC.

6.2.7.4 Control node execution

Finally, in a fourth terminal, the robot control node is started. This node transforms the detected gestures into movement commands for the robot (Figure 6.18):

```
cd ~/ws
source install/setup.bash
ros2 run turtlebot_control_pkg control_node
```

Listing 6.11: Launching the robot control node

```
jaumemil@jaumemil:~/tfg_jaume_vf$ ros2 run turtlebot_control_pkg control_node
[WARN] [1767888180.611277257] [rcl]: ROS_LOCALHOST_ONLY is deprecated but still honored if it is enabled. Use ROS_AUTOMATIC_DISC
OVERY_RANGE and ROS_STATIC_PEERS instead.
App Center 67888180.611313660] [rcl]: 'localhost_only' is disabled, 'automatic_discovery_range' and 'static_peers' will be used.
[INFO] [1767888180.903555956] [control_node]: CONTROL NODE LLEST.
[INFO] [1767888180.904720584] [control_node]: Topics: gesture='gesture_command', scan='scan', cmd_vel='/cmd_vel' (TwistStamped=T
rue)
[INFO] [1767888215.090856928] [control_node]: START ROTATE: lin=0.00 ang=0.80 dur=8.0s
[INFO] [1767888227.497745502] [control_node]: START TURN_LEFT: lin=0.00 ang=-0.80 dur=2.0s
[INFO] [1767888246.985939514] [control_node]: START TURN_RIGHT: lin=0.00 ang=0.80 dur=2.0s
[INFO] [1767888256.128043129] [control_node]: START ROTATE: lin=0.00 ang=0.80 dur=8.0s
[INFO] [1767888267.295955071] [control_node]: REPEAT last_action=9
[INFO] [1767888267.297135540] [control_node]: START ROTATE: lin=0.00 ang=0.80 dur=8.0s
[INFO] [1767888290.917863088] [control_node]: START BACKWARD: lin=-0.20 ang=0.00 dur=5.0s
[INFO] [1767888302.113077489] [control_node]: START FORWARD: lin=0.20 ang=0.00 dur=5.0s
[INFO] [1767888347.046950266] [control_node]: FOLLOW RIGHT START (30s max)
[INFO] [1767888347.062367933] [control_node]: FOLLOW[right] st=IDLE front=0.76 L=0.75(0) R=inf(0) t_left=30.0
[INFO] [1767888347.561270670] [control_node]: FOLLOW[right] st=IDLE front=0.75 L=0.75(0) R=inf(0) t_left=29.5
[INFO] [1767888348.109472369] [control_node]: FOLLOW[right] st=IDLE front=0.76 L=0.75(0) R=inf(0) t_left=28.9
[INFO] [1767888348.613751383] [control_node]: FOLLOW[right] st=IDLE front=1.15 L=0.75(0) R=inf(0) t_left=28.4
[INFO] [1767888349.159149621] [control_node]: FOLLOW[right] st=IDLE front=1.06 L=0.75(0) R=inf(0) t_left=27.9
[INFO] [1767888349.708782766] [control_node]: FOLLOW[right] st=IDLE front=0.97 L=0.77(0) R=inf(0) t_left=27.3
[INFO] [1767888350.209675717] [control_node]: FOLLOW[right] st=IDLE front=0.91 L=0.82(0) R=inf(0) t_left=26.8
[INFO] [1767888350.758701121] [control_node]: FOLLOW[right] st=IDLE front=0.86 L=0.77(0) R=inf(0) t_left=26.3
[INFO] [1767888351.259004996] [control_node]: FOLLOW[right] st=IDLE front=0.77 L=0.69(0) R=inf(0) t_left=25.8
[INFO] [1767888351.809365135] [control_node]: FOLLOW[right] st=IDLE front=0.69 L=0.62(0) R=inf(0) t_left=25.2
[INFO] [1767888352.359888714] [control_node]: FOLLOW[right] st=IDLE front=0.60 L=0.55(0) R=inf(0) t_left=24.7
[INFO] [1767888352.908717409] [control_node]: FOLLOW[right] st=IDLE front=0.52 L=0.47(1) R=inf(0) t_left=24.1
[INFO] [1767888353.457813849] [control_node]: FOLLOW[right] st=IDLE front=0.43 L=0.38(1) R=inf(0) t_left=23.6
```

Figure 6.18: Terminal output corresponding to the execution of the robot control node responsible for publishing movement commands.

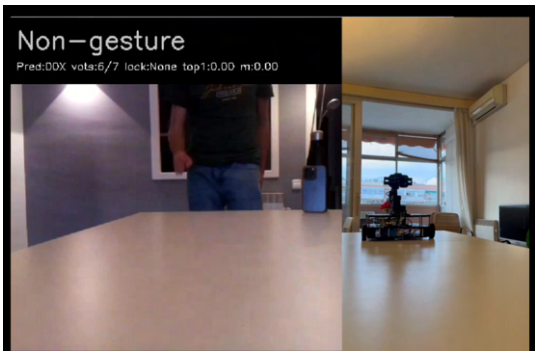
The system is considered correctly operational when it is verified that the image published by the vision node can be visualized on the PC and that, by performing gestures, the robot responds with the expected movements.

6.3 Phase III: On-Robot integrated interaction

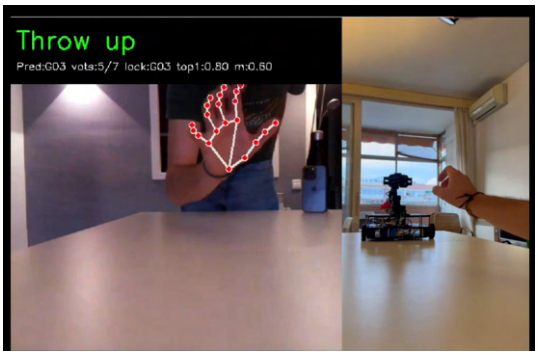
This phase represents the final experiment of the system and constitutes its definitive validation. Once the accuracy of the gesture recognition system had been verified and the mobile robot used in the project had been selected, the actual teleoperation of the TurtleBot3 Waffle Pi was carried out. The objective of this experiment is to demonstrate the robot's capacity to interpret semantic commands and translate them into fluid, fast, and safe movements, all of this using the `TwistStamped` protocol in ROS 2 to guarantee proper temporal synchronization.

6.3.1 Real-time teleoperation and navigation scenarios

Before presenting the experiments gesture by gesture, it is worth highlighting two practical conditions observed during the tests. First, the user-camera distance directly influences the detection, a correct and sufficient distance is recommended in an approximate range of 0.3 m to 1.0 m, as can be seen in the comparison in Figure 6.19, on the left more than a meter which does not allow the gesture to be detected well and on the right closer.



More than 1m



Approximately 0.3m

Figure 6.19: Comparison of user-camera distance: on the left, a distance greater than 1 m makes it difficult to detect the gesture; on the right, a closer distance (approx. 0.3 m) allows for more stable detection.

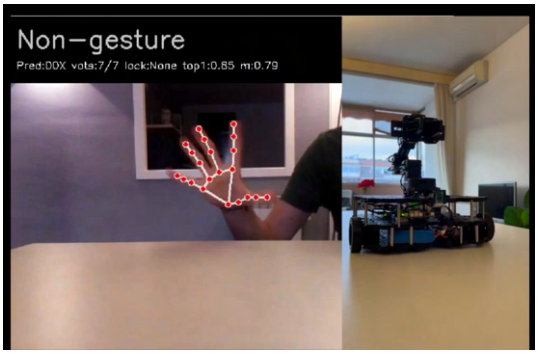
Secondly, the lighting must be functional but not demanding: normal ambient light is sufficient, avoiding especially situations of intense backlight.

In order to make a good analysis of the operation of the system with the robot, a double simultaneous video has been recorded and edited for each gesture, in each frame the execution of the gesture appears on the left and the physical response of the robot on the right. In each of the experiments, a few representative frames are shown that validate the sequence: *User posture → detection → publication of the command → robot movement.*

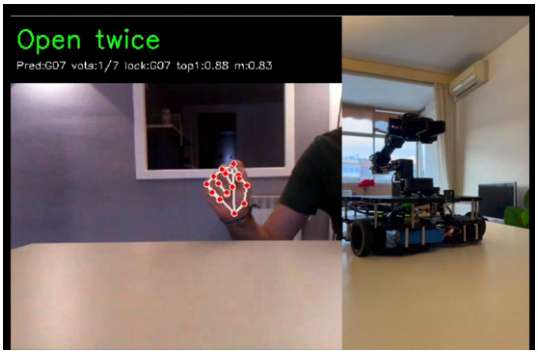
6.3.1.1 Gesture: Open twice (360º rotation)

This first gesture is designed so that the robot can rotate around itself (`G07`, ID 9).

