UNIVERSIDAD CATÓLICA DE LA SANTÍSIMA CONCEPCIÓN

FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

Sistema robótico para la asistencia de actividades cotidianas en el hogar

Alexander Marcelo Thompson Ferreira

Informe Habilitación Profesional
Ingeniería Civil Eléctrica

Profesor Patrocinante:

Dra. Silvia E. Restrepo M.

Profesores Guía:

Dr. Ricardo Bustos
Dr. Yasmany Prieto

Concepción, mayo de 2025

UNIVERSIDAD CATÓLICA DE LA SANTISIMA CONCEPCION
Facultad de Ingeniería
Departamento de Ingeniería Eléctrica

Profesor Patrocinante:
Dra. Silvia E. Restrepo M.

Sistema robótico para la asistencia de actividades cotidianas del hogar

Alexander Marcelo Thompson Ferreira

Informe Habilitación Profesional

Ingeniería Civil Eléctrica

Mayo 2025

Resumen

El proyecto consiste en el desarrollo de un sistema robótico basado en el robot TonyPi, diseñado para asistir en actividades cotidianas en el hogar, especialmente para personas con movilidad reducida. Estas personas con frecuencia encuentran dificultades para realizar tareas básicas de manera autónoma y efectiva en la actualidad, lo que puede limitar su independencia y calidad de vida. El objetivo del sistema robótico es mejorar la rutina de las personas, actuando como un asistente diario.

Este sistema robótico, utiliza una Raspberry Pi con 4GB de memoria como controlador central y una tarjeta de expansión Raspberry Pi. Está equipado con una cámara de alta definición que puede capturar imágenes en tiempo real y enviarlas a un celular o computadora. Tiene servomotores de alta tecnología con buses seriales. Esto le permite realizar actividades como caminar y hacer flexiones, entre otras. Su batería de litio recargable tiene una duración aproximada de una hora.

El sistema robótico destaca por su flexibilidad, ya que puede personalizarse a través de programación en Python para ajustarse a las necesidades específicas de los usuarios. Esta capacidad lo hace ideal no solo para actividades de entretenimiento, sino también para brindar apoyo en áreas críticas como la seguridad y el bienestar. Por ejemplo, puede programarse para recordar la toma de medicamentos o enviar alertas en caso de emergencias. Además, se pueden integrar distintos sensores, como el de temperatura y humedad, que permiten monitorear estos parámetros directamente. Esto agrega una funcionalidad práctica que resulta especialmente útil para supervisar el entorno doméstico en tiempo real.

Para mejorar la conectividad y la integración del sistema robótico en el entorno doméstico, se agregó un receptor WiFi USB a la Raspberry, lo que permite emitir y recibir señales de WiFi simultáneamente. Esto optimiza su integración con diferentes aplicaciones. Esta mejora la conectividad como también asegura que el sistema robótico mantenga una comunicación estable y eficiente con aplicaciones externas, como Telegram.

Una característica implementada del sistema robótico es la integración con la aplicación Unity y el dispositivo Oculus Quest 2, permitiendo una experiencia de control inmersiva mediante realidad virtual. Esta conexión hace posible que los usuarios vean en tiempo real lo que la cámara del robot captura y controlen sus movimientos mediante el entorno virtual, lo que facilita la interacción directa y mejora el control sobre las acciones del robot.

En el sistema robótico también se implementó un chatbot integrado en Telegram, diseñado para enviar y recibir mensajes de manera automática y personalizable. Esta herramienta resulta

especialmente útil para configurar recordatorios y activar alertas de emergencia en caso de que el usuario lo indique. Además, el chatbot permite monitorear parámetros como la temperatura y la humedad a través de un sensor AHT10, ofreciendo información en tiempo real. Su capacidad para facilitar la comunicación con el usuario, el cual brinda una capa adicional de seguridad y conectividad.

Los resultados de las pruebas demostraron que el sistema robótico es funcional y capaz de asistir en tareas como recordatorios, monitoreo ambiental, control remoto por realidad virtual y una comunicación interactiva. Sin embargo, se identificaron aspectos que pueden optimizarse a futuro, como la reducción de latencia en la transmisión de video, integración de nuevos sensores o funcionalidades que permitan una asistencia más personalizada y eficiente.

Se espera que el proyecto del sistema robótico ofrezca una solución sofisticada y adaptable para mejorar la calidad de vida de las con movilidad reducida. Su desarrollo no solo facilitará las tareas diarias, sino que también brindará a sus usuarios una mayor independencia y bienestar, posicionándose como un asistente esencial en el hogar.

A los alumnos del pasado, presente y futuro de la carrera de Ingeniería Civil Eléctrica de la U.C.S.C.

Agradecimientos

Agradezco a toda mi familia, en especial a mi padre Marcelo, mi madre Paulina, mi hermano John y mi abuela Rosa, por su apoyo incondicional y su paciencia a lo largo de este proceso. A mis tíos, quienes también han estado presentes, y a mi sobrino Martín, les agradezco el apoyo que me han brindado. También quiero expresar mi gratitud a mi novia Carole, quien estuvo a mi lado en todo este proceso universitario.

Extiendo mi agradecimiento a los jefes de brigada de CONAF, Enrique Cabrales y Pedro Oñate, por su comprensión y por permitirme asistir a clases, sin importar las horas comprometidas.

Quiero agradecer a José Díaz y Renato Pomeri por su ayuda desinteresada, la cual fue invaluable en momentos clave de este proceso.

Por último, agradecer al proyecto ANID FONDECYT N°11221231 por el incentivo de tesis que fue recibido para poder mejorar e investigar a fondo mi proyecto de habilitación profesional

Tabla de Contenidos

Lista de Tablas

Lista de Figuras

Abreviaciones

Mayúsculas

I.A.	: Inteligencia Artificial.
IoT	: Internet de las cosas
DOF	: Degrees of Freedom (Grado de libertad)
PID.	: Control Proporcional Integral Derivativo.
PLN.	: Procesamiento del lenguaje natural.
ASR.	: Reconocimiento automatico del habla.
PWM.	: Pulse Width Modulation

Capítulo 1. Introducción

1.1. Introducción General

A lo largo de la evolución de la tecnología, el ser humano ha creado y mejorado herramientas destinadas a satisfacer y solucionar necesidades o tareas cotidianas del hogar, generando un notable desarrollo en la calidad de vida de las personas. En la era actual de la tecnología, la Inteligencia Artificial (IA) ha emergido como uno de los campos más destacados en las ciencias computacionales, atrayendo un gran interés debido a su amplia variedad de aplicaciones. (Ramírez-Pereira et al., 2023; Aguilar Villegas, 2008).

El creciente interés en la IA se ha visto respaldado por el rápido desarrollo que ha experimentado en los últimos tiempos, lo que es evidente en diversos campos como la domótica, el transporte y la industria. La domótica, en particular, ha revolucionado la manera en que interactuamos con nuestros hogares, permitiendo la automatización de tareas domésticas y mejorando la eficiencia energética. En el transporte, la IA ha facilitado avances en la conducción autónoma y la gestión del tráfico, mientras que en la industria ha optimizado procesos de producción y mantenimiento predictivo. (Muñiz Castañeda, 2015).

La integración de la inteligencia artificial y la robótica ha dado paso a innovaciones revolucionarias que buscan mejorar significativamente la calidad de vida de las personas, ofreciendo soluciones tecnológicas para diversos desafíos. En este contexto de avance tecnológico, se presenta un sistema robótico, el que se basa en la modificación del robot humanoide Tony Pi que cuenta con inteligencia artificial desarrollada por Hiwonder. Su controlador central es una potente Raspberry Pi 4B con 4GB de memoria, programado en el lenguaje de programación Python y respaldado por la robusta biblioteca de procesamiento de imágenes OpenCV.

Mediante el uso de Tony Pi se espera diseñar y programar con el objetivo de brindar asistencia en actividades rutinarias dentro del hogar. Se enfocará especialmente en ayudar a personas con movilidad reducida. Este robot humanoide no solo representará una convergencia prometedora entre la robótica y la programación, sino que también tendrá el potencial de transformar radicalmente la manera en que las personas interactúan con la tecnología.

El robot Tony Pi está equipado con una cámara de alta definición y varios sensores que le permiten reconocer y responder a su entorno. Con la capacidad de identificar colores, imágenes y seguir líneas visuales; también puede realizar una variedad de actividades, desde jugar fútbol,

reconociendo el balón, hasta navegar por el hogar, evitando obstáculos.

El desarrollo del sistema robótico incluye la posibilidad de personalizar sus capacidades a través de la programación en Python, lo que permite a los usuarios crear y modificar grupos de acciones complejas, directamente desde un computador o smartphone. Se espera que, gracias a la versatilidad del sistema robótico, se convierta en una herramienta poderosa para la asistencia doméstica, y proporcionar un apoyo significativo a quienes más lo necesitan y mejorar su calidad de vida mediante la tecnología.

1.2. Trabajos Previos

En esta sección se presentarán trabajos previos relacionados con inteligencia artificial, sensores, robótica en la implementación dentro de un hogar, para tener una visión más amplia y detallada de los puntos que se quieren abarcar.

1.2.1 Robótica

- Barrios Sánchez, R. y Rodríguez, R. "ASISTENCIA DE ADULTOS MAYORES MEDIANTE LA IMPLEMENTACION DE ROBOTS DE SERVICIO EN ENTORNOS DOMÓTICOS", tesis, Universidad Piloto de Colombia, 2018 [1].

En este trabajo integran herramientas tecnológicas que permiten la implementación de un sistema de comunicación y control de una placa en un robot móvil, con la capacidad de desplazarse en ambientes predeterminados e interactuar para el monitoreo de los estados de salud de los integrantes y la asistencia en actividades cotidianas, por medio de la recopilación de datos acerca del estado del hogar para mantenerlo controlado a través de un dispositivo móvil.

- Valencia Redrovan, D. P. y González Delgado, L. E. "DISEÑO E IMPLEMENTACION DE UN PROTOTIPO DE ROBOT ASISTENTE PARA PERSONAS CON DISCAPACIDAD MOTRIZ Y ADULTOS MAYORES, BASADOS EN INTELIGENCIA ARTIFICIAL", tesis, Universidad Politécnica Salesiana de Ecuador, 2014 [2].

Este estudio desarrolló un prototipo de robot para ayudar a personas adultas y con movilidad reducida. Se recopiló información sobre discapacidades, enfermedades y leyes del país donde se desarrolló, para que el robot pudiera satisfacer las necesidades. Para comunicarse con el robot, se

utilizaron técnicas de reconocimiento facial y detección de rostros, así como audio bidireccional. Se llevaron a cabo numerosas pruebas de interacción entre robots y humanos el cual los resultados fueron muy satisfactorios y proporcionaron algunas sugerencias para el futuro.

- Revista de Robots. "Robot Pepper: qué es, historia y precio de Pepper." Revista de Robots, 8 de junio de 2023, https://revistaderobots.com/robots-y-robotica/robot-pepper-que-es-historia-y-precio-de-pepper/?cn-reloaded=1. [3].

Pepper, robot humanoide desarrollado por SoftBank Robotics, es un robot humanoide diseñado para interactuar con humanos de la forma más natural posible a través del dialogo y de su pantalla táctil, este robot humanoide es capaz de identificar rostros y emociones humanas clave. Utiliza inteligencia artificial y reconocimiento de voz para realizar tareas como la atención al cliente, la enseñanza y la asistencia en el hogar. Se puede programar utilizando varios lenguajes de programación, incluidos Python y Java.

- VERA REPULLO, José Alfonso, et. al. Diseño de un sistema integral de atención domiciliaria con robot móvil asistencial. En: XLI Congreso Anual de la Sociedad Española de Ingeniería Biomédica. Cartagena: Universidad Politécnica de Cartagena, 2023. [4].

En este trabajo se diseñó un sistema integral de atención domiciliaria con un robot móvil asistencial. Este navega de manera autónoma dentro de la casa para atender las necesidades del usuario o sugerir actividades según su estado anímico. Para su funcionamiento e integración, se utilizaron distintos componentes, como un asistente virtual controlado, una base móvil que alimenta parte del sistema, como sensores, una cámara 3D para el reconocimiento facial o gestual de las personas, un sensor láser para la detección de obstáculos, un miniordenador y servomotores para la movilidad del robot. Además, cuenta con un sistema de Ambient Assisted Living integrado en el hogar y una serie de sensores biométricos para monitorear el estilo de vida del usuario.

Mediante los datos de los sensores se realiza un algoritmo con las actividades cotidianas de las personas para obtener los patrones de comportamiento, como las horas de sueño, la actividad física y las visitas al baño. Esto se puede saber gracias a una pulsera de actividad (Empactica E4) y un sensor de movimiento (Mi Motion Sensor Xiaomi). Además, detecta la alteración de hábitos o estado de ánimo. Este robot cuenta con un coaching emocional que propone realizar alguna actividad lúdica,

realizar ejercicio cuando se detecte que las persona llevan mucho tiempo recostada o sentada, llamar algún familiar si se detecta estrés o cambio de ánimo, poner música, etc. Siempre se respeta la privacidad del usuario, al cual se le pregunta si quiere realizar la actividad o no.

- Escobar Ruiz, D. "Ameca: el robot humanoide más avanzado del mundo y que vale más de USD 100 mil." Infobae, 15 de julio de 2024, https://www.infobae.com/tecno/2024/07/15/ameca-el-robot-humanoide-mas-avanzado-del-mundo-y-que-vale-mas-de-usd-100-mil/.[5].

Androide robótico creado por la compañía robótica Engineered Arts. Este robot inteligente puede moverse, conversar e incluso expresar una amplia variedad de emociones, desde la felicidad hasta la rabia. Uno de sus avances más significativos es que puede ejecutar gestos faciales muy realistas, con una alta carga emocional.

Su software se llama Tritium y está formado por algoritmos de inteligencia artificial capaces de identificar patrones de comportamiento y elaborar predicciones. Aunque está diseñado con extremidades artificiales conectadas con la tecnología más avanzada

- TUTIVEN REYES, Jorge Enrique; VILLAGÓMEZ GALARZA, Daniel Euclides. Diseño e Implementación de SLAM y control PID de un robot autónomo Robomaster S1 utilizando Python. 2021. Tesis de Licenciatura. [6].

En este trabajo se puede visualizar y comprender que toda investigación en el campo de la robótica tiene el objetivo común de mejorar la autonomía de los robots. Además, se analizan los diversos hardware y software necesarios para la elaboración de robots, utilizando el lenguaje de programación Python. Esta investigación hace un recorrido histórico sobre la palabra "robot", sus inicios y avances a lo largo del tiempo, las diferentes clasificaciones de robots, y sus arquitecturas.

También, se detallan los sensores e instrumentos necesarios para la construcción de un robot, como las placas de Arduino, Raspberry Pi, entre otros. Se describen los sistemas operativos de cada instrumento para facilitar una mejor comprensión de su funcionamiento. El trabajo se centra en un robot de cuatro ruedas con apariencia de tanque, equipado con cámara y altavoces. Se presenta la arquitectura del sistema, junto con los códigos y librerías utilizados. Este trabajo será de gran ayuda en la realización del sistema robotico, proporcionando información valiosa y consejos prácticos para su desarrollo.

1.2.2 Inteligencia artificial

Para automatizar la robótica, se necesitará el apoyo de la inteligencia artificial, para que detecte o genere algún algoritmo para asistir y monitorear a las personas en sus hogares

- Ramírez-Pereira M, Figueredo-Borda N, Opazo Morales E. La inteligencia artificial en el cuidado: un reto para Enfermería. Enfermería: Cuidados Humanizados. 2023;12(1):e3372. doi: 10.22235/ech.v12i1.3372. [7].

Este trabajo explora la viabilidad de emplear la inteligencia artificial en el ámbito del cuidado de la salud, tanto en entornos hospitalarios como en el área de la salud en general. Se destaca el potencial de esta tecnología para recopilar y analizar los datos personales de cada paciente, lo que podría ofrecer beneficios predictivos respecto a su estado de salud. Además, se plantea que la IA podría mejorar la relación entre pacientes y personal médico, así como aumentar la eficiencia en la atención. Se sugiere que el uso de la inteligencia artificial podría facilitar tratamientos personalizados al analizar datos clínicos y desarrollar o identificar terapias más efectivas basadas en el historial médico de cada paciente.

- AGUILAR VILLEGAS, Jamil Esteban. Inteligencia Artificial y Microcontroladores. RITS, La Paz, n. 1, 2008. [8].

En esta investigación el autor comenta sobre la inteligencia artificial, desde un punto de ayuda a las personas y que logre imitar las habilidades humanas con sensores y conversores análogo/digitales, el reconocimiento de objetos, colores, etc. Todo esto gracias a los microcontroladores que incluyen 3 unidades funcionales de una computadora, que es programable. Además, comenta sobre el carácter ético de la aplicación de la inteligencia artificial, las leyes que rigen a un robot, y el uso de estos en la industria, lo cual conlleva a un proceso más automatizado, aumentando la producción sin bajar la calidad y seguridad.

- Muñiz Castañeda, J. "MEJORAR LA CALIDAD DE VIDA DE LAS PERSONAS EN EL HOGAR MEDIANTE UN SISTEMA DE TOMA DE DECISIONES INTELIGENTES ", Instituto tecnológico de colima, México, 2015 [9].

En esta investigación, el autor llevó a cabo la implementación de la tecnología Arduino para automatizar las funciones de un hogar. Esta automatización se complementó con la integración de la inteligencia artificial, con el objetivo de lograr una sinergia entre ambas tecnologías. El propósito principal de este enfoque fue crear un entorno doméstico más eficiente y cómodo para sus habitantes. Al combinar la tecnología Arduino con la inteligencia artificial, se esperaba lograr un estilo de vida más agradable y conveniente para los usuarios, permitiendo el control y la gestión automatizados de una variedad de dispositivos y sistemas dentro del hogar. Este enfoque también tenía como objetivo abrir nuevas oportunidades en el mercado de la automatización del hogar, al ofrecer soluciones más flexibles y asequible

1.2.3 Sensores

- COLOMER BARBERA, Javier. Estudio de los sensores para la detección de obstáculos aplicables a robots móviles. 2018. [10].

En este trabajo de tesis de la universidad Oberta de Catalunya describen los principales sensores para la detección de obstáculos, se dividen en dos grupos: con contacto físico y sin contacto. Se detallan sus principales características y los escenarios ideales para la utilización de cada sensor, así como la naturaleza de la señal de detección. Esta señal puede ser binaria, indicando la presencia o ausencia de un obstáculo, o analógica/digital, permitiendo determinar la distancia del obstáculo al sensor. Además, esta investigación amplía el vocabulario relacionado con los sensores, ya que se describen varios conceptos sobre estos, como, por ejemplo, el campo de medida, sensibilidad e interferencias, entre otros. Además, esta investigación explica en detalle los sensores de tipo inductivo y capacitivo. Se describen su funcionamiento, características y aplicaciones reales.

Por último, y lo más importante, este trabajo proporciona los criterios más importantes para la selección de sensores, teniendo en cuenta factores como el material, tamaño, interferencias y compatibilidad. Esto resulta de gran ayuda para la correcta implementación de los sensores en diversos proyectos y aplicaciones.

- O. Torres Acuitlapa, "Exploración basada en sensores para robots móviles", Tesis de maestría, Benemérita Universidad Autónoma de Puebla, 2014. [11].

En este trabajo se busca que los robots sepan dónde se encuentran ubicados con el propósito de que puedan tomar decisiones correctas de movimiento o enviar señales de alerta al manipulador del robot, asegurando que siempre se conozca su posición. Para ello, el robot debe ser capaz de construir un mapa y, al mismo tiempo, localizarse en él. Se utiliza un método basado en sensores para la exploración de ambientes desconocidos, describiendo dos métodos como el SRT que es un método que se basa en la generación de una estructura de datos incremental aleatoria llamada árbol aleatorio de exploración usando sensores (SRT, del inglés Sensor-Based Random Tree), el que representa un roadmap del área explorada asociada a una región segura y el SET que es un método para la exploración basada en sensores de entornos desconocidos, utilizando un sistema robótico equipado con telémetros.

El método se basa en la generación incremental de una estructura de datos del espacio de configuraciones llamado árbol de exploración basado en sensores (SET, del inglés Sensor-based Exploration Tree). Además, en este trabajo se implementó un algoritmo de visibilidad, el que permite la corrección de la información obtenida a través de un sensor telemétrico y ayuda a obtener una mejor aproximación de los objetos. Este enfoque mejora significativamente la precisión de la localización y el mapeo, facilitando una navegación más eficiente y segura en entornos desconocidos.

