UNIVERSIDAD POLITÉCNICA DE MADRID


**GRADO EN INGENIERÍA DE COMPUTADORES**

PROYECTO DE FIN DE GRADO

# ARQUITECTURA SOFTWARE ADAPTABLE A DISTINTAS INTERFACES Y PROTOCOLO DE COMUNICACIONES SEGURO PARA EL CONTROL REMOTO DE UN ROBOT

AUTORA: VIOLETA TORIBIO RIVERA

TUTOR: JAVIER GARCÍA MARTÍN

MADRID, JUNIO 2025

RESUMEN

Este proyecto se centra en el desarrollo de un sistema de control remoto de robots mediante gestos humanos, capturados a través de sensores corporales. El sistema ofrece además comunicaciones seguras con el robot y se ha estructurado a partir de componentes de software individuales con el fin de ofrecer mayor flexibilidad y modularidad. También se ha diseñado para incorporar nuevos atajos de movimiento que complementan los comandos básicos.

El objetivo principal es permitir una interacción más natural, adaptable y segura para manejar robots en tiempo real. Para lograrlo, se diseñó una arquitectura de software flexible, capaz de integrarse con distintas interfaces de forma sencilla, y se implementó una comunicación segura entre dispositivos utilizando el protocolo Secure Sockets Layer (SSL). Este protocolo garantiza que los datos transmitidos estén cifrados, sean íntegros y que ambas partes —usuario y robot— estén correctamente autenticadas, asegurando una conexión exclusiva y fiable.

El sistema admite la incorporación de nuevos gestos y dispositivos de control de forma sencilla, lo cual facilita su escalabilidad futura. Esta capacidad de adaptación permite que el sistema evolucione y se integre fácilmente en distintos entornos y aplicaciones. Los resultados obtenidos hasta el momento demuestran un control funcional, seguro y adaptable del robot, con una respuesta adecuada a los gestos del usuario, aunque con margen de mejora en términos de precisión y fluidez en ciertos movimientos.

Esta área de investigación abre la puerta a diversas aplicaciones prácticas, como el fomento de la autosuficiencia en personas con movilidad reducida o la realización de tareas en entornos de alto riesgo para el ser humano, tales como operaciones de rescate en desastres naturales, trabajos en minería o exploración de superficies planetarias.

**Palabras clave:** Robot; Control remoto; Secure Sockets Layer (SSL); Movilidad reducida; Acelerómetros; Software escalable; Autómata.

ABSTRACT

This project focuses on the development of a secure remote-control system for robots through human gestures, captured by body-worn sensors. The system has been designed to include new movement shortcuts that complement the basic commands and is structured using individual software components to ensure greater flexibility and modularity.

The main objective is to enable a more natural, adaptable, and secure interaction for controlling robots in real time. To achieve this, a flexible software architecture was designed, capable of integrating with different interfaces in an effortless way, and secure communication between devices was implemented using the Secure Sockets Layer (SSL) protocol. This protocol ensures that data is transmitted encrypted, remains intact, and that both ends—user and robot—are properly authenticated, securing an exclusive and reliable connection.

The system supports the easy integration of new gestures and control devices, which facilitates its future scalability. This adaptability allows the system to evolve and be easily integrated into various environments and applications. The results obtained at the end demonstrate a functional, secure, and adaptable control of the robot, with an adequate response to user gestures, although there is room for improvement in terms of precision and fluidity in certain movements.

This field of study opens the door to various practical applications, such as promoting self-sufficiency for people with reduced mobility or carrying out tasks in high-risk environments for humans, including rescue operations during natural disasters, mining work, or the exploration of planetary surfaces.

**Key words**: Robot; Remote control; Secure Sockets Layer (SSL); Reduced mobility; Accelerometers; Scalable Software; Automaton.

## DEDICATORIA

*A Kika, Pepa y Dorita.*

*Por su cálida energía, que me sigue acompañando.*

Tabla de contenido:

Índice de figuras:

## Índice de tablas:



## Índice de ecuaciones

# Capítulo 1: Introducción

## 1.1. Motivación:

Los sistemas de control remoto y la robótica permiten la manipulación y el control del entorno físico a distancia. Este avance ha conseguido grandes logros en distintas áreas de investigación: como en el área militar, pudiendo estudiar entornos que resultan peligrosos para la interacción humana o en el área de la medicina, un sector que requiere de tanta asistencia que resultaría todo un reto el depender exclusivamente de la intervención humana.

El hecho de poder mandar instrucciones a una máquina con capacidades específicas desde la distancia facilita tanto tareas cotidianas como las no tan comunes para el ser humano. Las distintas posibilidades de combinación entre sensores de entrada y tipos de robots existentes en el mercado permiten gran flexibilidad a la hora de crear proyectos útiles y personalizados y con la comunicación a distancia se reduce la complejidad y tamaño final del sistema hardware donde, gracias a ello, se consigue abaratar costes y minimizar el impacto medioambiental para la obtención del producto final.

Actualmente, la tecnología de control remoto sobre robots de desplazamiento se ha implementado en diversos proyectos prácticos donde, muchos de ellos, sirven de sustitución a humanos en trabajos que ocasionan, frecuentemente, accidentes graves. Por ejemplo, se han creado robots tele-operados que permiten desactivar explosivos gracias a distintos sensores que interpretan el movimiento que harían unas manos humanas [1]. También, se han desarrollado robots que mapean minas en 3D de forma remota y segura [2], evitando la exposición humana, o robots que permiten el rescate de personas en zonas peligrosas como derrumbamientos o terrenos inestables [3].

Otro ejemplo de aplicación actual, pero referente al área médica, son las sillas de ruedas motorizadas. Ayudan a dotar de autonomía en el desplazamiento a las personas con dificultades motrices. Este proyecto se ha desarrollado múltiples veces, con distintos tipos de sensores y controles avanzados, en base a las limitaciones específicas que pueden presentar las personas.

Las sillas de ruedas tradicionales pueden resultar incómodas y limitantes para el usuario, siendo complicado su uso para según qué discapacidad. Con las sillas motorizadas [4] y controladas remotamente resulta sencillo incorporar nuevas características con distintos sensores que mejoren la funcionalidad y aprovechen los recursos de los microprocesadores. Se pueden adaptar a casos concretos y resulta beneficioso para la salud mental de los usuarios, ya que promueven la independencia de los mismos.

Fig. 1 Ejemplos de robots de asistencia en tareas humanas [3], [2], [1], [4]

## 1.2. Proyectos relacionados

Algunos proyectos recientes relativos al control de sillas de ruedas se describen a continuación:

A. En este proyecto [5], Abhay Kumar describe un sistema en el que se centra en la comodidad y bienestar del usuario, utilizando IoT o Internet-of-Things para monitorear la salud a través de varios sensores conectados a una Raspberry Pi 4 que gestiona el cómputo de los datos en tiempo real gracias a algoritmos basados en inteligencia artificial. Incorpora también una comunicación vía bluetooth entre esta y una aplicación móvil para el control del movimiento y gestión de alertas.
B. En el Proyecto [6], el Dr. N Krishna Chaitanya se centra en conseguir una conducción más segura incorporando sensores para la detección de obstáculos en el camino, acelerómetros para detectar caídas del usuario y un sensor de latidos. Todo queda conectado a un Arduino UNO, que a su vez manda los datos del sensor de salud a través de la ESP8266 Wi-Fi hasta una aplicación móvil para alertar al usuario en caso de necesidad.
C. En este Proyecto [7], utilizan la tecnología VR o de realidad Virtual para el control de una silla de ruedas. Utilizan un sistema cliente - servidor vía wifi para la comunicación entre el casco de VR y la Raspberry Pi 4 que comanda los motores e incorpora una cámara que representa la vista del usuario. También tiene sensores de infrarrojos que permite una visualización más nítida en entornos oscuros.

Los tres trabajos consiguen obtener mediciones precisas gracias a sensores o entradas que envían datos en tiempo real, comandan los datos con microprocesadores y son sistemas compactos. Todos ellos buscan la comodidad y facilidad de uso al usuario.

Los proyectos mencionados han utilizado diferentes combinaciones de tecnologías para mejorar la autonomía y seguridad de los usuarios. Sin embargo, cada uno presenta sus propias limitaciones frente al resto, como: la falta de control de impactos y caídas, en los proyectos A y C; la menor capacidad de procesamiento y no integración de medios de comunicación propios que presenta el Arduino UNO, usado en el proyecto B, frente a la Raspberry PI; o el alto costo de los materiales seleccionados en el proyecto C frente al resto.

El siguiente cuadro resumen (**Tabla 1**) muestra una comparativa entre los tres proyectos:

| **Caso** | **A** | **B** | **C** |
|---|---|---|---|
| **Sensores** | - Heartbeat<br>- Temperatura<br>- Oxígeno<br>- EGG | - Heartbeat<br>- Acelerómetro<br>- Ultrasonido<br>- Buzzer | - Cámara<br>- Infrarrojos |
| **Micro-controlador** | Raspberry PI 4 | Arduino UNO | Raspberry PI 4 |
| **Comunicación** | Bluetooth integrado en Raspberry | Wifi proporcionado por la ESP8266 | Wifi integrado en Raspberry |
| **Interfaz de usuario** | Joystick + Pantalla táctil | Aplicación móvil | Joystick + gafas VR |
| **Características** | - Monitoreo de salud<br>- Uso adaptado al usuario<br>- Sistema compacto<br>- Alerta de emergencia<br>- Monitoreo meteorológico | - Monitoreo de salud<br>- Uso adaptado al usuario<br>- Sistema compacto<br>- Alertas audibles<br>- Alerta ante caídas<br>- Control choques | - Sistema compacto<br>- Uso adaptado al usuario<br>- Interactivo para el usuario<br>- Adaptado al medio nocturno |

| | | | |
|---|---|---|---|
| | | - Notificaciones de salud por APP móvil | |

Tabla 1 Comparativa de varios proyectos A **[5]**, B **[6]** y C **[7]**

## 1.3. Objetivos del proyecto

El proyecto presente trata de controlar remotamente un robot, mediante gestos del usuario, a través de sensores corporales en la interfaz de entrada, como acelerómetros o cámara. El uso de sensores corporales como medio de control, consigue un sistema más adaptativo que las opciones de usar tablets, móviles o joysticks, que pueden acarrear cierta incomodidad o dificultad para las personas con distintos tipos de movilidad reducida.

El usuario puede realizar gestos con las manos, que serán capturados por los acelerómetros, o bien realizar movimientos con la cabeza, que se analizarán gracias a una cámara. Una vez obtenida la orden deseada por el usuario, será enviada al robot y procesada por el sistema. Una vez sea aclarada la orden, el sistema enviará las directrices a los motores mediante el protocolo GPIO (General-Purpose Input/Output). La comunicación se realiza en un esquema cliente servidor entre dos Raspberry Pi a través de protocolo TCP/IP y utiliza el protocolo de cifrado SSL (Secure Sockets Layer) para asegurar una comunicación exclusiva, íntegra y segura.

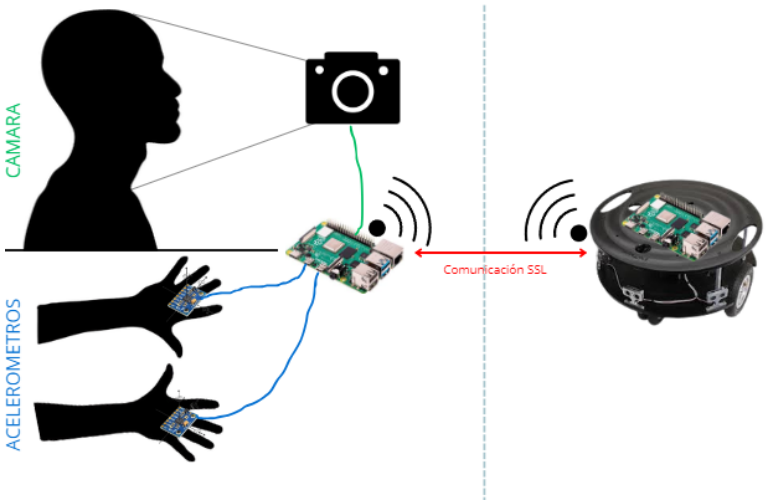



Fig. 2 Esquema simple del sistema

Este proyecto se basa en la implementación de nuevas funcionalidades y un enfoque más manejable del software implementado en el proyecto de fin de grado por Alba Payo Fernández [8].

Las nuevas funcionalidades que incorpora este sistema consisten en nuevos comandos de movimiento, tales como: giro de 180º en ambos sentidos, grabar un recorrido, reproducir un recorrido pregrabado y una parada de seguridad que anula cualquier orden hasta que el sistema reciba un comando específico. Estos atajos se complementan con los movimientos básicos ya definidos.

Este proyecto propone una nueva arquitectura al sistema, ofreciendo mayor flexibilidad, escalabilidad y consistencia de la arquitectura software a base de implementar componentes individuales para el control y la comunicación de los dispositivos. Esto permite implementar nuevas interfaces de entrada y nuevos atajos con mayor simplicidad en el código, consiguiendo la capacidad de personalización del control en base a las limitaciones o necesidades del usuario. También ofrece una comunicación segura y tolerante a fallos en la comunicación para que el sistema se detenga y cumpla las expectativas de seguridad del usuario.

En este proyecto se desarrolla una interfaz para el manejo de un robot Arlo, disponible en el laboratorio, pero el sistema se puede utilizar para el manejo remoto de otros dispositivos motorizados, como por ejemplo una silla de ruedas con tan solo adaptar el código de salida.

Se describen a continuación los objetivos generales que trata de satisfacer este proyecto:

1- Diseñar la arquitectura software de manera que sea fácilmente reutilizable para el desarrollo de nuevas interfaces.
2- Implementar comunicación segura entre la interfaz de usuario y el sistema de control del robot: Garantizar que la comunicación a través de sockets SSL para conseguir autenticación en la comunicación e integridad de los datos.
3- Permitir movimientos básicos del robot: como avanzar, retroceder, girar a la derecha, girar a la izquierda (hacia delante y hacia atrás) y detenerse.
4- Integrar atajos de movimiento: Implementar modos adicionales a los movimientos básicos:
    - Girar 180 grados en sentido horario o antihorario.
    - Grabar un recorrido.
    - Ejecutar un recorrido pregrabado.
    - Realizar una parada total del sistema hasta nuevo aviso del usuario.
5- Incorporar mecanismos de seguridad ante fallos de la comunicación o los datos para asegurar que el sistema quede parado en caso de fallo.
6- Facilitar la incorporación de nuevos modos: Diseñar el software de control de manera que sea fácil desarrollar nuevos atajos que permitan mayor autonomía al usuario.
7- Adaptar la interfaz de los acelerómetros definiendo otros movimientos de las manos para las nuevas órdenes desarrolladas.
8- Adaptar el interfaz de la cámara para funcionar en el nuevo sistema.
9- Cifrado en la comunicación: Se incorpora la autenticación del usuario y del robot con el fin de preservar la autenticidad e integridad de los comandos de movimiento recibidos por el robot y asegurar la exclusividad de la comunicación.

Aunque las diferencias entre los proyectos revisados y el presente proyecto pueden parecer sutiles, el enfoque actual introduce mejoras clave que aumentan la funcionalidad. Este proyecto incorpora cuatro nuevos modos de movimiento, permitiendo a los usuarios desplazarse con mayor fluidez en su entorno.

Para facilitar futuras actualizaciones, los datos procesados se presentan en un formato fácilmente adaptable a otros sistemas de entrada usados como interfaz de usuario. La comunicación entre el usuario y el robot está asegurada mediante cifrado SSL, donde tanto cliente como robot presentan sus propios certificados para conseguir una comunicación privada y exclusiva a las dos entidades (usuario y robot). Esto permite evitar que peticiones externas no autorizadas intenten comandar el sistema. Además, se centra en integrar control de seguridad ante fallos en la comunicación como desconexiones y errores de entrada para asegurar la seguridad para el usuario.

## 1.4. Metodología de trabajo

Para la realización de este proyecto se ha utilizado un robot comercial proporcionado por el departamento de sistemas. El montaje electrónico necesario para el control de los motores ha sido llevado a cabo en un PFM (Proyecto de Fin de Máster) [9] y la programación de los movimientos básicos mediante acelerómetros y cámara ha sido realizada en un PFG (Proyecto de Fin de Grado) [8]. El trabajo se ha dividido en las siguientes fases para asegurar una ejecución ordenada y eficiente del proyecto:

- **Fase de investigación:** En esta fase inicial, se realizó un análisis de la problemática a resolver y se investigaron sistemas previos que pudieran ofrecer soluciones similares. Esto incluyó la revisión de tecnologías y métodos existentes para el control remoto de robots a través de gestos humanos, así como, el estudio de diferentes arquitecturas de software y protocolos de comunicación utilizados en proyectos de robótica. Se evaluaron las ventajas y limitaciones de cada enfoque, lo que permitió identificar las mejores prácticas para la implementación del sistema con los recursos existentes.
- **Fase de definición:** Con la información obtenida en la fase anterior, se procedió a definir el sistema que se iba a desarrollar. En esta etapa, se diseñaron los distintos módulos del sistema y sus interacciones, con el objetivo de facilitar la futura incorporación de nuevos interfaces. Para modelar el comportamiento del robot, se empleó un diagrama de estados (Statechart), que permitió representar de forma clara los nuevos modos de funcionamiento incorporados. Asimismo, se definieron los gestos manuales mediante los cuales el usuario puede seleccionar el modo deseado, a través de una interfaz basada en dos acelerómetros, uno en cada mano.
- **Fase de programación:** Una vez definida la estructura y los componentes del sistema, se procedió a la programación de los distintos módulos que componen el sistema de control remoto. Esta fase incluyó el desarrollo del código para los nuevos gestos, la implementación del protocolo de comunicación seguro (SSL) para garantizar la transmisión de datos entre los dispositivos y la programación del autómata y el sistema de control. El software se diseñó de forma modular y escalable, lo que permite su fácil ampliación en el futuro, tanto para integrar nuevos gestos como distintas interfaces de control.
- **Fase de pruebas:** Una vez completada la programación, se realizó un exhaustivo proceso de pruebas para verificar que el sistema funcionara correctamente y cumpliera con los requisitos definidos (pág. 27). Las pruebas se centraron en la comprobación de la encriptación en la comunicación, la precisión del control de los gestos y la estabilidad del sistema bajo diferentes condiciones. Se realizaron pruebas tanto en entornos simulados como en el robot final, permitiendo identificar áreas de mejora y ajustar el sistema para optimizar su rendimiento.
- **Fase de redacción:** Finalmente, una vez completadas las fases anteriores, se redactó el informe detallado del proyecto. En este documento se describen todos los aspectos del desarrollo, incluyendo la arquitectura del sistema (pág. 36), los detalles técnicos de la programación (págs. 39, 48, 54), y los resultados obtenidos en las pruebas (pág. 57). Además, se incluyen recomendaciones para futuras mejoras del sistema (pág. 66).

# Capítulo 2: Contexto

## 2.1. Contexto social:

Las personas con dificultades motrices a menudo se enfrentan a múltiples barreras en actividades cotidianas, desde la dificultad para moverse de manera independiente hasta la falta de acceso adecuado a espacios y servicios públicos.

Desde hace décadas, en la sociedad actual se ha trabajado para adaptar del entorno y desarrollar tecnologías en el ámbito de la salud. Al reducir esas barreras limitantes, se consigue dotar de cierta independencia y autonomía a estas personas con necesidades especiales. Al no sentir la necesidad de asistencia constante, se provoca un sentimiento de libertad y bienestar que llega a mejorar considerablemente el estado de ánimo y la salud mental de los individuos.