- **Action:** The robot rotates around its own axis at 0.8 rad/s for 8.0 seconds (`t_rotate`), thus completing a full 360° rotation (see Figure 6.20).

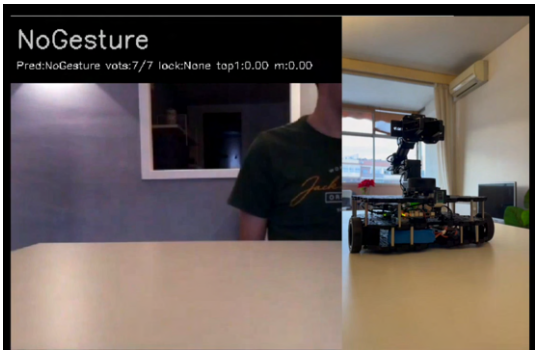


Frame 1



Frame 2

Frame 3



Frame 4

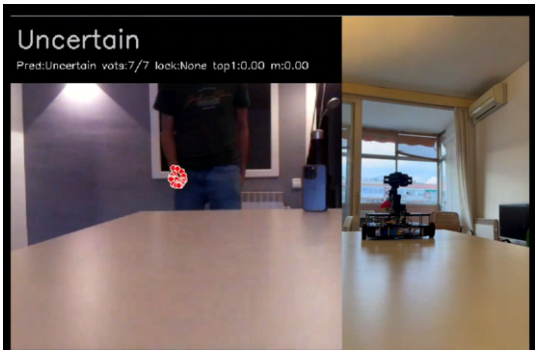
Figure 6.20: Integrated execution of *Open twice*: gesture detection and a full 360° rotation for environment inspection.

It can be clearly observed how, in the first frame, the gesture has just started, corresponding to the first of the two hand openings. In the second frame, the gesture has already finished and been detected, that is why the robot starts to rotate (the robot is already slightly displaced). In the third frame, the robot is almost halfway through the rotation. Finally, the fourth frame shows the completed action.

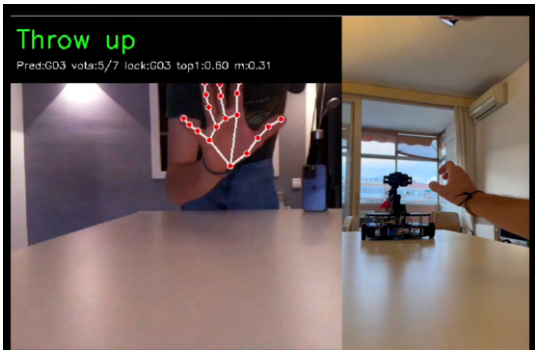
6.3.1.2 Gesture: Throw up (Forward)

Identified by the code `G03` (ID 5), this gesture activates a forward action for a specific time (see Figure 6.21).

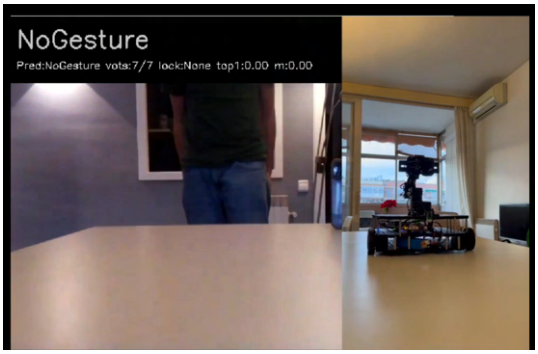
- **Execution:** The robot advances at a linear velocity of 0.2 m/s for a time (`t_fwd`) of 5.0 seconds.
- **Active safety:** The system uses the `_obstacle_ahead()` function to monitor the front sector (20°). If an object is detected within 0.35 m, the action is immediately canceled by calling `_cancel_all()`.



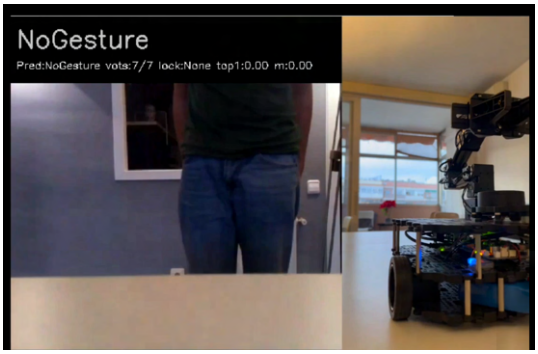
Frame 1



Frame 2



Frame 3



Frame 4

Figure 6.21: Integrated execution of *Throw up*: gesture performance and forward motion of the TurtleBot3.

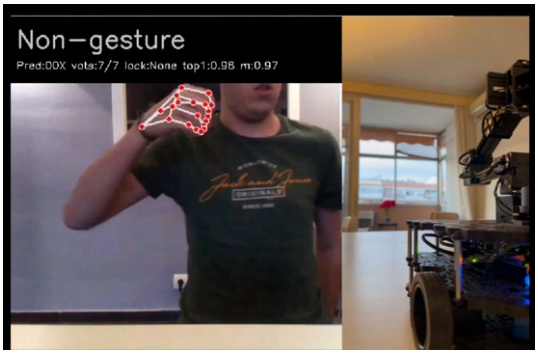
It can be observed how, in the first frame, the system has not yet found any gesture and the detection is in an uncertain state, the movement is just about to begin. It is already in the second frame when the Throw up gesture has already been correctly recognized: the hand is open and moved upwards, thus the system clearly shows the landmarks activates the corresponding command and starts the robot's forward movement.

In the third and fourth frames, it can be clearly observed how the robot is progressively moving forward. In the third the robot is at a halfway point along the route and in the fourth the robot has finished the route and is much closer to the recording camera and the person controlling it.

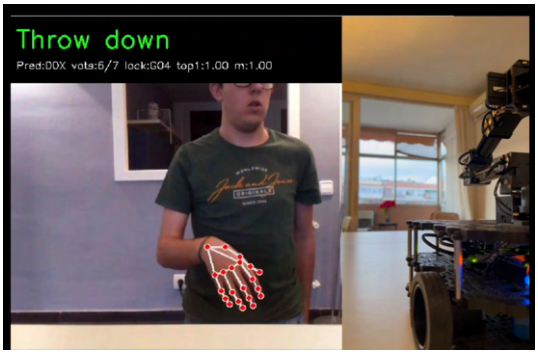
6.3.1.3 Gesture: Throw down (Reverse)

Mapped as `G04` (ID 3), it allows controlled backward maneuvers (see Figure 6.22).

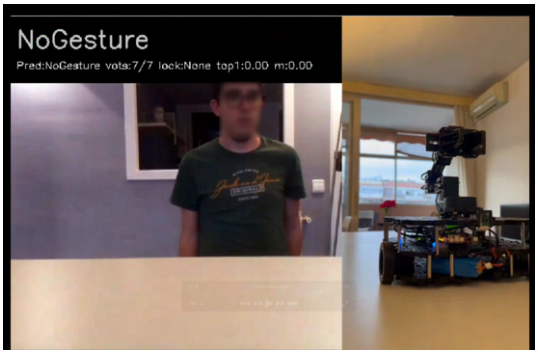
- **Action:** The robot moves backward at a speed of -0.2 m/s for 5.0 seconds (`t_bwd`).



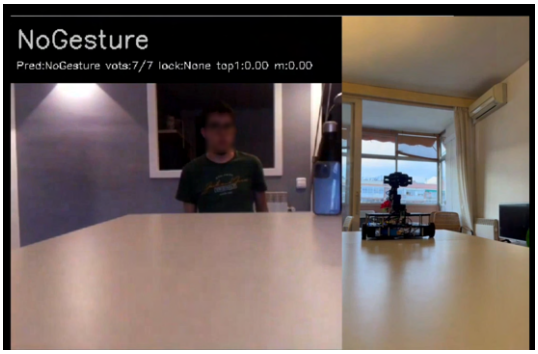
Frame 1



Frame 2



Frame 3



Frame 4

Figure 6.22: Integrated execution of *Throw down*: gesture performance and backward motion of the TurtleBot3.