1.3. Discusión

La revisión de trabajos previos o similares, así como de páginas web, proporcionó una valiosa fuente de ayuda y conocimiento, además de ampliar la perspectiva sobre los temas de robótica, IA y sensores.

Para los trabajos [1],[2] y [4] se destaca que son trabajos similares a lo que se quiere llegar, ya que se centran en la asistencia para las personas con movilidad reducida, mediante un robot, ya sea integrado con inteligencia artificial o automatizado. El [1] utiliza sistemas domóticos para saber las condiciones del usuario y brindar la asistencia con el robot, mientras que el [2] y [4], son diseños de un robot móvil que pueda asistir a sus usuarios. En el caso del [2] se utiliza inteligencia artificial que complementa de una gran manera al robot.

En [3] y [5] se presentan robots humanoides inteligentes, que están desarrollados para interactuar con personas lo más natural posible, además de identificar rostros y emociones humanas con inteligencia artificial. Mientras que [6] aborda la implementación de un robot, sus piezas, lenguaje de programación, software y hardware para desarrollar un robot móvil.

[7] En este trabajo se exploró la viabilidad sobre la inteligencia artificial en el área de medicina para la toma de decisiones, la que puede ser muy efectivo y siempre respetando las leyes.

[8] [9] Son investigaciones que aportan en el desarrollo de la IA, para ayudar a las personas en su día a día con eficacia y las reglas que ellos propongan.

[10] [11] Estos trabajos abordan el estudio sobre los sensores para la utilización en un robot móvil y que pueda localizarse o detectar obstáculos en su camino.

1.4. Hipótesis de Trabajo

El desarrollo e implementación de un sistema robótico basado en el robot Tony Pi de Hiwonder, permitirá brindar asistencia personalizada a personas con movilidad reducida. Al incorporar un chatbot integrado con Telegram y comandos de voz, junto con un nuevo método de control mediante el uso de Oculus Quest 2, el sistema facilitará la interacción intuitiva y en tiempo real.

Esto permitirá configurar recordatorios automáticos, recibir alertas de emergencia, revisar parámetros de temperatura y humedad, enviar y recibir mensajes, como controlar el robot de forma remota mediante realidad virtual. Además, el sistema podrá asistir en actividades como el monitoreo del entorno y la emisión de alarmas.

Estas funcionalidades permitirán que los usuarios realicen tareas cotidianas sin requerir asistencia constante, promoviendo así su autonomía e independencia, y reduciendo la necesidad de apoyo permanente por parte de cuidadores.

1.5. Objetivos

1.5.1 Objetivo General

Realizar e implementar un sistema robótico que proporcione asistencia integral en actividades cotidianas en el hogar con el fin de facilitar sus rutinas diarias

1.5.2 Objetivos Específicos

- Investigar sobre tópicos relevantes a robots asistentes para identificar las mejores prácticas
- Identificar las necesidades más urgentes de los usuarios dentro de sus hogares, priorizando aquellos con movilidad reducida.
- Proponer un sistema de asistencia de actividades cotidianas en el hogar, basado en la integración del robot Tony Pi y componentes físicos
- Implementar un sistema de Wi-fi que permita al robot comunicarse con diferentes dispositivos.
- Implementar un nuevo método de control y un chatbot que mantenga comunicación constante entre usuario y cuidador.
- Realizar pruebas del sistema robótico propuesto.

1.6. Alcances y Limitaciones

El proyecto se centrará en el uso del lenguaje de programación Python para desarrollar y probar las funcionalidades del robot Tony Pi. Se llevarán a cabo pruebas experimentales utilizando una Raspberry Pi y el robot.

El sistema robótico por desarrollar contará con al menos la asistencia de 3 actividades

1. Implementar que el robot emita y reciba wifi
2. Nuevo método de control
3. Implementación de chatbot

1.7. Temario y Metodología

En el capítulo 2 del informe detalla una descripción del robot, luego, las características técnicas, incluyendo una descripción de sus componentes, como la Raspberry Pi 4B, la tarjeta de expansión HAT, los servomotores LX-824, la batería de litio, y la cámara 2 DOF. Se describe el sistema operativo con su lenguaje de programación Python junto con la biblioteca OpenCV para facilitar la personalización y expansión del robot. Se presentan cuatro métodos de control: software de PC, aplicación móvil, control de escritorio remoto VNC y joystick PS2. Además, se destaca la capacidad de expansión del robot mediante la adición de diversos sensores.

En el capítulo 3 detalla los resultados de una encuesta realizada para identificar las principales necesidades de las personas con movilidad reducida dentro de sus hogares. El objetivo de esta encuesta fue comprender mejor las áreas donde el sistema robótico, basado en el robot Tony Pi, puede ofrecer asistencia significativa. Además, el capítulo recoge comentarios y sugerencias de los encuestados sobre posibles mejoras que podrían implementarse en el sistema robótico para incrementar su efectividad y utilidad. Se analizan los resultados de la encuesta, destacando algunos posibles usos del sistema robótico.

En el Capítulo 4, se describen las modificaciones realizadas al robot Tony Pi, destacando la personalización y expansión de su hardware como software, incluyendo la incorporación de un receptor WiFi USB para mejorar la conectividad, permitiendo la emisión y recepción simultánea de señales y asegurando una comunicación estable con dispositivos externos. Se destaca la integración de Unity y Oculus Quest 2 para un control inmersivo en realidad virtual, permitiendo la visualización en tiempo real desde la cámara del robot. Además, se desarrolló un chatbot en Telegram que facilita el envío de comandos, configuración de recordatorios y activación de alertas de emergencia, con la capacidad de monitorear parámetros ambientales gracias a un sensor AHT10. Estas mejoras optimizan la interacción y funcionalidad del sistema en el entorno doméstico.

En el capítulo 5, se realiza un sumario de todo lo abordado en este informe, con su respectiva conclusión, en la que se describen las realizaciones más importantes del informe, se comentan de los posibles trabajos futuros necesarios para mejorar y expandir las capacidades del robot, que puedan adaptarse de manera efectiva a las diversas necesidades de sus usuarios.

Capítulo 2. Tony Pi

2.1. Introducción

En este capítulo, se proporcionará una descripción de los componentes, estructura, características técnicas del robot humanoide Tony Pi, así como una visión general de su lenguaje de programación y biblioteca de procesamiento de imágenes. El propósito fundamental de este capítulo es presentar en detalle el diseño y funcionamiento del robot Tony Pi.

2.2. Descripción

El robot está equipado con una potente Raspberry Pi 4B con 4GB de RAM como su controlador central, servomotores de alta tecnología y una cámara de alta definición con dos grados de libertad (2-DOF). Esta configuración le permite realizar la identificación de imágenes mediante inteligencia artificial, facilitando el reconocimiento de colores e imágenes. La cámara, montada en un soporte de metal y equipada con un servo antibloqueo, captura imágenes en tiempo real, y las envía directamente a un teléfono inteligente o Tablet, proporcionando una visión instantánea y remota del entorno.

Tony Pi utiliza la programación en lenguaje Python y la biblioteca de procesamiento de imágenes OpenCV. Se proporciona el código fuente, tanto para el software de PC como para Python, permitiendo una personalización y modificación completa para adaptarse a las necesidades del usuario. Los servomotores de bus serial de 11.1V pueden reducir la corriente en más del 60%, lo que aumenta la capacidad de la batería y lo hace más eficiente. El robot también cuenta con una pantalla de voltaje para monitorear el estado de energía de la batería.

Además, el software para PC, la aplicación para teléfono y el control de escritorio remoto VNC ofrecen una variedad de opciones para controlar y operar el robot de forma remota y conveniente desde diferentes dispositivos. Tony Pi incorpora una batería de litio recargable de gran capacidad para aumentar su tiempo de uso, también puede ser programado desde una computadora o un teléfono celular para realizar actividades complejas como bailar, caminar o boxear.

2.3. Estructura Tony Pi

2.3.1 Componentes

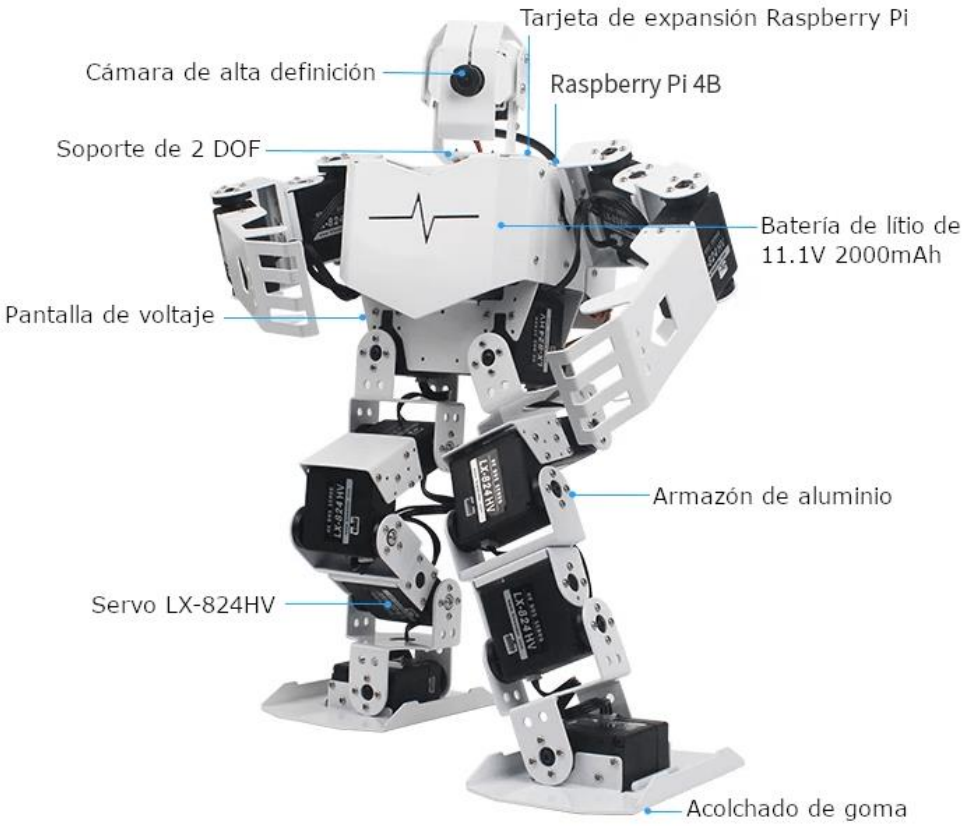

Fig. 2.1: Estructura Tony Pi. Fuente: Hubot Chile, 2024 [13]

En la imagen y descripción, se puede apreciar que el robot está construido sobre un armazón de aluminio, lo que le confiere una estructura más liviana y resistente, con un peso de 1.7 Kg. en el área del pecho, se aloja la batería de litio, mientras que en la parte posterior lleva la Raspberry Pi junto con su tarjeta de expansión. En la parte baja de la espalda, se encuentra la pantalla de voltaje, la que proporciona información, del porcentaje de batería. En la cabeza del robot se destaca una cámara de alta definición. Además, por todo el cuerpo del robot se distribuyen servomotores.

2.3.2 Dimensiones

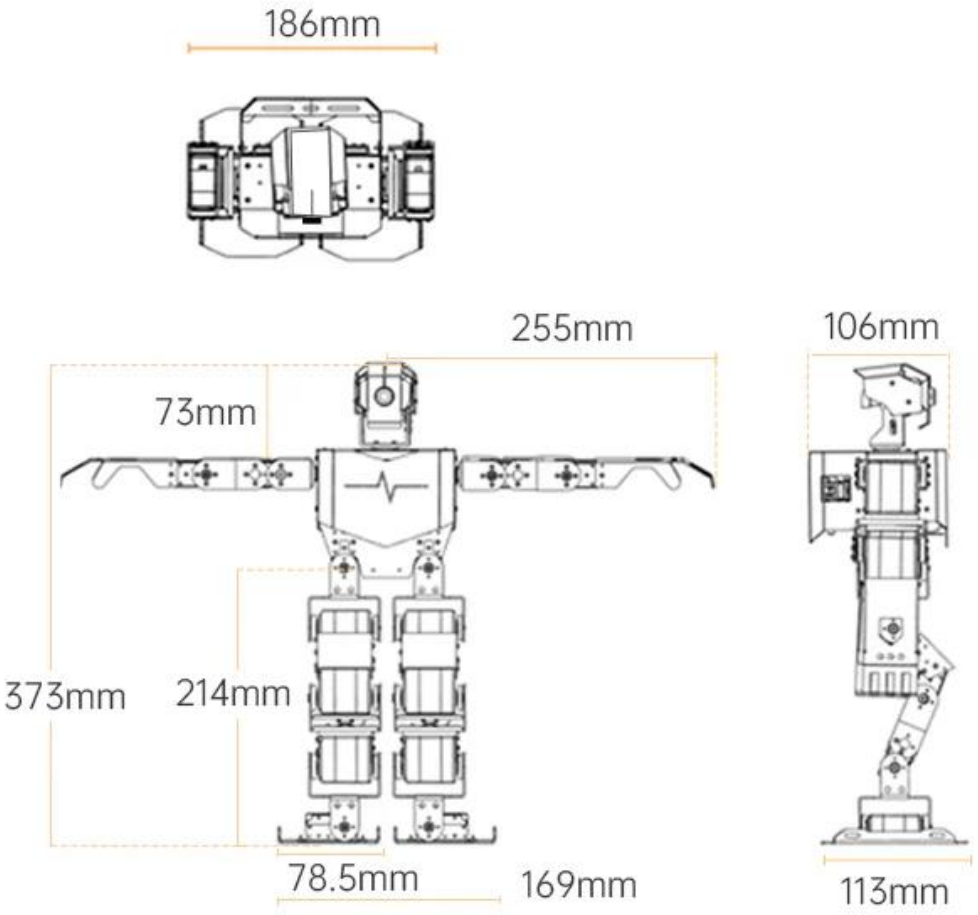

Fig. 2.2: Medidas Tony Pi. Fuente: Hubot Chile, 2024 [13]

El robot Tony Pi presenta dimensiones de 37.3 cm de altura, 18.6 cm de ancho y 11.3 cm de profundidad. Cuando abre completamente los brazos, de mano a mano, sus medidas alcanzan los 25.5 cm.

2.4. Características técnicas

2.4.1 Raspberry Pi

La Raspberry Pi es una computadora de bajo costo y con un tamaño compacto; Puede ser conectada a un monitor de computador o un TV, y usarse con un mouse y teclado. La característica principal de la Raspberry Pi 4B es que está equipada con un procesador Broadcom BCM2711 Quad-core Cortex-A72 (ARM v8) SoC de 64 bits a 1.5GHz, junto con 4GB de memoria LPDDR4,

proporciona una capacidad de procesamiento robusta para ejecutar diversas tareas. En cuanto a la conectividad, dispone de 2.4 GHz y 5.0 GHz IEEE 802.11b/g/n/ac Wi-Fi, Bluetooth 5.0, BLE, y Gigabit Ethernet, lo que permite una amplia gama de opciones para la comunicación inalámbrica y por cable. Además, cuenta con 2 puertos USB 3.0 y 2 puertos USB 2.0 para la conexión de dispositivos.

Para la visualización y el audio, la Raspberry Pi 4B cuenta con 2 puertos micro HDMI, un puerto de pantalla MIPI DSI de 2 carriles, un puerto de cámara MIPI CSI de 2 carriles y un puerto de audio y video, compuesto de 4 polos. Esto permite la conexión a pantallas y dispositivos de audio para interactuar con el entorno de manera efectiva. En términos de multimedia, la Raspberry Pi 4B es compatible con la decodificación de video H.265 (4Kp60) y H.264 (1080p60), así como con gráficos OpenGL ES 3.0, lo que la hace adecuada para aplicaciones multimedia exigentes. La tarjeta Micro SD proporciona la capacidad de cargar el sistema operativo y almacenar datos, mientras que la alimentación se puede suministrar a través del conector USB-C o del encabezado GPIO, con un mínimo de 3 A.

Fig. 2.3: Raspberry Pi 4B. Fuente: Fuente: Raspberry Pi Chile, 2024.[12]

El sistema operativo con que cuenta la Raspberry Pi es el Raspberry Pi OS, el que anteriormente era conocido como Raspbian, es un sistema operativo basado en Linux, específicamente adaptado para la serie de microcomputadoras Raspberry Pi. Basado en Debian, es optimizado para el hardware de la Raspberry Pi, ofreciendo una interfaz amigable y herramientas preinstaladas como Python y Scratch, ideales para la educación y el desarrollo. Hereda la estabilidad y seguridad de Debian, con actualizaciones regulares y un amplio soporte comunitario. Raspberry Pi OS es utilizado para enseñar programación, desarrollar proyectos electrónicos, crear centros de medios con Kodi, automatizar hogares, configurar servidores personales y en proyectos de Internet de las Cosas (IoT), destacándose por su versatilidad y eficiencia en diversas aplicaciones prácticas y recreativas.

2.4.2 HAT (Hardware Attached on TOP):

Se trata de una placa de expansión de hardware diseñada para aprovechar al máximo las capacidades de la Raspberry Pi. Su función principal es permitir que la Raspberry Pi interactúe con el entorno y amplíe sus funciones, mediante la conexión de sensores y actuadores. Esta tarjeta se conecta al conector GPIO de 40 pines de la Raspberry Pi. El HAT proporciona a la Raspberry Pi la capacidad de realizar mediciones analógicas, una función que de otro modo no sería posible debido a la falta de un convertidor analógico-digital integrado en la Raspberry. Además, el HAT puede utilizarse como convertidor de niveles lógicos o como relés o SCR para el control de cargas elevadas, lo que amplía aún más las capacidades de la Raspberry Pi.

La placa de expansión Raspberry Pi incluye protección contra sobre corriente para el puerto PWM de 8 canales, un circuito eléctrico en serie de un solo bus para controlar directamente el servo, luces LED programables de dos canales para mostrar el estado del sistema, y botones programables de dos canales para configurar el sistema de manera conveniente. Además, cuenta con puertos reservados IIC y UART para la compatibilidad con diversos sensores y servomotores.

Fig. 2.4: HAT o tarjeta de expansión Raspberry Pi. Fuente: Hubot Chile, 2024 [13]

2.4.3 Servomotor

También conocido como servo, es un dispositivo que permite controlar con máxima precisión la posición y movimientos de su eje. Esto significa que se puede mover en un ángulo, posición y a una velocidad determinada en cada momento. Los servomotores se destacan por permitir una gran precisión de movimientos, por tener tiempos de respuesta muy cortos y por su larga vida útil y robustez, todo ello por un precio relativamente bajo.

Los 16 servomotores LX824 de Hiwonder que utiliza Tony Pi se destacan por su conector de servo de bus serie que facilita la conexión individual del servo, simplificando el cableado. Además, cuenta con un potenciómetro de alta precisión que proporciona retroalimentación de posición de

ángulo. El LX824 ofrece una amplia gama de datos de retroalimentación, como la posición, temperatura y voltaje, para un control preciso. Su capacidad para proporcionar un torque máximo de 17 kg*cm con rodamiento de bolas doble garantiza un rendimiento confiable en diversas aplicaciones.

Fig. 2.5: Servomotor LX-824. Fuente: Hubot Chile, 2024 [13]

2.4.4 Batería de litio

Es un tipo de batería recargable en la que el litio es el componente activo del electrodo, apreciada por su alta densidad de energía, en relación con su peso y volumen. Además, tienen una tasa de autodescarga más baja que muchas otras tecnologías de baterías recargables, puede perder menos carga cuando no está en uso.

Tony Pi cuenta con una batería de litio de 11.1 V 2000mAh con una duración de funcionamiento aproximado de 1 hora.

Fig. 2.6: Batería de litio. Fuente: Hubot Chile, 2024 [13]

2.4.5 Cámara 2 DOF (Degrees of Freedom)

Es un tipo de cámara que puede moverse y rotar en dos direcciones específicas. Los grados de libertad se refieren a los movimientos que puede realizar un objeto en un espacio tridimensional. En el caso de una cámara 2 DOF, puede moverse en dos direcciones, típicamente giro horizontal y giro vertical, 180° y 130° respectivamente. La inteligencia artificial a través de esta cámara de alta

definición automatiza la interpretación visual del entorno, capturando la imagen desde la cámara procesándola mediante la conversión de color, aplicación de filtros y mascaras para colores específicos, detección de contorno para identificar objetos y finalmente otorgándole comandos al robot para moverse según la posición del objeto detectado.