La incorporación de rampas, por ejemplo, supuso un gran avance para facilitar el paso a las sillas de ruedas mecánicas, sin embargo, el complemento que aporta la incorporación de tecnología, como las sillas motorizadas y controladas remotamente, ha conseguido liberar la dependencia asistencial por completo en muchos casos. En los casos más especiales o delicados, se ha mejorado considerablemente la calidad de vida, y aliviado la carga asistencial recaída en terceros (como cuidadores o familiares) gracias a personalizar cada caso, según sean las limitaciones motrices del usuario, con distintos tipos de sensores.

Los avances en investigación también han provocado que los costes se abaraten y productividad y eficiencia de las herramientas tecnológicas aumente, consiguiendo reducir la brecha social latente en la sociedad, siendo estas tecnologías más asequibles para un mayor porcentaje de la población [10].

Realizar estos proyectos con sistemas de control ante fallos provocan confianza en la tecnología, comodidad y seguridad. La facilidad de incorporar sensores de monitoreo continuo de salud [11] también aporta la capacidad de un seguimiento constante del estado de los pacientes, permitiendo intervenciones más rápidas y eficaces, aparte de ayudar a la descongestión de los hospitales.

El desarrollo de un sistema de control remoto eficiente y seguro para sillas de ruedas motorizadas tiene un profundo impacto en la sociedad, especialmente en el ámbito de la inclusión y la mejora de la calidad de vida para las personas con movilidad reducida. Con una mayor autonomía, las personas con discapacidades motrices pueden participar más activamente en la vida social, educativa y laboral, lo que contribuye a su integración plena en la sociedad y a su realización personal [12].

## 2.2. Contexto tecnológico:

La tecnología actual está en constante innovación y evolución. El progreso en robótica, IoT y telecomunicaciones juegan un papel crucial para el desarrollo de proyectos enfocados en la accesibilidad y cuidado de las personas.

La industria tecnológica está centrándose cada vez más en la eficiencia energética, produciendo microchips más optimizados, con mayor durabilidad, más económicos y enfocados a cuidar el medio ambiente. Las nuevas baterías también consiguen reducir cada vez más el tiempo de carga y ser más eficientes en el suministro electrónico [13], alargando el tiempo de vida entre cada recarga y mejorando la funcionalidad de los proyectos inalámbricos.

Los robots de servicio, diseñados para asistir a personas en actividades diarias [14], han mejorado en términos de movilidad, capacidad de carga y navegación autónoma. La fácil configuración de las conexiones vía TPC/IP proporciona una ágil comunicación en tiempo real, incluso a larga distancia gracias a las comunicaciones Wi-Fi o redes 4G o 5G [15], que se pueden cifrar mediante SSL para permitir una comunicación autenticada, confidencial e íntegra de los datos.

Los componentes electrónicos, cada vez más accesibles para la población, permiten desarrollar proyectos caseros personalizados [16] gracias también al apogeo de las impresoras 3D, que facilitan el diseño de piezas específicas para dar un acabado más profesional a los prototipos [17]. Actualmente se pueden encontrar en el mercado múltiples sensores y actuadores eficientes con los que poder tener una mejor percepción del entorno y un control de movimientos que mejoran la seguridad y experiencia del usuario.

Se pueden encontrar sensores de monitoreo de salud [18] como: sensor de oxígeno, de frecuencia cardiaca o de presión arterial, de monitoreo medioambiental como: los de humedad, temperatura o luz, de posición como: ultrasonidos, acelerómetros o geolocalización, y también distintos tipos de actuadores, como alarmas sonoras, leds, mecanismos de frenado o los motores, entre otros muchos.

Gracias al desarrollo de algoritmos de aprendizaje automático o inteligencia artificial se ha transformado la manera en que los sistemas operan, siendo éstos mucho más intuitivos con los deseos del usuario, mejorando el reconocimiento de voz o gestos y dotando de cierta autonomía a los sistemas con una navegación más inteligente, por ejemplo [19]. Cuanto más desarrollo hay en las tecnologías, más incorporación de elementos *open source* se pueden encontrar en la comunidad, como las distintas librerías de Python, que aportan mucha flexibilidad y facilidad para realizar proyectos, además permiten mayor compatibilidad entre elementos.

En general, la sociedad actual está dotada de múltiples recursos con los que trabajar para la realización de proyectos funcionales y personalizados de fácil acceso para la población media.

## 3.1. Conceptos teóricos

En esta sección se van a explicar los fundamentos teóricos y protocolos esenciales que se han utilizado en la creación del proyecto, y así facilitar su comprensión.

### Comunicación por sockets y protocolos SSL/TLS

En términos sencillos, un socket es un punto final de comunicación bidireccional entre dos programas que se ejecutan en una red. Se podría decir que son como "enchufes virtuales" que permiten el flujo de datos entre aplicaciones. Aceptan la comunicación simultánea en ambas vías y se puede combinar con protocolos de seguridad para establecer una comunicación de datos privada e íntegra.

En este proyecto se han utilizado los sockets para facilitar la comunicación mediante el protocolo TCP/IP entre la interfaz de usuario y el robot y también se ha integrado el protocolo SSL/TLS (*Secure Sockets Layer/Transport Layer Security*) para agregar una capa extra de seguridad. En realidad, SSL y TLS se refieren al mismo protocolo ya que TLS es la versión mejorada de SSL, aunque usualmente se encuentra el término SSL para generalizar este protocolo.

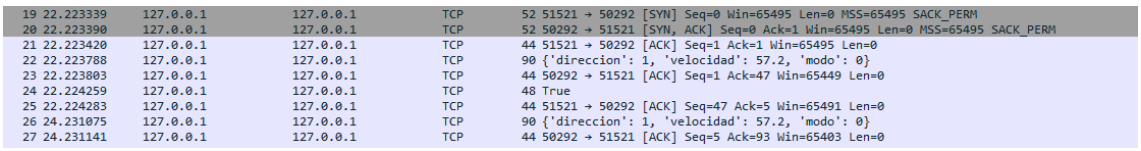
Para la creación de un socket, en el lado del cliente se solicitará establecer una conexión a la IP del servidor, y por parte del servidor se podrá aceptar o rechazar dicha solicitud. Una vez establecido el canal, se mantendrá hasta que alguna de las partes implicadas decida finalizar la conexión.

Para establecer la seguridad SSL/TLS en una comunicación, es necesario crear certificados digitales (pág. 25). Estos certificados permiten que el servidor, el cliente o ambas partes se autentiquen antes de iniciar la comunicación, un proceso conocido como handshake o intercambio de claves de sesión. Una vez autenticados, los datos se cifrarán mediante un algoritmo de encriptación, utilizando las claves intercambiadas durante el handshake. En este proyecto se ha implementado la doble autenticación, tanto del cliente como del servidor para asegurar la comunicación exclusiva de esos dos agentes.

En los siguientes párrafos se pueden observar las diferencias entre una comunicación por sockets sin cifrado, con solo una vía de autenticación (*handshake* a una vía) y con doble autenticación (handshake bidireccional).

#### *Comunicación simple por sockets*

En la siguiente figura (Fig. 3) se puede observar cómo los datos son transferidos por el protocolo TCP. Las tres primeras líneas representan la petición del usuario para la conexión (*SYN*), la respuesta por parte del robot y petición del *ACK* al usuario (*SYN, ACK*) y por último la respuesta *ACK* del usuario. A partir de ahí, ambas partes pueden enviar datos por ese canal. En la cuarta línea el usuario entrega unos datos al robot (*{'direccion': 1, 'velocidad': 57.2, 'modo': 0}*), y en la quinta el servidor responde con un *ACK* por la correcta recepción. La sexta indica unos datos enviados por el robot (*True*) y la séptima el *ACK* del usuario. La comunicación continúa de la misma manera.



| | | | | | |
|---|---|---|---|---|---|
| 19 | 22.223339 | 127.0.0.1 | 127.0.0.1 | TCP | 52 51521 → 50292 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM |
| 20 | 22.223390 | 127.0.0.1 | 127.0.0.1 | TCP | 52 50292 → 51521 [SYN, ACK] Seq=0 Ack=1 Win=65495 Len=0 MSS=65495 SACK_PERM |
| 21 | 22.223420 | 127.0.0.1 | 127.0.0.1 | TCP | 44 51521 → 50292 [ACK] Seq=1 Ack=1 Win=65495 Len=0 |
| 22 | 22.223788 | 127.0.0.1 | 127.0.0.1 | TCP | 90 {'direccion': 1, 'velocidad': 57.2, 'modo': 0} |
| 23 | 22.223803 | 127.0.0.1 | 127.0.0.1 | TCP | 44 50292 → 51521 [ACK] Seq=1 Ack=47 Win=65449 Len=0 |
| 24 | 22.224259 | 127.0.0.1 | 127.0.0.1 | TCP | 48 True |
| 25 | 22.224283 | 127.0.0.1 | 127.0.0.1 | TCP | 44 51521 → 50292 [ACK] Seq=47 Ack=5 Win=65491 Len=0 |
| 26 | 24.231075 | 127.0.0.1 | 127.0.0.1 | TCP | 90 {'direccion': 1, 'velocidad': 57.2, 'modo': 0} |
| 27 | 24.231141 | 127.0.0.1 | 127.0.0.1 | TCP | 44 50292 → 51521 [ACK] Seq=5 Ack=93 Win=65403 Len=0 |

Fig. 3 Captura de fragmento en Wireshark de comunicación simple

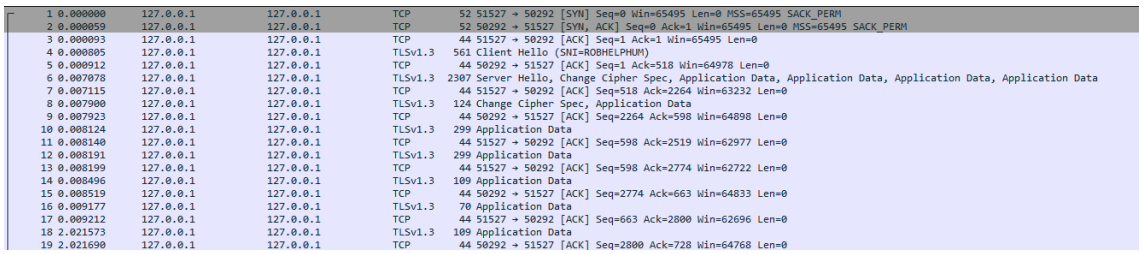
En estas trazas se ve que cualquiera que intercepte esta comunicación puede saber qué datos exactos son transmitidos, ya que se muestran en texto plano, y le resultaría fácil llegar a alterarlos.

### *Handshake de una vía*

Es uno de los escenarios más utilizados, se puede encontrar, por ejemplo, en la navegación web.

El cliente y el servidor intercambian información para establecer una conexión segura. En el handshake, el cliente solicita el certificado (*Client Hello*) del servidor correspondiente al SNI (Server Name Indication) y este le responde (*Server Hello* + datos relacionados con el intercambio de claves). Este certificado digital contiene su clave pública y es revisado por el cliente para comprobar su validez con una entidad certificadora. Se negocia un algoritmo de cifrado y se generan claves de sesión para cifrar la comunicación. Cuando se acepta la comunicación, se realizan los intercambios de datos de forma cifrada.

Se puede ver como los paquetes de datos se mandan cifrados con el protocolo de seguridad TLS mientras que las confirmaciones de recepción siguen enviándose mediante el protocolo TCP.



| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 0.000000 | 127.0.0.1 | 127.0.0.1 | TCP | 52 | 51527 → 50292 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM |
| 2 | 0.000059 | 127.0.0.1 | 127.0.0.1 | TCP | 52 | 50292 → 51527 [SYN, ACK] Seq=0 Ack=1 Win=65495 Len=0 MSS=65495 SACK_PERM |
| 3 | 0.000093 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 51527 → 50292 [ACK] Seq=1 Ack=1 Win=65495 Len=0 |
| 4 | 0.000805 | 127.0.0.1 | 127.0.0.1 | TLSv1.3 | 561 | Client Hello (SNI=ROBHELPHUM) |
| 5 | 0.000912 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 50292 → 51527 [ACK] Seq=1 Ack=518 Win=64978 Len=0 |
| 6 | 0.007078 | 127.0.0.1 | 127.0.0.1 | TLSv1.3 | 2307 | Server Hello, Change Cipher Spec, Application Data, Application Data, Application Data, Application Data |
| 7 | 0.007115 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 51527 → 50292 [ACK] Seq=518 Ack=2264 Win=63232 Len=0 |
| 8 | 0.007900 | 127.0.0.1 | 127.0.0.1 | TLSv1.3 | 124 | Change Cipher Spec, Application Data |
| 9 | 0.007923 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 50292 → 51527 [ACK] Seq=2264 Ack=598 Win=64898 Len=0 |
| 10 | 0.008124 | 127.0.0.1 | 127.0.0.1 | TLSv1.3 | 299 | Application Data |
| 11 | 0.008140 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 51527 → 50292 [ACK] Seq=598 Ack=2519 Win=62977 Len=0 |
| 12 | 0.008191 | 127.0.0.1 | 127.0.0.1 | TLSv1.3 | 299 | Application Data |
| 13 | 0.008199 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 51527 → 50292 [ACK] Seq=598 Ack=2774 Win=62722 Len=0 |
| 14 | 0.008496 | 127.0.0.1 | 127.0.0.1 | TLSv1.3 | 109 | Application Data |
| 15 | 0.008519 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 50292 → 51527 [ACK] Seq=2774 Ack=663 Win=64833 Len=0 |
| 16 | 0.009177 | 127.0.0.1 | 127.0.0.1 | TLSv1.3 | 70 | Application Data |
| 17 | 0.009212 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 51527 → 50292 [ACK] Seq=663 Ack=2800 Win=62696 Len=0 |
| 18 | 2.021573 | 127.0.0.1 | 127.0.0.1 | TLSv1.3 | 109 | Application Data |
| 19 | 2.021690 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 50292 → 51527 [ACK] Seq=2800 Ack=728 Win=64768 Len=0 |

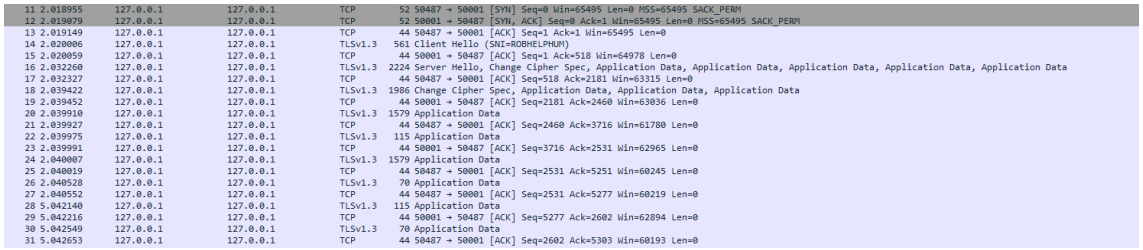
Fig. 4 Captura de fragmento en Wireshark de handshake simple

### *Handshake bidireccional*

Tanto el cliente como el servidor muestran sus certificados digitales. Se utiliza en aplicaciones que requieren una mayor seguridad.

Como se puede ver es muy similar al handshake de una vía, pero ahora se ven más paquetes de *Application Data* intercambiados después del *Server Hello*, relacionados con el certificado del cliente.

La conexión fallará si el certificado no es válido o no se proporciona por alguna de las partes.



| | | | | | | |
|---|---|---|---|---|---|---|
| 11 | 2.018955 | 127.0.0.1 | 127.0.0.1 | TCP | 52 | 50487 → 50001 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM |
| 12 | 2.019079 | 127.0.0.1 | 127.0.0.1 | TCP | 52 | 50001 → 50487 [SYN, ACK] Seq=0 Ack=1 Win=65495 Len=0 MSS=65495 SACK_PERM |
| 13 | 2.019149 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 50487 → 50001 [ACK] Seq=1 Ack=1 Win=65495 Len=0 |
| 14 | 2.020006 | 127.0.0.1 | 127.0.0.1 | TLSv1.3 | 561 | Client Hello (SNI=ROBHELPHUM) |
| 15 | 2.020059 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 50001 → 50487 [ACK] Seq=1 Ack=518 Win=64978 Len=0 |
| 16 | 2.032260 | 127.0.0.1 | 127.0.0.1 | TLSv1.3 | 2224 | Server Hello, Change Cipher Spec, Application Data, Application Data, Application Data, Application Data, Application Data |
| 17 | 2.032327 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 50487 → 50001 [ACK] Seq=518 Ack=2181 Win=63315 Len=0 |
| 18 | 2.039422 | 127.0.0.1 | 127.0.0.1 | TLSv1.3 | 1986 | Change Cipher Spec, Application Data, Application Data, Application Data |
| 19 | 2.039452 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 50001 → 50487 [ACK] Seq=2181 Ack=2460 Win=63036 Len=0 |
| 20 | 2.039910 | 127.0.0.1 | 127.0.0.1 | TLSv1.3 | 1579 | Application Data |
| 21 | 2.039927 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 50487 → 50001 [ACK] Seq=2460 Ack=3716 Win=61780 Len=0 |
| 22 | 2.039975 | 127.0.0.1 | 127.0.0.1 | TLSv1.3 | 115 | Application Data |
| 23 | 2.039991 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 50001 → 50487 [ACK] Seq=3716 Ack=2531 Win=62965 Len=0 |
| 24 | 2.040007 | 127.0.0.1 | 127.0.0.1 | TLSv1.3 | 1579 | Application Data |
| 25 | 2.040019 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 50487 → 50001 [ACK] Seq=2531 Ack=5251 Win=60245 Len=0 |
| 26 | 2.040528 | 127.0.0.1 | 127.0.0.1 | TLSv1.3 | 70 | Application Data |
| 27 | 2.040552 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 50487 → 50001 [ACK] Seq=2531 Ack=5277 Win=60219 Len=0 |
| 28 | 5.042140 | 127.0.0.1 | 127.0.0.1 | TLSv1.3 | 115 | Application Data |
| 29 | 5.042216 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 50001 → 50487 [ACK] Seq=5277 Ack=2602 Win=62894 Len=0 |
| 30 | 5.042549 | 127.0.0.1 | 127.0.0.1 | TLSv1.3 | 70 | Application Data |
| 31 | 5.042653 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 50487 → 50001 [ACK] Seq=2602 Ack=5303 Win=60193 Len=0 |

Fig. 5 Captura de fragmento en Wireshark de Handshake bidireccional

### Autómata o máquina de estados finitos (SFM)

Un autómata es un modelo matemático de computación que describe el comportamiento de un sistema mediante un conjunto finito de estados, junto con transiciones que definen cómo el sistema pasa de un estado a otro en respuesta a eventos o condiciones. Además, este modelo incluye acciones asociadas a la entrada, salida, y permanencia en cada estado, proporcionando una representación estructurada y predecible del comportamiento del sistema [20]. El autómata se representa con los siguientes componentes:

- **Estados:** Diferentes condiciones o situaciones en las que puede encontrarse el sistema. Se representan mediante círculos o celdas. Los *estados finales* indican un estado en el que no hay transición, este caso se representa con un doble circulo. Los *estados iniciales* se señalan con una flecha o marca de inicio. Dentro de los estados se pueden especificar acciones de entrada al estado, mientras se mantiene en el estado o de salida del estado.
- **Transiciones:** Reglas que determinan el cambio de un estado a otro en respuesta a eventos o condiciones. Se representa mediante flechas que van de un estado a otro.
- **Eventos/Condiciones:** Factores externos o internos que desencadenan transiciones entre estados. Vienen definidos encima de la línea de transiciones.
- **Acciones:** Operaciones que se realizan cuando se entra, se sale o se mantiene en un estado.
- **Diagrama de Estados:** Una representación gráfica que muestra los estados del sistema y las transiciones entre ellos. Facilita la visualización del comportamiento del sistema.