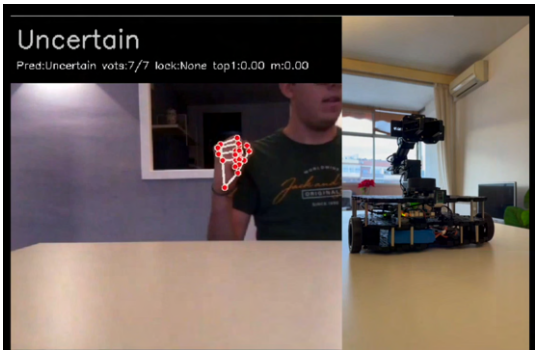
This *Throw down* case is very similar to the previous one. In fact, it corresponds to the same behavior but executed in reverse. In the first frame, once again, the system does not yet detect any active gesture and classifies the scene as Non-gesture, while the hand is in an initial position before starting the movement. It is in the second frame, once the gesture has been completed, that it is correctly detected as *Throw down* and the backward movement begins.

The third and fourth frames illustrate the robot's backward motion. In the third frame, the robot has already moved back slightly but has not yet reached the final position, which is reached in frame 4. The smooth progression of these frames confirms a coherent execution between the timely detection of the gesture and the physical response of the robot.

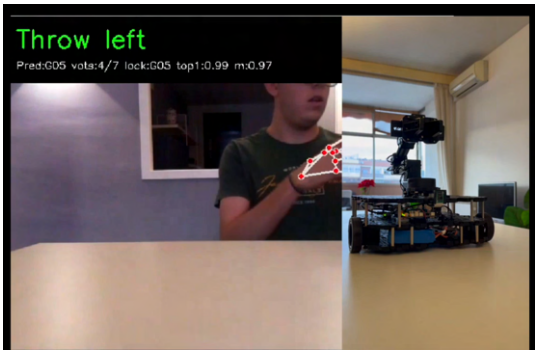
6.3.1.4 Gesture: Throw left

Mapped as `G05` (ID 4), it is used to change the orientation of the robot to the left (see Figure 6.23).

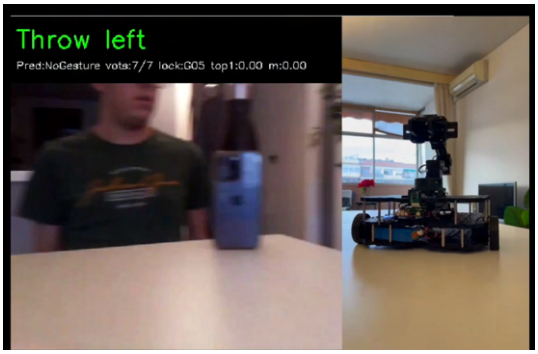
- **Parameters:** Angular velocity of 0.8 rad/s for 2.0 seconds (`t_turn_90`).



Frame 1



Frame 2



Frame 3

Frame 4

Figure 6.23: Integrated execution of *Throw left*: lateral gesture and approximately 90° left rotation.

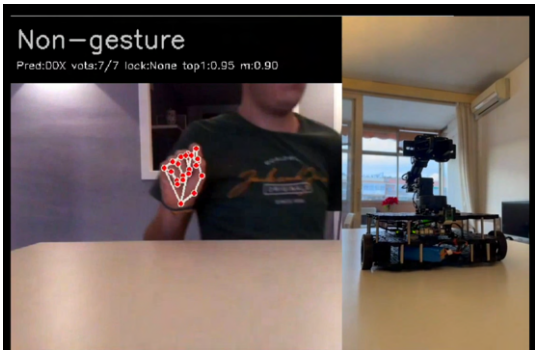
All throw gestures are very similar; in this case, the *Throw left* gesture is presented. This gesture starts in the first frame with the hand stopped, without having started either the gesture or its detection. It is already in the second frame when the complete *Throw left* gesture is detected and the robot begins to rotate slightly to the left.

In the third and fourth frames, it can be observed how the action is completed. In this case, however, there is a particular detail worth noting. If we look at the previous gestures, once the gesture is detected, the classification usually returns to the *Non-gesture* state. This is not the case here, but for a reasonable reason. The robot does not have time to observe a *Non-gesture* state because the hand leaves the field of view immediately after the *Throw left* gesture ends and remains outside the camera view until the end of the movement. Only when the hand reappears will the system classify it again as *Non-gesture*.

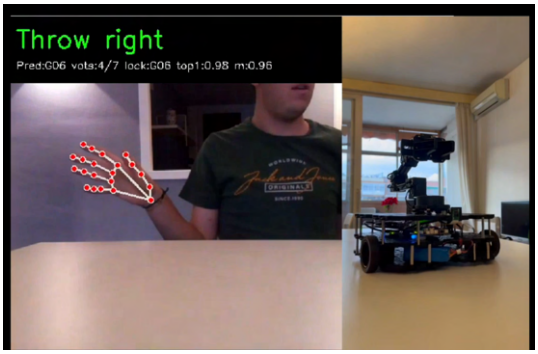
6.3.1.5 Gesture: Throw right

Mapped as `G06` (ID 10), it is used to change the orientation of the robot to the right (see Figure 6.24).

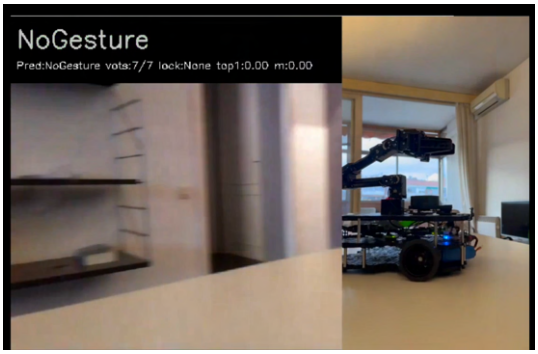
- **Parameters:** Angular velocity of 0.8 rad/s for 2.0 seconds (`t_turn_90`).



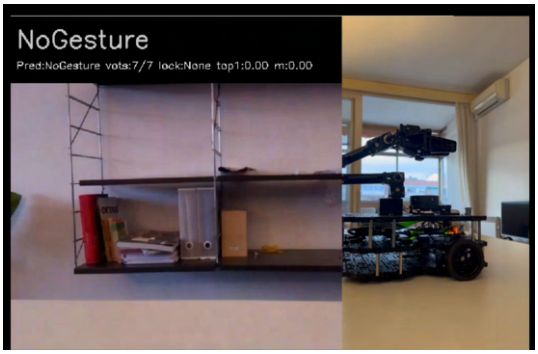
Frame 1



Frame 2



Frame 3



Frame 4

Figure 6.24: Integrated execution of *Throw right*: lateral gesture and approximately 90° right rotation.

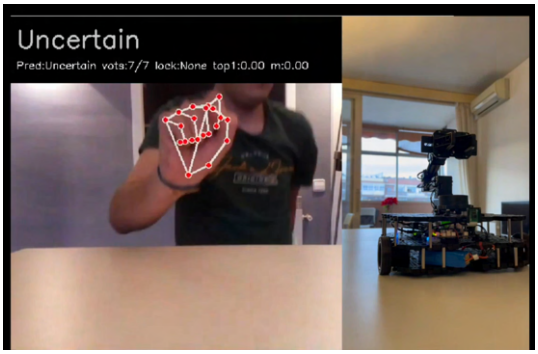
Here, the *Throw right* gesture is presented, which is the same than the previous example but to the other side. This gesture starts in the first frame with the hand stopped, without having started either the gesture or its detection. It is in the second frame when the complete *Throw right* gesture is detected and the robot begins to rotate slightly to the right.

In the third and fourth frames, it can be observed how the action is completed. In this case, however, the behavior is slightly different from the previous example. Once the gesture is detected, the classification returns to the *Non-gesture* state, since the hand is no longer performing the gesture and the system can still observe it briefly. This is why, in **frames 3 and 4**, the system already shows *Non-gesture*, while the robot continues executing the temporized rotation until the movement ends.

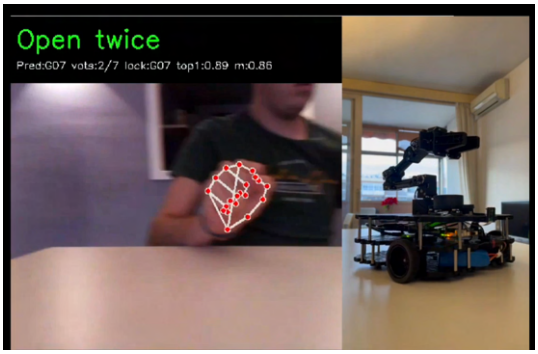
6.3.1.6 Gesture: Pointing with one finger (Repeat action)

The gesture `BOA` (ID 1) allows repeating the last valid command executed.