Fig. 2.7: Cámara 2 DOF. Fuente: Solectroshop, 2024 [14]

Tabla 2.1: Características técnicas. Fuente: Hubot Chile, 2024 [13]

Raspberry Pi 4B	**Procesador:** - Broadcom BCM2711, Quad-core Cortex-A72 (ARM v8) SoC de 64 bits a 1.5GHz **Memoria:** - 4GB LPDDR4 **Conectividad:** - 2.4 GHz y 5.0 GHz IEEE 802.11b / g / n / - LAN inalámbrica de CA, Bluetooth 5.0, BLE - Gigabit Ethernet - 2 × puertos USB 3.0 - 2 × puertos USB 2.0 **GPIO:** - encabezado GPIO estándar de 40 pines **Video y sonido:** - 2 × puertos micro HDMI (hasta 4Kp60 compatible) - Puerto de pantalla MIPI DSI de 2 carriles - Puerto de cámara MIPI CSI de 2 carriles - Puerto de audio compuesto y video compuesto de 4 polos **Multimedia:** - H.265 (decodificación 4Kp60) - H.264 (decodificación 1080p60, codificación 1080p30) - OpenGL ES, gráficos 3.0 **Compatibilidad con tarjeta SD:** - Ranura para tarjeta Micro SD para cargar el sistema operativo y el almacenamiento de datos **Potencia de entrada:** - 5 V CC a través del conector USB-C (mínimo 3 A) - 5 V CC a través del encabezado GPIO (mínimo 3 A) - Alimentación a través de Ethernet (PoE) habilitada (requiere PoE separado HAT) **Medio ambiente:** - Temperatura de funcionamiento 0–50ºC
Placa de expansión Raspberry Pi	- La protección contra sobre corriente incorporada del puerto PWM de 8 canales evita que el servo se dañe. - El circuito eléctrico en serie de un solo bus incorporado puede controlar directamente el servo en serie - Luces LED programables de dos canales, muestran claramente el estado de funcionamiento del sistema - Botones programables de dos canales para configurar más convenientemente el sistema - Puertos reservados IIC y UART que pueden ser compatibles con varios sensores
Servomotores	16 unidades LX-824 servo bus de tres pines
Fuente de alimentación	Batería de litio de 11.1 V 2000mAh, duración de funcionamiento: 1 hora aprox.
Peso del Robot	1.7 Kg aproximadamente

2.5. Métodos de control Tony Pi

2.5.1 Control de escritorio remoto VNC

Tony Pi ofrece múltiples métodos de control, entre los que destaca el uso de un control de escritorio remoto VNC, que sirve para conectarse a la interfaz de la Raspberry Pi inalámbricamente, mediante el control de un PC. Este software proporciona una interfaz intuitiva donde se muestra la estructura completa del robot, así como las diferentes partes que pueden ser controladas individualmente. Como se ilustra en la figura 2.8, los usuarios pueden seleccionar las articulaciones específicas o áreas del robot que desean mover, lo que brinda un control preciso y detallado sobre sus acciones y movimientos.

Además, Tony Pi ofrece una opción adicional para aquellos usuarios que deseen personalizar aún más el comportamiento del robot. Se proporciona un código fuente en Python el que puede ser modificado a gusto del usuario. Esto permite una mayor personalización y adaptabilidad, ya que los usuarios pueden ajustar el comportamiento del robot, según sus necesidades específicas y preferencias individuales.

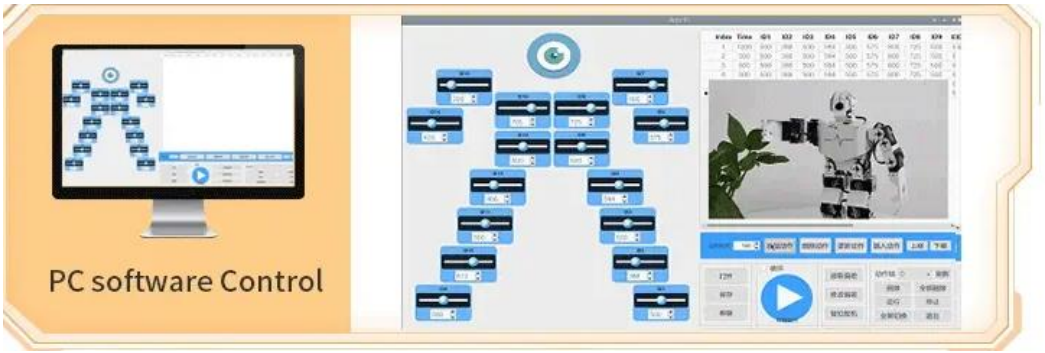

Fig. 2.8: Software de control a través del computador. Fuente: Hubot Chile, 2024 [13]

2.5.2 Control aplicación móvil

Ofrece una opción adicional de control a través de una aplicación móvil, la que proporciona una variedad de movimientos predefinidos que el usuario puede seleccionar fácilmente. Estos movimientos incluyen seguir una línea o un color detectado por la cámara del robot, patear una pelota, así como un control completo del robot que permite al usuario manejar tanto la cámara como el cuerpo del robot de manera directa.

Fig. 2.9: Control mediante aplicación móvil. Fuente: Elaboración propia.

Movimientos predefinidos Tony Pi a través de aplicación:

- **Seguimiento de color:** La cámara de Tony Pi puede procesar imágenes través de la biblioteca de computación visual OpenCV, para recibir las posiciones de objetos de colores rojo, azul y verde, de las que el robot hace un seguimiento visual a través de su cámara del color previamente seleccionado en tiempo real.

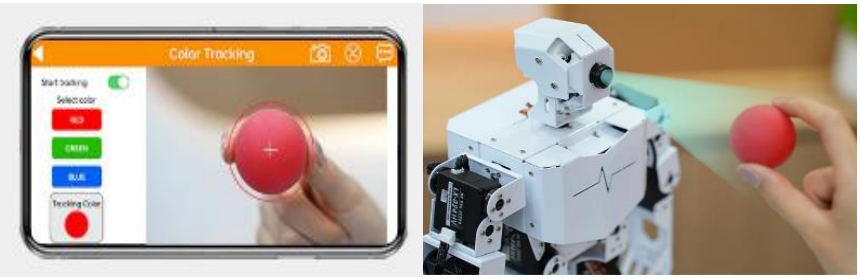

Fig. 2.10: Seguimiento de color en aplicación y demostración. Fuente: Hiwonder, 2024 [15]

- **Seguimiento de línea:** Tony Pi puede reconocer líneas de color negro, blanco o rojo a través de su cámara y de OpenCV. A través de esta línea, el robot calcula su posición para avanzar en ella y la ajustar la ruta, por si hay alguna curva o vuelta.

Fig. 2.11: Seguimiento de línea y demostración. Fuente: Hiwonder, 2024 [15]

- **Reconocimiento de etiquetas:** Tony Pi puede realizar acciones al escanear un código Qr previamente guardado. Además, se pueden programar más códigos Qr para realizar más tipos de acciones o gestos.

Fig. 2.12: Escaneo Qr y demostración. Fuente: Hiwonder, 2024 [15]

- **Reconocimiento facial:** Al usar esta acción predeterminada, Tony Pi, a través de OpenCV, puede realizar la acción de detección facial, la cual, al reconocer un rostro humano, el robot saluda a la persona que detecto. El algoritmo de inteligencia artificial que se utiliza para el reconocimiento facial es el clasificador en cascada de Haar, una técnica basada en aprendizaje supervisado que emplea una serie de clasificadores simples entrenados con el método AdaBoost. Esta técnica permite detectar características faciales como ojos, nariz y boca de manera eficiente en tiempo real.

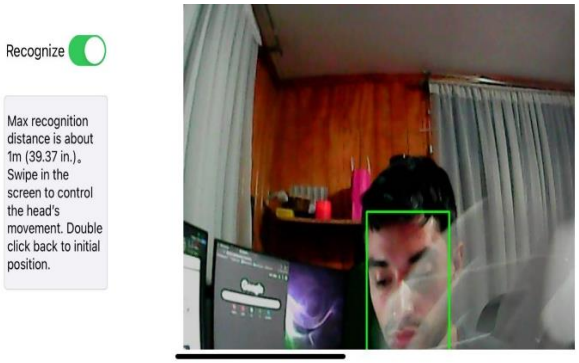

Fig. 2.13: Reconocimiento facial y demostración. Fuente: Elaboración propia

- **Tiro automático:** Tony Pi puede al realizar un procesamiento de imágenes, a través de OpenCV y obtener la posición de una pelota, luego, utilizar un algoritmo PID para seguirla y patearla automáticamente**.**

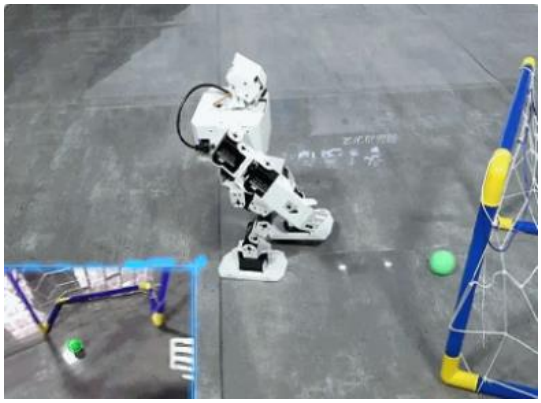

Fig. 2.14: Demostración tiro automático. Fuente: Hiwonder, 2024 [15]

- **Transporte automático:** Tony Pi puede identificar visualmente la distancia de un objeto de color verde, azul o rojo, para, posteriormente, transportarlo a la etiqueta o código Qr que representa el color.

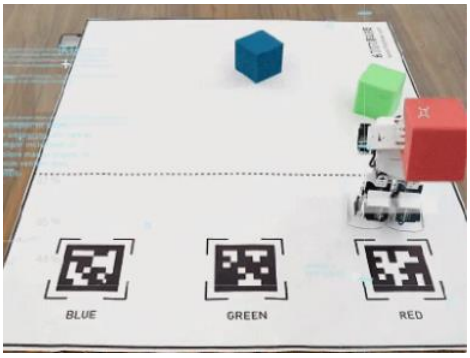

Fig. 2.15: Demostración de transporte automático. Fuente: Hiwonder, 2024 [15]

2.5.3 Control a través de mando de PS2

Tony Pi puede ser controlado mediante un joystick de PlayStation 2 con Bluetooth, conectado a un puerto USB, que permite manejar el robot de manera cómoda y precisa, utilizando un dispositivo de juego popular y ampliamente disponible.

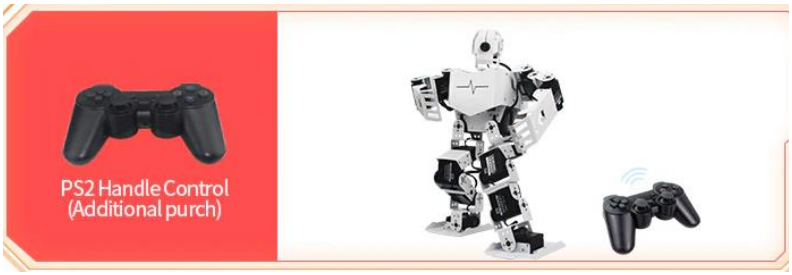

Fig. 2.16: Control PS2 compatible. Fuente: Hubot Chile, 2024 [13]

2.5.4 Control a través de software para PC

Se debe conectar al Wifi que emite el robot. Una vez conectado, el uso de Tony Pi en el computador se realiza de manera similar a su uso a través del teléfono. Esto implica acceder a la interfaz de usuario desde el navegador web o una aplicación específica en el PC, permitiendo el control y monitoreo del robot.

Fig. 2.17: Control mediante PC. Fuente: Hiwonder, 2024 [15]

2.6. Expansión de Tony Pi

Tony Pi tiene una alta capacidad de expansión, se pueden agregar sensores en diferentes partes de su cuerpo o módulos electrónicos adicionales. Esta capacidad de expansión hace que el robot sea más práctico y versátil. Por ejemplo, se pueden incorporar sensores de proximidad, temperatura, humedad y otros, para mejorar su percepción del entorno.

Capítulo 3. Identificación y análisis del problema

3.1. Introducción

Este capítulo se centra en la identificación y análisis de las necesidades de personas con movilidad reducida dentro de sus hogares, basándose en los resultados de una encuesta específica. También, se describen los resultados, destacando las áreas donde el robot Tony Pi puede proporcionar una asistencia significativa. Se exploran posibles usos del sistema robótico, incluyendo recordatorios de medicación y asistencia.

3.2. Encuesta sobre necesidades de un adulto con poca movilidad

Se realizó una encuesta a un total de 50 personas, donde se incluían hombres y mujeres de las ciudades de Concepción, Santiago y Victoria del rango de edad de 20 a 77 años, con el objetivo de identificar y analizar las principales necesidades de asistencia para mejorar la calidad de vida de personas con movilidad reducida a través del uso de un sistema robótico de asistencia en el hogar. Dentro de la encuesta se le dio una serie de alternativas por cada pregunta, para que escogieran la más adecuada según su preferencia.

3.2.1 ¿Qué tipo de asistencia considera más necesaria en su vida diaria?

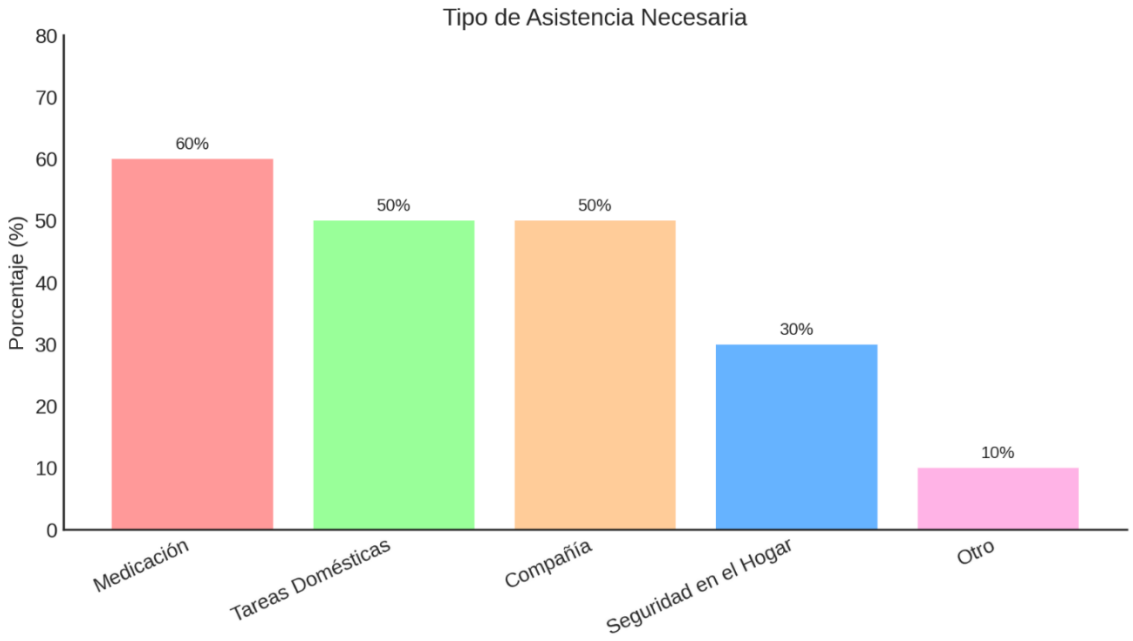

Fig. 3.1: Resultados sobre asistencia necesaria en el hogar. Fuente: Elaboración propia.

La necesidad más destacada es la ayuda con la toma de medicación, seleccionada por el 60% de los encuestados. La asistencia en tareas domésticas con la compañía y el apoyo emocional, también son cruciales con un 50% de las votaciones, seguidos por la seguridad en el hogar, mencionada por el 30%. Además, el 10% de los encuestados identificaron otras necesidades específicas como asistencia para las mascotas.

3.2.2 ¿Con qué frecuencia necesita ayuda para realizar tareas domésticas?

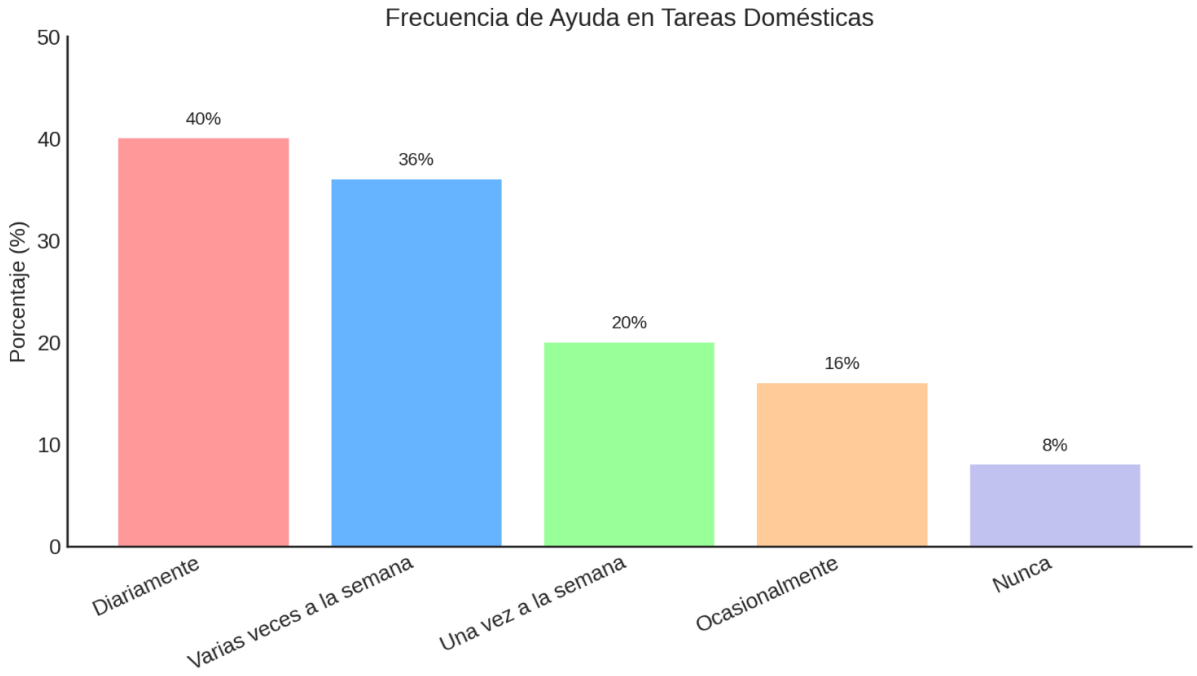

Fig. 3.2: Resultados de frecuencia de ayuda para tareas domésticas Fuente: Elaboración propia.

El gráfico muestra que el 40% de los encuestados necesita ayuda diaria con las tareas domésticas, mientras que el 36% requiere asistencia varias veces a la semana. El 20% necesita ayuda una vez a la semana, el 16% ocasionalmente, y el 8% no requiere ayuda en absoluto. Estos resultados extraídos de la figura 3.2 destacan la variabilidad en las necesidades de apoyo doméstico, destacando la importancia de ofrecer asistencia personalizada que se ajuste a las demandas específicas de cada persona.

3.2.3 ¿Cuánto valoraría tener un recordatorio automático para tomar sus medicamentos?