Los autómatas se utilizan en programación para diseñar sistemas que deben reaccionar de manera predecible a una serie de eventos, como los sistemas de control en tiempo real. Son muy prácticos para determinar y visualizar fácilmente el funcionamiento deseado del sistema.

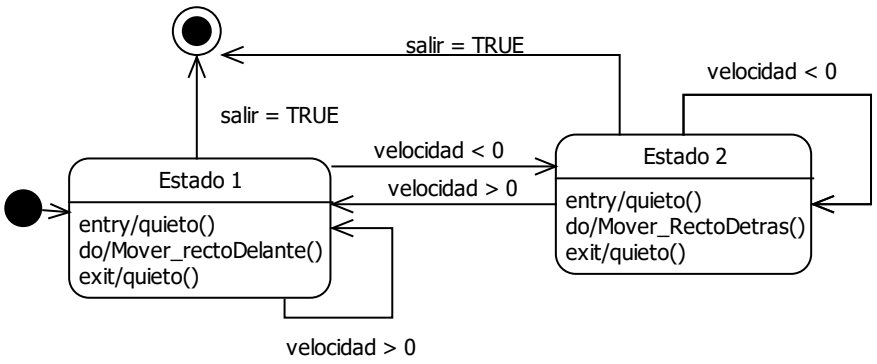



Fig. 6 Ejemplo básico de Statechart con todos los componentes

En este pequeño ejemplo (Fig. 6) hay dos estados. El estado inicial, el estado 1, representa un movimiento hacia delante, el estado 2, un movimiento hacia atrás. Se puede ver como en ambos casos la acción de entrada y salida del estado es permanecer quieto, mientras que, si se permanece en el estado, se ejecutará el movimiento que corresponda. Se mantendrá en estado 1 siempre que la velocidad sea mayor que cero, se cambiará a estado 2 cuando la velocidad sea menor que cero. En ambos estados se puede finalizar la ejecución si se establece el evento salir en TRUE.

### Inter-Integrated Circuit (I2c)

Es un protocolo de comunicación serial que permite que múltiples dispositivos se comuniquen entre sí mediante solo dos líneas: una para datos (SDA) y otra para reloj (SCL). Tiene una topología maestro-esclavo que permite a un dispositivo (maestro), controlar a varios dispositivos (esclavo) conectados. Cada componente que hace uso del bus de comunicación debe tener una dirección única, para poder ser comandado individualmente. Los datos se transmiten serializados en bits y sincronizados gracias a la señal de reloj generada por maestro.

Este protocolo es ampliamente utilizado en electrónica para conectar componentes como sensores, memorias, pantallas y otros periféricos, especialmente en sistemas embebidos y microcontroladores.

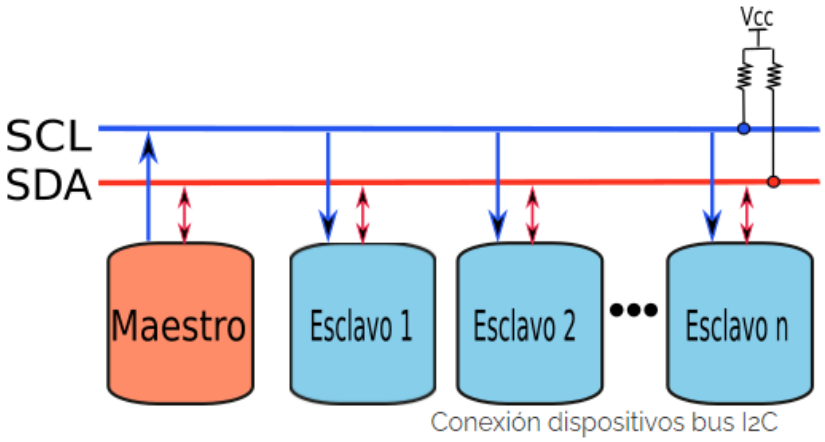



Fig. 7 Esquema básico conexión bus dispositivos I2C **[21]**

## 3.2. Tecnologías utilizadas

En este apartado se hace una breve descripción de las tecnologías esenciales que se han usado para este proyecto.

### Raspberry PI 3 b+ GPIO

Los dispositivos Raspberry PI son minicomputadores de bajo costo y alta capacidad de procesamiento equipados con puertos GPIO (entrada/salida de propósito general), Wi-Fi, Bluetooth y una amplia gama de puertos de conexión. Ha ganado gran popularidad para el desarrollo de proyectos educativos y domóticos gracias a su accesibilidad, flexibilidad y comunidad activa de desarrolladores.

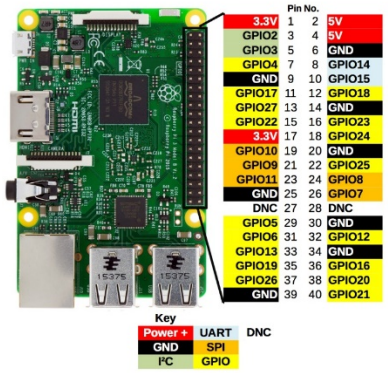



Fig. 8 Raspberry PI 3 b+ y descripción GPIO **[22]**

### MPU-6050

El MPU-6050 es un sensor de movimiento de seis ejes que combina un acelerómetro de tres ejes y un giroscopio de tres ejes en un solo chip. Este dispositivo compacto y de bajo consumo de energía es comúnmente utilizado en aplicaciones de electrónica y robótica para medir la aceleración y la velocidad angular en tres dimensiones. Utiliza una interfaz de comunicación I2C, que es compatible con una amplia variedad de microcontroladores y placas de desarrollo, como la Raspberry PI. Sus conexiones son muy simples ya que solo requiere de entrada de corriente VCC (Fig. **9** punto rojo), conector a tierra GND (Fig. **9** punto negro) y dos conexiones más compartidas con todos los dispositivos I2C conectados: una bidireccional para la comunicación de los datos SDA (Fig. **9** punto verde), y otra que proporciona la señal de reloj para sincronizar la transferencia de datos, SCL (Fig. **9** punto naranja), que será controlada por el máster.

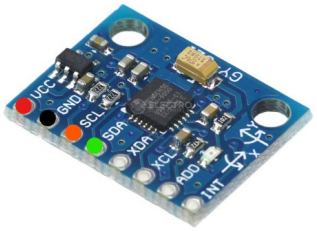



Fig. 9 Microchip MPU-6050 **[23]**

## 3.3. Software empleado

En este apartado se hace una breve descripción del software utilizado en este proyecto.

### Python

Python es un lenguaje de programación de alto nivel y orientado a objetos, compatible con la mayoría de sistemas operativos como Windows, Linux y macOS. Gracias a su extensa comunidad existen múltiples librerías de libre distribución que se pueden incorporar fácilmente en los proyectos personales y que consiguen reducir enormemente la complejidad de la programación.

Fig. 10 Logotipo Python **[24]**

### Androidcam

AndroidCam es un cliente web que consigue reconocer la cámara de un móvil Android como entrada en otros dispositivos como Windows o Linux. Simplemente se necesita instalar el programa en ambos dispositivos (PC y móvil) y que éstos se encuentren conectados a la misma red. Desde la aplicación del PC se configura la IP que se muestra en la aplicación del dispositivo Android y se selecciona a que puertos te quieres conectar (audio, vídeo o ambos). Una vez queda conectado, se reciben imágenes al PC desde la cámara del móvil.

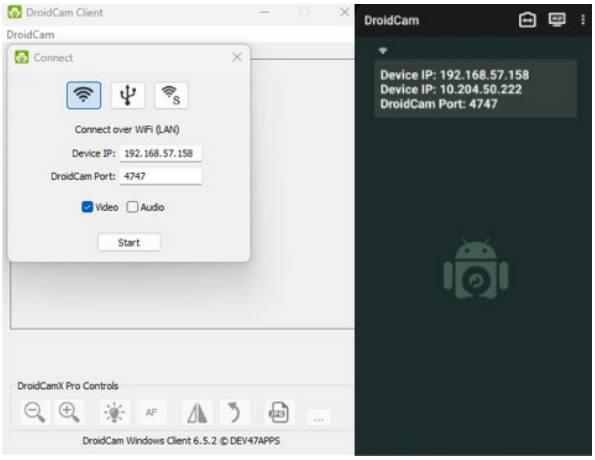



Fig. 11 Vista DroidCam Client Windows (derecha) y android (izquierda)

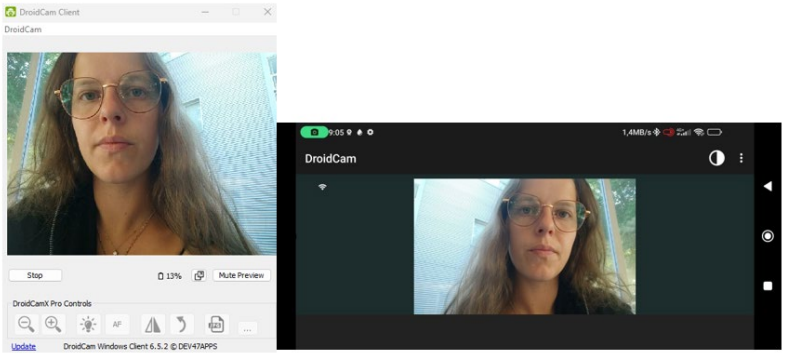



Fig. 12 Vista de las dos perspectivas de la cámara

### Wireshark

Es una herramienta de análisis de protocolos de red que permite la captura y el análisis del tráfico de red en tiempo real. Es altamente utilizado por profesionales de seguridad informática y por administradores de red para investigar tramas de datos, detección de intrusiones o análisis de rendimiento de red, entre otras cosas, gracias a sus capacidades de filtrado, estadísticas detalladas, decodificación de protocolos y su interfaz fácil e intuitiva [25].

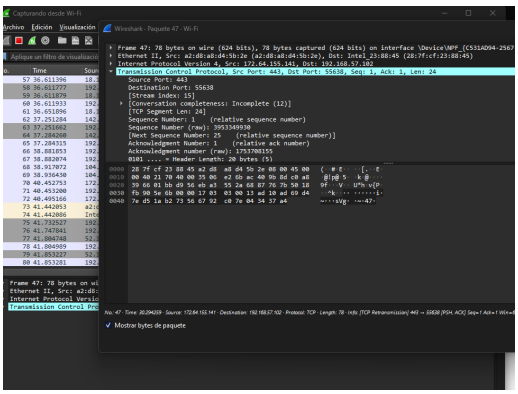


Fig. 13 Vista captura de tráfico WireShark

### Raspbian SO

El sistema operativo Raspbian SO está adaptado a los dispositivos Raspberry PI y les otorga la capacidad de actuar como un minicomputador, con una interfaz muy práctica. Trabaja sobre el software libre Debian Linux adaptado para trabajar sin dificultad sobre esta placa y poder interactuar sobre todas las entradas y protocolos que la componen.


Fig. 14 Imagen simple escritorio Raspberry Pi OS **[26]**

### OpenSSL

OpenSSL es una robusta herramienta de software ampliamente utilizada para la implementación de protocolos de seguridad de red y criptografía. Ofrece una rica biblioteca de funciones escritas en el lenguaje de programación C, que permiten la utilización de criptografía de clave pública y privada, así como protocolos SSL (*Secure Sockets Layer*) y TLS (*Transport Layer Security*) [27]. Sus características principales son:

- Establecimiento de conexiones seguras entre servidores y clientes en aplicaciones web y de red. *OpensSSL* facilita la creación de conexiones seguras en redes, proporcionando cifrado de datos en tránsito gracias a los protocolos SSL/TLS.

- Cifrado y descifrado de datos para asegurar la confidencialidad e integridad de la información, gracias a su gran biblioteca criptográfica que incluye gran variedad de algoritmos criptográficos como AES, RSA, SHA, MAD5, …
- Generación y gestión de certificados digitales y claves criptográficas. Ofrece herramientas para gestionar certificados SSL/TLS, generar claves criptográficas, crear solicitudes de firma de certificados (CSR), y más.

A continuación, se explica cómo se realiza el último punto ya que en este proyecto se ha utilizado esta característica para la comunicación de datos entre el usuario y el robot.

### *Generación de certificados SSL*

Para la generación de los certificados y claves para clientes y servidores es muy común el uso de OpenSSL. El proceso para la obtención de certificados autofirmados se consigue fácilmente con esta herramienta para gran variedad de SO. Los pasos a seguir en un sistema Windows son los siguientes:

1. Descargar OpenSSL desde el sitio oficial [28] e instalar siguiendo los pasos marcados por el agente de instalación hasta finalizar el proceso.

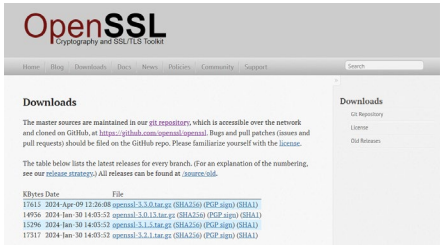



Fig. 15 Página oficial de descargas OpenSSL **[28]**

2. Situarse en el directorio donde se ha instalado el programa *openssl.exe* y abrir la consola de comandos con permisos administrativos en esa ruta.

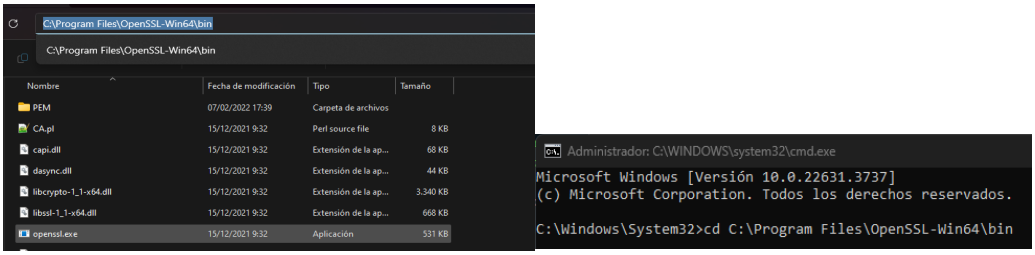



Fig. 16 Carpeta de ubicación del archivo *OpenSSL.exe*

3. Generar los certificados del cliente y del servidor desde la consola:

Las siguientes dos líneas muestran cómo se han generado los certificados del servidor y del cliente respectivamente:

```
> openssl req -nodes -x509 -newkey rsa:4096 -keyout robhelphum_k.key -out robhelphum_cer.cer -days 10000 -subj /CN=robhelphum
```

```
> openssl req -nodes -x509 -newkey rsa:4096 -keyout user_k.key -out user_cer.cer -days 10000 -subj /CN=user
```

- ***-nodes***: (No descifrado) Indica que los certificados no tengan contraseña, facilita la implementación de ciertos códigos.
- ***-x509*** genera un certificado x.509 autofirmado, en vez de solo una solicitud de firma de certificado (CSR).

- ***-newkey rsa:4096***: indica que la clave privada que se va a generar debe ser de tipo RSA y tener una longitud de 4096 bits.
- ***-keyout robhelphum_k.key -out robhelphum_cer.cer***: indica el nombre de la clave privada (.key) y el de la clave pública (.cer) que se quiere dar a los nuevos certificados.
- ***-days 10000***: Indica el tiempo de validez del certificado.
- ***-subj /CN=robhelphum***: Indica el nombre común que va a resolver este certificado. Poniendo esta línea se evita rellenar más información.

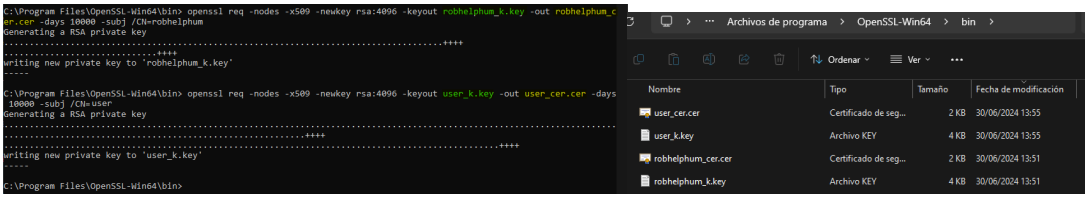



Fig. 17 Generación de certificados

4. Para obtener el hash o huella única de un certificado se sitúa el certificado en la ubicación de OpenSSL y desde la línea de comandos en la misma carpeta, se ejecuta el siguiente comando:

```
> openssl x509 -in user_cer.cer -noout -fingerprint -sha256
```

```
C:\Program Files\OpenSSL-Win64\bin>openssl x509 -in user_cer.cer -noout -fingerprint -sha256
SHA256 Fingerprint=70:5C:1F:39:FD:67:0F:4E:17:4D:9A:26:DC:E7:03:86:32:52:5E:E5:1C:A6:93:0D:0D:B1:92:A5:C0:76:DC:48
```

Fig. 18 Obtención del hash de un certificado desde la línea de comandos

5. Con herramientas como Whireshark, explicada anteriormente, se puede observar el flujo de datos con texto claro cuando no se usan certificados y el texto encriptado cuando se aplican los certificados en la comunicación del programa:

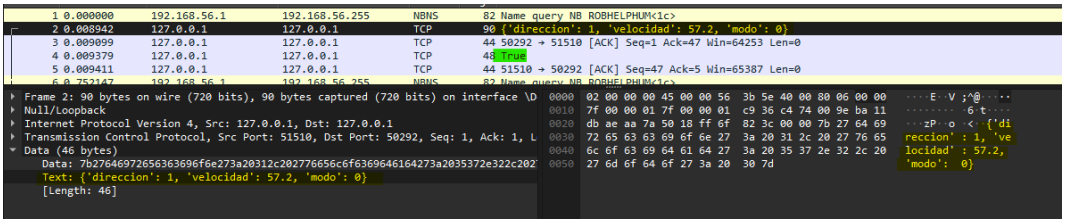



Fig. 19 Flujo de datos sin cifrado SSL

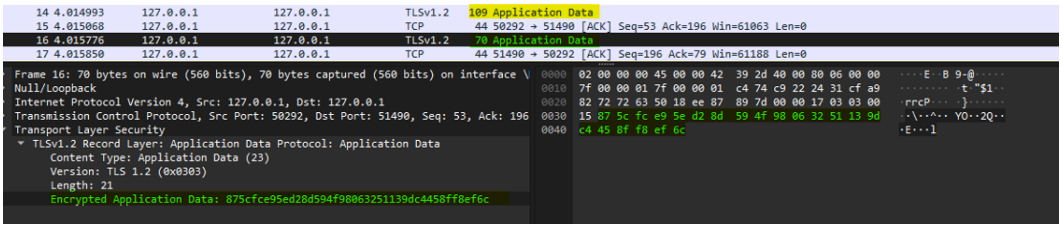



Fig. 20 Flujo de datos con cifrado SSL

# Capítulo 3: Sistema desarrollado

## 3.1. Especificación de requisitos del sistema

En base a los objetivos generales descritos en el apartado 1.2 y de los componentes del sistema descritos en la Fig. **2** de este documento, se han desarrollado los requisitos específicos que cumple el proyecto.