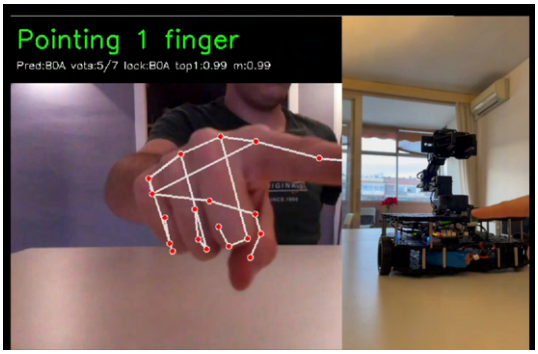
- **"One-shot" logic:** A *latch* has been implemented to avoid indefinite repetitions while the gesture remains active and also to be able to repeat the last gesture with a predefined repetition gesture. In this way, the immediately previous gesture can only be repeated with the *Pointing with one finger* gesture and does not work if you try to do the same gesture twice in a row.



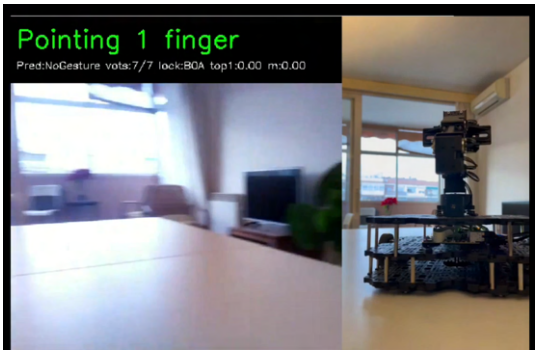
Frame 1



Frame 2



Frame 3



Frame 4

Figure 6.25: Integrated execution of *Pointing with one finger*: gesture detection and repetition of the last valid action.

As explained above, the *Pointing with one finger* gesture allows the robot to repeat the last valid command that it was able to execute. That is, in this case, no new action is generated, but the most recent internal action is repeated, which can correspond to any action within the entire set of implemented commands.

By analyzing the sequence in Figure 6.25 frame by frame, we can observe how, once again, in **frame 1**, the gesture has not yet started and the classifier maintains the *Non-gesture* state.

In **frame 2**, an *Open twice* gesture appears (which will later be repeated). This initial gesture is intentionally shown in the sequence so that the behavior of the repetition mechanism becomes clear.

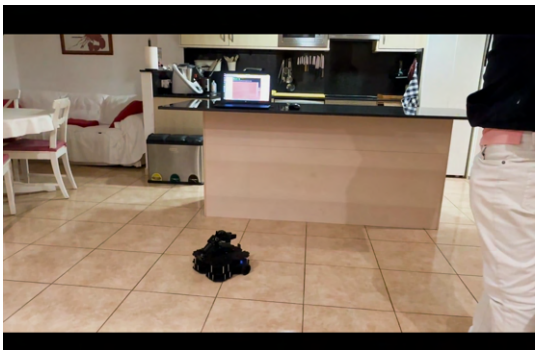
It is in **frame 3** when the *Pointing with one finger* gesture is finally detected in a clear and stable manner. At this moment, the control node retrieves the value of `last_action_id` and executes the last valid action again, as can be clearly observed in **frame 4**, where the robot performs a rotation, corresponding to the previously executed *Open twice* action.

In addition, thanks to the *latch* mechanism, the action is executed only once per gesture activation, preventing indefinite repetitions while the user maintains the gesture.

6.3.1.7 Gesture: Zoom in (Following wall on the right)

- **Logic:** The robot enters a state-based behavior and uses LiDAR to keep a wall on its right whenever possible.
- **Timeout:** For safety reasons, a deadline of 30 seconds (`follow_timeout_s`) has been implemented.

Frame 1



Frame 2

Frame 3

Frame 4

Figure 6.26: Integrated execution of *Zoom in*: gesture activation and wall-following behavior using LiDAR feedback.

The case corresponding to the *Zoom in* gesture, represented in Figure 6.26, is described below.

In this experiment, the *Zoom in* gesture activates the wall-following behavior, with the condition that the robot tries to keep a wall on its right whenever possible using LiDAR information. As can be observed throughout the four frames, the gesture is executed correctly and the system enters the autonomous navigation mode associated with this behavior.

In **frame 1**, the robot is in its initial state after the gesture is detected, while the user adopts a low position to facilitate the visibility of the robot's movement during the recording. In **frames 2, 3, and 4**, it can be observed how the robot moves through the environment while trying to identify a wall located on its right. However, in this specific scenario, the robot does not find a clear wall on the right

at any moment.

As a result, the observed behavior is not wall following per se, but rather an active exploration phase, during which the robot continues attempting to locate a suitable wall until the maximum operating time of 30 seconds is reached.

It should be emphasized that this situation does not represent a system error, but instead reflects the expected behavior when the environmental conditions do not allow the wall-following criterion to be satisfied. This behavior can be contrasted with the following case of the *Zoom out* gesture, in which the robot does find a wall on the left and is able to maintain it throughout the execution time.

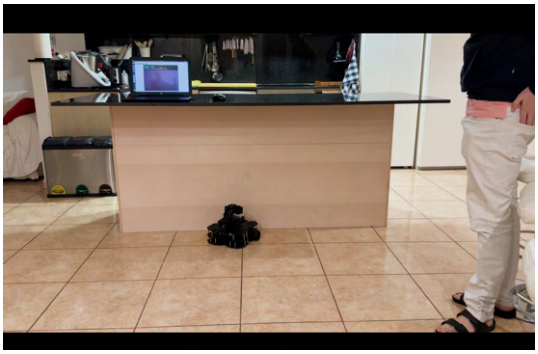
Finally, it should be noted that, unlike other experiments, a dual-screen format (gesture and robot view) has not been used in this sequence. This choice was made to prioritize a clear visualization of the robot's motion in a different real environment, thereby facilitating the analysis of its navigation behavior. Nevertheless, gesture detection can still be observed indirectly on the computer screen visible in the background, confirming that the vision system remains active and functional during the experiment.

6.3.1.8 Gesture: Zoom out (Following wall on the left)

- **Logic:** The robot enters a state-based behavior and uses LiDAR to leave a wall on the left whenever possible.
- **Timeout:** For safety, a deadline of 30 seconds (`follow_timeout_s`) has been implemented.

Frame 1

Frame 2



Frame 3

Frame 4

Frame 5

Frame 6

Figure 6.27: Integrated execution of *Zoom out*: gesture activation and the corresponding navigation state transition.

The case corresponding to the gesture *Zoom out*, represented in Figure 6.27, shows the activation of the autonomous navigation behavior with the inverse criterion to the previous case, that is, maintaining a wall or lateral surface to the left whenever possible using LiDAR information. Unlike the previous case, here the wall on the left is found quickly.

In frame 1, the robot is in its initial state just after the activation of the gesture, while the user adopts a low position to facilitate the recording of the robot's movement. In frame 2, the robot begins to

move and adjusts its orientation to identify a suitable surface on the left. In frame 3, this behavior is already stable and the robot clearly starts lateral tracking.

From frames 4, 5, and 6, it can be observed how the robot continues its trajectory while maintaining the lateral surface on the left, describing a contouring movement around a fixed element of the environment, which effectively acts as a wall.

In this case, six frames have been included instead of the usual four, with the aim of showing more clearly the evolution of the behavior and the continuity of lateral tracking throughout the trajectory, during the 30 seconds of execution.

Similarly to the previous case, a dual-screen format (gesture + robot) has not been used. This decision was made to prioritize the visibility of the robot's movement in a different real domestic environment, thus facilitating the analysis of the navigation behavior. However, once again, gesture detection can be observed indirectly on the computer screen visible in the background of the scene, showing the correct functioning of the vision system during the experiment.

6.3.2 Performance and stability analysis

This section describes the evaluation of the system's performance and stability under real-world conditions. Key factors such as response latency over the WiFi network, the accuracy of the neural network's trust thresholds, and the robustness of the data flow are reviewed to ensure secure control of the TurtleBot3.