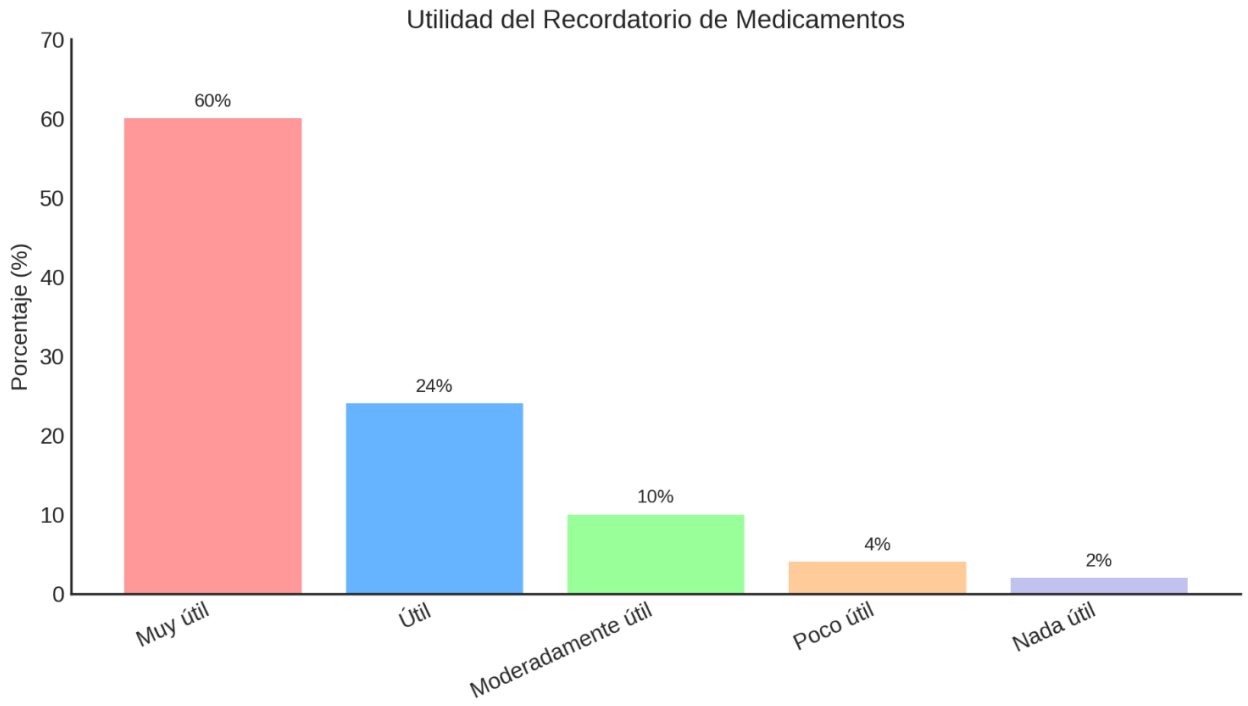

Fig. 3.3: Resultados de recordatorio automático. Fuente: Elaboración propia.

El gráfico muestra que el 60% de los encuestados considera al recordatorio de medicamentos como muy útil, mientras que el 24% lo ve como útil, el 10% lo considera moderadamente útil, el 4% poco útil, y el 2% nada útil. Estos resultados reflejan la alta valoración de esta funcionalidad, ya que ayuda a mantener la salud y el bienestar de los adultos mayores al asegurar que las medicinas se tomen a tiempo.

3.2.4 ¿Cuáles de las siguientes tareas le gustaría que el sistema robótico realizara por usted?

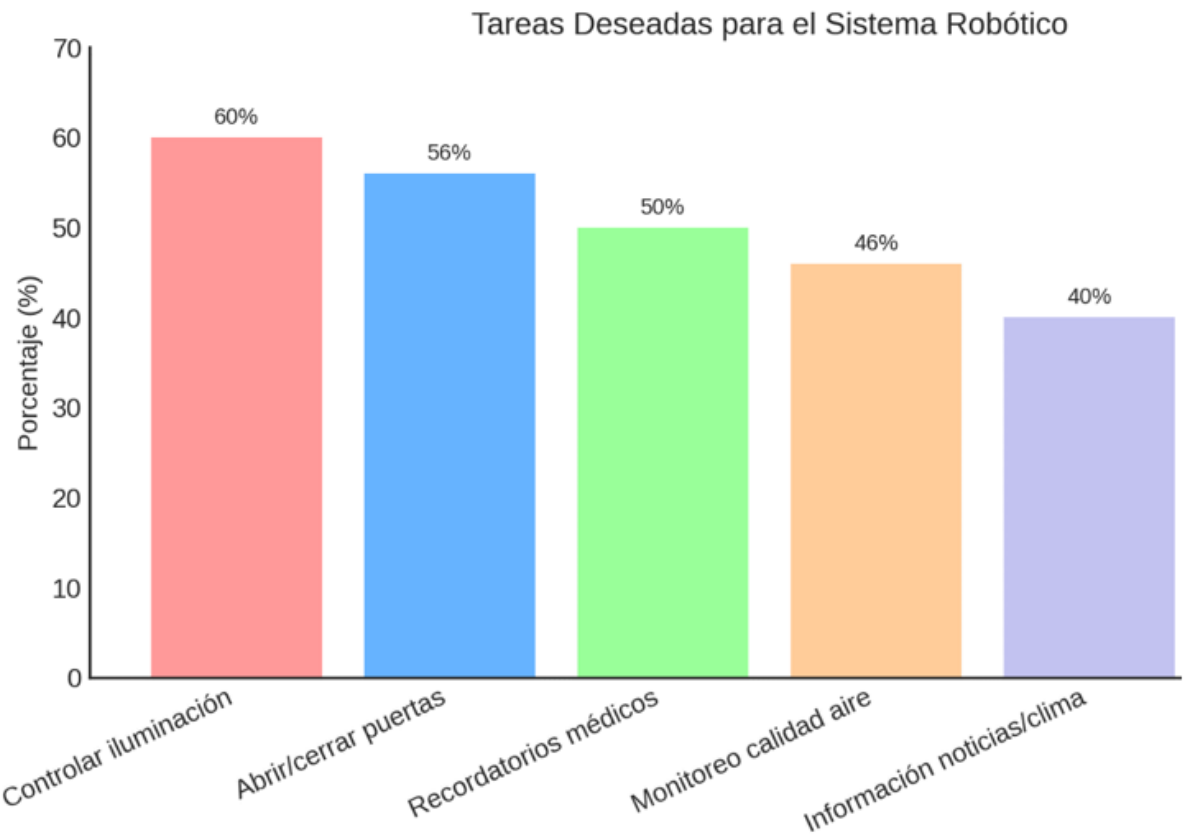

Fig. 3.4 Resultados de tareas realizada por el sistema robótico. Fuente: Elaboración propia.

El gráfico muestra las tareas más solicitadas para el sistema robótico por los encuestados. El 60% desea que el robot controle la iluminación, mientras que el 56% quiere que abra y cierre puertas. Los recordatorios médicos son importantes para el 50%, el 46% considera útil que el robot monitoree la calidad del aire. Además, el 40% de los encuestados desea que el robot proporcione información sobre noticias y clima. Estos resultados indican cómo la tecnología puede integrarse en la vida diaria para facilitar diversas actividades con las que se puedan facilitar y mejorar la calidad de vida de los adultos mayores.

3.2.5 ¿Qué nivel de interacción le gustaría tener con el sistema robótico?

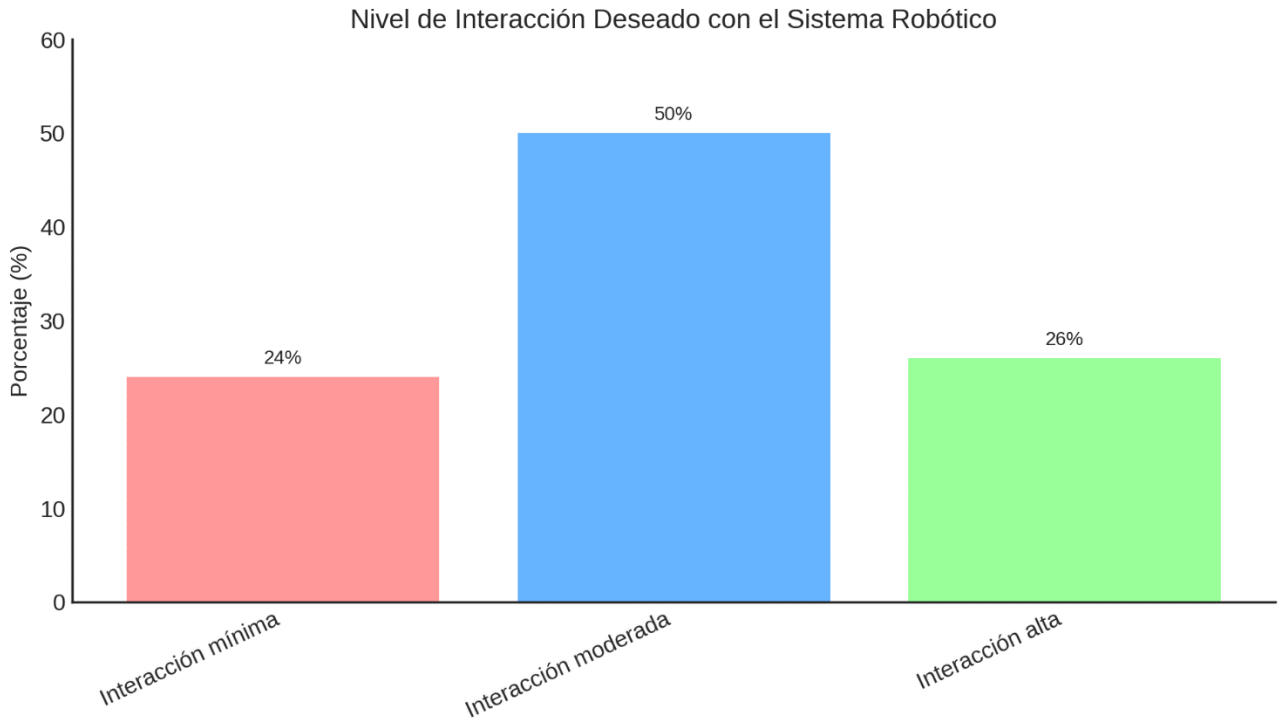

Fig. 3.5: Resultados de nivel de interacción. Fuente: Elaboración propia.

El gráfico muestra las preferencias de los encuestados en cuanto al nivel de interacción deseado con el sistema robótico. El 24% de los encuestados prefiere una interacción mínima, en la que solo realice alertas y recordatorios, mientras que el 50% desea una interacción moderada, en la que el sistema robótico realice comandos simples (información) y respuestas básicas. Un 26% prefiere una interacción alta en la que el sistema robótico pueda realizar conversaciones y una asistencia personalizada. Estos resultados sugieren que, aunque algunos prefieren un enfoque más automatizado, la mayoría de los encuestados valora una interacción más equilibrada y personalizada con el sistema robótico.

3.2.6 ¿Qué tipo de funcionalidades adicionales le gustaría que tuviera el sistema robótico?

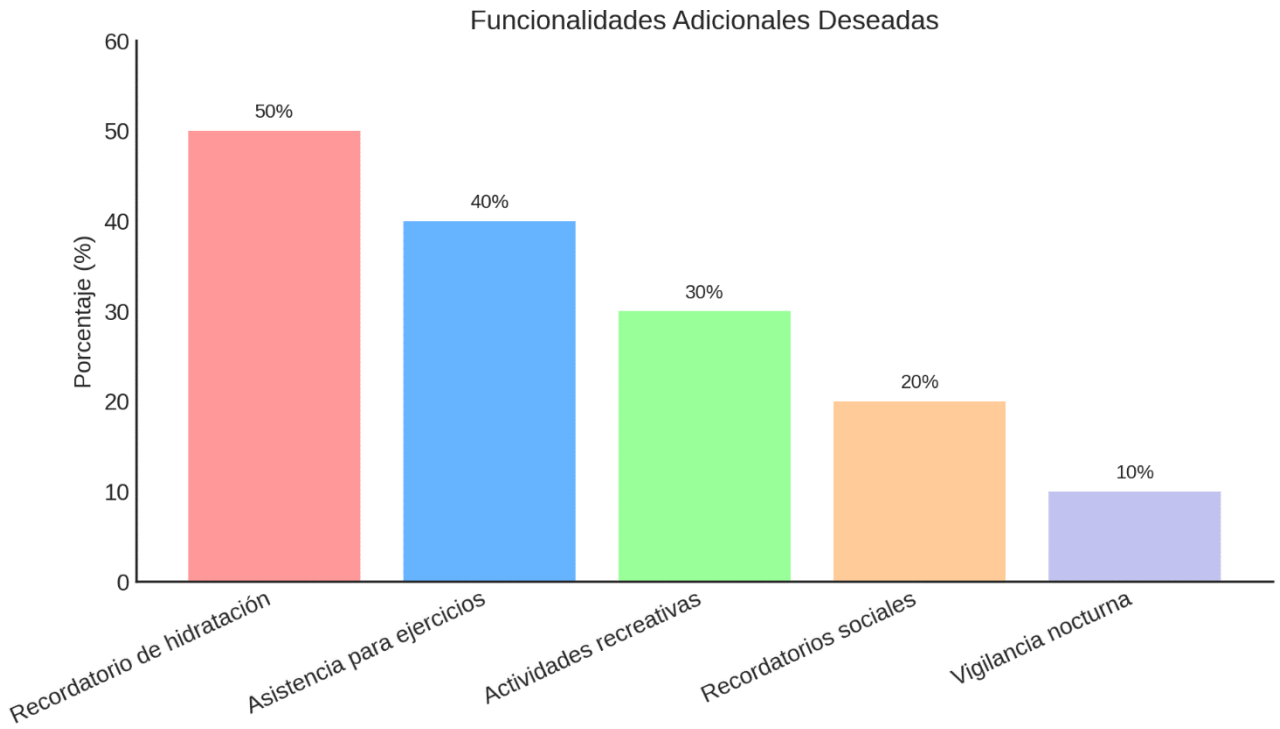

Fig. 3.6: Resultados sobre funcionalidades adicionales. Fuente: Elaboración propia.

El 50% de los encuestados considera importante tener recordatorios de hidratación, mientras que el 40% desea asistencia para ejercicios. Las actividades recreativas son solicitadas por el 30%, y el 20% ve útiles los recordatorios sociales, por último, el 10% de los encuestados desea funcionalidades de vigilancia nocturna. Estos resultados reflejan una preocupación integral por el bienestar, abarcando tanto la salud física como la interacción social y la seguridad.

3.2.7 Comentarios Adicionales

- "Me gustaría que el robot pudiera detectar caídas y alertar a cuidador."
- "La capacidad de programar el robot de manera sencilla".

3.2.8 Análisis de los resultados

Los resultados de las encuestas destacan la importancia de un sistema robótico de asistencia integral que no solo ayude con tareas prácticas, como la toma de medicación y las tareas domésticas, sino que también ofrezca un monitoreo de seguridad. La variabilidad en los requisitos y preferencias de interacción demuestran la necesidad de un diseño flexible y personalizable para el sistema robótico.

3.3. Posibles usos del sistema robótico

Algunos de los posibles usos o configuraciones que se le podrían realizar al sistema robótico podrían ser los siguientes:

- Recordar citas médicas u horarios de medicación, mediante alarmas sonoras y mensajes de voz.
- Ajustar automáticamente la iluminación, la temperatura del hogar para crear un ambiente cómodo y seguro.
- Realizar pequeñas tareas como encender o apagar luces, abrir y cerrar cortinas o puertas, y verificar los niveles de humedad y temperatura del ambiente.

Con estas funcionalidades, Tony Pi no solo mejora la calidad de vida de los adultos mayores, sino también, proporciona tranquilidad a sus familiares y cuidadores, sabiendo que están asistidos en sus necesidades diarias y cuidados en su bienestar general.

Capítulo 4. Modificaciones Tony Pi

4.1. Introducción

En este capítulo, se detalla lo referido a la personalización y expansión que puede realizar, además del proceso de modificación del robot Tony Pi. Las modificaciones realizadas en Tony Pi están diseñadas para aumentar su funcionalidad y efectividad en el entorno doméstico, asegurando que los usuarios puedan llevar una vida más autónoma y segura.

Luego de la encuesta realizada el trabajo se enfocó principalmente en implementar una herramienta de conexión entre usuario y cuidador para mantener una comunicación constante y fluida, además de agregar un nuevo método de control del robot dentro del hogar

Asimismo, se incorporaron nuevas funcionalidades mediante la integración de hardware adicional, como sensores ambientales, micrófono y altavoces, permitiendo al sistema monitorear en tiempo real variables como temperatura y humedad. También se estableció una conexión entre la Raspberry Pi y Unity para habilitar el control remoto del robot a través de gafas de realidad virtual Oculus Quest 2, ofreciendo una experiencia inmersiva y accesible para el usuario. La implementación de un chatbot en Telegram complementó estas mejoras, facilitando la interacción por voz y texto, y permitiendo activar alarmas, configurar recordatorios o enviar mensajes automáticos, todo esto con el objetivo de fortalecer la asistencia personalizada y la seguridad del entorno doméstico.

4.2. Personalización

El sistema robótico, se espera que distinga por su notable capacidad de personalización. Su característica más destacada será la capacidad del robot para programar recordatorios personalizados, como la hora de la ingesta de medicación o eventos importantes.

Esta interfaz incluirá opciones para ajustar el horario de los recordatorios, configurar respuestas específicas del robot y añadir nuevas tareas de manera intuitiva, utilizando solo comandos de voz. Además, se integrará un sistema de reconocimiento de voz mejorado que permita a los usuarios interactuar con el sistema robótico de manera más natural e intuitiva, tanto ajustando el tono y la velocidad de los mensajes de voz del robot, como añadiendo nuevos comandos específicos para tareas personalizadas.

4.3. Expansión

Además de la personalización, la expansión del robot Tony Pi es una característica fundamental que permitirá que el robot evolucione y se adapte a las necesidades cambiantes de los usuarios. La capacidad de expansión incluye tanto el software como el hardware del robot:

4.3.1 Expansión de Software

- **Integración de Nuevas Funcionalidades:** Se añadirán nuevas funcionalidades mediante actualizaciones de software, como las alertas de seguridad y recordatorios.

4.3.2 Expansión de Hardware

Se integrarán componentes físicos, como sensores, micrófono, altavoces y receptores Wi-Fi para optimizar funciones del software.

- **Sensores adicionales**

Se incorporarán sensores adicionales como sensores de temperatura y humedad, con el fin de proporcionar un monitoreo ambiental más completo y ofrecer alertas en tiempo real sobre posibles riesgos.

- **Micrófono USB compatible**

Para el proyecto de configuración del sistema robótico se utilizará el Mini Micrófono Portátil USB 2.0 Adaptador de Audio. Este micrófono se caracteriza por su facilidad de uso, no requiere controladores adicionales, simplemente se conecta y está listo para usarse (plug and play). Una de sus ventajas principales es la calidad de audio superior proporcionada por la conexión digital USB avanzada, lo que garantiza claridad en la captura de voz. Además, el micrófono está equipado con cancelación de ruido, lo que permite filtrar el ruido de fondo no deseado.

Fig. 4.1: Mini Micrófono Portátil USB 2.0. Fuente: Mercado libre Chile, 2024 [16]

- **Altavoces**

Para que el robot pueda proporcionar retroalimentación audible y recordatorios claros, es esencial utilizar altavoces de calidad, los que deben ser lo suficientemente potentes para ser escuchados claramente en diferentes partes del hogar.

- **Configuración de la red Wi-Fi del robot**

Inicialmente, el robot Tony Pi solo podía emitir su propia señal Wi-Fi. Esta configuración dificultaba la interacción con dispositivos externos, ya que únicamente podían conectarse a la red emitida por el robot.

Como solución a este problema, se implementó un receptor de Wi-Fi USB en la Raspberry Pi del robot, lo que permitió que el sistema robótico no solo emitiera su propia señal, sino también pudiera conectarse a redes Wi-Fi externas. Esto facilitó la comunicación del robot con una variedad de dispositivos y, especialmente, a aplicaciones como Unity y Telegram.

Fig. 4.2: Adaptador Wifi USB. Fuente: Mercado libre Chile, 2024 [16]

A través de la interfaz de Raspberry Pi se configuraron dos servidores, uno responsable de recibir Wi-Fi y otro de emitirlo. Esto optimiza el rendimiento de los códigos de Python, permitiendo actualizaciones, sincronización de datos y una mayor capacidad para ejecutar funcionalidades en tiempo real.

Para establecer la conexión a una red Wi-Fi, el código se debe ejecutar directamente en la Raspberry Pi del robot Tony Pi, ya que es este dispositivo el que gestiona las interfaces de red. Esta utiliza el sistema operativo Raspberry Pi OS, basado en Linux.

El procedimiento se realiza abriendo una ventana de terminal en la Raspberry Pi. Esto puede hacerse de dos formas:

1. De forma directa, conectando un teclado, mouse y pantalla a la Raspberry Pi.

2. De forma remota, utilizando un software de escritorio remoto como VNC Viewer desde un computador externo.

Una vez abierta la terminal, se ejecuta el siguiente comando para editar el archivo de configuración de la red Wi-Fi asociada al receptor USB (interfaz wlan1):

```
sudo nano /etc/wpa_supplicant/wpa_supplicant-wlan1.conf
```

Este comando usa el editor de texto nano con permisos de superusuario para poder modificar el archivo. En este archivo se debe ingresar:

- En ssid: el nombre exacto de la red Wi-Fi (entre comillas).
- En psk: la contraseña de dicha red (también entre comillas).

Después de ingresar los datos, se guarda presionando Ctrl + O, luego Enter, y se cierra con Ctrl + X. Finalmente, se reinicia la Raspberry Pi para aplicar los cambios y establecer la conexión a internet.