### 3.1.1. Requisitos funcionales

*Generales*

FG1. El sistema global debe ser capaz de reconocer gestos del usuario que transformará en órdenes para el control del movimiento de un robot. Con los gestos, el usuario envía información sobre la velocidad, dirección y el modo.

FG2. Las órdenes al robot se enviarán de forma inalámbrica desde el subsistema de entrada al subsistema de salida.

*Interfaz de Usuario*

FI1. El sistema de entrada debe ser capaz de interpretar los movimientos de dos acelerómetros para determinar órdenes del usuario.

FI2. El sistema de entrada debe ser capaz de interpretar los movimientos de la cabeza del usuario a través de una cámara para determinar las órdenes.

FI3. El sistema de entrada debe ser capaz de calibrar las entradas de los acelerómetros en función de las capacidades de movimiento del usuario.

*Movimiento*

FM1. El sistema de salida será capaz de interpretar tres datos mandados por el usuario mediante la interfaz de entrada: modo, dirección y velocidad.

FM2. El sistema de salida debe ser capaz de hacer que el robot ejecute los movimientos básicos de ir hacia delante, hacia atrás, girar hacia la derecha o a la izquierda en ambos sentidos o parar. El evento u orden que indicará esta acción se nombrará de aquí en adelante como MODO 0, o **M0**.

FM3. El sistema de salida debe ser capaz de hacer que el robot dé un giro de 180º en sentido horario o antihorario según lo indique el usuario. El evento u orden que indicará esta acción se nombrará de aquí en adelante como MODO 1, o **M1**.

FM4. El sistema de salida debe ser capaz de grabar en un fichero una secuencia de movimientos realizados cuando el usuario indique la orden de grabación, el evento u orden que indicará esta acción se nombrará de aquí en adelante como MODO 3, o **M3**. Otra orden indicada por el usuario se utilizará para guardar el fichero, el evento u orden que indicará esta acción se nombrará de aquí en adelante como MODO 5, o **M5**.

FM5. El sistema de salida debe ser capaz de exigir al robot que replique los movimientos pregrabados en el fichero guardado a través de una orden concreta del usuario. El evento u orden que indicará esta acción se nombrará de aquí en adelante como MODO 2, o **M2**.

FM6. El sistema de salida debe ser capaz de realizar una parada total en cualquier tipo de movimiento que se esté ejecutando si el usuario lo indica, el evento u orden que indicará esta acción se nombrará de aquí en adelante como MODO 4, o **M4**. El robot se mantendrá en parada hasta que el usuario indique **M5**, evento que en este caso indica la reanudación del movimiento con las órdenes básicas M0 (FM2).

*Control estados*

FC1. Las transiciones entre los distintos estados de movimiento del robot respetarán el diagrama de estados de la Fig. 23 y serán provocadas por órdenes o eventos indicados por el usuario mediante gestos: M0, M1, M2, M3, M4 y M5.

FC2. El estado 0, **E0**, será aquél donde el sistema retransmita al robot las órdenes de los movimientos básicos del usuario, indicados como M0, y aceptará también los eventos M1, M2, M3 y M4, para un cambio de estado.

FC3. El estado 1, **E1**, será aquél donde el sistema se mantenga indicando un giro de 180º al robot y únicamente aceptará M4 como evento del usuario (abortar el movimiento). Si el giro finaliza sin M4, el sistema volverá al E0 automáticamente.

FC4. El estado 2, **E2**, será aquél donde el sistema reproduce en el robot los movimientos almacenados en un fichero, enviando al robot las indicaciones anteriormente grabadas. En este estado únicamente aceptará M4 como evento del usuario (abortar el movimiento). Si la lectura del fichero acaba sin haber recibido la señal M4, el sistema volverá al E0 automáticamente.

FC5. El estado 3, **E3**, será aquel donde el sistema envíe los movimientos básicos recibidos como M0 por parte del usuario a la vez que los plasma en un fichero. Aceptará también como eventos M4, donde se abortará el movimiento y quedará vacío el fichero, y M5, donde se guardará el fichero con los movimientos recibidos y se volverá al E0.

FC6. El estado 4, **E4**, será aquel donde el sistema mantendrá al robot en parada hasta que reciba la señal M5, donde se volverá al E0.

FC7. Para realizar una transición de estado, el sistema esperará a que el usuario solicite dicha acción durante un tiempo estimado inicialmente en 3 segundos, aunque en la fase de pruebas se experimentará para determinar cuál es el retardo adecuado.

### 3.1.2. Requisitos no funcionales

*Generales*

NF1. El sistema será capaz de detener el movimiento del robot cuando detecte una desconexión en la comunicación entre el usuario y el robot.

NF2. El sistema podrá gestionar lecturas erróneas de entrada, o gestos desconocidos para mantener el robot quieto. Desde ahora se mencionará como MODO 999 o **M999**.

NF3. La jerarquía de ficheros realizados en este sistema será intuitiva y fácilmente ampliable para agregar una nueva interfaz de usuario.

NF4. El sistema se programará mediante un autómata o máquina de estados que permitirá incorporar nuevos estados de funcionamiento y sus transiciones de forma fácil.

*Comunicación*

NFC1. La comunicación entre los dos subsistemas será continua.

NFC2. La comunicación entre los dos subsistemas será capaz de mantener la integridad y veracidad de los datos mediante el cifrado SSL.

NFC3. La comunicación solo se permitirá a un usuario concreto, comprobando el HASH de su certificado digital, manteniendo la exclusividad y autenticidad de la comunicación.

NFC4. La comunicación tendrá un sistema de detección de errores para que se desconecte de forma segura, indicando al robot que quede quieto antes de la desconexión.

## 3.2. Casos de uso

| | | | | |
|---|---|---|---|---|
| **Código** | **CU-00** | | | |
| **Nombre** | Modo 0 – Movimiento del robot hacia delante. (FM2) | | | |
| **Descripción** | El usuario indica el gesto de ir hacia delante (la interfaz de entrada envía velocidad positiva, dirección nula y M0 al sistema de salida). El sistema envía las directrices pertinentes a los motores del robot, que ejecutan el movimiento. | | | |
| **Actores** | Usuario. | | | |
| **Precondición** | Encontrarse en E0 o E3. | | | |
| **Flujo normal** | 1 | Usuario indica el movimiento de velocidad positiva y dirección nula y M0. | 4 | Se envían las directrices a los motores mediante GPIO. |
| | 2 | El sistema de entrada envía los datos cifrados al subsistema de salida. | 5 | El robot realiza el movimiento. |
| | 3 | El subsistema de salida interpreta los datos en E0. | | |
| **Flujos Alternos** | *Flujo Alterno 1 - Movimiento del robot hacia delante en E3.* | | | |
| | 3 | El subsistema de salida interpreta los datos en E3. | 5 | Se envían las directrices a los motores mediante GPIO. |
| | 4 | Se guardan los datos de velocidad y dirección en un fichero. | 6 | El robot realiza el movimiento. |
| **Postcondición** | Ninguna. | | | |
| **Excepciones** | El subsistema entradas y el subsistema salidas pierden la comunicación por un fallo de entradas y el robot cancela el movimiento, quedando parado. El programa termina su ejecución. | | | |
| **Comentarios** | Si está en un estado distinto de la precondición, el sistema ignora la señal M1. | | | |

Tabla 2 (CU-00) Movimiento del robot hacia delante

| **Código** | **CU-01** | | | |
|---|---|---|---|---|
| **Nombre** | MODO 1 – Inicio del giro robot 180º derecha. (FM3) | | | |
| **Descripción** | El usuario indica M1 hacia la derecha mediante la interfaz de entrada durante unos segundos (en este caso la dirección será 1 y la velocidad será nula), manteniéndose quieto durante el cambio de estado. El sistema realiza un cambio de estado al E1 para realizar un giro de 180º a la derecha y cuando finaliza el giro vuelve al E0 tras dos segundos en espera estática. | | | |
| **Actores** | Usuario. | | | |
| **Precondición** | Encontrarse inicialmente en el E0. | | | |
| **Flujo normal** | 1 | Usuario indica M1 hacia la derecha. | 5 | El robot completa el movimiento. |
| | 2 | El sistema de entrada envía los datos cifrados al subsistema de salida. | 6 | El robot se mantiene 2 segundos en parada. |
| | 3 | El subsistema de salida da por bueno el cambio de estado al E1. | 7 | El subsistema de salida vuelve al E0 y está listo para recibir nuevas directrices. |
| | 4 | Se inicia el M1 enviando las señales de giro a los motores. | | |
| **Flujos Alternos** | *Flujo Alterno 1 – Giro 180º con parada segura* | | | |
| | 5 | El usuario indica M4. | 8 | El estado actual, E1, cambia a E4. |
| | 6 | El subsistema salidas recibe la señal e indica parada al robot. | 9 | El robot queda en parada. |
| | 7 | El subsistema salidas recibe todas las señales M4 necesarias para el cambio de estado. | | |
| **Postcondición** | Ninguna | | | |
| **Excepciones** | El subsistema entradas y el subsistema salidas pierden la comunicación por un fallo de entradas y el robot cancela el movimiento, quedando parado. El programa termina su ejecución. | | | |
| **Comentarios** | El funcionamiento del M1 con giro de 180º a la izquierda sería el mismo caso, pero girando a la izquierda.<br>Si está en un estado distinto de la precondición, el sistema ignora la señal M1. | | | |

Tabla 3 (CU-01) Inicio del giro robot 180º derecha

| **Código** | **CU-02** | | | |
|---|---|---|---|---|
| **Nombre** | Modo2 – Reproducir movimiento guardado. (FM5) | | | |
| **Descripción** | El usuario indica cambio a M2 (con dirección y velocidad nulas) durante unos segundos. El robot se mantiene quieto durante el cambio de modo. El robot comienza a reproducir las ordenes guardadas en fichero ignorando las entradas del usuario, salvo si recibe M4. Cuando finaliza el fichero el sistema vuelve al E0, el robot queda detenido durante 2 segundos y vuelve a hacer caso de las entradas del usuario. | | | |
| **Actores** | Usuario. | | | |
| **Precondición** | El sistema esta inicialmente en E0. | | | |
| **Flujo normal** | 1 | El usuario indica M2 desde las interfaces de entrada. | 5 | El sistema manda los datos al robot y realiza el movimiento. |
| | 2 | El subsistema entradas envía los datos cifrados al subsistema salidas. | 6 | El fichero acaba, indicativo de cambio de estado al E0. |
| | 3 | El autómata verifica el cambio de estado y se inicia el cambio de estado al E2. | 7 | El robot queda quieto durante el cambio de estado. |
| | 4 | El sistema lee una a una las entradas del fichero con la misma frecuencia con la que se reciben datos del usuario. | 8 | El robot vuelve a hacer caso de las entradas recibidas del usuario. |
| **Flujos Alternos** | *Flujo Alterno 1 – Reproducir movimiento con fichero vacío.* | | | |
| | 5 | El fichero se encuentra vacío, por lo que se indica un cambio de estado al E0. | 7 | El robot vuelve a hacer caso de las entradas recibidas del usuario. |
| | 6 | El robot queda quieto durante el cambio de estado. | | |
| | *Flujo Alterno 2 – Reproducir movimiento con parada segura.* | | | |
| | 6 | El usuario indica M4. | 8 | El subsistema cambia a E4. |
| | 7 | El robot queda quieto durante el cambio de modo. | 9 | El robot se mantiene quieto en E4. |
| **Postcondición** | Ninguna. | | | |
| **Excepciones** | El subsistema entradas y el subsistema salidas pierden la comunicación por un fallo de entradas y el robot cancela el movimiento, quedando parado. El programa termina su ejecución. | | | |
| **Comentarios** | Si está en un estado distinto de la precondición, el sistema ignora la señal M2. | | | |

Tabla 4 (CU-02) Reproducir movimiento guardado

| **Código** | **CU-03** | | | |
|---|---|---|---|---|
| **Nombre** | MODO 3 – Grabación de movimiento en fichero. (FM4) | | | |
| **Descripción** | El usuario indica M3 (con dirección y velocidad nulas) durante unos segundos y el robot inicia el E3. El robot realiza los movimientos básicos, M0, indicados por el usuario a la vez que guarda las entradas de dirección y velocidad en un fichero. El usuario indica M5 durante unos segundos para finalizar la grabación. El robot queda quieto durante el cambio de estado a E0. | | | |
| **Actores** | Usuario. | | | |
| **Precondición** | El sistema esta inicialmente en E0. | | | |
| **Flujo normal** | 1 | El usuario indica M3 desde las interfaces de entrada. | 5.2 | El robot guarda en fichero las entradas del usuario de velocidad y dirección. |
| | 2 | El subsistema entradas envía los datos cifrados al subsistema salidas. | 6 | El usuario indica señal M5. |
| | 3 | El autómata aprueba la transición de estado. | 7 | El robot queda quieto durante el cambio de estado y el subsistema vuelve al E0. |
| | 4 | Se cambia del E0 al E3. | | |
| | 5.1 | El robot se mueve según lo indica el usuario. | | |
| **Flujos Alternos** | *Flujo Alterno 1* | | | |
| | 6 | El usuario indica señal M4. | 9 | Se borran las órdenes del fichero, dejándolo vacío. |
| | 7 | El robot queda quieto durante el cambio de estado y el subsistema cambia a E4. | 10 | El robot queda quieto, en E4. |
| **Postcondición** | Ninguna | | | |
| **Excepciones** | El subsistema entradas y el subsistema salidas pierden la comunicación por un fallo de entradas y el robot cancela el movimiento, quedando parado. El programa termina su ejecución. | | | |
| **Comentarios** | Si está en un estado distinto de la precondición, el sistema ignora la señal M3. | | | |

Tabla 5 (CU-03) Grabación de movimiento en fichero

| **Código** | **CU-04** | | | |
|---|---|---|---|---|
| **Nombre** | Modo 4 – Parada absoluta del robot. (FM6) | | | |
| **Descripción** | El usuario indica cambio a M4 (con dirección y velocidad nulas) durante unos segundos. El robot se mantiene quieto durante el cambio de estado. El sistema cambia a E4 y el robot queda quieto ignorando las entradas del usuario. | | | |
| **Actores** | Usuario. | | | |
| **Precondición** | Ninguna. | | | |
| **Flujo normal** | 1 | El usuario indica M4 desde la interfaz de entrada. | 3 | El autómata verifica el cambio de estado y el sistema realiza el cambio de estado al E4. |
| | 2 | El subsistema entradas envía los datos cifrados al subsistema salidas. | 4 | El sistema mantiene al robot quieto. |
| **Flujos Alternos** | Ninguno. | | | |
| **Postcondición** | Ninguna. | | | |
| **Excepciones** | El subsistema entradas y el subsistema salidas pierden la comunicación por un fallo de entradas y el robot cancela el movimiento, quedando parado. El programa termina su ejecución. | | | |
| **Comentarios** | La única señal del usuario que aceptará el sistema será M5. Si el usuario nunca indica M5, el robot permanecerá quieto. El sistema acepta la seña M4 en cualquier estado. | | | |

Tabla 6 (CU-04) Parada absoluta del robot

| **Código** | **CU-05** | | | |
|---|---|---|---|---|
| **Nombre** | Setup – Iniciación de rangos máximos y mínimos de los acelerómetros. (FI3) | | | |
| **Descripción** | El usuario ejecuta el programa de setup de los acelerómetros. El usuario mantiene los movimientos de las manos indicados por texto en pantalla durante unos segundos mientras el sistema recoge datos y guarda en un fichero la media de los valores obtenidos para el rango máximo y mínimo de los movimientos sobre el eje 'x' y el eje 'y'. | | | |
| **Actores** | Usuario. | | | |
| **Precondición** | Ninguna. | | | |
| **Flujo normal** | 1 | El usuario ejecuta el programa setup de los acelerómetros y espera indicaciones. | 5 | El usuario mantiene la posición durante unos segundos. El programa obtiene la posición máxima y mínima del eje 'y' de cada mano e indica por texto movimiento registrado. |
| | 2 | Se indica por pantalla que se giren ambas manos hacia la derecha. | 6 | Se indica por pantalla que se giren ambas manos hacia arriba. |
| | 3 | El usuario mantiene la posición durante unos segundos. El programa obtiene la variable x_derecha de cada mano e indica por texto movimiento registrado. | 7 | El usuario mantiene la posición durante unos segundos. El programa obtiene la variable y_delante de cada mano e indica por texto movimiento registrado. |
| | 4 | Se indica por pantalla que se giren ambas manos hacia la izquierda. | 8 | Se indica por pantalla que se giren ambas manos hacia abajo. |
| | 5 | El usuario mantiene la posición durante unos segundos. El programa obtiene la variable x_izquierda de cada mano e indica por texto movimiento registrado. | | El usuario mantiene la posición durante unos segundos. El programa obtiene la variable y_detrás de cada mano e indica por texto movimiento registrado. |
| | 6 | Se indica por pantalla que se mantengan las palmas paralelas al suelo. | | El programa guarda en un fichero "*setup_file.txt*" los datos obtenidos e indica una correcta inicialización. |
| **Postcondición** | Ninguna. | | | |
| **Excepciones** | Ninguna. | | | |
| **Comentarios** | Se debe ejecutar este programa para configurar los valores para cada usuario antes de la ejecución del programa principal. | | | |

Tabla 7 (CU-05) Setup de los acelerómetros

## 3.3. Desarrollo software

### 3.3.1. Explicación general del sistema

#### Diagrama de bloques

El siguiente diagrama de bloques (Fig. 20) representa el esquema de los ficheros del proyecto y cómo interactúan entre sí:

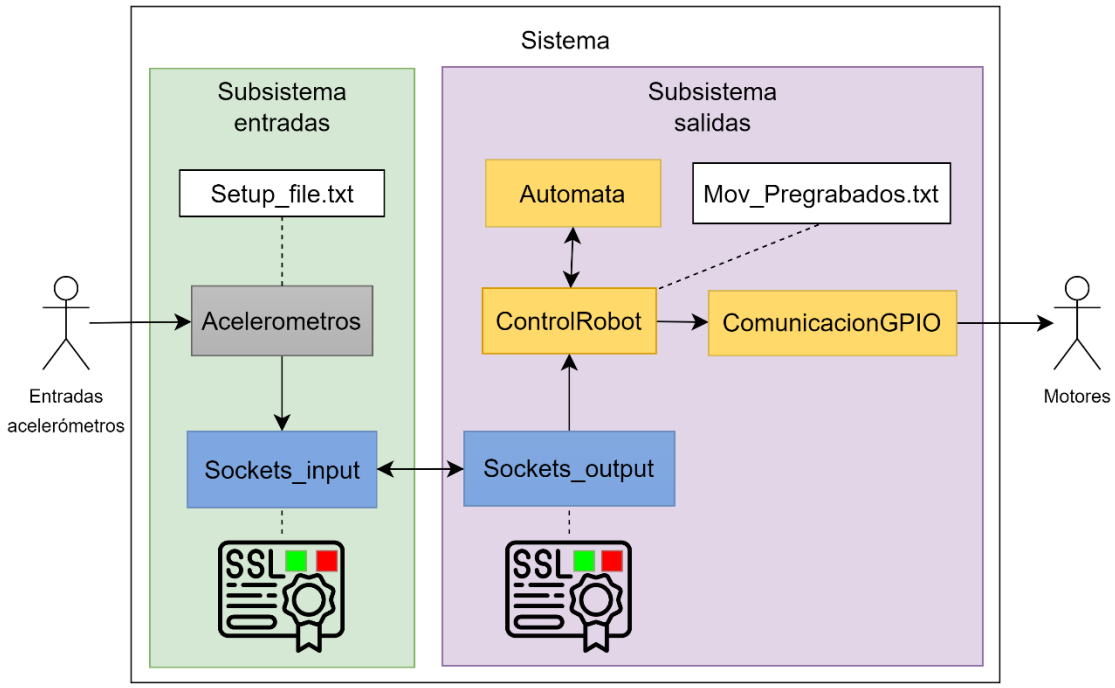



Fig. 21 Esquema general de los elementos del sistema con acelerómetros

Se puede observar cómo el conjunto queda dividido en dos subsistemas. El bloque verde, subsistema entradas, representa la gestión de los datos obtenidos desde la interfaz de entrada controlada por el usuario. El bloque morado, subsistema salidas, representa la interpretación de los datos por parte de la Raspberry PI ubicada en el robot, para producir la salida adecuada para cada motor.