6.3.2.1 Terminal output during execution

In order to increase the explainability of the system, the control node has been configured to continuously print its output and the decisions taken to the terminal. This output makes it possible to verify whether the sequence *gesture detection* → *command publication* → *physical execution* is executed correctly. An example of the terminal output corresponding to the *Zoom in* gesture is shown in Figure 6.28.

```
[INFO] [1767888347.046950266] [control_node]: FOLLOW RIGHT START (30s max)
[INFO] [1767888347.062367933] [control_node]: FOLLOW[right] st=IDLE front=0.76 L=0.75(0) R=inf(0) t_left=30.0
[INFO] [1767888347.561270670] [control_node]: FOLLOW[right] st=IDLE front=0.75 L=0.75(0) R=inf(0) t_left=29.5
[INFO] [1767888348.109472369] [control_node]: FOLLOW[right] st=IDLE front=0.76 L=0.75(0) R=inf(0) t_left=28.9
[INFO] [1767888348.613751383] [control_node]: FOLLOW[right] st=IDLE front=1.15 L=0.75(0) R=inf(0) t_left=28.4
[INFO] [1767888349.159149621] [control_node]: FOLLOW[right] st=IDLE front=1.06 L=0.75(0) R=inf(0) t_left=27.9
[INFO] [1767888349.708782766] [control_node]: FOLLOW[right] st=IDLE front=0.97 L=0.77(0) R=inf(0) t_left=27.3
[INFO] [1767888350.209675717] [control_node]: FOLLOW[right] st=IDLE front=0.91 L=0.82(0) R=inf(0) t_left=26.8
[INFO] [1767888350.758701121] [control_node]: FOLLOW[right] st=IDLE front=0.86 L=0.77(0) R=inf(0) t_left=26.3
[INFO] [1767888351.259004996] [control_node]: FOLLOW[right] st=IDLE front=0.77 L=0.69(0) R=inf(0) t_left=25.8
[INFO] [1767888351.809365135] [control_node]: FOLLOW[right] st=IDLE front=0.69 L=0.62(0) R=inf(0) t_left=25.2
[INFO] [1767888352.359888714] [control_node]: FOLLOW[right] st=IDLE front=0.60 L=0.55(0) R=inf(0) t_left=24.7
[INFO] [1767888352.908717409] [control_node]: FOLLOW[right] st=IDLE front=0.52 L=0.47(1) R=inf(0) t_left=24.1
[INFO] [1767888353.457813849] [control_node]: FOLLOW[right] st=IDLE front=0.43 L=0.38(1) R=inf(0) t_left=23.6
[INFO] [1767888353.960572664] [control_node]: FOLLOW[right] st=IDLE front=0.36 L=0.33(1) R=inf(0) t_left=23.1
[INFO] [1767888354.508332556] [control_node]: FOLLOW[right] st=TURNING front=0.28 L=0.26(1) R=inf(0) t_left=22.5
[INFO] [1767888355.008935850] [control_node]: FOLLOW[right] st=TURNING front=0.22 L=0.20(1) R=inf(0) t_left=22.0
[INFO] [1767888355.511168350] [control_node]: FOLLOW[right] st=TURNING front=0.26 L=0.18(1) R=inf(0) t_left=21.5
[INFO] [1767888356.059025077] [control_node]: FOLLOW[right] st=RECOVER front=0.36 L=0.18(1) R=inf(0) t_left=21.0
[INFO] [1767888356.564000141] [control_node]: FOLLOW[right] st=RECOVER front=0.66 L=0.17(1) R=inf(0) t_left=20.5
[INFO] [1767888357.108334782] [control_node]: FOLLOW[right] st=RECOVER front=2.30 L=0.18(1) R=inf(0) t_left=19.9
[INFO] [1767888357.617325730] [control_node]: FOLLOW[right] st=RECOVER front=2.64 L=0.20(1) R=inf(0) t_left=19.4
[INFO] [1767888358.157741730] [control_node]: FOLLOW[right] st=RECOVER front=2.61 L=0.23(1) R=inf(0) t_left=18.9
[INFO] [1767888358.661122664] [control_node]: FOLLOW[right] st=RECOVER front=2.62 L=0.32(1) R=inf(0) t_left=18.4
```

Figure 6.28: Output from the terminal during the execution of a gesture, showing the activated action and the internal state of the control node in real time.

The terminal output follows a structured and consistent format that allows the internal state of the system to be understood at all times. Each line includes the node name (`control_node`) and the currently active operating mode. In the case of autonomous behaviors, like the one of the example, the state of the finite state machine is explicitly shown, together with the distances measured by the LiDAR in the frontal and lateral sectors and time left.

This mechanism is particularly useful both for debugging the system and for justifying the robot's observed behavior during the experiments.

6.3.2.2 Confidence thresholds and locking mechanism

The tests performed with the robot confirm that the locking system implemented in the `VisionNode` works correctly to guarantee a stable interaction. In addition, the use of different confidence thresholds for finger gestures and for the *Non-gesture* class helps to reduce visual noise and spurious detections. It can be clearly observed that, once a gesture is recognized, the system blocks further detections by setting a `locked_code`. This mechanism allows the user to remove the hand from the field of view without losing the command. The lock is released only after five consecutive detections of the *Non-gesture* class.

6.3.2.3 Latency and QoS over WiFi

During the experimentation, a very low latency has been observed between the execution of the gesture and the start of the robot's movement. This behavior is mainly achieved thanks to the use of compressed image streams and the publication of motion commands using the `TwistStamped` interface. Under normal operating conditions, the measured latency remains within the range of 200 to 300 ms.

6.3.3 Conclusions of Phase III

The results of this phase show that the integrated system is robust and fully functional in real environments. The combination of vision-based gesture recognition with LiDAR-based control logic enables smooth and safe robot motions, including more complex behaviors such as wall following. Overall, the system provides an intuitive and reliable human–robot interaction interface suitable for real-time, vision-based teleoperation.

Chapter 7

Conclusions and future work

This chapter attempts to summarize and collect in a single section the reflections and conclusions drawn from the development of this project. In addition, the results obtained are summarized, an analysis of the system's limitations is presented, and future improvements are proposed that will allow for further advances in the field of human–robot interaction (HRI).

7.1 Summary of the work

To summarize the work, the central objective of this project was to improve human-robot interaction and minimize the difficulties posed by traditional physical controllers for all those people who are not experts or are not used to working with robots.

The entire project report and implementation describes the design and implementation of a complete system on `ROS 2 Jazzy Jalisco` that allows teleoperating a TurtleBot3 Waffle Pi based on everyday human-gestures.

In addition, the work has followed an incremental development. It started with a very basic heuristic model that turned out to be insufficient to ensure good teleoperation in the face of the variability of gestures between different people. For this reason, a new detection system based on a deep learning model `EurekaNet` was proposed.

To sum up, we have ended up obtaining a model that, through the use of `MediaPipe` for the extraction of landmarks and specialized training with the `IPN Hand` dataset, is able to correctly recognize the vast majority of gestures in real time with very low latency. This then allows us to map the detections and convert them into real gestures of the robot, which are combined with the data from the LiDAR sensor to control the robot.

7.2 Main achievements

Although all the general contributions have been listed in the introduction, it is important to highlight the real value that each of the implemented solutions brings:

- **Robustness in real environments:** The first point is the robustness of the MLP in real environments. It achieves accuracy figures higher than 90% in both the training and inference phases.
- **Lightweight model:** The fact that the neural network is lightweight allows inference to be performed on a standard computer without the need for powerful GPUs.
- **Modernization of robot software:** In the robotics subjects of the degree, robots based on `ROS1` were used. Therefore, updating the robot to `ROS 2 Jazzy Jalisco` (the most recent distribution at the time of development) ensures that the project is up to date.
- **Batch approach:** The strategy of using 30, 60, and 90-frame time windows is important to address the problem of indefinite gesture duration and allows continuous detection without the need for manual intervention.
- **End-to-end real-time system:** The developed solution operates as a complete end-to-end system, integrating perception, gesture recognition, decision logic, and robot control in real time to enable intuitive and continuous teleoperation.
- **Safety prioritization:** A conservative solution has been prioritized, and control and safety mechanisms such as the use of LiDAR and an emergency stop gesture have been implemented.

7.3 Limitations of the current system

Despite achieving very good results, a complete analysis of the experiments and their errors has revealed several considerations that must be taken into account. In addition, incompatibilities between the firmware of the OpenManipulator-X arm and this version of ROS 2 also represent a limitation with respect to the initial objective.

First of all, it is necessary to consider light sensitivity. In general, even in low-light environments, the system works correctly. The problem arises in situations of extreme backlighting, which causes the system to lose finger definition.

In addition, the confusion matrix has revealed some errors. In particular, the *throw down* gesture presents a higher error rate than the other gestures, as it requires a sufficiently large motion path, and the vertically positioned camera sometimes does not provide enough space for this movement.

Although it has not been observed in any case, it is also necessary to consider that network instability could increase latency and cause the robot to temporarily lose control or even stop responding altogether. This is because the robot connects to the remote PC running the nodes via WiFi.

It is also clear that the system is not capable of detecting multiple hands simultaneously and always selects the most confident one. For this reason, it must be clear at all times which hand is intended to control the robot when more than one hand is present in the scene.