4.4. Copia de seguridad

Para trabajar y modificar el código de Tony Pi, se debe realizar una copia de seguridad de su configuración inicial. Para ello, se utilizó el programa Win32DiskImager, software que permite crear una imagen exacta de la tarjeta de memoria del robot.

4.5. Nuevo método de control

El proceso de implementación de un nuevo método de control para el sistema robótico permitirá su manejo a través de los lentes de realidad virtual Oculus Quest 2. La motivación detrás de esta implementación es ofrecer una experiencia de control inmersiva, facilitando que los usuarios, especialmente aquellos con movilidad reducida, puedan controlar el robot y desplazarlo dentro de su hogar mediante un entorno de realidad virtual.

4.5.1 Conexión con Unity

Esta funcionalidad de red es crucial para la conexión con dispositivos externos, como Unity y el propio Oculus Quest, que será utilizados en este proyecto para establecer la interacción entre el robot y el dispositivo de realidad virtual

Una vez que el sistema robótico se conecta y opera a través de la red Wi-Fi, el siguiente paso fue integrar Unity, una plataforma para el desarrollo de juegos, a los lentes de realidad virtual, en el proceso de control del robot.

Unity es un motor de desarrollo de software especializado en la creación de aplicaciones gráficas y videojuegos en 2D y 3D. Su versatilidad permite no solo crear videojuegos, sino también desarrollar simulaciones y aplicaciones interactivas en una amplia variedad de plataformas, como consolas, dispositivos móviles y realidad virtual. Unity fue elegido para este proyecto debido a su capacidad de integrar gráficos tridimensionales y conectar dispositivos externos, lo que lo convierte en una herramienta ideal para la interacción entre el robot y el Oculus Quest 2.

4.5.2 Conexión de la cámara con Unity

El primer paso en el desarrollo fue configurar la cámara 2-DOF, conectada a la Raspberry Pi del sistema robótico, para la conexión de esta con la aplicación Unity. La conexión se realiza utilizando un servidor en Python en la Raspberry Pi que transmite las imágenes en tiempo real desde la cámara hacia un cliente en Unity, implementado en C#.

Para optimizar la transmisión, las imágenes capturadas se comprimen en formato JPEG, lo que reduce el tamaño de los datos transmitidos, mejorando la eficiencia de la conexión a través de la red Wi-Fi. En este sistema, el servidor de cámara en Python utiliza la biblioteca OpenCV para capturar imágenes, las que son comprimidas y enviadas mediante una conexión TCP/IP establecida con Unity.

- **Establecimiento de la conexión TCP/IP**

Para enviar las imágenes capturadas desde la Raspberry Pi hacia Unity, se emplea un protocolo de comunicación basado en sockets TCP/IP, que permite establecer una conexión confiable y continua entre el servidor (la Raspberry Pi) y el cliente (la aplicación Unity). El servidor ejecuta las conexiones entrantes en el puerto 8000, enviando los datos de imagen comprimidos en paquetes a intervalos regulares. La Raspberry Pi actúa como servidor y realiza conexiones entrantes desde Unity, enviando los frames de video en tiempo real.

Este sistema de transmisión está diseñado para asegurar que los datos de video se envíen de forma eficiente, minimizando la pérdida de paquetes y garantizando que el cliente Unity reciba cada imagen en el orden correcto. Para mantener una tasa de actualización de aproximadamente 15 FPS, el servidor realiza un pequeño retraso entre cada frame enviado.

- **Recepción de imágenes en Unity**

En Unity, se desarrolló un sistema en C# para recibir los frames de video transmitidos desde la Raspberry Pi y procesarlos para su visualización. El código CameraClient se encarga de recibir los datos de la cámara y convertirlos en texturas que se aplican a superficies dentro de la escena 3D. Esta superficie funciona como una pantalla virtual, mostrando lo que la cámara del robot captura en tiempo real. La conexión TCP permite que las imágenes se reciban de manera continua y en orden.

Cada vez que se recibe un paquete de datos de la Raspberry Pi, Unity lo procesa y actualiza la textura mostrada en la escena, proporcionando al usuario una vista en tiempo real del entorno del robot. El cliente recibe los datos del tamaño de la imagen, seguido del contenido de la imagen misma, que se convierte en una textura para la visualización.

Fig. 4.3: Diagrama de flujo de conexión con la camara. Fuente: Elaboración propia.

- **Pruebas de rendimiento**

Se observó que la transmisión de video tenía un retardo promedio de aproximadamente 1 segundo. Este retardo se debió principalmente a la conexión de internet. Si bien este retraso es perceptible, se consideró aceptable para la mayoría de las aplicaciones de control remoto del robot, donde la prioridad es la estabilidad de la transmisión más que la velocidad instantánea.

Fig. 4.4: Prueba de transmisión de video en Unity. Fuente: Elaboración propia.

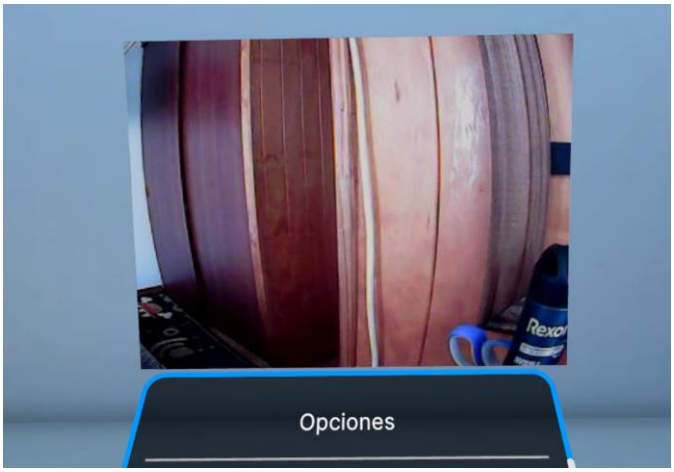

Fig. 4.5: Prueba de transmisión de video desde el sistema robótico a los Oculust Quest 2. Fuente: Elaboración propia.

4.5.3 Conexión de movimientos con Unity

Para controlar los movimientos del sistema robótico mediante Unity, se configuró un servidor de control de movimientos en la Raspberry Pi. Este servidor recibe comandos específicos de movimiento y los traduce en acciones que el robot ejecuta mediante el sistema de ActionGroups. Estos comandos permiten realizar movimientos básicos como avanzar, retroceder y girar, además de movimientos especiales o gestos preprogramados.

- **Establecimiento de la conexión TCP/IP**

La conexión de control de movimientos se realiza mediante una comunicación de socket TCP/IP, en la que Unity actúa como cliente y la Raspberry Pi como servidor. El servidor de movimiento en la Raspberry Pi ejecuta en el puerto 8001, esperando recibir comandos en formato de texto desde Unity, como “go_forward_one_step” o “wave.” Cada comando que se recibe desencadena un ActionGroup en el sistema robótico, que ejecuta la secuencia de movimientos correspondiente.

Este enfoque garantiza una conexión estable, permitiendo que los comandos de movimiento se envíen de forma continua y se ejecuten de inmediato. La comunicación TCP asegura que los datos lleguen de forma confiable, minimizando la posibilidad de pérdida de comandos.

- **Ejecución de comandos en Unity**

En Unity, se utiliza el script MovementClient para enviar comandos de movimiento a la Raspberry Pi. Este script recibe las entradas de los botones del controlador derecho del Oculus Quest 2 y las traduce en comandos que se envían a través de la conexión TCP. Cada vez que el usuario

mueve el joystick o activa un botón en el controlador, Unity envía el comando correspondiente, lo que permite un control intuitivo y en tiempo real del robot.

La respuesta del sistema es inmediata, permitiendo una experiencia de control fluida donde el robot ejecuta los movimientos conforme a las entradas del usuario. Esto permite realizar movimientos complejos en secuencia y ajustar la dirección del sistema robótico en tiempo real.

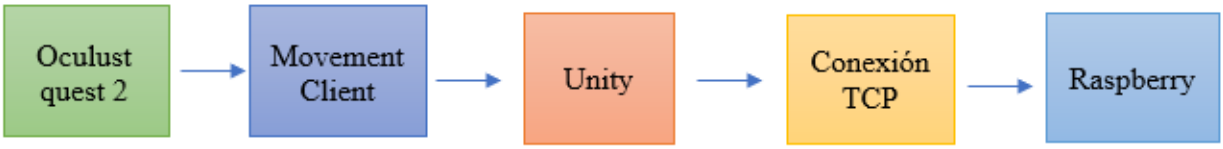

Fig. 4.6: Diagrama de bloques de conexión de movimientos. Fuente: Elaboración propia.

- **Pruebas de rendimiento**

Durante las pruebas de rendimiento, se observó que los comandos de movimiento se ejecutaban con una latencia mínima de aproximadamente 0,5 segundos, lo que es crucial para una experiencia de control inmersiva y sin interrupciones. El sistema TCP permitió que los movimientos se realizaran en el orden correcto, sin pérdida de comandos, incluso cuando se envían en rápida sucesión.

4.5.4 Conexión de los servomotores de la cámara con Unity

El control de los servomotores de la cámara, que permiten ajustar su ángulo vertical y horizontal, se realizó mediante un servidor adicional en la Raspberry Pi, operando en el puerto 8002. Este servidor recibe comandos para modificar los ángulos de los servos, lo que permite al usuario manipular la vista de la cámara en tiempo real.

- **Establecimiento de la conexión TCP/IP**

La conexión TCP/IP para los servomotores utiliza un servidor dedicado en la Raspberry Pi, que realiza las conexiones de Unity. Cuando se envía un comando desde Unity, el servidor ajusta los ángulos de los servomotores de la cámara en función de las entradas recibidas. Este servidor permite cambios rápidos de ángulo en tiempo real, proporcionando una experiencia visual adaptable al entorno del robot.

El servidor limita los ángulos de movimiento para evitar que los servos excedan su rango físico, asegurando un control seguro y estable. Además, se agrega un pequeño retraso entre comandos para permitir que los servos respondan completamente antes de recibir la siguiente instrucción.

- **Ejecución de comandos de Servo de la cámara del robot en Unity**

En Unity, el código ServoClient maneja el envío de comandos a los servos de la cámara. El script detecta las entradas del joystick izquierdo del controlador de Oculus Quest 2, asignando comandos para subir, para bajar, para mover a la izquierda y para mover a la derecha. Estos comandos se envían al servidor de la Raspberry Pi, que ajusta la posición de los servos en tiempo real, permitiendo al usuario controlar la vista de la cámara.

Esta capacidad de ajuste en tiempo real permite al usuario inspeccionar el entorno del robot y cambiar la perspectiva de visualización sin necesidad de mover el cuerpo del robot.

- **Pruebas de rendimiento**

Las pruebas mostraron que los servomotores de la cámara respondieron con una latencia mínima a los comandos enviados desde Unity. La estabilidad de la conexión TCP aseguró que los ángulos se ajustaran de forma precisa y continua, proporcionando una experiencia de control visual detallada y sin interrupciones.

Video demostrativo: https://drive.google.com/file/d/1WzBUggeqyFIPyTkV9Csl4mYhNOIc3dlA/view?usp=drive_link

4.5.5 Conexión del control de servomotores de las manos del sistema robótico

Para proporcionar un control detallado sobre los servomotores de las manos del sistema robótico, se implementó un servidor adicional en el puerto 8003, el que utiliza sockets TCP. Este servidor está diseñado para recibir comandos específicos desde Unity, ajustando los ángulos de los servos de las manos en función de la presión aplicada en los botones de los mandos del Oculus Quest 2. La adición de este servidor amplía la capacidad de interacción y precisión del robot, permitiendo que el usuario ajuste la apertura de las manos en tiempo real.

- **Configuración del control de servomotores de las manos**

Este servidor recibe comandos con el formato id, ángulo, tiempo, donde:

- **Id**: se refiere al identificador del servo (en este caso, el servo derecho es el 17 y el izquierdo es el 18).
- **Ángulo:** representa el ángulo de apertura o cierre deseado.
- **Tiempo:** indica la duración en milisegundos para realizar el movimiento, permitiendo un control suave y progresivo.

En el código implementado en la Raspberry Pi, la función mover_servo procesa los comandos recibidos. Esta función traduce los valores de ángulo en señales PWM, las que son interpretadas por los servos para ajustar su posición. Además, se incluyeron validaciones para garantizar que los ángulos estén dentro de los límites físicos seguros del robot, previniendo daños en el hardware.

- **Integración del Control de Presión en Unity**

Desde Unity, el control de los servos de las manos se integra a través del script denominado GripTesting, diseñado específicamente para capturar la presión ejercida en los mandos del Oculus Quest 2. Este código permite que el usuario ajuste dinámicamente la apertura de las manos del robot, de acuerdo con la presión aplicada.

- **Asignación de Valores:**
    - El script de Unity lee los valores de presión en tiempo real y los convierte en un rango de ángulos para los servos de las manos, utilizando la función ReturnDistance. Esta función mapea la presión aplicada en el botón del mando a un valor entre los límites de apertura de la mano.
    - Dependiendo de la presión, el ángulo de los servos se ajusta para abrir o cerrar la mano del robot, lo que proporciona un control natural para el usuario.

- **Sincronización y Control en Tiempo Real:**

Para evitar una sobrecarga de comandos y lograr un movimiento fluido, se implementaron corrutinas en Unity. Estas funciones introducen un pequeño retraso entre comandos consecutivos, lo que permite que los servos respondan de manera eficiente, evitando comandos redundantes.

El script de Unity recibe las entradas de presión de los mandos del Oculus Quest 2 y ajusta los ángulos de los servos de las manos en función de estos datos. De esta manera, la presión que el usuario aplica en los mandos se traduce en la apertura o cierre de las manos del robot en tiempo real.

- **Pruebas de Funcionamiento**

Se realizaron pruebas para asegurar que el servidor responda correctamente a los comandos enviados desde Unity, además, se evaluó la precisión y la capacidad de respuesta del sistema. Durante las pruebas, se verificó que los servos de las manos ajustan su posición de acuerdo con la presión aplicada en los mandos, logrando una respuesta rápida y un movimiento suave.

4.5.6 Control del movimiento de los brazos del sistema robótico

Para expandir las capacidades de interacción del sistema robótico, se desarrolló un sistema de control específico para los movimientos de los brazos del robot. Este desarrollo permite que los brazos del robot se muevan de manera precisa en los planos vertical y horizontal, sincronizados con los comandos generados en tiempo real por los mandos del Oculust Quest 2, con una comunicación a través del puerto TCP/IP 8003.

- **Control del movimiento vertical**

El control del movimiento vertical de los brazos se basa en la inclinación generada por los movimientos del mando del Oculus Quest 2. Utilizando las posiciones relativas de dos puntos virtuales, pointA y pointB, definidos en el entorno de Unity, se calcula el ángulo necesario mediante la función Mathf.Atan2. Este cálculo toma en cuenta la diferencia de altura y la distancia horizontal entre los puntos.

Los datos de movimiento generados por los mandos del Oculus Quest 2 son transformados en valores angulares que se normalizan dentro de un rango predefinido de seguridad. Estos valores se mapean a un formato interpretable por los servomotores del robot y se envían mediante el método MoverServo. Esto permite que los brazos del robot respondan de manera fluida y precisa a los movimientos verticales realizados por el usuario en tiempo real.

- **Control del movimiento horizontal**

El movimiento horizontal de los brazos se controla mediante la dirección y orientación del mando del Oculus Quest 2 en el plano XZ. La función Vector3.SignedAngle calcula el ángulo relativo entre la orientación del robot y la línea que conecta su posición con el punto objetivo (pointB), ajustándose dinámicamente a los movimientos del usuario.

Los valores angulares generados se ajustan a un rango seguro definido por los límites (minLimitdAngleXZ y maxLimitdAngleXZ). Posteriormente, estos valores son mapeados y transmitidos al servomotor responsable del movimiento horizontal. Esto asegura que los movimientos del robot reflejen las interacciones realizadas con el mando del Oculus Quest 2.

- **Pruebas de funcionamiento**

Durante las pruebas realizadas, al activar los brazos del robot respondieron a los comandos enviados desde Unity, y los movimientos se mantuvieron dentro de los límites establecidos para garantizar la seguridad del hardware. Sin embargo, se observó un retraso en la detección de la posición del mando del Oculus Quest 2 y en la ejecución de los movimientos correspondientes, de parte del robot.

Fig. 4.7: Prueba de activación de los brazos del robot desde los Oculust Quest 2. Fuente: Elaboración propia.
Video demostrativo: https://drive.google.com/file/d/1BW4wAjtKnz98OeF8xgJLmTTaZNzct8WU/view?usp=drive_link

4.5.7 Conexión del controlador Oculus Quest 2 con Unity

La integración del Oculus Quest 2 permite un control intuitivo e inmersivo sobre el sistema robótico, utilizando los joysticks y otros elementos de entrada del controlador VR para enviar comandos a Unity, que luego se transmiten al robot para ejecutar movimientos y ajustar la posición de la cámara.

- **Configuración de controles de acciones en Unity**

Para capturar las entradas del Oculus Quest 2, se utiliza un archivo de InputActionAsset en Unity que contiene el mapeo de las acciones. Este archivo incluye el "ActionMap", denominado XRActionsMovement, que define los eventos de entrada para los joysticks izquierdo y derecho del controlador. Se configuran dos acciones principales:

- **LeftJoystickAction:** Mapea los movimientos del joystick izquierdo, que controlan los ajustes de la cámara del robot.
- **RightJoystickAction:** Mapea los movimientos del joystick derecho, que controlan los desplazamientos del robot en distintas direcciones.

Ambas acciones se activan al inicio del código OnEnable y se asignan los controles para ejecutar los comandos correspondientes de movimiento y ajuste de cámara en Unity.

- ## Ejecución de comandos en base a la entrada del Oculus

En el código OculusControllerInput, el método Update evalúa el valor de los joysticks y, dependiendo de su posición, envía comandos al robot:

- **Inputs de Movimiento (InputsMovement):** Cuando el joystick derecho del Oculus se mueve, el código verifica su dirección (hacia adelante, atrás, izquierda o derecha) y ejecuta comandos específicos de movimiento en el robot a través del cliente MovementClient en Unity.
- **Inputs de Cámara (InputsCamera):** El joystick izquierdo controla los servomotores de la cámara en el robot, permitiendo ajustar su orientación. Dependiendo de la posición del joystick, el código ejecuta comandos que modifican el ángulo de la cámara, enviándolos a través del cliente ServoClient.

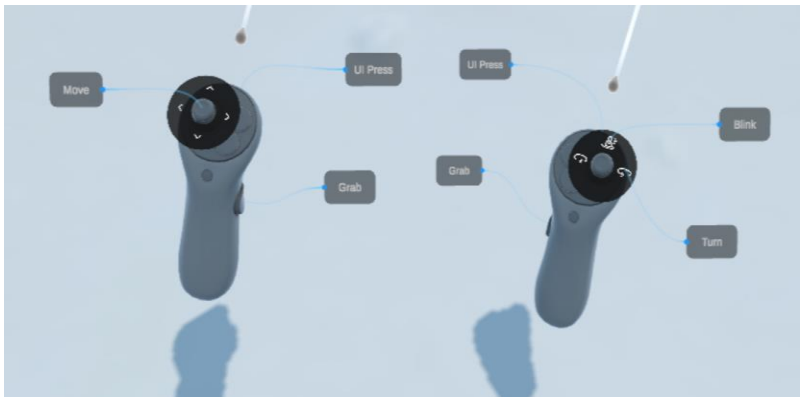

Fig. 4.8: Controles dentro de los Oculust Quest 2. Fuente: Elaboración propia.

- ## Temporización de entradas

Para evitar que múltiples comandos se ejecuten en rápida sucesión y causen un retraso en el sistema, el código utiliza una función de espera (StartDelay) que crea un breve intervalo entre cada entrada, asegurando que cada comando se ejecute de manera adecuada antes de procesar el siguiente. Esto proporciona una experiencia de control más fluida y evita sobrecargar el sistema con comandos repetitivos.

4.5.8 Gestión de conexión centralizada en Unity

El script TonyManager en Unity se encarga de centralizar las configuraciones de conexión, asegurando que los módulos MovementClient, CameraClient, ServoClient y ServoControl estén correctamente configurados y utilicen la misma dirección IP de la Raspberry Pi. Esto permite una configuración rápida y asegura que todos los módulos de control de Unity interactúen de manera consistente con el robot.