El subsistema entradas está compuesto por el código *Acelerometros.py*, que utiliza el fichero *Setup_file.txt* (Tabla 7) para inicializar los extremos de entrada del eje x y del eje y en ambas manos, y envía los datos (modo, dirección y velocidad) a través de un diccionario nombrado *pautas*, realizando una llamada al fichero *Sockets_input.py*.

El módulo *Sockets_output.py*, de salidas, se encarga de establecer y aceptar la comunicación con el subsistema entradas y recibe los datos enviados por *Sockets_input.py*. Estos datos los gestiona *ControlRobot.py*, que llama a *Automata.py* para analizar los cambios de estado del sistema. Finalmente, las órdenes se transmiten a los motores desde el fichero *ComunicacionGPIO.py*, mediante puertos digitales GPIO (General Port Input Output).

En la comunicación, bloques azules, será necesario tener los certificados SSL de cada extremo (la clave pública ajena, indicada con un cuadrado verde y la privada y pública propia, indicado con un cuadrado rojo).

El funcionamiento en profundidad de cada fichero queda explicado más adelante.

El esquema del sistema en el caso de controlar el robot mediante gestos realizados con la cabeza a través de una cámara sigue a continuación. Como se puede ver en la Fig. 21, el subsistema de salidas sigue siendo igual, solo se ha cambiado el fichero de recepción de entradas del subsistema entradas.

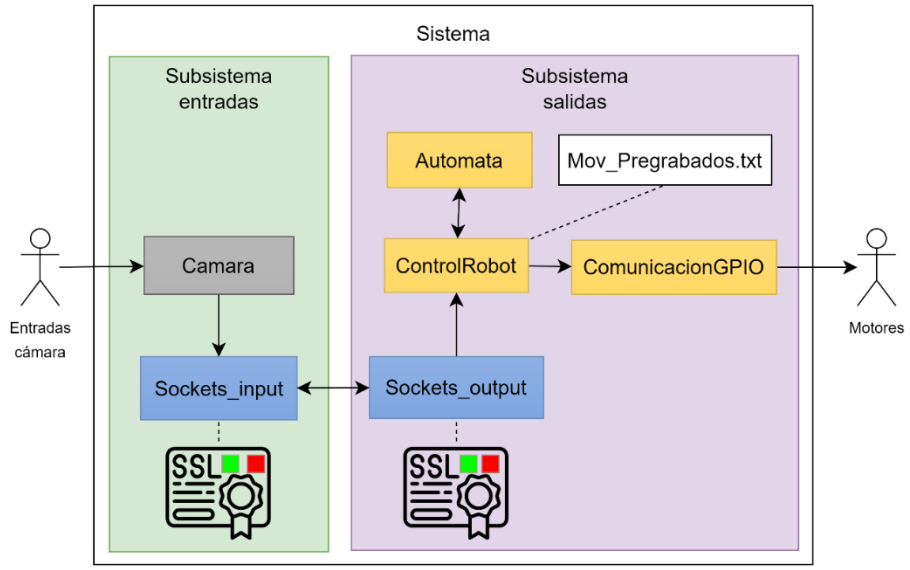



Fig. 22 Esquema general de los elementos del sistema con cámara

## Diagrama de secuencia

En el siguiente diagrama de secuencia se puede observar más a fondo cómo se comunican los ficheros entre sí desde que se inicia el programa (punto 1) y para cada dato leído por el sensor (a partir del punto 6):

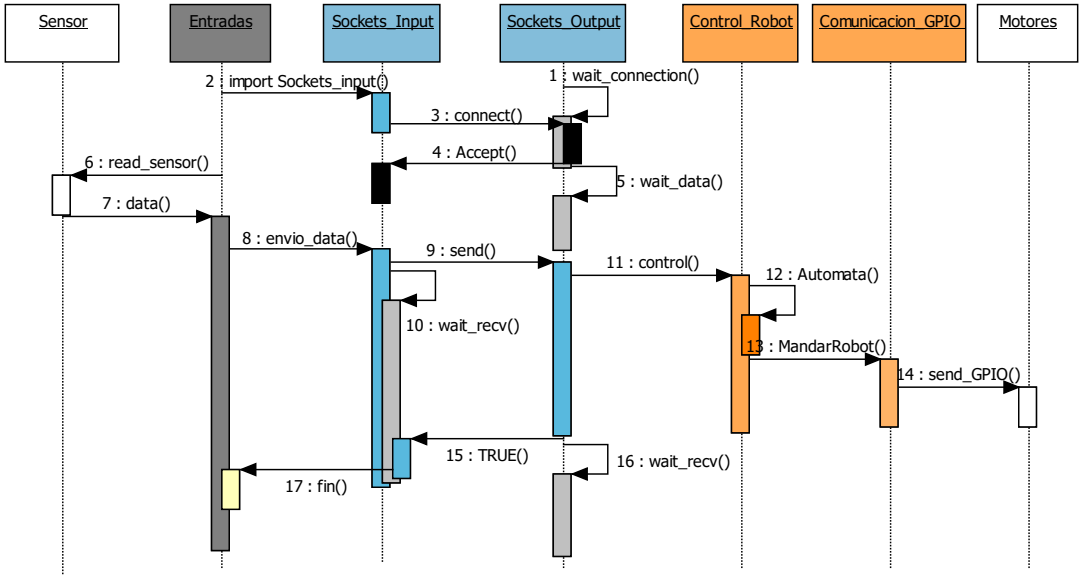



Fig. 23 Diagrama de flujo del sistema

Lo primero que se debe ejecutar es *Sockets_output.py*, que va a esperar hasta que algún usuario se conecte (1). Cuando se lanza el código *Entradas*, que representa el fichero *Acelerometros.py* o

*Camara.py*, se importa *Sockets_input.py* (2) y, cuando se inicializa este código, es cuando se establece la conexión entre ambos subsistemas (3, 4).

*Sockets_output.py* queda esperando hasta recibir un dato (5). *Entradas* lee los datos del sensor (6, 7) y los transforma en datos legibles para el sistema. Los envía por Sockets (9) tras realizar una llamada a *Sockets_input.py* (8). *Sockets_input.py* queda esperando hasta recibir confirmación del otro subsistema (10).

Cuando *Sockets_output.py* recibe los datos, los envía a *ControlRobot.py* (11) para determinar el tratamiento de ese dato. Con ayuda del código *Automata.py* (12) se determina el comportamiento de los motores y se llama a *ComunicacionGPIO.py* para que transmita la orden a los motores (14).

Cuando este proceso acaba, *Sockets_output.py* indica al subsistema entradas que ya está listo para analizar otro dato (15) y vuelve a quedar en espera hasta la recepción de otro dato (16).

Los datos enviados de *Sockets_input.py* a *Sockets_output.py* tienen un formato diccionario llamado pautas, mientras que lo que se envía de *Sockets_output.py* a *Sockets_input.py* es un *True*, confirmación de haber procesado el dato.

### 3.3.2. Tratamiento de los datos: Bloque de control

El bloque de control representa al procesamiento de las órdenes recibidas en el subsistema de salida. Está compuesto de los ficheros *ControlRobot.py*, *Automata.py* y *ComunicacionGPIO.py*.

Los datos recibidos en este subsistema son tres: *modo*, *direccion* y *velocidad*. La variable *modo* puede tener los valores 0, 1, 2, 3, 4, 5 y 999, que hacen referencia a M0, M1, M2, M3, M4, M5 y M999. la variable *direccion* acepta los valores -1, 0 y 1, que indica izquierda, recto y derecha del robot respectivamente. La variable *velocidad* puede tener un valor comprendido entre -100 y 100, que indicará el ciclo de trabajo de la señal PWM que se enviará a los motores, siendo los valores positivos movimiento hacia delante y los negativos marcha atrás.

Estos tres datos se recogen en un diccionario llamado *pautas* en el subsistema de entrada y es la variable que se recibe y procesa en este bloque de control. Esta variable debe admitir únicamente los datos contemplados en la Tabla 8 para que el subsistema salidas pueda interpretarlos.

| **Valores permitidos** | | | |
|---|---|---|---|
| ***modo*** | ***direccion*** | ***velocidad*** | **Resultado** |
| 0 | -1, 0 y 1 | -100 .. 100 | Ejecución de los movimientos básicos. |
| 1 | -1 y 1 | 0 | Indica cambio de estado E1, giro en sentido horario o antihorario según *direccion*. |
| 2 | 0 | 0 | Indica cambio de estado E2. |
| 3 | 0 | 0 | Indica cambio de estado E3. |
| 4 | 0 | 0 | Indica cambio de estado E4. |
| 5 | 0 | 0 | Indica señal M5, que permite la vuelta al E0 cuando se encuentra en E4 o E3. |
| 999 | 0 | 0 | Indica señal desconocida o errónea en la entrada, sirve para el control de errores. |

Tabla 8 Valores permitidos de la variable pautas

Para procesar la variable *pautas*, *ControlRobot.py* hace uso de *Automata.py* para conocer en qué estado se encuentra el sistema y si debe hacer algún cambio de estado en base a la entrada actual. Con ayuda de estos resultados, *ControlRobot.py* trata la entrada de un modo u otro según se mantenga, salga o comience la ejecución de un estado. Tras este procesamiento, *ControlRobot.py* trasmite la orden final a *ComunicacionGPIO.py* para que comande los motores pertinentemente.

Los cinco estados posibles tratados en *ControlRobot.py* y las distintas transiciones con condición definidos en el autómata programado cumplen el diagrama de estados de la Fig. 23.

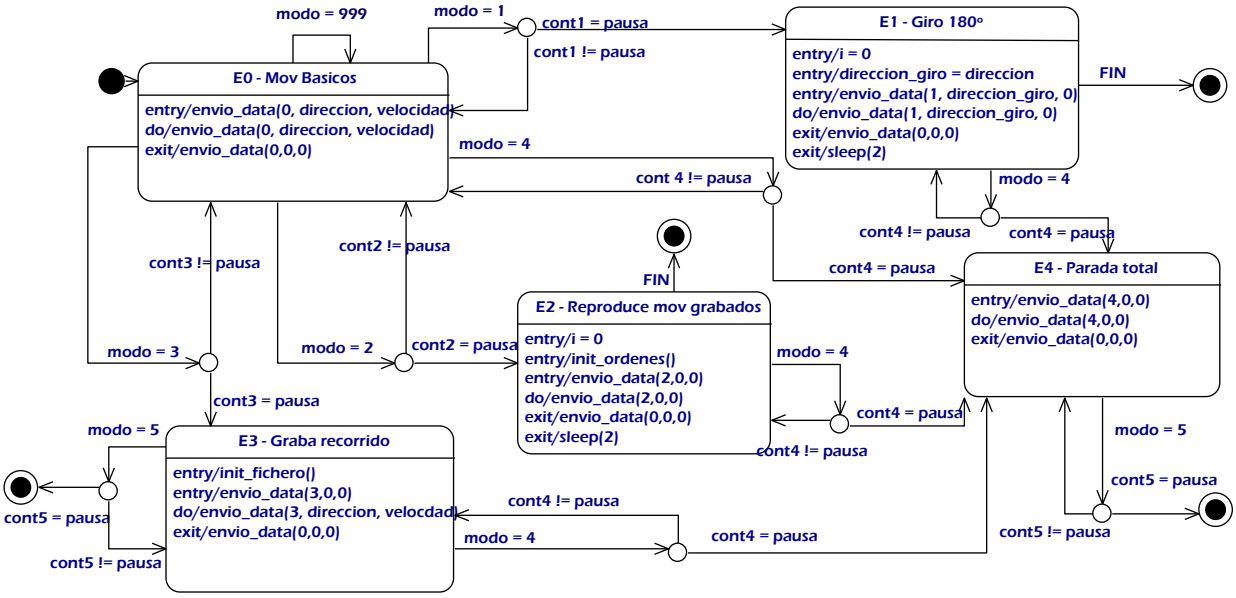



Fig. 24 Diagrama de estados del autómata del sistema SysML

Hay que recalcar que la comunicación es constante, por lo que los estados finales se han definido con el único fin de simplificar el esquema. Todos los estados finales que se muestran en el diagrama representan la vuelta al E0, junto con el envío de parada al robot (*velocidad* = 0, *direccion* = 0).

## Condiciones para las transiciones

Hay dos tipos de transiciones de estado en este autómata: las solicitadas por el usuario a través de la variable *modo* y las que indica el sistema tras el fin de la ejecución de una secuencia de movimientos predefinida.

```
self.lista_modos = [0, 1, 2, 3, 4, 5]
self.pausa = 35
self.contador = {modo: 0 for modo in self.lista_modos}
self.transiciones = {
    Estado.ESTADO0: {   1: (False , Estado.ESTADO1),
                        2: (False, Estado.ESTADO2),
                        3: (False, Estado.ESTADO3),
                        4: (False, Estado.ESTADO4)},

    Estado.ESTADO1: {   'FIN': (True, Estado.ESTADO0),
                        4: (False, Estado.ESTADO4)},

    Estado.ESTADO2: {   'FIN': (True, Estado.ESTADO0),
                        4: (False, Estado.ESTADO4)},

    Estado.ESTADO3: {   4: (False, Estado.ESTADO4),
                        5: (False, Estado.ESTADO0)},

    Estado.ESTADO4: {   5: (False, Estado.ESTADO0)}
    }
```

Fig. 25 Fragmento de Automata.py: Transiciones factibles

- Las transiciones dependientes de la variable ***modo*** se cumplen solo si se recibe esa misma señal un número de veces seguidas igual a la variable *pausa* definida. La variable *pausa* es igual al número de señales que procesa el robot en aproximadamente 3 segundos. El control de la cuenta de entradas se realiza a través de un array de contadores para cada modo (**Fig. 26**). Este contador ayuda a verificar que la orden del usuario es clara y no una señal errónea, donde,

mientras exista transición y no transite, el robot quedará quieto(**Fig. 25**). La variable pausa se puede ajustar según las necesidades de cada usuario.

- ***FIN*** es una señal que se recibe en el autómata cuando *ControlRobot.py* quiere indicar que el robot ha terminado la ejecución del M1 o del M2. Estos son los modos que cuentan con movimientos predefinidos del robot, es decir, que no requieren de indicaciones por parte del usuario para su ejecución. En cuanto se recibe esta señal se realiza el cambio de estado.

```
def transitar(self, modo):
    if modo in self.transiciones[self.estado_actual]:
        cond, nuevo_estado = self.transiciones[self.estado_actual][modo]
        if not cond:
            cond = self.cuenta_entradas(modo)
        if cond:
            self.estado_actual = nuevo_estado
            self.tiempo_reset()
            return True, True
        else: return True, False
    else: return False, False
```

Fig. 26 Fragmento de Automata.py: Función *transitar()*

```
def cuenta_entradas(self, modo):
    self.contador[modo] += 1
    return self.contador[modo] >= self.pausa

def tiempo_reset(self):
    for i in self.contador:
        self.contador[i] = 0
```

Fig. 27 Fragmento de Automata.py: Tratamiento del array de contadores

### Estados y transiciones

Para aclarar la imagen del diagrama de estados (Fig. 23), se explica con palabras y tablas resumen el movimiento que representa cada estado en el robot, cuáles son las transiciones de estado factibles y las acciones del estado ejecutadas en el fichero *ControlRobot.py*:

- El **E0** representa el estado inicial donde el usuario podrá ejecutar los movimientos básicos. En este estado se permite transitar hacia cualquier otro estado. Si el bloque de control recibe un valor de *modo* distinto de 0 o 5, en *Automata.py* se iniciará la cuenta para el cambio de estado. Cuando la cuenta llegue al valor determinado por *pausa*, se realizará la transición. En este estado, *ControlRobot.py* simplemente manda a *ComunicacionGPIO.py* la velocidad y dirección que debe tomar el robot (Fig. 25).

| | | | |
|---|---|---|---|
| **Autómata** | **Transiciones en E0:** Entrada and condición → nuevo estado | | |
| | **Entrada** | **Condicion** | **Nuevo estado actual** |
| | M1 | *contador*[1]=*pausa* | E1 |
| | M2 | *contador*[2]=*pausa* | E2 |
| | M3 | *contador*[3]=*pausa* | E3 |
| | M4 | *contador*[4]=*pausa* | E4 |
| **ControlRobot.p** | **Acción de entrada al E0** | Se envía al robot señal de parada. | |
| | **Acción dentro del E0** | Se envía la dirección y velocidad determinada por el usuario, al robot. | |

Tabla 9 Operaciones en estado 0

```
if modo == 0:
    m.mandar_robot(direccion, velocidad)
```

Fig. 28 Acción dentro de E0 en *ControlRobot.py*

- El **E1** representa el proceso de ejecución del modo 1 o el giro 180º. En este estado el usuario solo podrá ejecutar la orden de parada (M4) para detener el movimiento, de lo contrario, se mantiene en E1 hasta que el robot complete el giro y el sistema establezca la variable *fin_modo* a *True*. En este estado, *ControlRobot.py* realiza el conteo de instrucciones de giro mandadas a *ComunicacionGPIO.py* para poder mantener activa la comunicación con el usuario y, cuando acaba, establece *fin_modo* a *True* para indicar al autómata el final de la ejecución de este estado (Fig. 27). Para entrar a este estado será necesario guardar la dirección de giro deseada por el usuario y poner el contador *i* a 0 (Fig. 26).

| | | | |
|---|---|---|---|
| **Automata.py** | **Transiciones en E1:** Entrada and condición → nuevo estado | | |
| | **Entrada** | **Condición** | **Nuevo estado** |
| | M4 | *contador*[4]=*pausa* | E4 |
| | *'FIN'* | *True* | E0 |
| **ControlRobot.py** | **Acción de entrada al E1** | Se guarda la dirección a la que el robot debe hacer el giro de 180º y se inicializa la variable *i*, que servirá de contador auxiliar como tiempo de giro. | |
| | **Acción dentro del E1** | Se manda señal de giro hacia la dirección guardada un número de veces. Al usuario se le escucha continuamente, pero solo se le permite la orden de M4. Cuando acaba, se espera 2 segundos en parada y vuelve a M0. *Fin_modo = True.* | |

Tabla 10 Operaciones en estado 1

```
if e_act == 1:
    direccion_giro = direccion
    i = 0
```

Fig. 29 Acción de entrada a E1 en *ControlRobot.py*

```
elif modo == 1:
    if i < 25:                          # Aproximacion de instrucciones para que el robot gire 180°
        m.mandar_robot(direccion_giro, 0)
        i += 1
        fin_modo = False
    else:
        m.mandar_robot(0, 0)
        time.sleep(2)
        fin_modo = True
```