Finally, as mentioned at the beginning, one of the main limitations has been the OpenManipulator-X robotic arm. Despite significant effort and the initial intention to incorporate it and teleoperate it together with the robot, the lack of a firmware update fully compatible with this version of ROS has prevented its use.

7.4 Future improvements and extensions

Based on the experience acquired, the following future lines of work are proposed to further evolve the project:

- **OpenManipulator-X arm integration:** Once the arm can be correctly installed and becomes fully functional, extend the system to enable its teleoperation as well.
- **Multimodal integration (voice and vision):** Combine gesture recognition with voice commands to make the interaction more natural and comprehensive.
- **Point-to-go navigation via gestures:** Implement the ability for the user to point to a target location, allowing the robot to navigate to it while avoiding obstacles.
- **Model integration to improve explainability:** Integrate an explainability-oriented model so that the robot can provide feedback on its actions or detections, potentially through an integrated large language model (LLM).

Bibliography

[1] A. Bazarevsky, V. Bazarevsky, and S. Tsai, "BlazePalm: Real-time Hand Detection and Tracking," *Google Research*, 2019.

[2] G. Benitez-Garcia, J. Olivares-Mercado, G. Sanchez-Perez, and K. Yanai, "IPN Hand: A Video Dataset and Benchmark for Real-Time Continuous Hand Gesture Recognition," *arXiv preprint arXiv:2005.02134*, 2020. [Online]. Available: `https://arxiv.org/abs/2005.02134`

[3] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, vol. 25, pp. 120–125, 2000.

[4] Z. Cao, G. Hidalgo, T. Simon, S. E. Wei, and Y. Sheikh, "OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 1, pp. 172-186, 2021. [Online]. Available: `https://arxiv.org/abs/1812.08008`

[5] J. Cassell, "Gesture in Human-Robot Interaction," *The Cambridge Handbook of Multimedia Learning*, Cambridge University Press, 2014.

[6] A. Dosovitskiy et al., "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," *ICLR*, 2021. [Online]. Available: `https://arxiv.org/abs/2010.11929`

[7] European Data Protection Board, "Guidelines on data minimisation," 2020. [Online]. Available: `https://edpb.europa.eu/our-work-tools/our-documents/guidelines_en`

[8] European Union, "Regulation (EU) 2024/1689 of the European Parliament and of the Council laying down harmonised rules on artificial intelligence (Artificial Intelligence Act)," Official Journal of the European Union, 2024. [Online]. Available: `https://eur-lex.europa.eu/eli/reg/2024/1689/oj`

[9] European Union, "Regulation (EU) 2016/679 of the European Parliament and of the Council (General Data Protection Regulation)," Official Journal of the European Union, 2016. [Online]. Available: `https://eur-lex.europa.eu/eli/reg/2016/679/oj`

[10] J. Han, L. Shao, D. Xu, and J. Shotton, "Enhanced Computer Vision with Microsoft Kinect Sensor: A Review," *IEEE Transactions on Cybernetics*, vol. 43, no. 5, pp. 1318–1334, 2013. [Online]. Available: `https://doi.org/10.1109/TCYB.2013.2265378`

[11] International Organization for Standardization, "ISO 13482:2014 — Robots and robotic devices — Safety requirements for personal care robots," ISO, 2014. [Online]. Available: `https://www.iso.org/standard/53820.html`

[12] A. Kendon, M. Argyle, and A. C. Wilson, "Vision-Based Human–Robot Interaction: A Survey," *Robotics and Autonomous Systems*, vol. 100, pp. 1–18, 2018.

[13] C. Li et al., "YOLOv8: Real-Time Geometry-Informed Keypoint Detection," *Ultralytics*, 2023. [Online]. Available: `https://docs.ultralytics.com/tasks/pose/`

[14] J. Lin et al., "Hand Gesture Recognition via 2D Joint Landmarks and Temporal Logic," *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021.

[15] H. Liu, L. Wang, and M. Q.-H. Meng, "Hand and Arm Gesture-based Human-Robot Interaction: A Review," *IEEE Transactions on Automation Science and Engineering*, 2022. [Online]. Available: `https://arxiv.org/abs/2209.08229`

[16] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot Operating System 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, 2022.

[17] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot Operating System 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, 2022. [Online]. Available: `https://www.science.org/doi/10.1126/scirobotics.abm6074`

[18] MediaPipe, "Hands - MediaPipe Solutions Guide," 2024. [Online]. Available: `https://mediapipe.readthedocs.io/en/latest/solutions/hands.html`

[19] S. Mitra and T. Acharya, "Gesture Recognition: A Survey," *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, vol. 37, no. 3, pp. 311–324, 2007.

[20] H. Molchanov, S. Gupta, K. Kim, and J. Kautz, "Hand Gesture Recognition with 3D Convolutional Neural Networks," in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2015, pp. 1–7.

[21] Open Robotics, "ROS 2 Jazzy Jalisco Documentation," 2024. [Online]. Available: `https://docs.ros.org/en/jazzy/`

[22] A. Paszke *et al.*, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*, Vancouver, Canada, 2019, pp. 8024–8035. [Online]. Available: `https://arxiv.org/abs/1912.01703`

[23] M. Peral, A. Sanfeliu, and A. Garrell, "Efficient Hand Gesture Recognition for Human-Robot Interaction," *IEEE Robotics and Automation Letters*, 2022.

[24] S. S. Rautaray and A. Agrawal, "Vision based hand gesture recognition for human computer interaction: a survey," *Artificial Intelligence Review*, vol. 43, pp. 1–54, 2015. [Online]. Available: `https://doi.org/10.1007/s10462-012-9356-9`

[25] ROBOTIS, "OpenManipulator-X e-Manual," 2024. [Online]. Available: `https://emanual.robotis.com/docs/en/arm/openmanipulator_x/overview/`

[26] ROBOTIS, "OpenManipulator-X Quick Start Guide for ROS 2 Jazzy," 2024. [Online]. Available: `https://emanual.robotis.com/docs/en/platform/openmanipulator_x/quick_start_guide/`

[27] ROBOTIS, "TurtleBot3 Waffle Pi e-Manual," 2024. [Online]. Available: `https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/`

[28] X. Shi et al., "Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting," *Advances in Neural Information Processing Systems*, 2015.

[29] Ultraleap, "Getting Started with Ultraleap Hand Tracking," 2024. [Online]. Available: `https://docs.ultraleap.com/hand-tracking/getting-started.html`

[30] Ultralytics, "What is OpenPose? Exploring a Milestone in Pose Estimation," 2024. [Online]. Available: `https://www.ultralytics.com/es/blog/what-is-openpose-exploring-a-milestone-in-pose-estimation`

[31] Ultralytics, "YOLO11 Pose: Real-Time Pose Estimation with Ultralytics YOLO," 2024. [Online]. Available: `https://docs.ultralytics.com/tasks/pose/`

[32] United Nations, "Sustainable Development Goals," [Online]. Available: `https://sdgs.un.org/goals`

[33] F. Zhang, V. Bazarevsky, A. Vakunov, A. Tkachenka, G. Sung, C.L. Chang, and M. Grundmann, "MediaPipe Hands: On-device Real-time Hand Tracking," *arXiv preprint arXiv:2006.10214*, 2020. [Online]. Available: `https://arxiv.org/abs/2006.10214`

Appendix A

Work plan

This project has been carried out over about 4-5 months, following the usual timeline of a TFG. In this case, the work has been divided into four main parts:

1. **Setup and state of the art:** First, `ROS 2 Jazzy Jalisco` was installed on the Raspberry Pi and the `Docker/Rocker` environment was configured on the Remote PC. Also, the documentation for the TurtleBot3 and MediaPipe was studied.
2. **Setting up the robot:** All the necessary packages were installed and the migration from ROS 1 to ROS 2 Jazzy Jalisco was finished, as well as creating the workspace and compiling all the new installations.
3. **Initial prototyping:** Second, a basic vision node based on geometry was implemented to understand the difficulties of the problem. In addition, the basic control logic was developed.
4. **Deep learning integration:** The heuristic logic was replaced by the EUREKA architecture. This phase included collecting the dataset, training and integrating the model.
5. **Control node creation:** The control node was created to be able to translate the commands of the vision node into actions of the robot.
6. **Validation and writing:** Finally, real experiments were done to check the performance of the system and this document was written.