- **Configuración de Dirección IP en los Clientes**

El script TonyManager asigna automáticamente la dirección IP de la Raspberry Pi a cada cliente (movimiento, cámara y servos) al iniciar el programa. Esto permite que cualquier cambio en la IP solo tenga que actualizarse en un lugar (en TonyManager), facilitando la administración del sistema.

- **Inicialización de Clientes de Control**

Durante la inicialización (Awake), el script TonyManager configura todos los clientes, permitiendo que Unity se conecte correctamente a cada servidor en la Raspberry Pi. Este enfoque asegura que cada cliente esté listo para recibir y enviar comandos cuando se inicie la aplicación en Unity. Con esta gestión centralizada, el sistema mantiene una configuración consistente, lo que reduce la posibilidad de errores de conexión y mejora la estabilidad de la comunicación.

Fig. 4.9: Demostración del código TonyManager desde los Oculust Quest 2. Fuente: Elaboración propia.

4.5.9 Manejo de errores y cierre de conexiones

Es importante gestionar adecuadamente las conexiones de red para evitar problemas en el caso de errores o cierres inesperados. En cada script de cliente de Unity (MovementClient, CameraClient,

ServoClient, ServoControl), se incluye un método OnApplicationQuit que cierra correctamente la conexión de red cuando el programa de Unity se cierra.

- **Implementación del cierre de conexiones**

En cada cliente, el método OnApplicationQuit cierra el NetworkStream y la conexión de red antes de que finalice la aplicación. Este paso es esencial para evitar problemas de conexión en futuras ejecuciones y liberar recursos de red en la Raspberry Pi.

- **Manejo de errores en la conexión**

En cada cliente de Unity, se captura cualquier error de conexión durante la recepción y envío de datos. En el caso de un error (como pérdida de conexión o tiempo de espera), se muestra un mensaje de depuración que facilita la identificación del problema. Esto permite realizar diagnósticos y asegurar que el sistema se mantenga operativo ante interrupciones menores de conexión.

4.5.10 Proceso de conexión entre Raspberry Pi, Unity y Oculus Quest 2

Para establecer la conexión entre el sistema robótico, Unity, y Oculus Quest 2, se sigue un proceso que permite al usuario controlar el robot en un entorno de realidad virtual. A continuación, se detalla cada paso del proceso de sincronización de los múltiples dispositivos y aplicaciones.

- **Conexión del Oculus Quest 2 con el PC mediante Meta Quest Link**

El primer paso para habilitar el control del sistema robótico mediante Oculus Quest 2 consiste en establecer la conexión entre el visor y el computador que ejecuta Unity. Para ello, se utiliza Meta Quest Link, una aplicación que facilita la comunicación entre los lentes de realidad virtual y Unity en la computadora. Meta Quest Link ofrece tanto conexión física mediante un cable USB-C como una conexión inalámbrica, siempre que la computadora y Oculus Quest se encuentren en la misma red Wi-Fi.

Al establecer la conexión con Meta Quest Link, la aplicación Unity sincroniza el visor y los controladores de Oculus Quest 2, lo que permite que los datos de entrada (joysticks y botones) se integren directamente en el entorno Unity.

- **Ejecución del código en la Raspberry Pi a través de VNC Viewer**

Una vez configurada la conexión entre Oculus Quest 2 y Unity, el siguiente paso es encender el robot. Para luego establecer conexión de forma remota desde la computadora mediante VNC

Viewer, lo que permite al usuario acceder a la interfaz de la Raspberry Pi y ejecutar los scripts necesarios. Estos códigos en Python configuran y activan servidores de control, encargados de transmitir la señal de la cámara, recibir comandos de movimiento y ajuste de servomotores desde Unity.

La ejecución del código Python en la Raspberry Pi establece conexiones TCP/IP en los puertos 8000, 8001, 8002 y 8003 para diferentes funcionalidades. Este proceso asegura que Unity pueda interactuar con el robot en tiempo real, mediante comandos precisos, y que la señal de video, captada por el robot, sea transmitida de forma continua al visor de Oculus.

- **Ejecución del entorno de control en Unity para manipular el sistema robótico**

Con la conexión establecida en Meta Quest Link y el código en la Raspberry Pi en funcionamiento, el usuario puede iniciar el entorno de control en Unity. Esta aplicación, ejecutada en la computadora, proporciona una interfaz gráfica-interactiva, en la que el usuario observa la transmisión de video captada por la cámara del sistema robótico y, mediante los controles de Oculus Quest 2, puede manipular los movimientos y la posición de la cámara del robot.

Las entradas del controlador en Oculus Quest 2 envían comandos directamente al robot a través de la conexión con la Raspberry Pi, permitiendo el control de sus movimientos y la orientación de la cámara. Esta integración de Unity y Oculus, en un entorno de realidad virtual inmersiva, facilita la navegación del sistema robótico y mejora significativamente la experiencia del usuario al ofrecer control en tiempo real y ajustes de cámara precisos en función del entorno.

- **Diagrama de conexión entre dispositivos y aplicaciones**

A continuación, se presenta un diagrama que ilustra las conexiones entre los dispositivos y aplicaciones involucrados en el sistema. Este esquema visualiza el flujo de datos entre Oculus Quest 2, Unity, VNC Viewer, Raspberry Pi y sistema robótico, proporcionando una referencia visual sobre la estructura de comunicación del sistema.

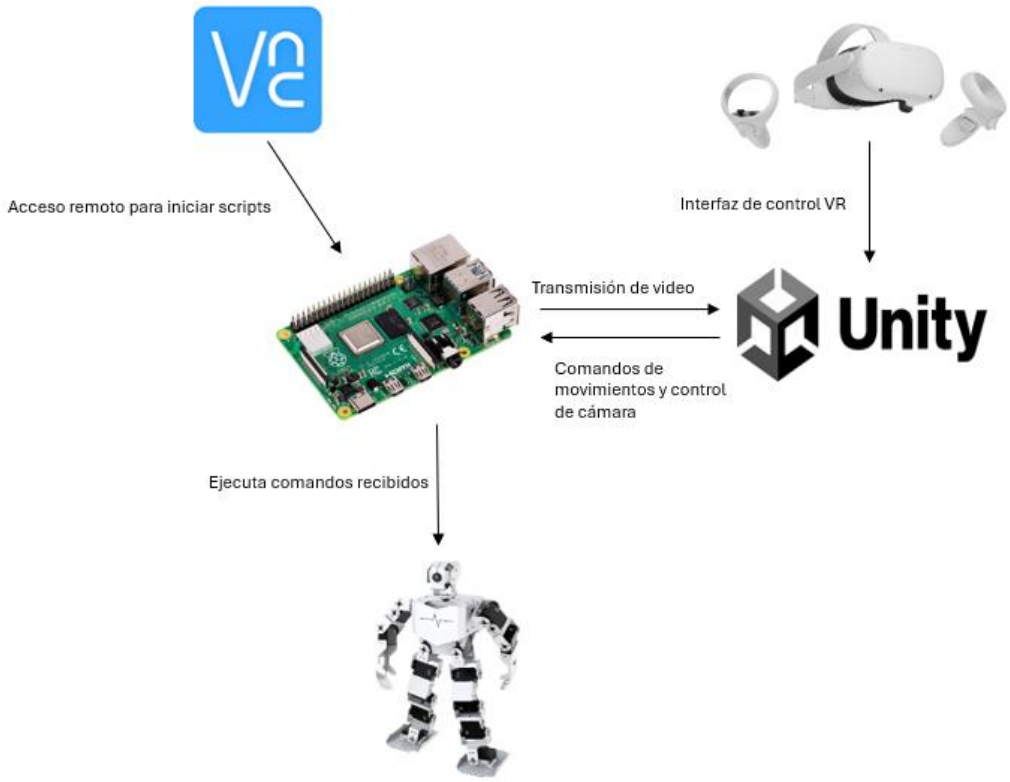

Fig. 4.10: Diagrama de conexión entre dispositivos y aplicaciones. Fuente: Elaboración propia.

4.5.11 Instalación de aplicación en Oculust Quest 2

Para poder simplificar la conexión entre los dispositivos, se utilizó la aplicación realizada en unity con la finalidad de subirla como "juego" dentro del visor Oculust Quest 2, utilizando Oculust Developer hub. Esto permite una conexión más rápida y eficiente, ya que solo se debe contar con el sistema robótico y el visor conectados a la misma red Wi-Fi, evitando el uso de intermediario como el computador

El proceso de instalación se debe conectar el Oculust Quest 2 al computador mediante un cable USB-C, posterior a esto aparecerá una notificación solicitando autorización de depuración USB, luego de confirmar se abre el programa Oculust Developer Hub, que mostrará el dispositivo como conectado si esta correctamente enlazado, con el visor reconocido en la plataforma, se procede a exportar la aplicación desde unity. Para ello, se accede a la configuración del proyecto en Edit > Project setting > Player > Android, donde se ajustan los siguientes parámetros:

- Minimum API level: El cual se establece en 23 (Android 6.0)
- Target API level: Se configura en 30 (Android 11)
- Graphics API: Vulkan o OpenGLES3
- Scripting backend: IL2CPP

- ARM64: activar (requisito fundamental)

Posteriormente, se habilita el soporte para el oculus en la sección en XR Plugin Management, en la cual se debe acceder a Project Settings > XR Plug-in Management > Android y se debe marcar "Oculus"

Finalizada estas configuraciones, se debe generar el archivo APK. En el menú de Build settings, se selecciona la plataforma Android. Si el dispositivo aparece como como opción "run device", puede seleccionarse para realizar pruebas. Al selecciona "Build", unity solicitara un nombre y ubicación para el archivo.

Una vez obtenido la APK, se debe ir a la interfaz de Oculust Developer Hub. En "Device Manager", se selecciona la opción de Oculust Quest 2, se accede a la sección "Manage files" y allí se selecciona "Install APK". Se busca el archivo generado y se confirma la instalación, luego de esto la aplicación se transfiere al visor.

Para poder ejecutar la aplicación solo es necesario usar el Oculust Quest 2, ir al menú principal, seleccionar biblioteca y la categoría "orígenes desconocidos", donde estará disponible la aplicación instalada para la ejecución, este método agiliza la instalación y el uso de aplicaciones desarrollas por unity, integrándolas sin complicaciones al oculust. Gracias a esta implementación, la conexión entre el robot y el visor se establece directamente a través de la red local, permitiendo un control remoto inmersivo, eficaz y libre de configuraciones complejas, ideal para un entorno domestico

4.6. Implementación de chatbot a través de Telegram

La implementación de un chatbot de Telegram para el sistema robótico permitirá una comunicación interactiva entre el usuario y el sistema robótico. El objetivo principal de esta modificación es permitir que los usuarios se comuniquen con el robot de manera remota o directa, utilizando mensajes de texto o comandos de voz. Actúa como la interfaz principal para que los usuarios interactúen con el sistema robótico de forma remota. Permite gestionar alarmas, enviar mensajes, activar alertas de emergencia y revisar parámetros de temperatura y humedad. Esta integración mejora la accesibilidad y autonomía del sistema, especialmente para usuarios con movilidad limitada.

4.6.1 Creación del Bot en Telegram

La creación del bot en Telegram se realizó a través de BotFather, una herramienta oficial de Telegram que permite registrar y gestionar bots de manera segura y fácil. El bot fue registrado con un

nombre único y se generó un token de autenticación exclusivo. Este token actúa como un "identificador" que permite al sistema robótico conectarse de forma segura con la API de Telegram.

Para proteger la seguridad del bot, el token de autenticación se guarda en el código fuente del sistema robótico, pero se maneja de manera cifrada. Esto asegura que solo el robot pueda acceder a las funcionalidades del bot en Telegram, evitando el acceso de usuarios no autorizados que pudieran comprometer la privacidad o seguridad del sistema.

Este bot, una vez creado y configurado, puede ser utilizado desde cualquier dispositivo que soporte Telegram. Esto significa que el usuario y el cuidador pueden interactuar con el robot desde un teléfono móvil o computadora, eliminando la necesidad de una interfaz de usuario adicional. Esto es particularmente útil para los cuidadores que necesitan monitorear o controlar al robot mientras están fuera de casa.

4.6.2 Configuración del reconocimiento de voz

El sistema robótico está equipado con un módulo de reconocimiento de voz, implementado mediante la biblioteca de Python SpeechRecognition. Esta función permite que el usuario se comunique con el bot a través de comandos de voz en lugar de depender exclusivamente de mensajes de texto. El sistema escucha activamente los comandos de voz y utiliza el reconocimiento de voz de Google para interpretar las órdenes emitidas por el usuario.

Para asegurar una experiencia de uso cómoda y precisa, el sistema ajusta su sensibilidad al ruido ambiental. Esto se logra configurando el "umbral de silencio" en el micrófono, que permite al bot diferenciar entre ruido de fondo y comandos de voz claros. Esto es especialmente importante en entornos domésticos, donde el ruido ambiental puede variar considerablemente. El bot ignora los sonidos de fondo y responde solo a los comandos específicos, como "pon una alarma", "mensaje rápido" o "emergencia", que son frases clave para desencadenar funciones específicas.

Por ejemplo, cuando el usuario dice "necesito ayuda", el robot interpreta esta frase, la convierte a texto y envía un mensaje de emergencia al chat de Telegram, asegurando que el cuidador reciba la alerta.

4.6.3 Integración en tiempo real con el Chat de Telegram

La biblioteca python-telegram-bot proporciona el marco para la conexión continua y en tiempo real entre el sistema robótico y Telegram. Esta biblioteca permite que el bot reciba y responda a mensajes de Telegram al instante, lo que es esencial para ofrecer una experiencia de usuario fluida y sin interrupciones.

La biblioteca configura al bot para monitorear continuamente el chat de Telegram. Cuando el usuario envía un mensaje, la biblioteca permite que el bot lo procese en tiempo real y responda de inmediato.

Este sistema de comunicación bidireccional en tiempo real asegura que el bot pueda confirmar cada acción solicitada por el usuario, como la configuración de una alarma o la activación de una alerta de emergencia. Cada vez que el usuario configura una alarma o solicita información, el bot responde inmediatamente.

Las principales funciones implementadas en el sistema son:

- **Envío de mensajes desde el robot**: el robot puede enviar mensajes cuando detecta situaciones específicas, como el sonido de una alarma o una solicitud de emergencia.
- **Recepción y respuesta a mensajes:** el robot está configurado para recibir mensajes desde el chat de Telegram, interpretarlos y realizar las acciones correspondientes, como activar alarmas, responder con la hora actual o enviar recordatorios.

4.6.4 Conversión de texto a voz y respuesta automática

Para mejorar la accesibilidad, el chatbot convierte los mensajes de texto en audio a través de un sistema de texto a voz, basado en el software eSpeak (instalado en la Raspberry Pi). Esto permite que el robot lea en voz alta los mensajes que recibe, asegurando que los usuarios con dificultades visuales o que prefieren la comunicación auditiva, puedan acceder a la información.

Cuando el robot recibe un mensaje, pregunta al usuario si desea responder o ignorar el mensaje. Si el usuario decide no responder, el sistema envía automáticamente un mensaje en el chat indicando que no se enviará una respuesta en ese momento. Esta función permite al usuario ignorar mensajes sin preocuparse por responder de inmediato, mientras que la respuesta automática notifica al remitente que el usuario está ocupado.

Este sistema de conversión de texto a voz también facilita la interacción del usuario con el robot, permitiendo que el sistema le lea mensajes importantes y lo mantenga informado de cualquier notificación en tiempo real.

4.6.5 Flujo de trabajo del Chatbot

El flujo de trabajo del chatbot está diseñado para que cada interacción del usuario siga un proceso lógico y ordenado.

El flujo de trabajo comienza desde el momento en que el bot se activa y empieza a escuchar comandos de voz o recibir mensajes. Cada función del chatbot, desde la configuración de alarmas

hasta la gestión de alertas de emergencia, sigue una serie de pasos bien definidos que ayudan al usuario a completar cada tarea de manera intuitiva y sin errores.

Para asegurar una atención eficiente, el sistema robótico prioriza las funcionalidades en el siguiente orden:

1. **Alertas de ayuda:** Estas son atendidas de forma inmediata y tienen la máxima prioridad dentro del flujo del sistema, ya que pueden representar situaciones críticas.
2. **Alarmas y recordatorios:** Se priorizan para garantizar el cumplimento de tratamientos médicos o actividades importantes del usuario.
3. **Envío y recepción de mensajes:** Permiten mantener la comunicación con familiares o cuidadores.
4. **Funciones adicionales:** Como consultas sobre temperatura, humedad, y acciones personalizadas como preguntar la hora, se ejecutan siempre que no haya tareas urgentes en curso.

El sistema cuenta con comandos de voz específicos, botones interactivos y confirmaciones auditivas que guían al usuario en cada paso del proceso. Por ejemplo, al iniciar una alarma, el usuario selecciona el tipo de alarma y la hora, el sistema verifica cada paso antes de confirmar que la alarma ha sido configurada correctamente.

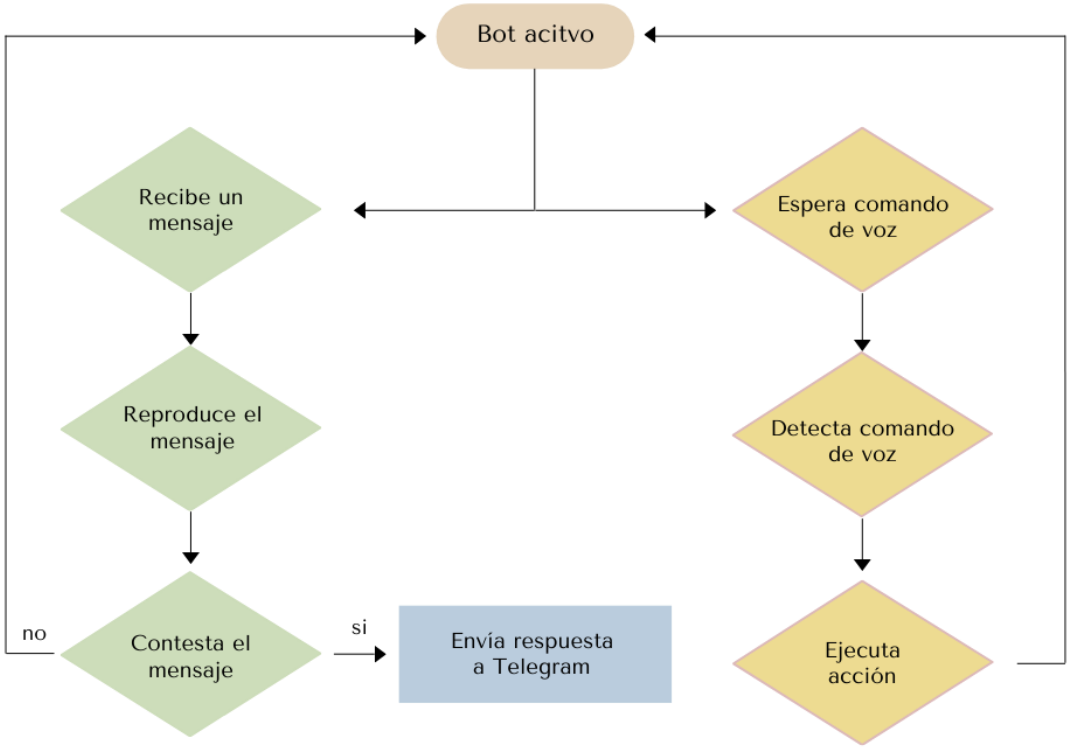

Fig. 4.11: Diagrama de flujo chabot. Fuente: Elaboración propia.

4.6.6 Sistema de emergencia y activación de alertas

El sistema de emergencia del chatbot es una funcionalidad crítica para la seguridad del usuario, permitiéndole activar una alerta en caso de una situación urgente o emergencia. Mediante comandos de voz como “ayuda” o “emergencia”, el usuario puede enviar una alerta automática a través del chat de Telegram. Esta alerta notifica de inmediato a los cuidadores o familiares, quienes pueden ver la notificación en tiempo real.

Una vez activada, la alerta se repite en intervalos regulares, asegurando que el mensaje de emergencia no pase desapercibido. Además, el sistema emite un aviso audible a través del altavoz del robot, confirmando que la alerta ha sido enviada y que los cuidadores han sido notificados. Esta repetición de la alerta es esencial para situaciones donde el usuario puede necesitar ayuda urgente y garantizar que los cuidadores actúen rápidamente.

Para desactivar esta alerta, el cuidador simplemente debe responder al mensaje con cualquier respuesta en texto, lo que desactiva automáticamente la notificación y detiene la repetición de la alerta. Esta facilidad de desactivación permite a los cuidadores reconocer la alerta sin necesidad de emitir comandos específicos, haciendo el proceso más rápido y accesible.