Fig. 30 Acción dentro de E1 en *ControlRobot.py*

- El **E2** representa el proceso de ejecución del M2 o "ejecución de movimiento pregrabado". Este estado se parece a E1, de modo que solo transita si el usuario desea abortar la ejecución del movimiento (mediante la señal M4), o se completa la lectura del fichero de los datos de movimiento del recorrido (*movimientos_Pregrabados*.txt), donde el sistema establece *fin_modo* a *True* para indicar al autómata el fin del estado (Fig. 29). Al entrar en este estado se

debe inicializar *ordenes* que es la variable que contempla el número de líneas escritas en el fichero de texto, así como el contador auxiliar *i* (Fig. 28).

| | | | |
|---|---|---|---|
| Automata.py | **Transiciones en E2:** Entrada and condición → nuevo estado | | |
| | **Entrada** | **Condición** | **Nuevo estado** |
| | M4 | *contador*[4]=*pausa* | E4 |
| | *'FIN'* | *True* | E0 |
| ControlRobot.py | **Acción de entrada al E2** | Se cuenta las líneas escritas en fichero *Movimientos_Pregrabados.txt*, se guarda el dato en *ordenes* y se inicializa la variable *i*, que servirá de contador auxiliar. | |
| | **Acción dentro del E2** | Se leen una a una las líneas del fichero para mantener activa la comunicación con el usuario y se manda la orden al robot, a la vez que se incrementa el contador *i* hasta que *i = ordenes*. Cuando acaba, se espera 2 segundos en parada y vuelve a M0. *Fin_modo = True*. | |

Tabla 11 Operaciones en estado 2

```
elif e_act == 2:
    init_ordenes()
    i = 0
```

Fig. 31 Acción de entrada a E2 en ControlRobot.py

```
elif modo == 2:
    if i < ordenes:
        direccion, velocidad = leer_orden()
        m.mandar_robot(direccion, velocidad)
        i += 1
        fin_modo = False
    else:
        print("FIN FICHERO")
        m.mandar_robot(0, 0)
        time.sleep(2)
        fin_modo = True
```

Fig. 32 Acción dentro de E2 en ControlRobot.py

- El **E3** representa el proceso de ejecución de M3 o "grabación de movimiento". En este estado solo se transita cuando se quiere cancelar la grabación (señal M4) o se quiere dejar de grabar y guardar el fichero de movimientos (señal M5). Al transitar a este estado, se vacía el fichero (Fig. 30) y dentro de este estado, *ControlRobot.py* enviará los movimientos a *ComunicacionGPIO.py* para que los ejecute y también les escribirá en el fichero *Movimientos_Pregrabados.txt (*Fig. 31*)*.

| | | | |
|---|---|---|---|
| Automata.py | **Transiciones en E3:** Entrada and condición → nuevo estado | | |
| | **Entrada** | **Condición** | **Nuevo estado** |
| | M4 | *contador*[4]=*pausa* | E4 |
| | M5 | *contador*[5]=*pausa* | E0 |
| ControlRobot.p | **Acción de entrada al E3** | El fichero "*Movimientos_Pregrabados.txt*" se vacía. | |
| | **Acción dentro del E3** | Se manda la velocidad y el movimiento mandado por el usuario al robot a la vez que se escribe en el fichero. | |

Tabla 12 Operaciones en estado 3

```
elif e_act == 3:
    init_fichero()
```

Fig. 33 Acción de entrada a E3 en ControlRobot.py

```
elif modo == 3:
    m.mandar_robot(direccion, velocidad)
    escribir_orden(direccion, velocidad)
```

Fig. 34 Acción dentro de E3 en ControlRobot.py

- El **E4** representa el proceso de ejecución de M4 o "parada absoluta". En este estado se permanece cuando el usuario ha decidido realizar una parada de emergencia. Solo se retomará el movimiento cuando el usuario indique la señal M5, donde se volverá al E0. Si al transitar a E4 el estado anterior fue E3, en este cambio se vacía el fichero de *Movimientos_Pregrabados.txt (*Fig. 32*)* y durante este estado *ControlRobot.py* le mandará continuas ordenes de parada a *ComunicacionGPIO.py* para mantener activa la escucha al usuario (Fig. 33).

<table>
<tr><td rowspan="2">Automata.py</td><td colspan="3">Transiciones en E4: Entrada and condición → nuevo estado</td></tr>
<tr><td>Entrada</td><td>Condición</td><td>Nuevo estado</td></tr>
<tr><td></td><td>M5</td><td>contador[5]=pausa</td><td>E0</td></tr>
<tr><td rowspan="2">ControlRobot.py</td><td colspan="2">Acción de entrada al E4</td><td>Si el estado anterior fue el E3, se vacía el fichero Movimientos_Pregrabados.txt y se envía parada al robot. Si el estado anterior es cualquier otro, solo se envía parada al robot.</td></tr>
<tr><td colspan="2">Acción dentro del E4</td><td>Manda parada al robot constante mientras se sigue escuchando al usuario en espera de recibir señal M5.</td></tr>
</table>

Tabla 13 Operaciones en estado 4

```
elif e_act == 4 and e_ant == 3:
    init_fichero()
```

Fig. 35 Acción de entrada a E4 tras encontrarse en E3 en ControlRobot.py

```
elif modo == 4:
    m.mandar_robot(0, 0)
```

Fig. 36 Acción dentro de E4 en ControlRobot.py

Para que durante las transiciones entre modos el robot se mantenga quieto, *Automata.py* manda a *ControlRobot.py* dos booleanos: uno indica si existe transición (*existe_t*) y otro si transita (*realiza_t*). Si no existe transición, entonces el robot se mantendrá en el estado que estaba. Si realiza la transición, se analiza si debe hacerse alguna acción de cambio de estado (*gestión_transiciones()*) y si existe la transición y no transita, entonces se manda señal de parada al robot para que se mantenga en espera hasta que se complete la señal de la transición (Fig. 34).

```
if not existe_t:
    envio_data(e_act, direccion, velocidad)

elif realiza_t:
    gestion_transiciones(e_act, e_ant)

else: envio_data(0, 0, 0)
```

Fig. 37 Fragmento de código para evaluar las transiciones en ControlRobot.py

Como se puede apreciar, cada entrada que se recibe por el subsistema de entrada se interpreta en *ControlRobot.py* y se trasmite la orden final de dirección y velocidad que ha de cumplir el robot a *ComunicacionGPIO.py* mediante la función *mandar_robot(direccion, velocidad).*

### Salidas finales GPIO

El archivo *ComunicacionGPIO.py* es el encargado de enviar, a través del protocolo GPIO, el porcentaje de ciclo de trabajo PWM correspondiente a la velocidad final de cada motor.

Para comprender mejor la programación de este fichero, es necesario comprender cómo se han establecido las conexiones del hardware entre la Raspberry y el Robot:

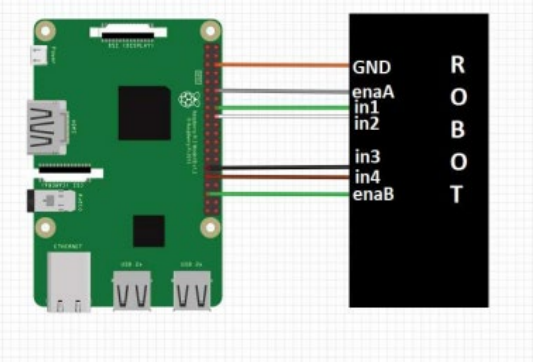



Fig. 38 Diagrama de conexiones del subsistema de salida **[8]**

En la imagen anterior se ven los pines utilizados para la comandancia de los motores. Para el giro de la rueda derecha, *in1* indicará el movimiento hacia delante e *in2* el movimiento hacia atrás. Para el movimiento de la rueda izquierda, *in3* indicará el movimiento hacia detrás e *in4* el movimiento hacia delante.

El montaje del robot, incluyendo sus componentes y la etapa de control de potencia para los motores, se desarrolló en la tesis de Arash Soofi Alizadeh [9]. Posteriormente, Alba Payo [8] utilizó este robot y parte de su código, adaptándolo y ampliándolo con nuevas funciones en su propio proyecto. En este trabajo, se ha tomado como base el código de Alba Payo, donde define los movimientos básicos del robot y se ha modificado para incorporar un nuevo movimiento: el giro en el sitio.

Esta nueva función es necesaria para realizar el giro de 180 grados en sentido horario y antihorario y se ejecutará cuando la velocidad sea cero y la dirección sea distinta de cero.

```
def giro_180_d(self):
        # SENTIDO HORARIO
    GPIO.output(self.In1, GPIO.LOW)
    GPIO.output(self.In2, GPIO.HIGH)
    GPIO.output(self.In3, GPIO.LOW)
    GPIO.output(self.In4, GPIO.HIGH)
    self.pwmMotorA.ChangeDutyCycle(50)
    self.pwmMotorB.ChangeDutyCycle(50)
```

Fig. 39 Función para el giro de 180 grados en el sitio en sentido horario

Para hacer posible este movimiento, ambos motores deben girar a la misma velocidad, el motor de la derecha gira hacia adelante y el motor de la izquierda gira hacia atrás (Fig. 37). El sentido antihorario se ha programado de la misma manera, pero el movimiento de los motores será el contrario.

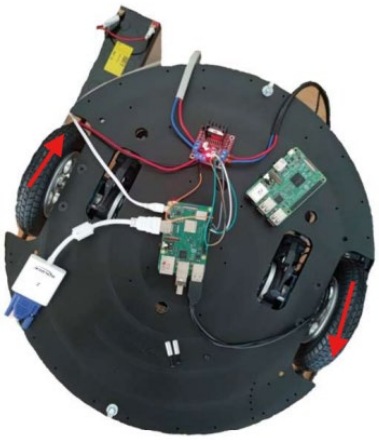


Fig. 40 Movimiento de las ruedas para giro horario

### 3.3.3. Interfaces de entrada

Las interfaces de entrada que admite el sistema son todas aquellas que puedan traducir datos obtenidos del usuario mediante sensores, teclas o mandos a los datos de entrada que admite el sistema, explicados en la tabla (Tabla 8). Se pueden usar todos los modos programados o elegir el que más convenga, según la cantidad de movimientos que se permitan en el hardware empleado.

En este proyecto se ha adecuado la entrada de dos acelerómetros situados en las manos para leer y traducir los movimientos a todos los modos programados, y la interfaz de cámara de los movimientos básicos de Alba Payo [8] se ha adaptado para que funcione mediante el nuevo sistema para demostrar la fácil usabilidad y escalabilidad del mismo.

#### Acelerómetros

La interfaz de los acelerómetros consta de un fichero de inicialización de las posiciones máximas y mínimas de ambas manos tanto sobre el eje *x* como el eje *y (Setup_code.py,* Tabla 7*).* Lo que se obtiene tras la ejecución de este código son las variables máximas y mínimas en la rotación de las muñecas tanto sobre el eje *x* como sobre el eje *y,* que se recogen en un fichero de texto (*setup_file.txt*) del que hará uso el código principal de la interfaz (*Acelerometros.py*).

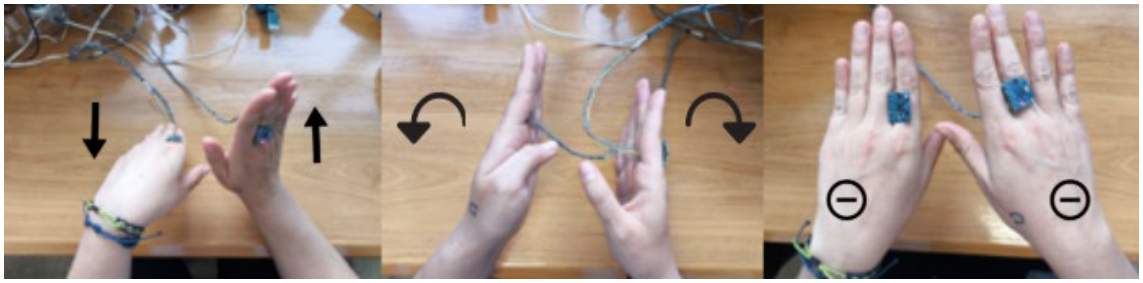


Fig. 41 Ejemplo movimientos: rotación sobre el eje x, rotación sobre el eje y y posición neutra

Para el código Acelerometros.py se han utilizado las siguientes librerías:

```
import RPi.GPIO as GPIO
from mpu6050 import mpu6050
import math
import time
import sys
#Importación fichero comunicación por sockets
import Sockets_input as sk
```

Fig. 42 Librerías importadas en Acelerometros.py

**RPi.GPIO** es la librería que va a permitir a la Raspberry la lectura de datos de los acelerómetros en sus puertos de entrada/salida.

**Mpu6050** va a permitir obtener los datos de posición *x*, *y* y *z* de cada acelerómetro.

**Math** sirve para poder ejecutar los cálculos de la posición para obtener los distintos giros de muñeca.

**Time** ejecuta una pequeña espera en la lectura de los acelerómetros para no saturar el sistema de envío.

Por último, la librería **Sys** se ha utilizado para permitir el control de errores.

Para recibir los datos de posición de ambos acelerómetros, hay dos tipos de control de error:

```
while True:
    try:
#Leemos los datos de los acelerómetros
        x, y, z = leer_datos(mpu.get_accel_data())
        x2, y2, z2 = leer_datos(mpu2.get_accel_data())
##Control de error división por cero, si llega una entrada mal leida,
#                       llega en formato  (x = 0.0, y = 0.0, z = 0.0)
        if (x == z and y == z) or (x2 == z2 and y2 == z2): error = True
```

Fig. 43 Tratamiento de errores en Acelerometros.py: try

El primero consiste en intentar leer los datos de los acelerómetros, si hay algún error en el hardware, no se podrán leer, con lo que saltará una excepción que indicará el error y enviará los datos a cero para que se mantenga el robot quieto y evitar desconexiones del programa (Fig. 42, Fig. 43).

El segundo control se encarga de procurar no dar fallo en las funciones matemáticas del código 1 y 2 . Cuando algún acelerómetro ha trasmitido algún dato extraño, esto se interpreta como que el eje *x*, *y* y *z* tienen el valor 0.0, con lo cual, al calcular la posición resulta una división entre cero que el sistema no soporta (Fig. 42).

Para evitar que salte ese error en el sistema, se interpreta directamente como M999, que es el que indica fallo en la interfaz de entrada. Este modo el sistema de salida lo interpretará como que debe mantener quieto al robot (Fig. 43).

```
        else: #Hay error (se ha leído mal la entrada)
            modo = 999
            velocidad = 0
            direccion = 0
            error = False
        sk.envio_data(modo, direccion, velocidad)
#Si salta error en la mpu6050, se captura para finalizar
#la ejecución del código y que queden quietos los motores
except Exception:
        print("Error entrada")
        sk.envio_data(0,0,0)
```

Fig. 44 Tratamiento de errores en *Acelerometros.py*: *except* y modo 999

Si el dato leído pasa los controles de errores, se calcula para ambas manos el ángulo de rotación en el eje x y el ángulo de rotación en el eje y de las siguientes maneras:

$$y\_rotation = \tan^{-1}\left(\frac{y}{\sqrt{x^2 + z^2}}\right)$$

1 Ecuación del ángulo de rotación sobre el eje y

$$x\_rotation = \tan^{-1}\frac{x}{\sqrt{y^2 + z^2}}$$

2 Ecuación del ángulo de rotación sobre el eje x

Con los ángulos de rotación obtenidos, se va comparando con los datos máximos y mínimos registrados en *Setup_file.txt* para obtener la posición final de cada mano.

En la siguiente tabla, se resumen los movimientos simultáneos de ambas manos para ejecutar cada movimiento, incluyendo todos los modos, siendo los valores finales de la variable *pautas* los detallados en la Tabla 8:

<table>
<tr><th>Modo</th><th>Mano izquierda</th><th>Mano derecha</th></tr>
<tr><td rowspan="2">M0 – Ejecución de movimientos básicos.</td><td>Controla la velocidad aplicada al robot. Si la muñeca izquierda rota sobre el eje x hacia arriba, se consigue una velocidad positiva (de 1 a 100 según la amplitud del ángulo) y el robot avanzará hacia delante, mientras que, si rota hacia abajo, la velocidad será negativa (de -1 a -100) y el robot se moverá marcha atrás. Si la palma de la mano queda paralela al suelo, el robot se quedará quieto (velocidad igual a 0).</td><td>Controla la dirección de giro del robot. Si la muñeca derecha rota sobre el eje y a la derecha, indica al robot un giro a la derecha (la dirección es 1), si rota hacia la izquierda, el robot gira a la izquierda (la dirección es -1). Si la palma de la mano queda paralela al suelo, indica no giro y mantendrá la dirección (la dirección es 0).</td></tr>
<tr><td colspan="2">  <br>Fig. 45 Ejemplo movimiento básico giro derecha hacia delante</td></tr>
<tr><td rowspan="2">M1 – Movimiento de giro 180º</td><td>La muñeca izquierda rota sobre el eje y hacia la izquierda si se quiere un movimiento de 180º en sentido antihorario (dirección es -1) y gira a la derecha si el sentido se quiere horario (dirección es 1). Se mantiene posición varios segundos para asegurar el cambio.</td><td>La muñeca derecha rota sobre el eje x hacia arriba. Se mantiene posición varios segundos para asegurar el cambio.</td></tr>
<tr><td colspan="2">  <br>Fig. 46 Ejemplo solicitud M1 giro antihorario</td></tr>
<tr><td>M2 – Ejecutar recorrido pregrabado</td><td>La muñeca izquierda rota sobre el eje x hacia abajo. Se mantiene posición varios segundos para asegurar el cambio.</td><td>La muñeca derecha rota sobre el eje x hacia arriba. Se mantiene posición varios segundos para asegurar el cambio.</td></tr>
</table>

<table>
<tr><td></td><td colspan="2">  <br>Fig. 47 Ejemplo solicitud M2</td></tr>
<tr><td rowspan="2">M3 – Iniciar grabación de recorrido</td><td>La muñeca izquierda rota sobre el eje x hacia arriba. Se mantiene posición varios segundos para asegurar el cambio.</td><td>La muñeca derecha rota sobre el eje x hacia abajo. Se mantiene posición varios segundos para asegurar el cambio.</td></tr>
<tr><td colspan="2">  <br>Fig. 48 Ejemplo solicitud M3</td></tr>
<tr><td rowspan="2">M4 – Cancelar movimiento</td><td>La muñeca izquierda rota sobre el eje x hacia arriba. Se mantiene posición varios segundos para asegurar el cambio.</td><td>La muñeca derecha rota sobre el eje x hacia arriba. Se mantiene posición varios segundos para asegurar el cambio.</td></tr>
<tr><td colspan="2">  <br>Fig. 49 Ejemplo solicitud M4</td></tr>
<tr><td rowspan="2">M5 – Terminar grabación || Volver a M0 tras M4.</td><td>La muñeca izquierda rota sobre el eje x hacia abajo. Se mantiene posición varios segundos para asegurar el cambio.</td><td>La muñeca derecha rota sobre el eje x hacia abajo. Se mantiene posición varios segundos para asegurar el cambio.</td></tr>
<tr><td colspan="2">  <br>Fig. 50 Ejemplo solicitud M5</td></tr>
</table>

Tabla 14 Movimientos de las manos para el control con acelerómetros

### Cámara

Para adaptar el código de Alba Payo en la interpretación del movimiento de la cabeza a través de una cámara [8], se han aprovechado las funciones del procesamiento de los distintos movimientos de la cabeza para los movimientos básicos y se ha suprimido todo el código que implicaba el envío de información mediante sockets.

De este modo, una vez interpretado el movimiento deseado, se envía la dirección y velocidad a la que debe operar el robot, siempre junto al modo cero gracias a la función *sk.envio_data(0, direccion, velocidad)* del fichero *Sockets_input.py*.