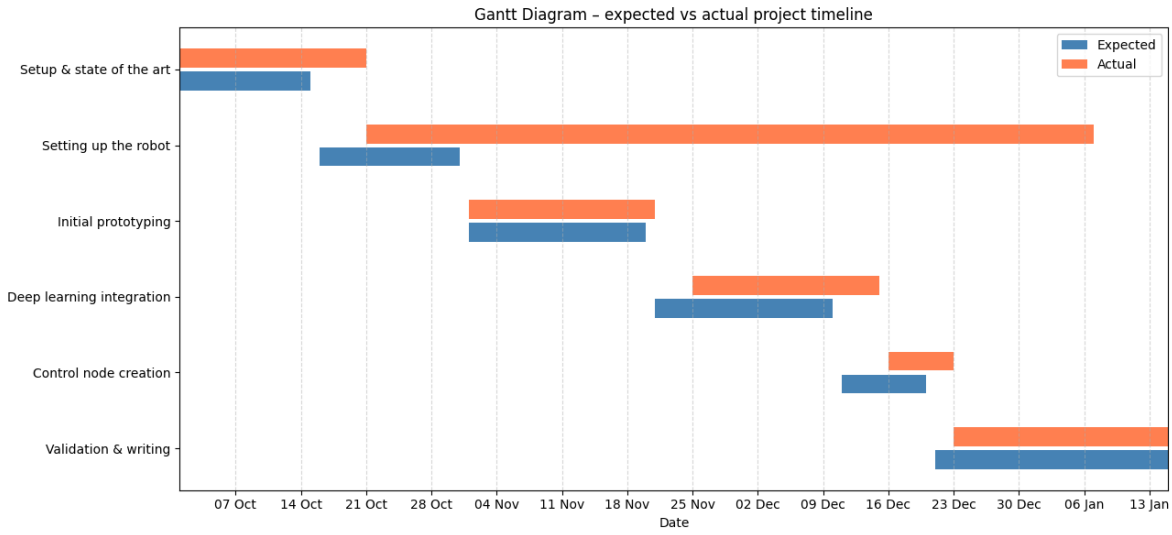


Figure A.1: Comparison between the expected and actual project timeline using a Gantt diagram (see Figure A.1).

You can clearly see that the actual Gantt fits quite well with the expected plan, except for the "Setting up the robot" phase. This is because the initial plan for this section was to install and compile all the packages at once and leave it as is, but this phase turned out to be the one that caused the most issues during the project.

Initially, the robotic arm was intended to be installed, but incompatibilities between the firmware, hardware, and `ROS 2 Jazzy Jalisco` on the TurtleBot3 Waffle Pi and the OpenManipulator-X made this integration impossible. In addition, delays during compilation and installation prevented this phase from following the originally proposed Gantt schedule.

Appendix B

Cost analysis and economic viability

In projects such as this one, related to mobile robotics and computer vision, where costs can be significant, it is essential to establish an economic viability plan detailing the costs associated with materials and project development, divided into fixed assets and human resources (see Table B.1).

B.1 Hardware and equipment

Concept	**Units**	**Estimated Cost (€)**
TurtleBot3 Waffle Pi (incl. LiDAR/RPi4)	1	2,050.00
Raspberry Pi Camera Module V2	1	35.00
Remote PC (Apple M2 – Training/Inference)	1	1,800.00
Network infrastructure (Router/WiFi)	1	100.00
Total hardware		**3,985.00**

Table B.1: Estimated hardware and equipment costs for the project.

B.2 Human resources

Considering a total dedication of 540 hours (corresponding to 30 hours for each of the 18 ECTS credits of the TFG) with an estimated salary for a junior engineer in training of 20 €/hour:

Personnel cost: 540 hours × 20 €/hour = 10,800.00 €.

B.3 Total project budget

The total cost of the project is estimated at **14,785.00 €**. It should be noted that most of the hardware cost has been covered by the Universitat Politècnica de Catalunya (UPC) and the Institute of Industrial and Control Engineering (IOC), making the project economically viable.

Appendix C

Environmental impact and sustainability considerations

This chapter analyzes the ecological impact and energy consumption derived from the development and execution of this project. In the current context of the global climate crisis, and considering the environmental challenges that cities like Barcelona, or even the entire European Union, are facing, the integration of sustainability criteria is a fundamental ethical responsibility in any engineering project such as this. The specific SDGs related to this work are shown in Figure C.1.

C.1 SDG alignment (Sustainable Development Goals)

This project aligns with the following United Nations Sustainable Development Goals (SDGs) [32]:

- **SDG 5 (Gender Equality):** By designing a gender-neutral gesture-based control interface that has involved both men and women in system training and experimentation, and avoids access barriers associated with physical strength, technical training, or traditional stereotypes in the use of technology.
- **SDG 9 (Industry, Innovation and Infrastructure):** By developing new low-cost human–machine interaction interfaces applied to mobile robotics.
- **SDG 10 (Reduced Inequalities):** By creating intuitive control systems that do not require prior technical knowledge, facilitating the use of robotics for elderly people or individuals with limitations.
- **SDG 12 (Responsible Production and Consumption):** By promoting the reuse and extension of the useful life of existing hardware (circular economy) and avoiding the use of specialized sensors or additional devices, thus reducing the need to acquire new equipment.

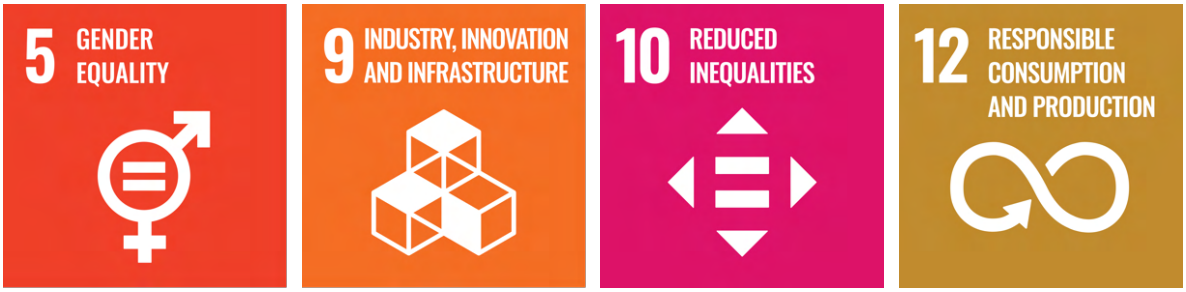


Figure C.1: Sustainable Development Goals (SDGs) of this project.

C.2 Hardware efficiency and circular economy

C.2.1 Power usage of the TurtleBot3 platform

From an energy point of view, the TurtleBot3 is a robot designed to be lightweight and to have fairly low power consumption, since it is intended to operate with a 1800 mAh battery in teaching and research environments. In this robot, the energy expenditure is mainly associated with:

- the network connection with the remote PC,
- the mobile base (Raspberry Pi, OpenCR, motors, and controllers),
- sensors (such as the camera and LiDAR),
- and the OpenManipulator-X arm (if used).

This energy is invested in obtaining a specific experimental functionality (natural teleoperation by gestures) rather than in executing intensive on-board computation. Therefore, the sustainability of this system depends less on the robot itself and more on how the overall system is organized (using a remote PC).

C.2.2 Reusing and extending existing hardware

A factor that has been taken into account during the development of the project, and that directly affects sustainability, is the reuse of a TurtleBot3 and part of the hardware from other TurtleBots to fix components that were not working.

Instead of designing a robot from scratch, we worked with a commercial laboratory robot and installed new software, updating the entire robot to `ROS 2 Jazzy Jalisco`. This component recovery practice is part of the principles of the Circular Economy, extending the useful life of existing hardware and reducing the generation of electronic waste. This has avoided the environmental costs associated with manufacturing and acquiring new equipment, and focuses the effort on improving the usefulness of an already available system.

In addition, the project avoids specialized sensors or wearables and is based exclusively on the hardware that the robot already incorporates (such as the standard RGB camera). This approach favors a more accessible and replicable design.

C.3 Sustainable software development lifecycle

C.3.1 Training the Eureka model and "Green AI"

Unlike other components of the project that use pre-trained models such as MediaPipe, the core of the vision system has ultimately been based on training the *Eureka* model from scratch. This process represents the phase with the greatest environmental impact of the entire project, since it requires the use of GPUs throughout the entire training phase.

Training reinforcement learning algorithms (such as Eureka) involves both high computational and, consequently, energy consumption. In an attempt to reduce the footprint and training time, the MacBook accelerator `Metal Performance Shaders (MPS)` has been used, which has made it possible to speed up the training process.

It should also be noted that the fact of having previously identified the best hyperparameters and network configuration, as a result of the Eureka research [23], has made it possible to significantly reduce the training time and, therefore, the resources used.