4.6.7 Configuración de alarmas

El chatbot de Telegram incluye un sistema de alarmas que permite a los usuarios configurar recordatorios personalizados, una función especialmente útil para personas que necesitan alarmas recurrentes para recordar medicación, citas o hidratación. Cuando el usuario simplemente escribe la letra “M” en el chat, el bot despliega una serie de opciones interactivas en forma de botones, permitiéndole elegir el tipo de alarma que desea establecer.

Los botones incluyen opciones como "alarma de cita", "alarma de medicación" y "alarma de hidratación". Este enfoque visual simplifica la experiencia de usuario y reduce la posibilidad de errores en la configuración.

Después de seleccionar el tipo de alarma mediante los botones, el sistema guía al usuario para que ingrese la hora en el formato “hh.mm”. Las alarmas configuradas se guardan en una lista de alertas activas y el sistema las revisa periódicamente para activarlas en el momento adecuado. Cuando llega la hora de la alarma, el bot envía una notificación en el chat de Telegram y reproduce un mensaje de voz en el robot, asegurando que el usuario reciba el recordatorio tanto en formato visual como auditivo.

Esta doble notificación mejora la accesibilidad del sistema, especialmente para usuarios que pueden no estar constantemente mirando su dispositivo móvil, ya que recibirán tanto una confirmación visual como auditiva de que la alarma ha sido activada.

4.6.8 Desactivación de alarmas con confirmación de voz

El sistema está diseñado para que el usuario pueda desactivar alarmas activas mediante comandos de voz específicos, como “desactiva alarma” o “desactivar alarma”.

Una vez que el sistema reconoce el comando correcto para desactivar la alarma, el bot envía una notificación de confirmación en el chat de Telegram, informando al usuario que la alarma ha sido desactivada. Además, el sistema emite un mensaje de voz en el altavoz del robot que confirma verbalmente la desactivación. Esta doble confirmación visual y auditiva asegura al usuario que la alarma ha sido apagada con éxito.

Este proceso también permite que el usuario tenga un control total sobre sus alarmas, eliminando la necesidad de verificar manualmente en el chat de Telegram si la alarma ha sido desactivada.

4.6.9 Mensajes rápidos y personalizados

El chatbot permite a los usuarios enviar mensajes rápidos mediante el comando “mensaje rápido”, seguido de la información que desean enviar. Esta funcionalidad convierte las palabras habladas en texto.

Cuando el usuario dice “mensaje rápido” y proporciona el contenido del mensaje, el sistema convierte esta entrada de voz en texto y la envía directamente al chat de Telegram. Esta característica es especialmente útil para situaciones en las que el usuario necesita informar rápidamente a un cuidador o familiar, sin pasar por el proceso de escribir o confirmar el mensaje varias veces.

Además, el sistema confirma verbalmente al usuario que el mensaje se ha enviado con éxito.

4.6.10 Implementación de sensor AHT10 para monitoreo

El sensor AHT10 se incorporó al sistema robótico para facilitar el monitoreo de las condiciones ambientales de manera práctica y en tiempo real. Este dispositivo destaca por su alta precisión, con un margen de error de ±0.3 °C para la temperatura y ±2% para la humedad relativa, lo que garantiza mediciones confiables. Además, utiliza el protocolo I^2C, el que simplifica la conexión con la tarjeta de expansión de la Raspberry (HAT) al ofrecer una comunicación estable y eficiente. Su diseño compacto y su bajo consumo energético lo convierten en una opción ideal.

El proceso de configuración del sensor incluyó su conexión al bus I²C del HAT como su integración en el chatbot de Telegram y el sistema de comandos de voz del robot. Esto habilitó a los usuarios, incluidos cuidadores, a consultar los valores actuales de temperatura y humedad de manera sencilla. Por medio del chat de Telegram, basta con escribir palabras clave como "temperatura", "humedad", "parámetros" o "condiciones" para que el robot responda automáticamente con un mensaje que detalla las mediciones ambientales actuales. Además, se configuró la posibilidad de interactuar mediante comandos de voz, el usuario puede decir estas mismas palabras clave y el robot responderá con un mensaje hablado que informa sobre los valores detectados por el sensor. Los cuidadores también pueden desplegar los botones escribiendo la letra "M" en el chat de Telegram. Al realizar esta acción, el robot muestra opciones como "Alarma de cita", "Alarma de medicación", "Alarma de hidratación" y el botón "Parámetros". Al seleccionar "Parámetros", el sistema responde automáticamente con un mensaje que informa sobre los valores actuales de temperatura y humedad.

4.6.11 Pruebas del chatbot

Durante el desarrollo del chatbot, se realizó una serie de pruebas para garantizar que las interacciones entre el robot y los usuarios fueran fluidas y precisas. Estas pruebas incluyeron:

- **Pruebas en reconocimiento de voz:** se verificó que el robot pudiera identificar las palabras clave de manera consistente, incluso en entornos con ruido de fondo.
- **Pruebas en envío de mensajes:** se probó que el sistema enviara correctamente los mensajes de texto a Telegram, tanto a partir de comandos de voz como de texto.
- **Pruebas en situaciones de emergencia:** se realizaron simulaciones de emergencias para asegurarse de que el sistema enviara alertas repetitivas hasta recibir una respuesta.
- **Prueba en situaciones reales:** En esta prueba específica, el chatbot se activó en una sala con varias personas hablando simultáneamente. Durante la prueba, se observó que el micrófono del robot captaba constantemente las conversaciones de las personas, lo que resultó en que el sistema tratara de armar oraciones continuamente. Debido a que nunca se detectó un silencio de más de 5 segundos, el chatbot no pudo ejecutar ninguna acción correctamente, ya que no logró reconocer un comando claro entre las múltiples voces. Esta prueba permitió identificar la necesidad de implementar filtros de ruido o ajustes en la sensibilidad del reconocimiento de voz para ambientes más ruidosos.

4.6.12 Ejemplos de uso de las funcionalidades del Chatbot

A continuación, se presentan ejemplos prácticos de cómo utilizar cada funcionalidad del chatbot en Telegram, facilitando al usuario la comprensión de cómo interactuar con el sistema robótico para configurar alarmas, activar alertas de emergencia, enviar mensajes y revisar parámetros ambientales.

Ejemplo 1: Configuración de una alarma de medicación

Para configurar una alarma para recordatorios de medicación, el usuario debe:

1. Escribir la letra "M".

Después de enviar el mensaje, el bot desplegará botones interactivos con opciones como "Alarma de cita", "Alarma de medicación" y "Alarma de hidratación".

- **Paso 1:** El usuario selecciona "Alarma de medicación".
- **Paso 2:** El bot solicitará la hora de la alarma en el formato "hh.mm”
  - **Ejemplo de respuesta:** El usuario escribe "14:30".
- **Resultado:** El bot confirma que la alarma ha sido configurada para las 14:30, enviando un mensaje en el chat y reproduciendo un mensaje de voz en el robot que indica: "Alarma de medicación configurada para las 14:30".

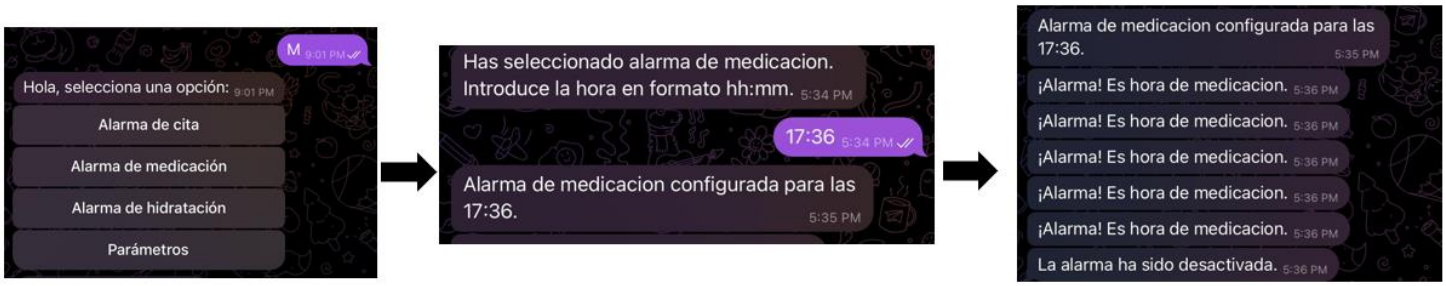

Fig. 4.12: Demostración de configuración de alarma. Fuente: Elaboración propia.

Ejemplo 2: Desactivación de una Alarma con Comando de Voz.

Cuando una alarma está activa, el usuario puede desactivarla utilizando comandos de voz. Esto evita que el cuidador al escribir o usar el dispositivo móvil pueda apagar la alarma.

- **Ejemplo:** El usuario dice “desactiva alarma” o “desactivar alarma”.
- **Resultado:** El bot desactiva la alarma y envía un mensaje de confirmación en el chat de Telegram: "La alarma ha sido desactivada". Además, el robot reproduce un mensaje de voz que confirma la desactivación de la alarma.

Ejemplo 3: Activación de una alerta de emergencia.

Para activar una alerta de emergencia, el usuario puede emitir un comando de voz diciendo “necesito ayuda” o “emergencia”. Al captar esta frase, el robot enviará automáticamente un mensaje de alerta a los cuidadores o familiares en el chat de Telegram.

- **Ejemplo:** El usuario dice en voz alta “necesito ayuda”.
- **Resultado:** El bot envía un mensaje en el chat de Telegram que dice: "ALERTA: La persona necesita ayuda". Este mensaje se repetirá cada 10 segundos hasta que el cuidador responda con cualquier mensaje en el chat, lo que detiene la alerta.

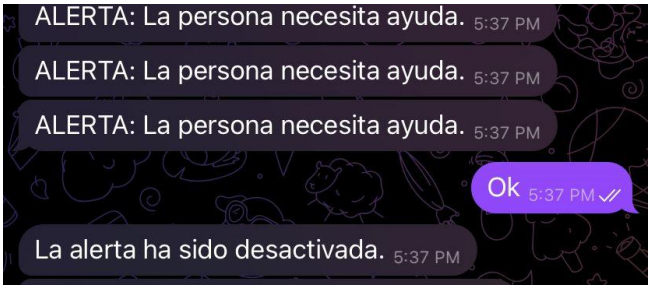

Fig. 4.13: Demostración de activación de alerta. Fuente: Elaboración propia.

Ejemplo 4: Envío de Mensaje Rápido.

Para enviar un mensaje de manera rápida y sencilla, el usuario puede utilizar el comando de voz “mensaje rápido” seguido del contenido del mensaje. Este mensaje se envía de inmediato sin necesidad de confirmación adicional.

- **Ejemplo:** El usuario dice “mensaje rápido: llego en 5 minutos”.
- **Resultado:** El bot convierte el mensaje de voz en texto y lo envía al chat de Telegram como: “llego en 5 minutos”. El robot también confirma el envío mediante un mensaje de voz: "Mensaje rápido enviado".

Ejemplo 5: Consulta y Respuesta a Mensajes Recibidos en el Chat de Telegram.

El chatbot está diseñado para leer en voz alta los mensajes recibidos en el chat de Telegram y preguntar al usuario si desea responder. Esto permite al usuario mantenerse informado sin tener que revisar el chat manualmente.

- **Paso 1:** El cuidador envía un mensaje en Telegram que dice “¿Necesitas algo?”.
- **Paso 2:** El robot lee el mensaje en voz alta y pregunta “¿Quieres responder el mensaje?”.
- **Paso 3:** Si el usuario dice “sí, quiero responder”, el robot le pide que dicte su respuesta.
- **Paso 4:** El usuario dice “Sí, necesito algo para comer”.

- **Resultado:** El bot envía la respuesta en el chat de Telegram y reproduce el mensaje de voz: "Respuesta enviada: Sí, necesito algo para comer".

Ejemplo 6: Consulta de parámetros ambientales.

El sistema permite a cuidadores y usuarios consultar las condiciones ambientales de manera sencilla, ya sea a través del chat de Telegram o mediante comandos de voz. Esto asegura que tanto quienes se encuentran cerca del robot como aquellos en ubicaciones remotas puedan acceder a la información en tiempo real.

Instrucciones:

[1] El cuidador puede escribir en el chat de Telegram cualquiera de las palabras clave: "temperatura", "humedad", "parámetros", "condiciones" o la letra “M”, para que despliegue los botones para seleccionar los parámetros.

[2] El usuario, desde la ubicación del robot, puede decir en voz alta cualquiera de estas mismas palabras clave.

Ejemplo:

- **Ejemplo 1:** El cuidador escribe "parámetros" en el chat de Telegram.
  - **Resultado:** El bot responde en el chat con la temperatura y humedad actual

Fig. 4.14: Demostración de ejemplo 6.1. Fuente: Elaboración propia.

- **Ejemplo 2:** El usuario pregunta por voz ¿Cuál es la temperatura y humedad?
  - **Resultado:** El sistema robótico responde con un mensaje de voz: “La temperatura actual es 21.46 grados Celsius y la humedad es 57.09 por ciento”.

Capítulo 5. Conclusiones

5.1. Sumario

Este proyecto aborda el desarrollo y la modificación del robot humanoide Tony Pi, con el objetivo de convertirlo en un asistente para actividades cotidianas dirigido a personas con movilidad reducida dentro del hogar. El propósito principal fue crear un sistema robótico capaz de asistir en tareas cotidianas, mejorar la calidad de vida de los usuarios y aumentar su autonomía. Para lograrlo, se realizaron diversas modificaciones que incluyeron la integración de comandos de voz, así como la conexión con plataformas externas como Telegram, Unity y Oculus Quest 2, lo que permitió un control inmersivo y una interacción en tiempo real.

El sistema de control del robot se basa en una Raspberry Pi 4B con 4 GB de memoria, que actúa como cerebro del dispositivo. Esta plataforma permite un alto nivel de personalización y expansión, gracias al uso de Python como lenguaje de programación y la biblioteca OpenCV para el procesamiento de imágenes. Además, el robot cuenta con servomotores que otorgan la capacidad de movimiento, una cámara 2-DOF para la detección visual y la integración de sensores adicionales como parte del diseño expandible.

Entre las funcionalidades del sistema, se encuentran la implementación de un chatbot en Telegram que facilita la interacción a través de mensajes de texto o comandos de voz, y un nuevo método de control mediante Oculus Quest 2. Este último permite que los usuarios vean lo que captura la cámara del robot en tiempo real y lo controlen mediante un entorno de realidad virtual.

Además, se realizó una encuesta para identificar las necesidades específicas de las personas con movilidad reducida, lo que permitió definir las principales funcionalidades que debía incorporar el robot. Entre los resultados más destacados se identificó la necesidad de asistencia en tareas domésticas, recordatorios de medicación y supervisión de parámetros ambientales. A partir de estas necesidades, se programaron las principales funcionalidades del robot, como la creación de alarmas personalizadas y la capacidad de interactuar a través de comandos de voz.

Las pruebas iniciales del sistema demostraron como resultado un rendimiento positivo en condiciones controladas, aunque se identificaron áreas de mejora en la precisión del reconocimiento de voz en ambientes ruidosos y en la estabilidad de la transmisión de datos a través de Wi-Fi. A pesar de estas limitaciones, el sistema demostró ser efectivo en la asistencia diaria y en la mejora de la calidad de vida de los usuarios.

5.2. Conclusiones

A lo largo de este trabajo, se llevaron a cabo diversas modificaciones para desarrollar un sistema robótico basado en el robot humanoide Tony Pi, con el objetivo de asistir en actividades cotidianas dentro del hogar.

Uno de los principales desafíos iniciales fue lograr que el robot pudiera compartir y emitir señal WiFi de manera estable, permitiendo así su conexión con otros dispositivos y garantizando una comunicación fluida con el entorno. Posteriormente, se avanzó en la integración de comandos de voz mediante bibliotecas como SpeechRecognition y pyttsx3, que permitieron que el robot interpretara y respondiera a instrucciones verbales en tiempo real. Como parte de este desarrollo, se implementó un sistema de alarmas y recordatorios, donde los usuarios pueden configurar alertas para la toma de medicamentos, citas médicas y recordatorios de hidratación mediante el chatbot de Telegram.

Una de las principales implementaciones del proyecto fue un método de control mediante Oculus Quest 2, que permite a los usuarios interactuar con el robot a través de un entorno de realidad virtual. Esta funcionalidad, combinada con la conexión a Unity, facilitó la visualización en tiempo real de lo que captura la cámara del robot y el control preciso de sus movimientos.

La implementación del chatbot supuso una de las fases más significativas del proyecto. Este componente permitió mejorar la interacción entre el usuario y el robot, tanto a nivel local como remoto. A través del chatbot, los usuarios pueden enviar mensajes al robot y recibir notificaciones importantes. En caso de emergencia, el chatbot envía automáticamente alertas a un familiar o cuidador, asegurando que el usuario esté monitoreado de forma continua y en tiempo real.

Durante el proceso de implementación, se enfrentaron diversos retos técnicos, como la estabilidad del reconocimiento de voz en ambientes con ruido de fondo y la transmisión fluida de información entre el robot y los dispositivos externos a través de la red Wi-Fi. En este sentido, fue necesario desarrollar un protocolo de comunicación eficiente para garantizar que los comandos y las respuestas se transmitieran sin interrupciones.

El proyecto incluyó la integración de un sensor adicional para monitoreo ambiental, lo que permitió medir parámetros como temperatura y humedad. Brindado información en tiempo real.

Aunque el sistema robótico ha demostrado ser capaz de proporcionar asistencia en el hogar, todavía se identificaron áreas que requieren mejoras, como la optimización del reconocimiento de voz en ambientes ruidosos y la expansión del sistema para incluir más sensores que ofrezcan monitoreo ambiental.

5.3. Trabajo Futuro

Se pretende configurar al sistema robótico como un asistente de voz, similar a otros asistentes de voz ampliamente utilizados como Google Home, Amazon Echo, Siri, entre otros.

Al configurar de esta manera, se espera lograr una integración efectiva entre la domótica y el robot. Esto permitiría que, mediante comandos de voz, pueda ejecutar diversas acciones en el hogar, como encender una lámpara, cerrar las cortinas o abrir puertas. Esta funcionalidad proporcionará un nivel adicional de independencia a personas con movilidad reducida en la realización de tareas cotidianas. Por ejemplo, el sistema robótico podría ser programado para ajustar los niveles de iluminación y temperatura, según las preferencias del usuario, o, también, si a cierta hora la persona enciende una luz, la podría programar para que todos los días se enciendan a esa hora o se cierren las cortinas. La capacidad del sistema robótico para interactuar con otros dispositivos inteligentes en el hogar mediante comandos de voz convertiría al robot en un centro de control domótico.

Además, otro trabajo a futuro, que el sistema robótico entre en un modo de suspensión cuando no esté en uso, similar al modo de espera en otros dispositivos, para optimizar su eficiencia energética. Se sugiere también la implementación de un cargador que funcione de manera similar al de las aspiradoras robóticas, permitiendo que el robot regrese de manera autónoma a la estación de carga cuando su batería esté baja o cuando no esté en uso. En esta estación de carga, el robot se mantendría en suspensión, conservando su energía hasta que sea activado mediante un comando de voz o una instrucción específica del usuario.

Este enfoque, además de asegurar que el robot esté siempre cargado y disponible para uso inmediato, facilitaría su integración en la rutina diaria del hogar sin necesidad de intervención manual para la recarga, aumentando así su eficiencia y autonomía en el entorno doméstico.

Bibliografía

[1] Barrios Sánchez, R. y Rodríguez, R. "Asistencia de adultos mayores mediante la implementación de robots de servicio en entornos domésticos", tesis, Universidad Piloto de Colombia, 2018.

[2] Valencia Redrovan, D. P. y González Delgado, L. E. "Diseño e implementación de un prototipo de robot asistente para personas con discapacidad motriz y adultos mayores, basados en inteligencia artificial", tesis, Universidad Politécnica Salesiana de Ecuador, 2014.

[3] Revista de Robots. "Robot Pepper: qué es, historia y precio de Pepper." Revista de Robots, 8 de junio de 2023, https://revistaderobots.com/robots-y-robotica/robot-pepper-que-es-historia-y-precio-de-pepper/?cn-reloaded=1.

[4] Vera Repullo, José Alfonso, et. al. Diseño de un sistema integral de atención domiciliaria con robot móvil asistencial. En: XLI Congreso Anual de la Sociedad Española de Ingeniería Biomédica. Cartagena: Universidad Politécnica de Cartagena, 2023.