De este modo, se evitan las librerías utilizadas para el envío y solo se requerirán las que controlan el análisis y cálculos de imagen (*cv2*, *mediapipe*, *numpy* y *math*) y la de comunicación de entrada del sistema (Sockets_input.py) siguientes:

```
#Importación de librerías
import cv2
import mediapipe as mp
import numpy as np
from math import acos,degrees
# Importación del fichero de comunicación por sockets
import Sockets_input as sk
```

Fig. 51 Librerías importadas en el código de MovimientosCabeza.py

Los movimientos que permite este código serán los movimientos básicos a una velocidad constante. Se ejecutan de la siguiente manera:


Fig. 52 Movimientos posibles mediante la cámara **[8]**

Para poder utilizar esta interfaz con comodidad, se recomienda instalar *DroidCamApp* en un teléfono móvil y en el ordenador donde se ejecuta el código, para tener una cámara móvil mucho más manejable, esto le aporta mayor usabilidad al sistema. La explicación detallada del uso de esta aplicación queda explicada en la pag.23.

### Plantilla para la incorporación de nuevas interfaces de usuario

Como se puede comprobar por las distintas interfaces de entrada que pueden comandar al robot, el sistema es adaptable fácilmente a nuevas interfaces de control con diverso hardware. En este punto se explican los pasos a seguir para poder controlar al robot mediante una nueva interfaz:

1. Importar la librería de comunicación por sockets del sistema:
   ```
   import Sockets_input.py as sk
   ```
2. Iniciar un bucle infinito para los puntos 3, 4 y 5.
   ```
   While True:
   ```
3. Leer entradas.
4. Analizar entradas y convertirlas a datos legibles por el sistema de control (Tabla 8).
5. Enviar los datos a través de la función *envio_data()* definida en el fichero *Sockets_input.py* de la siguiente manera:
   ```
   sk.envio_data(modo, direccion, velocidad)
   ```

NOTA: En la librería *Sockets_input.py*, se puede modificar la línea *time.sleep()* de acuerdo con la frecuencia de envío que se desee.

### 3.3.4. Módulo de comunicación

El módulo de comunicación está compuesto de los ficheros *Sockets_input.py*, que actúa en el lado del cliente, usuario, para solicitar la conexión y realizar los envíos y *Sockets_output.py*, que se ejecuta en el lado del servidor, robot, y se encarga de validar la conexión y trasmitir los envíos a que sean analizados por *ControlRobot.py*.

En el lado del cliente será necesario determinar la IP del servidor a la que va a enviar la solicitud, así como su nombre y el puerto. Se crea un contexto SSL para requerir la validación al servidor, para ello se necesita definir las rutas donde se sitúan los certificados público y privado del cliente y el público del servidor.

```
HOST = 'ROBHELPHUM'          #          Nombre del servidor
IP_HOST = 'localhost'        #          Direccion IP del servidor
PORT = 50001                 #          Puerto al que enviar los datos

# Definición de la ubicación de los certificados SSL
CLIENT_CERT = 'certificados_u/user_cer.cer'
CLIENT_KEY = 'certificados_u/user_k.key'
SERVER_CERT = 'certificados_u/robhelphum_cer.cer'

# Creacion del contexto SSL
context = ssl.create_default_context(ssl.Purpose.SERVER_AUTH)        ##
context.load_cert_chain(certfile=CLIENT_CERT, keyfile=CLIENT_KEY)    ##
context.load_verify_locations(cafile=SERVER_CERT)                    ##
```

Fig. 53 Definición de rutas y creación del contexto SSL en *Socket_input.py*

Después de definir las rutas y el contexto SSL, se crea el socket con la IP y puerto de la comunicación y se hace seguro comprobando que el nombre de la máquina servidor, en este caso *"ROBHELPHUM"*, sea el esperado. Por último, se comprueba que el servidor responde con el certificado válido.

```
# Creacion del socket
s = socket.create_connection((IP_HOST, PORT))
# Creacion del socket seguro
c_sec = context.wrap_socket(s, server_hostname=HOST)

# Revision de certificados
try:
        # Verificar el nombre del host
    ssl.match_hostname(c_sec.getpeercert(), HOST)
    print("La conexión SSL es segura.")
except ssl.CertificateError as e:
    print(f"Error de certificado: {e}")
    sys.exit(1)
```

Fig. 54 Creación del socket seguro en *Socket_input.py*

En este fichero se encuentra definida la función *envio_data(modo, direccion, velocidad)*, que es llamada por la interfaz para conseguir enviar los datos al otro sistema. En esta función, después del envío a través del socket seguro, se espera una confirmación del otro sistema que indicará que ha procesado el dato y está listo para recibir uno nuevo. Cuando acaba la función, el fichero de la interfaz continuará con la lectura de una nueva entrada.

Se han considerado dos excepciones en la función *envio_data* para cerrar el sistema de forma segura tras una mala lectura de entrada/salida o de una parada voluntaria de ejecución del programa. Con esto, el cliente se desconectará del socket y el otro subsistema se encargará de finalizar de forma segura.

```
# Tratamiento de excepciones
except IOError:
    print("Error I/O")
    sys.exit(1)

except KeyboardInterrupt:
    print("Detenido")
    sys.exit(1)
```

Fig. 55 Tratamiento de excepciones en *envio_data*

En el lado del servidor también se requiere definir la ruta de los certificados, en este caso, la clave privada y pública del servidor y la clave pública del cliente, al igual que se define cual es la IP de escucha del host (en este caso se deja vacía para que escuche a través de todas las interfaces de red que tiene disponibles; equivalente a definirlo como 0.0.0.0) y el puerto de escucha. También se define el hash del certificado público del único cliente al que se le va a conceder la conexión (este hash se obtiene como se explica en la sección de OpenSSL, pág. 26).

```
HOST = ''           #           Direccion de escucha (cualquier red disponible)
PORT = 50001        #           Puerto por donde escucha para recibir los datos.

# Definición de la ubicación de los certificados SSL
SERVER_CERT = 'certificados_r/robhelphum_cer.cer'
SERVER_KEY = 'certificados_r/robhelphum_k.key'
USER_CERT = 'certificados_r/user_cer.cer'

# Huella digital (hash) permitida (del certificado del único usuario autorizado)
ALLOWED_FINGERPRINT = "705C1F39FD670F4E174D9A26DCE7038632525EE51CA6930D0DB192A5C076DC48"
```

Fig. 56 Definición de variables usadas para la conexión en *Socket_Output.py*

Se crea el socket y se mantiene en escucha a la IP y puerto en espera a que algún cliente solicite la conexión. Se crea el contexto SSL para que se requiera una comprobación de certificado del cliente antes de la trasmisión de datos y se cargan los certificados.

```
# Creacion del socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((HOST, PORT))
s.listen(1)

# Creacion del contexto SSL
context = ssl.create_default_context(ssl.Purpose.CLIENT_AUTH)
context.verify_mode = ssl.CERT_REQUIRED
context.load_cert_chain(certfile=SERVER_CERT, keyfile=SERVER_KEY)
context.load_verify_locations(cafile=USER_CERT)
```

Fig. 57 Creación de socket y contexto SSL en *Socket_output.py*

Se utiliza la sentencia *'with'* para envolver el socket en el protocolo SSL y poder finalizar este socket limpiamente si da el caso de que falle la conexión o el programa. En esta sentencia, se acepta la conexión y se calcula la huella digital o hash del certificado recibido para compararlo con el que se admite válido.

```
with context.wrap_socket(s, server_side=True) as s_sec:
    conn, addr = s_sec.accept()
    # Obtener certificado del cliente
    client_cert = conn.getpeercert(binary_form=True)
    with conn:
        try:                                    # Intenta conexión
            if client_cert:
                # Calcular la huella digital SHA-256 del certificado recibido
                fingerprint = hashlib.sha256(client_cert).hexdigest().upper()
                print(f"Fingerprint recibido: {fingerprint}")
                if fingerprint != ALLOWED_FINGERPRINT:
                    print("Conexion rechazada: Certificado de cliente no autorizado.")
                    conn.close()
                else:
                    print("Conexion autorizada: Certificado valido.")
```

Fig. 58 Comprobación de certificado permitido en *Socket_output.py*

Tras este proceso, la conexión quedará establecida y encriptada y se recibe el primer dato dentro del bucle infinito, se trasmite a *ControlRobot.py* y, cuando finaliza el procesamiento, se envía la confirmación de estar listo para recibir un nuevo dato. El bucle finalizará cuando haya un fallo en la conexión.

```
while True:

    data = conn.recv(1024)
    if not data:break
    pautas = ast.literal_eval(data.decode('utf8'))
-------------------------------------------------#####
    e.control(pautas)
-------------------------------------------------####
    conn.send(bytes('True','utf-8'))
```

Fig. 59 Recepción final de los datos en *Socket_output.py*

Si el socket se desconecta por algún fallo en la autenticación del cliente o fallo en la conexión, se mandará una señal al robot para que quede detenido y finalmente se desconectará la conexión establecida.

```
finally:
    e.stop()
    conn.close()
```

Fig. 60 Control de desconexión segura en *Sockets_output.py*

# Capítulo 4: Pruebas unitarias y pruebas finales

## 4.1. Pruebas unitarias

Durante el desarrollo del software, se han realizado distintas pruebas para comprobar el correcto funcionamiento del sistema, verificando individualmente cada uno de los bloques que lo componen antes de su integración global. Cabe destacar que se utilizó el código de Alba Payo [8] como base para la creación de este proyecto.

### Interfaces

En la primera fase del código se modificaron los ficheros de interfaces, tanto de acelerómetros como de cámara para quitar las líneas de código relacionadas con la comunicación por sockets. Con simples impresiones por consola se probaron las correctas salidas en ambos códigos.

Tanto en el fichero *Acelerometros.py* como en el *Setup.py* se incorporaron nuevas detecciones de movimiento en ambas manos para incluir los movimientos programados para realizar los cambios a los nuevos modos. Se comprobó con texto por consola que se captaba correctamente la posición de ambas manos (arriba, abajo, giro izquierdo, giro derecho y neutral), aunque algunas lecturas erróneas provocaban una división por cero y hacían que se detuviese el programa. Se añadió un control de estas lecturas para evitar el fallo añadiendo el modo 999, también se añadió un control para cuando no se detectaba alguna MPU6050, como se ha explicado en la Pág. 49.


Fig. 61 Posición manos y ejemplo salida por pantalla

Cuando la lectura de ambas manos marcaba la posición exacta, se estableció el valor de las variables *modo*, *dirección* y *velocidad* según lo establecido (Tabla 14).

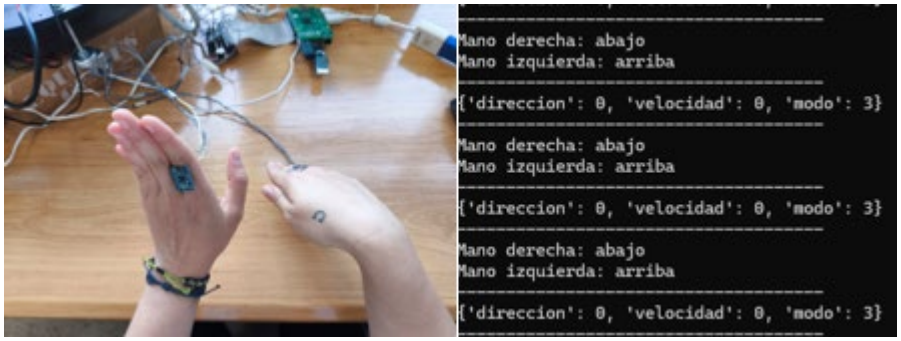



Fig. 62 Posición manos y ejemplo salida por pantalla

## Comunicación

Para programar la parte de la comunicación crearon dos ficheros independientes para contemplar sin ruido los flujos de datos: *Sockets_input.py* (cliente) y *Sockets_output.py* (servidor).

Primero se probó el envío con sockets simples para observar la comunicación fluida en estos ficheros más simples y luego se añadió por fases la encriptación SSL. Se crearon los certificados (pág. 25) de ambas partes y se realizó un handshake simple donde el cliente se aseguraba de que el servidor fuese el esperado, aquí se tuvo que cambiar el nombre de la Raspberry Pi que ejecuta la parte del servidor a "*ROBHELPHUM*" y se vio a través del programa Wireshark cómo cambió el flujo de datos, ya que se mostraban encriptados y enviados por el protocolo TLS en vez de ITP (pág. 18).

Para realizar la exclusividad en la comunicación, se sacó el hash del certificado del cliente y se incorporó una comprobación por parte del servidor en su código, también en el lado del cliente se habilitó la verificación del certificado del servidor. Se comprobó que la comunicación seguía siendo fluida, encriptada y sin latencia.

```
C:\Users\Violeta\OneDrive\Escritorio\PFG 2025\pruebas\prueba comunicacion>py Sockets_output.py
Fingerprint recibido: 705C1F39FD670F4E174D9A26DCE7038632525EE51CA6930D0DB192A5C076DC48
Conexión autorizada: Certificado válido.
{'direccion': '0', 'velocidad': '0', 'modo': '0'}
{'direccion': '0', 'velocidad': '0', 'modo': '0'}
{'direccion': '0', 'velocidad': '0', 'modo': '0'}
{'direccion': '0', 'velocidad': '0', 'modo': '0'}
{'direccion': '0', 'velocidad': '0', 'modo': '0'}
{'direccion': '0', 'velocidad': '0', 'modo': '0'}
```

Fig. 63 Comunicación con doble handshake, lado del servidor

```
C:\Users\Violeta\OneDrive\Escritorio\PFG 2025\pruebas\prueba comunicacion>py Sockets_input.py
La conexión SSL es segura.
{'direccion': '0', 'velocidad': '0', 'modo': '0'}
{'direccion': '0', 'velocidad': '0', 'modo': '0'}
{'direccion': '0', 'velocidad': '0', 'modo': '0'}
{'direccion': '0', 'velocidad': '0', 'modo': '0'}
{'direccion': '0', 'velocidad': '0', 'modo': '0'}
{'direccion': '0', 'velocidad': '0', 'modo': '0'}
Detenido
```

Fig. 64 Comunicación con doble handshake, lado del cliente

## Control y salida

Para desarrollar el bloque de control se programó una interfaz básica por teclado (Fig. 64) para evitar fallos de cualquier tipo en las conexiones GPIO de los acelerómetros y se adecuó el fichero de *ComunicacionGPIO.py* con texto por consola de los movimientos en vez de salidas GPIO con el fin de hacer más sencilla la tarea de las pruebas de este módulo (prueba con ambos módulos Fig. 66).

```
while True:

    num = int(input("num: ").strip())

    letra = input("modo: ").strip()
    on_key_event(num, letra)
```

```
def on_key_event(num, key):
    #cuando se presiona una tecla:
    for _ in range(num):
        if key == 'a':
            sk.envio_data(0,-1,-25)
        elif key == 's':
            sk.envio_data(0,0,-30)
```

Fig. 65 Fragmento de código para control por teclado

En este fichero sencillo de control por teclado, básicamente se indica un número de instrucciones a realizar y se pulsa una tecla según la instrucción que se desee mandar. Gracias a esta interfaz simple se pudo probar sin ruido los cambios de estado en el autómata, que cumple el diagrama de estados diseñado (Fig. 23) y ajustar el número de señales necesarias a recibir para efectuar el cambio de modo.

Para validar este módulo también se probaron las nuevas funciones individualmente y a conciencia: grabar en fichero, leer en fichero, giro 180º en ambos sentidos y parada de emergencia en cualquier circunstancia.

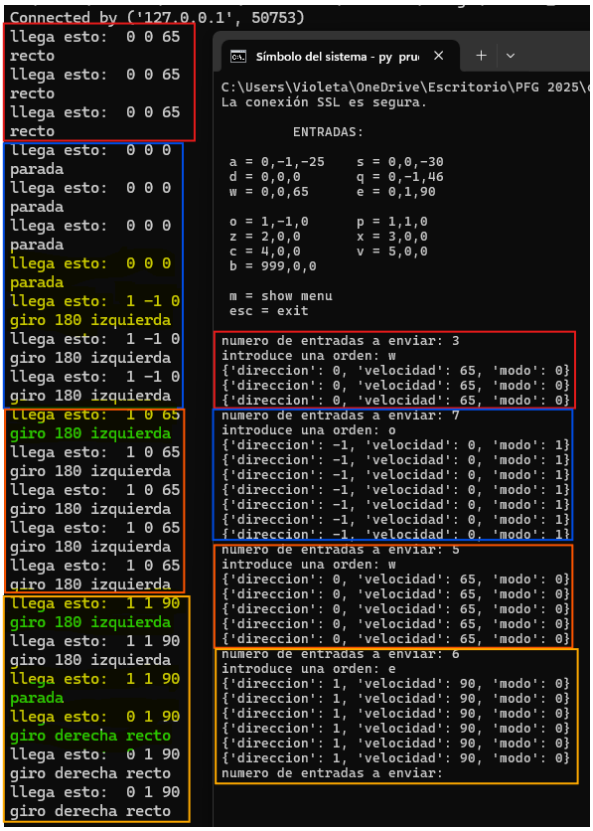



Fig. 66 Ejemplo prueba M1 sin GPIO: a la izquierda el servidor, a la derecha el usuario

En la imagen anterior se ve cómo trata el sistema un cambio del M0 al M1. En este caso se han establecido 5 órdenes iguales para el cambio de modo y 10 conteos para terminar el giro del M1. Se puede ver en el recuadro azul, cómo se mandan 7 órdenes de cambio a M1, y hasta la quinta no se realiza el cambio. Cuando comienza la ejecución del giro, ignora todas las órdenes (porque no recibe M4) hasta la décima donde se completa (cuadro naranja claro). Realiza una pequeña parada y vuelve a M0, donde escucha las siguientes órdenes del usuario.

## 4.2. Pruebas de integración y validación

Tras haber verificado el correcto funcionamiento de cada módulo de forma individual —interfaz, comunicación, control y salida—, se procedió a integrar todos los componentes del sistema para evaluar su comportamiento conjunto.

Las pruebas de integración consistieron en comprobar la interoperabilidad entre los distintos bloques. Se ajustaron algunas variables de tiempo como la frecuencia de la comunicación entre el sistema de entrada y el de salida, o la espera activa para el cambio de modo y para la ejecución de la media vuelta en M1. También se realizaron desconexiones simuladas para analizar el manejo de errores (Requisitos no funcionales) y se comprobó el funcionamiento del autómata poniendo a prueba transiciones y estados tanto los contemplados en el diagrama (Fig. 23) como los que no.

Una vez consolidados los tiempos, se realizaron pruebas de validación orientadas a comprobar que el sistema cumplía con los Requisitos funcionales definidos al inicio del proyecto. Estas pruebas fueron orientadas por los Casos de uso definidos, y se comprobó la correcta respuesta ante comandos en tiempo real y los posibles caminos secundarios.

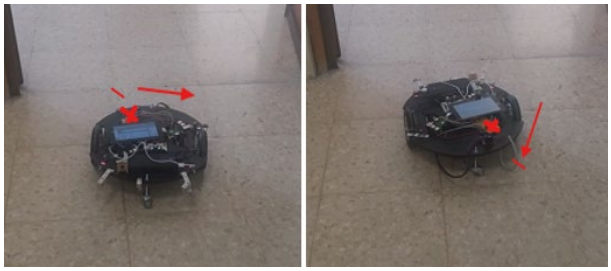


Fig. 67 Prueba de giro 180º sentido horario (M1), control vía acelerómetros


Fig. 68 Prueba de control por acelerómetros: giro derecha y movimiento recto


Fig. 69 Prueba de control por cámara: giro derecha y movimiento recto

## 4.3. Discusión de resultados

Los resultados obtenidos confirman que el sistema es capaz de operar el robot de forma remota, cumpliendo con los objetivos establecidos al inicio del proyecto. El sistema ha demostrado un funcionamiento correcto y coherente en su conjunto, integrando adecuadamente los distintos módulos desarrollados.