C.3.2 Software optimization and data efficiency

Once the training phase is complete, the project focuses on the efficiency of real-time execution. The use of `ROS 2 Jazzy Jalisco` allows for more efficient communication management thanks to the `Data Distribution Service` protocol, which optimizes bandwidth and reduces the processing load during message transmission between the PC and the TurtleBot3.

In addition, good practices have been applied to reduce computational consumption during testing, such as:

- **Asynchronous processing:** The artificial vision and motor control tasks are executed independently, avoiding bottlenecks that would force an unnecessary increase in processor clock frequencies.
- **Data filtering at the source:** Instead of sending the entire video stream at high resolution, only the essential *landmarks* are processed, drastically reducing the volume of data to be transmitted and processed per second.

C.4 Circular economy and project legacy

The project has been designed following principles of digital sustainability. The resulting code and models are delivered in a modular and well-documented manner, making it easier for any user who wishes to consult or reuse them to do so without having to repeat the processes from scratch. This is essential from a sustainability perspective, as it avoids repeated training and, therefore, unnecessary energy consumption.

Appendix D

Ethical and legal considerations

The quick evolution of robotics and computer vision requires an extensive analysis that goes beyond purely technical aspects. An examination of the ethical, legal, safety, and sustainability implications of the project is therefore necessary. For this reason, this chapter analyzes the impact of the project from a social, ecological, and responsibility-oriented perspective.

D.1 Legal framework and compliance

This project has been designed to comply with the current regulatory framework of the European Union, with particular attention to the `EU AI Act` [8]. Given the nature of the system—a gesture-based teleoperation system for a robot developed within the scope of an academic research project—it is expected to fall under the "minimum risk" category defined by this regulation. As such, it complies with the applicable transparency requirements, including informing users about the system's operation and potential risks, as well as providing indicators that warn when the camera is active in real time.

It is also important to highlight that the project complies with data processing requirements under the `General Data Protection Regulation (GDPR)` [9], particularly by applying the principle of data minimization [7]. The system does not process full images; instead, only the 21 hand landmarks are extracted, and any additional information captured by the camera is immediately discarded. Furthermore, the system operates exclusively in real time and has no capability to store any data, either temporarily or permanently.

D.2 Ethical aspects of camera-based control

D.2.1 Perception of being recorded

One of the main privacy-related challenges associated with this work is the use of cameras, which is currently one of the most relevant ethical concerns in modern technological systems.

Although the purpose of the camera in this project is strictly limited to detecting hand gestures for teleoperating the TurtleBot, its presence may generate discomfort or concern among individuals in the environment due to perceived privacy risks.

For this reason, it is essential to clearly state that, within the scope of this TFG, the camera remains active while the robot is powered on but never stores the images it captures. All processing is performed in real time, and no visual data is recorded. In addition, whenever experiments are conducted, it is important to inform all individuals who may be present in the environment and potentially affected by the system.

Furthermore, to minimize the risk of privacy violations, only the information strictly necessary for the correct operation of the system is captured and processed. Since the project is designed to be developed in controlled environments, all tests and experiments have been carried out exclusively in the *IOC* laboratory and in private testing spaces.

Finally, it can be guaranteed that the system is explicitly designed to avoid handling unnecessary information. As described in previous sections, once MediaPipe detects the 21 hand landmarks, all other visual data is automatically discarded. This reinforces the idea that the system is designed to recognize hand gestures rather than to observe or identify people.

D.2.2 Informed use and transparency

A second key ethical aspect is transparency in system operation and data usage. In any demonstration or experiment, it is crucial that participants clearly understand and consent to how the system works, including what the camera captures, what data is processed, and the fact that no data is stored. Moreover, the purpose of the system is strictly academic, within the framework of a TFG.

These considerations are addressed through concrete measures such as clearly indicating when the camera is active, delimiting the experimental space, and explaining the system's operation and limitations in an accessible manner, even to individuals without advanced technical knowledge.

Transparency is also considered at the design level. Visual indicators are displayed when the vision and control nodes are active, ensuring that the user is always aware of the system's state and whether the camera is currently operating.

Finally, to further guarantee safety, additional mechanisms such as an emergency stop gesture and motor safety limits have been implemented, ensuring that the user cannot abruptly lose control of the TurtleBot.

D.3 Gender perspective and inclusivity

Today, gender and population diversity considerations are increasingly being taken into account across a wide range of projects, both within engineering and technology and beyond these fields. For this reason, when analysing projects from this perspective, it is very important to consider computer vision systems, as not considering these aspects may lead the resulting models to introduce significant diversity-related biases.

In this project, these criteria have been taken into account mainly in the design of the human-robot interaction system. Control through hand gestures has been conceived as a natural and intuitive interface, which does not require physical strength, prior technical knowledge, or specific physical characteristics of the user. In this way, the aim is to avoid a conception of robotic control traditionally

associated with masculinized or excessively technical profiles, promoting a gender-neutral user experience.

In addition, both in the model training phase and in the system testing and validation, efforts have been made to include the participation of men and women. This diversity contributes to reducing possible biases in the system and to improving its generalization capability in real environments. The selected gestures have been defined to be simple and easily reproducible, trying to avoid any complex movement that could be uncomfortable or exclusionary.

It should also be noted that the use of vision models based on landmarks of the hand, instead of complete images, reduces the risk of introducing visual biases associated with physical appearance or gender, as the system does not identify or classify people, but only movement patterns, which reinforces a respectful and inclusive approach.

Finally, this project aligns with the principles of equal opportunities and inclusion, indirectly contributing to reducing potential gaps in the access and use of robotic technologies. Although this is not a project specifically oriented towards social or demographic analysis, the design decisions adopted avoid exclusions and promote universal use of the system.

D.4 Privacy and data handling

In a computer-vision-based teleoperation system, privacy depends largely on how and where data is processed, as well as on the overall data management and preprocessing pipeline. In this project, an RGB video stream is captured by the camera, converted into 21 hand landmarks, used to classify the gesture, and finally translated into motion commands for the TurtleBot3.

From a data protection perspective, it can be guaranteed that all data processing is performed locally and that the system does not rely on external services, reducing both privacy risks and latency. However, local processing does not automatically ensure complete security, as part of the computation is performed on a remote PC connected to the robot via a network. To mitigate potential risks, the use of trusted and secure networks is recommended.

D.5 Safety in human–robot interaction

D.5.1 Risk scenarios

Gesture-based teleoperation of a mobile robot introduces specific risks arising from imperfect perception, potential model misinterpretations, and the physical movement of the robot. In this project, these risks can be grouped into four main categories.

First, gesture misclassification may occur, resulting in an incorrect interpretation of the user's intention and leading to unintended robot movements, such as moving forward instead of stopping or turning in the wrong direction. This type of error is particularly critical when the robot operates close to obstacles or people.

Second, there is a risk of false positives, where the system detects a gesture that was not intended as a control command. This may happen when the control nodes are active but the user is not intentionally

interacting with the robot, or when involuntary hand movements occur within the camera's field of view. For this reason, it is essential to stop the robot when it is not actively being used.

Third, latency and network instability can lead to a loss of control. In a distributed architecture where visual processing and command publication depend on a WiFi connection, delays or packet loss may cause commands to be received late or intermittently, resulting in unstable or unpredictable robot behavior.

Finally, environmental factors such as poor lighting conditions, partial hand occlusions, extreme distances from the camera, or visual noise can degrade the performance of the vision system, increasing detection errors and the likelihood of unintended movements.

D.5.2 Mitigation strategies

Although risks cannot be completely eliminated, they can be significantly reduced through appropriate mitigation strategies combining control design, motion constraints, and emergency mechanisms.

All implemented strategies are aligned with the principles of the `ISO 13482:2014` standard on safety for personal care robots [11], particularly regarding collision prevention and safe operator control.

First, emergency and safety-related commands are given absolute priority over all other commands. This conservative approach ensures that the system behaves safely even under uncertain conditions.

Second, temporal stabilization of predictions is applied to reduce false positives and rapid gesture fluctuations. Instead of reacting to every instantaneous prediction, the system verifies that a gesture remains consistent over a short time window and meets a confidence threshold, significantly reducing the probability of unintended actions.

Third, the system enforces kinematic and acceleration limits and integrates `LiDAR` data to reduce collision risks and prevent unsafe movements. The control node constrains robot speed and continuously monitors the surrounding environment.

Finally, to address network-related issues, a watchdog-based safety mechanism has been implemented. If the robot stops receiving commands for a predefined time interval, it automatically comes to a complete stop, preventing uncontrolled motion in the event of communication loss.