Uno de los aspectos que presenta margen de mejora es el procesamiento de los datos provenientes de los acelerómetros. Al establecer nuevos movimientos en ambas manos, se ha detectado cierto nivel de ruido en la detección del gesto final. Esto ha obligado a ampliar los márgenes de la posición neutral de las manos para evitar activaciones involuntarias. Como consecuencia, la variación de la velocidad ha dejado de ser gradual y se ha vuelto más abrupta, ya que el rango útil de movimiento sobre el eje X de la muñeca izquierda —que determina la velocidad positiva o negativa— se ha reducido. Esto limita el control fino sobre la velocidad del robot, haciendo que pase de 0 a 100, o de 0 a -100, en un intervalo más corto del esperado.

Por otro lado, durante las pruebas también se han observado ciertas limitaciones físicas del sistema, derivadas del uso de un robot prestado por el laboratorio. Este equipo es compartido por varios proyectos y presenta algunas particularidades que afectan ligeramente a la trayectoria de desplazamiento. En concreto, una de las ruedas gira ligeramente más rápido que la otra, posiblemente debido a un desequilibrio de peso, una calibración imperfecta de los motores o interferencias en el cableado. Como resultado, el robot tiende a desviarse levemente, lo que impide una trayectoria perfectamente recta en algunos casos.

Además, el cableado de los acelerómetros no está correctamente soldado, lo que los vuelve sensibles a pequeñas interferencias físicas. Esto puede ocasionar errores puntuales en la lectura de los gestos, afectando la fiabilidad del control gestual en ciertos momentos.

Pese a estos inconvenientes, los diferentes modos de operación han arrojado resultados satisfactorios. Además, el control de errores incorporado evita interrupciones innecesarias debidas a lecturas erróneas y se asegura que el robot queda quieto ante una desconexión o error.

En cuanto a las interfaces, tanto la desarrollada mediante cámara como la versión simplificada por teclado se integraron sin dificultades gracias a la modularidad del sistema. Esta característica ha facilitado la implementación y adaptación de distintos métodos de control, demostrando la versatilidad de la arquitectura diseñada.

En general, el sistema ha cumplido con los requisitos funcionales definidos, si bien ciertos elementos —especialmente el control por acelerómetros y algunos aspectos físicos del hardware— podrían beneficiarse de una mejor calibración y un montaje más estable para mejorar la precisión y la experiencia de uso.

# Capítulo 5: Planificación y costes

## 5.1. Planificación

Para cumplir con los objetivos descritos en el apartado anterior se han seguido cinco fases representadas en la siguiente tabla y explicadas a continuación:

| FASE / MES | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| **REQUISITOS Y MODELADO** | | | | | | |
| **ARQUITECTURA Y DISEÑO** | | | | | | |
| **DESARROLLO E IMPLEMENTACIÓN** | | | | | | |
| **PRUEBAS DEL SISTEMA** | | | | | | |
| **REDACCIÓN DE DOCUMENTACIÓN** | | | | | | |

Tabla 15 Diagrama de Gantt

*Fase 1: Investigación y Recolección de Requisitos (Fase 1)*

En esta primera etapa el objetivo fue dejar fijados los objetivos, alcance y aplicación del proyecto, a la vez que se analizaron las necesidades y posibles expectativas de los usuarios.

*Fase 2: Diseño del Prototipo (Fase 1-2)*

Una vez claras las ideas principales del proyecto, se comenzó a diseñar la solución software, considerando las diferentes partes y dependencias que podrían tener los subsistemas mediante diagramas de bloques. Se estableció también el diseño de arquitectura del sistema de control y el diagrama de estados que debía cumplir el autómata.

Se dejaron claros los componentes hardware que figurarían en el sistema final.

*Fase 3: Desarrollo del Prototipo (Fase 2-4)*

Esta fase implica el desarrollo del código del proyecto y se puede segmentar en 3 subfases:

1. Lectura de los acelerómetros y ampliación de modos de movimiento: La primera parte desarrollada fue la adaptación de las entradas de los acelerómetros en los datos que el sistema sería capaz de interpretar. Se realizó la implementación de los atajos de movimiento que proporciona este sistema.

2. Mecanismos de comunicación: Se desarrolló un código cliente servidor básico para probar la comunicación, y después se estudió cuál sería el mecanismo de cifrado mejor adaptado a la finalidad del prototipo.

3. Sistema de control e integración de las partes: Una vez las otras partes quedan funcionando según lo esperado, se simplifica el código cambiando el sistema de control básico (basado en cadenas *if–else*) a un autómata de estados mucho más escalable y se unificaron todas las partes.

*Fase 4: Evaluación y Optimización (Fase 4-5)*

Se realizaron pruebas funcionales y ajustes durante la implementación del código. Se realizaron pruebas con usuarios para recoger otras opiniones y se realizaron optimizaciones en base a la retroalimentación obtenida. Se realizaron las pruebas finales recogidas en este documento.

*Fase 5: Documentación y Presentación (Fase 5-6)*

Preparar la documentación técnica completa del proyecto integrando las distintas investigaciones realizadas a lo largo del mismo. Se realiza la presentación final de cara a la exposición del proyecto.

## 5.2. Costes

Para el desglose de los gastos generados en la ejecución de este proyecto, se ha definido la siguiente tabla donde se pueden encontrar los distintos componentes integrados en el prototipo final basado en los acelerómetros, además de los gastos de ingeniería aplicados en el estudio, equivalentes a 360 horas:

| **Componente** | **Precio Unidad (inc.IVA)** | **Cantidad** | **Total** |
|---|---|---|---|
| Robot móvil ARLO | 437,50 € | 1 | 437,50 € |
| Batería de plomo ácido | 36,05 € | 1 | 36,05 € |
| Raspberry Pi 3 b+ | 43,17 € | 2 | 86,34 € |
| MPU6050 | 7,04 € | 2 | 14,08 € |
| Kit de desarrollo ARLO | 48,71 € | 1 | 48,71 € |
| VEEKTOMS power bank | 32,99 € | 1 | 32,99 € |
| **Total componentes:** | | | **655,67 €** |
| Gastos ingeniería | 15,00 €/h | 500 | 7.500,00 € |
| **Gastos totales:** | | | **8.155,67 €** |

Tabla 16 Tabla costes materiales del prototipo desarrollado

# Capítulo 6: Impacto social y medioambiental

## 6.1. Impacto social

Este proyecto responde a los principios del Objetivo de Desarrollo Sostenible (ODS) Nº10: Reducción de las desigualdades, al promover la accesibilidad en el control de sistemas robóticos para personas con movilidad reducida [29]. Se propone un sistema capaz de adaptarse a distintas capacidades motrices, con el objetivo de incrementar la autonomía personal y facilitar la interacción tecnológica sin requerir el uso de extremidades funcionales completas.

El sistema ha sido diseñado con interfaces adaptativas, donde el control se realiza mediante gestos corporales recogidos por acelerómetros o movimientos de la cabeza. En el primer caso, los acelerómetros pueden colocarse en una mano o muñeca, dependiendo de la movilidad del usuario. En el segundo caso, los movimientos de cabeza permiten emitir instrucciones como dirección y velocidad, adecuándose a usuarios con tetraplejia o lesiones más severas. Implementar el sistema en un robot es solo un caso de estudio ya que el interfaz adaptado se puede utilizar en otros dispositivos como electrodomésticos o maquinaria y se puede controlar con otros tipos de sensores y mandos.

Estas soluciones buscan mejorar la inclusión social y tecnológica, permitiendo que las personas con discapacidad severa participen activamente en tareas cotidianas y reduciendo su dependencia de terceros. Asimismo, liberan carga de trabajo a cuidadores y familiares, aumentando el bienestar emocional del entorno del paciente.

Gracias a su arquitectura modular y abierta, el sistema permite una futura integración con nuevas tecnologías (cámaras, electromiografía, pulsadores, etc.) y puede aplicarse a contextos de asistencia, educación especial o rehabilitación motora. En definitiva, el sistema se alinea con el concepto de diseño universal, tal como promueve la Convención sobre los Derechos de las Personas con Discapacidad (ONU, 2006) [30], fomentando el acceso equitativo a la tecnología.

## 6.2. Impacto medioambiental

El impacto medioambiental de este proyecto se relaciona principalmente con los componentes electrónicos empleados, como acelerómetros, microcontroladores, cámaras y la Raspberry Pi. Estos dispositivos requieren procesos industriales que implican consumo de recursos minerales, emisiones de $CO_2$ y generación de residuos electrónicos.

La fabricación y uso de estos dispositivos está regulada por varias normativas europeas:

La Directiva 2011/65/UE (RoHS 2) [31] prohíbe el uso de sustancias peligrosas como plomo, mercurio o cadmio en dispositivos eléctricos y electrónicos. Aunque el presente proyecto es un prototipo, se han seleccionado componentes que cumplen con esta directiva, asegurando una menor toxicidad en su ciclo de vida.

La Directiva 2012/19/UE (RAEE) [32] regula la recogida, tratamiento y reciclaje de residuos electrónicos. Se recomienda la correcta gestión de estos materiales al finalizar su vida útil, especialmente en laboratorios académicos o centros de investigación.

La norma ISO 14001:2015 [33], aunque su aplicación no es obligatoria en este contexto, el diseño modular del sistema (componentes intercambiables, reutilización de piezas existentes del laboratorio, posibilidad de actualización sin desechar hardware) está alineado con sus principios de reducción de impacto ambiental.

Los componentes empleados en el desarrollo del prototipo, como la Raspberry Pi, acelerómetros y cámaras, son dispositivos electrónicos de bajo consumo que se comercializan legalmente en el mercado europeo. Por tanto, cuentan con el marcado CE [34] y han sido fabricados cumpliendo las directivas europeas vigentes, como las mencionadas anteriormente. De este modo, se garantiza un cumplimiento básico de los estándares medioambientales y de seguridad en cuanto a materiales y compatibilidad electromagnética.

# Capítulo 7: Conclusiones y líneas futuras

## 7.1. Conclusiones

El desarrollo de este proyecto ha desembocado en la implementación de un sistema funcional para el control remoto de un robot, cumpliendo con los objetivos planteados en la fase inicial. Gracias a una arquitectura modular, ha sido posible integrar de forma efectiva los distintos componentes del sistema —interfaz, comunicación, control y salida— en una estructura integrada y funcional, capaz de responder de manera precisa a las órdenes del usuario.

Durante el proceso, se han superado diversos retos técnicos, destacando especialmente la implementación de un canal de comunicación cifrado y exclusivo entre un único usuario y el robot, así como, la programación de un autómata encargado de gestionar los distintos estados y transiciones del sistema.

Uno de los desafíos más laboriosos ha sido la correcta interpretación de los nuevos modos incorporados en la interfaz basada en acelerómetros. Aunque estos sensores presentan ciertas limitaciones, se ha logrado establecer un rango de funcionamiento estable y fiable. En este contexto, la incorporación de mecanismos de control de errores y sistemas de parada segura ha sido clave para garantizar la robustez del sistema frente a condiciones de uso variables.

La modularidad del diseño ha permitido integrar con facilidad distintas interfaces de control, como los acelerómetros, la cámara o el teclado, lo que demuestra la escalabilidad y adaptabilidad de la solución desarrollada.

En definitiva, esta experiencia ha contribuido de forma significativa al desarrollo de competencias técnicas y a la consolidación de habilidades clave en el ámbito del control y la interacción remota con sistemas físicos.

## 7.2. Líneas futuras

A partir de los resultados obtenidos y de la experiencia adquirida durante el desarrollo del proyecto, se identifican diversas líneas de mejora y ampliación que podrían explorarse en trabajos futuros:

**Optimización del control por acelerómetros:** Se plantea mejorar tanto el tratamiento de los datos como la calibración física de los sensores (conexiones, montaje, cableado), para reducir el ruido y aumentar la fiabilidad del control gestual. La aplicación de algoritmos de filtrado o incluso técnicas de aprendizaje automático podría mejorar notablemente la precisión y estabilidad de las órdenes interpretadas.

**Mejoras en la mecánica del sistema:** Ajustar la configuración de los motores, la distribución del peso del robot o su integración en plataformas móviles como sillas motorizadas, permitiría corregir desviaciones en la trayectoria y adaptar el sistema a nuevos escenarios de uso.

**Incorporación de sensores adicionales:** Para aumentar la seguridad y las capacidades del sistema, se sugiere integrar sensores de proximidad, cámaras con detección del entorno u otros dispositivos como sensores biométricos, que permitan funciones de monitorización de salud en aplicaciones asistenciales.

**Ampliación de las interfaces de control:** La arquitectura modular facilita la implementación de nuevas formas de interacción, como el control por voz, gestos avanzados o incluso mediante entornos de realidad aumentada, aumentando así la accesibilidad y versatilidad del sistema.

**Desarrollo de una interfaz gráfica avanzada:** Una interfaz más completa permitiría un mayor control visual del sistema, facilitando la configuración de parámetros, la supervisión en tiempo real y una experiencia de usuario más intuitiva y robusta.

**Adaptación a otras plataformas robóticas:** Gracias al diseño modular, sería posible aplicar la solución a diferentes tipos de robots sin requerir modificaciones estructurales significativas, lo que aumenta el alcance y la escalabilidad del sistema.

En conjunto, estas líneas futuras permitirían evolucionar el sistema hacia aplicaciones más complejas, fiables y adaptadas a distintos contextos, reforzando su utilidad en el ámbito de la robótica interactiva y la asistencia remota.

# Capítulo 8: Bibliografía

[1] UNSA, «La UNSA presenta prototipo de robot que desactiva explosivos utilizando sensores,» Vols. %1 de %2https://www.unsa.edu.pe/la-unsa-presenta-prototipo-de-robot-que-desactiva-explosivos-utilizando-sensores/ , 2021.

[2] J. F. SLAPAITIS FORERO y L. P. B. JORGE , «DISEÑO Y CONSTRUCCIÓN DE UN ROBOT DE EXPLORACIÓN CON UNSENSOR KINECT PARA EL MAPEO 3D DE ZONAS CON DIFÍCIL ACCESOUSANDO TÉCNICAS DE PROCESAMIENTO DE IMÁGENES.,» BARRANCABERMEJA, 2018.

[3] G.-J. M. Kruijff, V. Tretyakov, F. Pirri y E. Pianese, «Rescue Robots at Earthquake-Hit Mirandola, Italy:a Field Report,» Italia, 2012.

[4] D. Martín Castanier y E. V. Mendía Idrovo, «DISEÑO, CONSTRUCCIÓN E IMPLEMENTACIÓN DE UNA SILLA DE RUEDAS ELÉCTRICA PLEGABLE PARA UNA PERSONA CON PROBLEMAS DE MOVILIDAD,» Cuenca, 2018.

[5] A. Kumar, M. Singh, M. Singh, R. J, A. Dubey y S. Mujoo, «A Novel Electronic Wheel Chair Design using Artificial Intelligence Assisted Smart Sensors and Controller,» 2024.

[6] K. Chaitanya, S. Reddy, K. sreelakshmi, S. Sadik y A. Kumar, «IoT-enabled Moving Wheelchair with Obstacle,» 2024.

[7] Y. Kenzhetayev y I. Nagy, «A Real-time, Remotely Operated Vehicle, using,» Budapest, 2024.

[8] A. Payo, «Diseño de interfaces adaptadas a personas con movilidad reducida para el manejo remoto de un robot.,» Madrid, 2022.

[9] A. Soofi Alizadeh, «Interfaz adaptada para el manejo de un robot: infraestructura y caso práctico,» Madrid, 2022.

[10] D. Fernández Fernández, «Silla de ruedas de bajo coste, destinada a discapacitados en países de escasos recursos económicos,» Madrid, 2020.

[11] O. Sarmiento Gómez y J. E. Rubio Cristiano, «Monitoreo remoto de signos corporales y transmisión de datos y alertas a una aplicación Instalada en un Smartphone,» 2018.

[12] T. Pousada, «Impacto psicosocial de la silla de ruedas en la vida de las personas afectadas por una enfermedad neuromuscular,» La Coruña, 2011.

[13] V. Smil, «Ciencia para un nuevo sistema energético,» 2022.

[14] I. Arriandiaga, «Human Support Robot' (HSR),» 2018.

[15] A. M. Khamis Rashwan, «INTERACCIÓN REMOTA CON ROBOTS,» Madrid, 2003.

[16] A. Casals, «Robótica y tecnología como soporte a la discapacidad,» Cataluña, 2019.

[17] Telefónica, «Los robots asistenciales para personas con discapacidad,» Telefónica, 29 09 2023. [En línea]. Available: https://www.telefonica.com/es/sala-comunicacion/blog/robots-asistenciales-personas-discapacidad/.

[18] S. Majumder, T. Mondal y J. Deen, «Wearable Sensors for Remote Health Monitoring,» 2017.

[19] J. San Martín Garaluce, «Reconocimiento de gestos manuales mediante red neuronal artificial,» País Vasco, 2020.

[20] None, «Welcome to the world of Statecharts,» xxxx.

[21] U. Electronica, «PROTOCOLO I2C,» 2019.

[22] mbrobotics, «Raspberry PI - GPIOS,» xxxx.

[23] ElectroComponentes, «MPU-6050,» xxxx.

[24] python.org, «Python Tutorial,» xxxx.

[25] OpenWebinars, «Wireshark: Qué es y ejemplos de uso,» xxxx.

[26] Raspberrypi.cl, xxxx.

[27] SSLDragon, «¿Qué es un archivo .pem? Creación, uso y conversión de archivos SSL,» xxxx.

[28] OpenSSL.org, «Downloads,» xxxx.

[29] E. 2. D. sostenible, «ODS,» [En línea]. Available: https://estrategia2030.es/objetivo-10-reduccion-de-las-desigualdades/#:~:text=El%20ODS%2010%20promueve%20reducir,pol%C3%ADticas%20y%20la%20legislaci%C3%B3n%20pertinentes..

[30] Jefatura del Estado, «Instrumento de Ratificación de la Convención sobre los derechos de las personas con discapacidad,» de *https://www.boe.es/eli/es/ai/2006/12/13/(1)*, BOE-A-2008-6963, 21 de abril de 2008.

[31] D. 2. d. P. E. y. d. Consejo, «Boletin Oficial del Estado - DOUE-L-2011-81307,» 01 07 2011. [En línea]. Available: https://www.boe.es/buscar/doc.php?id=DOUE-L-2011-81307.

[32] B. O. d. Estado, «Directiva 2012/19/UE del Parlamento Europeo y del Consejo, sobre residuos de aparatos eléctricos y electrónicos (RAEE).,» 24 07 2012. [En línea]. Available: https://www.boe.es/buscar/doc.php?id=DOUE-L-2012-81320.

[33] I. 14001:2015, «Sistemas de gestión ambiental — Requisitos con orientación para su uso,» 2015. [En línea]. Available: https://www.iso.org/es/norma/14001.

[34] M. CE, «europa.eu,» 30 01 2025. [En línea]. Available: https://europa.eu/youreurope/business/product-requirements/labels-markings/ce-marking/index_es.htm.