[5] Escobar Ruiz, D. "Ameca: el robot humanoide más avanzado del mundo y que vale más de USD 100 mil." Infobae, 15 de julio de 2024, https://www.infobae.com/tecno/2024/07/15/ameca-el-robot-humanoide-mas-avanzado-del-mundo-y-que-vale-mas-de-usd-100-mil/.

[6] Tutiven Reyes, Jorge Enrique; Villagomez Galarza, Daniel Euclides. Diseño e Implementación de SLAM y control PID de un robot autónomo Robomaster S1 utilizando Python. 2021. Tesis de Licenciatura.

[7] Ramírez-Pereira M, Figueredo-Borda N, Opazo Morales E. La inteligencia artificial en el cuidado: un reto para Enfermería. Enfermería: Cuidados Humanizados. 2023;12(1): e3372.

[8] Aguilar Villegas, Jamil Esteban. Inteligencia Artificial y Microcontroladores. RITS, La Paz, n. 1, 2008.

[9] Muñiz Castañeda, J. "Mejorar la calidad de vida de las personas en el hogar mediante un sistema de toma de decisiones inteligentes", Instituto tecnológico de colima, México, 2015

[10] Colomer Barbera, Javier. Estudio de los sensores para la detección de obstáculos aplicables a robots móviles. 2018.

[11] O. Torres Acuitlapa, "Exploración basada en sensores para robots móviles", Tesis de maestría, Benemérita Universidad Autónoma de Puebla, 2014

[12] Raspberry Pi Chile, “Raspberry Pi 4 Modelo B, 8GB RAM,” imagen, 2024. [Online]. Disponible: https://raspberrypi.cl/producto/raspberry-pi-4-modelo-b-8gb-ram/

[13] Hubot Chile, “Robot humanoide con visión HD e inteligencia artificial, SKU: 875,” imagen,

2024. [Online]. Disponible: https://hubot.cl/producto/robot-humanoide-con-vision-hd-e-inteligencia-artificial-sku-875/

[14] Solectroshop, “2-DOF Pan-Tilt HAT para Raspberry Pi - soporte inclinable para cámara,” imagen, 2024. [Online]. Disponible: https://solectroshop.com/es/modulos-raspberry-pi-raspberry-pi-hats/5204-2-dof-pan-tilt-hat-para-raspberry-pi-soporte-inclinable-para-camara-16138.html

[15] Hiwonder, “TonyPi Pro Hiwonder Humanoid Robot Professional Development Kit Powered by Raspberry Pi 4,” 2024. [Online]. Disponible: https://www.hiwonder.com/products/tonypi-pro?variant=40213132935255

[16] Mercado Libre, “Plataforma de comercio electrónico en Latinoamérica,” [Online]. Disponible: https://www.mercadolibre.com

Anexos

6.1. Código de Python Chatbot de Telegram

```
import warnings
import telegram
from telegram import InlineKeyboardButton, InlineKeyboardMarkup
from telegram.ext import Updater, MessageHandler, Filters, CommandHandler,
CallbackQueryHandler, ConversationHandler
import logging
import speech_recognition as sr
import threading
import time
from datetime import datetime, timedelta
from sensor import SensorFormatado, SensorWrapper, SensorErrorLecturaException
from gtts import gTTS
from pydub import AudioSegment
from pydub.playback import play
from audio import Audio
from tareas.especificas import (
        Alarmas,
        TareaAlerta,
        TareaAgendarAlarmaEstandar,
        TareaAnunciarCondicionesAmbientales,
        TareaMensajeRapido,
        TareaReproducirHoraActual,
        TareaResponderMensaje,
        TareaTextoAAudio,
    )
from utils import BoolException, TelegramChat, Tiempo
from typing import Optional

warnings.filterwarnings("ignore", category=UserWarning, module='telegram.ext')

# Configura el nivel de logs y el logger del main
logging.basicConfig(format="[%(asctime)s][%(levelname)s][%(name)s][%(threadName)
s] %(message)s",
    level=logging.DEBUG)
logger = logging.getLogger("Main")

# Cambia el nivel de logging de telegram
    "apscheduler.scheduler",
    "pydub.converter",
    "telegram.ext.dispatcher",
    "telegram.ext.updater",
    "telegram.bot",
    "telegram.ext.conversationhandler",
    "gtts.lang",
    "gtts.tts",
    "urllib3.connectionpool"
]
logger.info("Configurando niveles de logs de dependencias...")
for l in loggers:
```

```
    logging.getLogger(l).setLevel(logging.ERROR)
    logger.info(f"'{l}' -> nivel: ERROR")

# Token del bot de Telegram
TOKEN = '7357187267:AAGCsYrBj7wWsiKQi5tEOq1ipWznfW2jTuQ'
# Estados para la conversación
SELECCION_TIPO, INGRESO_HORA = range(2)

# Variables globales
alerta = TareaAlerta()
alarmas = Alarmas()
recognizer = sr.Recognizer()
sensorFormateado = SensorFormatado()
sensor = SensorWrapper()

# Función para iniciar el micrófono en un hilo separado
def iniciar_hilo_microfono(context) -> None:
    hilo_microfono = threading.Thread(target=escuchar_comandos)
    hilo_microfono.daemon = True
    hilo_microfono.start()
    logger.info("Micrófono iniciado...")

# Mostrar botones de alarma sin necesidad de comandos específicos
def mostrar_botones_alarma(update, context) -> int:
    chat_id = update.message.chat_id if update.message else
update.callback_query.message.chat_id
    # Configura el chat id del chat que interactúa con el bot
    TelegramChat().id = chat_id
    botones = [
        [InlineKeyboardButton("Alarma de cita", callback_data='alarma_cita')],
        [InlineKeyboardButton("Alarma de medicación",
callback_data='alarma_medicación')],
        [InlineKeyboardButton("Alarma de hidratación",
callback_data='alarma_hidratación')],
        [InlineKeyboardButton("Parámetros", callback_data='parámetros')],
    ]
    reply_markup = InlineKeyboardMarkup(botones)
    context.bot.send_message(chat_id=chat_id, text="Hola, selecciona una
opción:", reply_markup=reply_markup)
    return SELECCION_TIPO

def manejar_seleccion(update, context) -> Optional[int]:
    query = update.callback_query
    seleccion = query.data
    # Configura el chat id del chat que interactúa con el bot
    TelegramChat().id = query.message.chat_id

    if seleccion == "parámetros":
        try:
            mensaje = f"Temperatura: {sensorFormateado.temperatura}" \
                      f"\nHumedad: {sensorFormateado.humedadRelativa}"
        except SensorErrorLecturaException as e:
            mensaje = str(e)
```

```
        TelegramChat().enviarMensaje(mensaje)
        query.answer()
    else:
        tipo_alarma = seleccion.split('_')[1]  # Extraer tipo de alarma (cita,
medicación, hidratación)
        context.user_data['tipo_alarma'] = tipo_alarma
        query.answer()
        query.edit_message_text(text=f"Has seleccionado alarma de {tipo_alarma}.
Introduce la hora en formato hh:mm.")
        return INGRESO_HORA

# Manejar la entrada de hora de una alarma
def ingresar_hora(update, context) -> int:
    global alarmas

    # Configura el chat id del chat que interactua con el bot
    TelegramChat().id = update.message.chat_id

    try:
        BoolException(not Tiempo.validarFormato(update.message.text))
        hora = Tiempo.extraerHora(update.message.text)
        tipo = context.user_data['tipo_alarma']

        # Configura alarma
        alarmas.agregar(hora, f"¡Alarma! Es hora de {tipo}")

        # Informa de alarma configurada
        msg = f"Alarma de {tipo} configurada para a las {hora}."
        TelegramChat().enviarMensaje(msg)
        TareaTextoAAudio(msg).iniciar()
        logger.info(msg)

        return ConversationHandler.END
    # Ocurre algún error al programar la alarma
    except Exception as e:
        logger.error(e)
        err = "Por favor, introduce la hora en el formato correcto (hh:mm)."
        TelegramChat().enviarMensaje(err)
        return INGRESO_HORA

# Función para manejar la alarma estándar
def manejar_alarma(respuesta) -> None:
    global alarmas
    if "a las" in respuesta:
        TareaAgendarAlarmaEstandar(alarmas, respuesta).iniciar()
    else:
        logger.info(f"Keyword 'a las' no encontrado en respuesta:
'{respuesta}'")

# Función para escuchar comandos de voz de forma continua con tiempo de espera
de silencio
def escuchar_comandos():
    global alarmas, alerta
    while True:
```

```
        # Escucha de comandos se desactiva mientras existe una tarea que
        # vocifera audio en altavoz
        if alerta.activa or Audio.EN_REPRODUCCION:
            logger.info("Escucha de comandos desactivada...")
            time.sleep(.1)
            continue

        # Escuchar comando
        respuesta = Audio.escuchar()
        if not respuesta:
            logger.info("No hay comando...")
            continue

        logger.info(f"Comando recibido: {respuesta}")


        # Comando para poner una alarma estándar
        if "pon una alarma a las" in respuesta:
            manejar_alarma(respuesta)

        # Comando para desactivar la alarma mediante las palabras clave
        elif "desactiva alarma" in respuesta or "desactivar alarma" in
respuesta:
            logger.info("Desactivando alarma...")
            if alarmas.detener():
                logger.info("Alarma detenida...")
            else:
                logger.info("No habían alarmas para detener")

        # Comando para enviar un mensaje rápido
        elif "mensaje rápido" in respuesta:
            # Extraer el mensaje después de "mensaje rápido"
            TareaMensajeRapido(respuesta).iniciar()

        # Comando para decir la hora actual
        elif "qué hora es" in respuesta or "dime la hora" in respuesta:
            TareaReproducirHoraActual().iniciar()

        # Comando para activar alerta de ayuda
        elif "ayuda" in respuesta or "emergencia" in respuesta:
            if not alerta.activa:
                alerta.activar()
                alerta.iniciar()

            # Comando para manejar parámetros del sensor AHT10
        elif "parámetros" in respuesta or "condiciones" in respuesta or
"temperatura" in respuesta or "humedad" in respuesta:
            TareaAnunciarCondicionesAmbientales().iniciar()

        # Comando de voz no registrado
        else:
            logger.info("¡Comando de voz no registrado!")

def telegram_anunciar_condiciones(update) -> int:
```

```
    '''Anuncia las condiciones ambientales al chat de telegram que las
solicita.'''
    TelegramChat().id = update.message.chat_id
    logger.info("Anunciando condiciones ambientales vía telegram...")

    try:
        mensaje = f"Temperatura: {sensorFormateado.temperatura}" \
                f"\nHumedad: {sensorFormateado.humedadRelativa}"
    except SensorErrorLecturaException as e:
        mensaje = str(e)

    TelegramChat().enviarMensaje(mensaje)
    return ConversationHandler.END

# Función para recibir mensajes de texto a través de Telegram
def recibir_mensaje(update, context) -> None:
    global alerta
    chat_id = update.message.chat_id
    # Configura el chat id del chat que interactua con el bot
    TelegramChat().id = update.message.chat_id
    texto = update.message.text.lower()

    # Confirmación de conexión con el chat al obtener el chat_id por primera vez
    if 'chat_id' not in context.bot_data:
        context.bot_data['chat_id'] = chat_id
        logger.info("Conexión establecida con el chat.")

    # Desactivar alerta si está activa al recibir cualquier mensaje
    if alerta.activa:
        alerta.desactivar()
        return

    # Manejo específico de palabras clave del sensor
    if texto in ["parámetros", "condiciones", "temperatura", "humedad"]:
        telegram_anunciar_condiciones(update)
        return

    # Manejo general de otros mensajes
    TareaResponderMensaje(texto).iniciar()

# Configuración del manejador de conversación
def cancelar(update, context):
    update.message.reply_text('Configuración de alarma cancelada.')
    return ConversationHandler.END

# Función principal para configurar el bot y el micrófono
def main():
    updater = Updater(TOKEN, use_context=True)
    dp = updater.dispatcher

    TelegramChat().context = dp

    # Configuración del manejador de conversación para alarmas
    conv_handler = ConversationHandler(
```

```
        entry_points=[CommandHandler('Botones', mostrar_botones_alarma),
MessageHandler(Filters.regex(r'\bM\b'), mostrar_botones_alarma)],
        states={
            SELECCION_TIPO: [CallbackQueryHandler(manejar_seleccion)],
# Manejar todas las selecciones
            INGRESO_HORA: [MessageHandler(Filters.text & ~Filters.command,
ingresar_hora)],
        },
        fallbacks=[CommandHandler('cancel', cancelar)]
    )

    dp.add_handler(conv_handler)

    # Manejador para mensajes de texto
    dp.add_handler(MessageHandler(Filters.text & ~Filters.command,
recibir_mensaje))

    # Iniciar hilo del micrófono
    iniciar_hilo_microfono(dp)

    # Iniciar el bot
    updater.start_polling()
    updater.idle()

# # Punto de entrada principal
if __name__ == '__main__':
    main()
```

6.2. Códigos de C# en Unity

6.2.1 Código de C# para el movimiento del robot (MovementClient)

```
using System.Net.Sockets;
using System.Text;
using System;
using UnityEngine;
using System.Collections;

public class MovementClient : TonyData
{
    public int port = 8001; // Puerto para conectarse a la Raspberry Pi
    private TcpClient client; // Cliente TCP para conectarse
    private NetworkStream stream;

    public float delayCommand; // Tiempo de espera entre comandos
    public bool canSendCommand; // Controlar para enviar un comando (evita spam de
comandos)

    // Sobrescribe el método SetTonyData para configurar la IP, texto, y estado
    public override void SetTonyData(string newIP, UITextUpdate newTextUpdate, bool
isEnable)
    {
        base.SetTonyData(newIP, newTextUpdate, isEnable);

        textUpdate.UpdateText("Intenta conectar Movement");

        if (enableComponent)
        {
            string message = raspberryPiIP; // Guarda IP para mostrar en mensajes

            try
            {
                // Intenta conectarse a la Raspberry Pi usando la IP y el puerto
definido
                client = new TcpClient(raspberryPiIP, port);
                stream = client.GetStream();
            }
            catch (SocketException ex)
            {
                textUpdate.UpdateText($"{message} : Error de conexion: {ex.Message}");
            }
            catch (Exception ex)
            {
                textUpdate.UpdateText($"{message} : Error inesperado: {ex.Message}");
            }
        }
        else
        {
            textUpdate.UpdateText("VR Movement Activado");
        }
    }

    public void PublicCommand(string command)
    {
        SendCommand(command);
```

```
    }

    // Envía comandos a la Raspberry
    private void SendCommand(string command)
    {
        if (stream.CanWrite && enableComponent && canSendCommand)
        {
            canSendCommand = false; // Bloquea envíos hasta que pase el delay

            byte[] data = Encoding.UTF8.GetBytes(command + "\n");
            stream.Write(data, 0, data.Length);
            stream.Flush();

            Debug.Log($"Comando: {command}");

            StartCoroutine(DelayCommand()); // Inicia la espera para permitir el
siguiente comando
        }
    }

    // Corutina que activa nuevamente el envío después del tiempo definido
    IEnumerator DelayCommand()
    {
        yield return new WaitForSeconds(delayCommand);
        canSendCommand = true;
    }

    // Cierra la conexión al cerrar la aplicación
    private void OnApplicationQuit()
    {
        if (enableComponent)
        {
            stream.Close();
            client.Close();
        }
    }
}
```

6.2.2 Código de C# para conectar la cámara del robot (CamaraClient)

```
using UnityEngine;
using System.Net.Sockets;
using System.IO;
using System;
using System.Threading;

public class CameraClient : TonyData
{
    public int port = 8000;
    public SizeCamera sizeCamera;

    private TcpClient client;
    private NetworkStream stream;
    private MemoryStream imageStream;
    private Texture2D texture;

    private Thread cameraThread;
    private bool isRunning = false;
```

```
    private byte[] latestImageData;
    private bool newImageAvailable = false;
// Configurar datos y establecer conexión
    public override void SetTonyData(string newIP, UITextUpdate newTextUpdate, bool
isEnable)
    {
        base.SetTonyData(newIP, newTextUpdate, isEnable);

        textUpdate = newTextUpdate;
        raspberryPiIP = newIP;

        if (enableComponent)
        {
            try
            {
                // Establece conexión con la Raspberry Pi
                client = new TcpClient(raspberryPiIP, port);
                stream = client.GetStream();
                imageStream = new MemoryStream();

                // Crea la textura con el tamaño definido
                texture = new Texture2D(sizeCamera.widht, sizeCamera.height);

                // Inicia el hilo para recibir imágenes
                isRunning = true;
                cameraThread = new Thread(ReceiveCameraData);
                cameraThread.IsBackground = true;
                cameraThread.Start();

                textUpdate.UpdateText("Cámara conectada");
            }
            catch (Exception ex)
            {
                Debug.LogError("Error al conectar la cámara: " + ex.Message);
            }
        }
    }

    private void ReceiveCameraData()
    {
        while (isRunning)
        {
            try
            {
                // Tamaño de la imagen (4 bytes)
                byte[] sizeInfo = new byte[4];
                int bytesRead = stream.Read(sizeInfo, 0, 4);
                if (bytesRead == 0) continue;

                int imageSize = BitConverter.ToInt32(sizeInfo, 0);
                byte[] imageData = new byte[imageSize];
                int totalBytesRead = 0;

                while (totalBytesRead < imageSize)
                {
                    bytesRead = stream.Read(imageData, totalBytesRead, imageSize -
totalBytesRead);
                    if (bytesRead == 0) break;
                    totalBytesRead += bytesRead;
```

```
                }

                lock (this)
                {
                    latestImageData = imageData;
                    newImageAvailable = true;
                }
            }
            catch (Exception ex)
            {
                Debug.Log("Error recibiendo imagen: " + ex.Message);
                isRunning = false;
            }
        }
    }

    void Update()
    {
        if (enableComponent && newImageAvailable && latestImageData != null)
        {
            lock (this)
            {
                texture.LoadImage(latestImageData);
                GetComponent<Renderer>().material.mainTexture = texture;
                newImageAvailable = false;
            }
        }
    }

    private void OnApplicationQuit()
    {
        isRunning = false;
        cameraThread?.Join();

        if (enableComponent)
        {
            stream?.Close();
            client?.Close();
        }
    }
}

[Serializable]
public class SizeCamera
{
    public int widht = 320;
    public int height = 240;
}
```

6.2.3 Código de C# para los servomotores de la cámara del robot (Servoclient)

```
using System.Net.Sockets;
using System.Text;
using UnityEngine;

public class ServoClient : TonyData
{
```

```
    public int port = 8002; // Puerto TCP usado para comunicarse con el servidor de
servos
    private TcpClient client;
    private NetworkStream stream;

    // Configura la IP, texto de estado y conexión al servidor
    public override void SetTonyData(string newIP, UITextUpdate newTextUpdate, bool
isEnable)
    {
        base.SetTonyData(newIP, newTextUpdate, isEnable);

        textUpdate.UpdateText("Intenta conecatar servo");
        if (enableComponent)
        {
            // Conexión TCP con la Raspberry Pi en el puerto 8002
            client = new TcpClient(raspberryPiIP, port);
            stream = client.GetStream();
            textUpdate.UpdateText("Servo Conectado");
        }
        else
        {
            textUpdate.UpdateText("VR Activado: Servo");
        }
    }

    //Para enviar comandos desde otros códigos
    public void PublicCommand(string command) { SendServoCommand(command); }

    // Envía el comando al servidor si el stream está disponible
    private void SendServoCommand(string command)
    {
        if (stream.CanWrite && enableComponent)
        {
            byte[] data = Encoding.UTF8.GetBytes(command + "\n");
            stream.Write(data, 0, data.Length);
            stream.Flush(); // Asegura que los datos se envíen inmediatamente
        }
    }

    // Cierra la conexión correctamente al salir de la aplicación
    private void OnApplicationQuit()
    {
        if (enableComponent)
        {
            stream.Close();
            client.Close();
        }
    }
}
```

6.2.4 Código de C# para los servomotores de las manos del robot (ServoControl)

```
using System.Net.Sockets;
using System.Text;
using UnityEngine;
using System.Collections;
```

```
public class ServoControl : TonyData
{
    private TcpClient client;
    private NetworkStream stream;

    public bool enableMovement = false;   // Permite activar/desactivar el movimiento
automático

    [Space(5)]
    public MovementRobotArms[] movementRobotArms; // Mover los brazos en horizontal y
vertical
    public GripTesting gripTesting;               // Control de la mano
    public float delayTime = 0.25f;               // Tiempo entre movimientos

    public int port = 8003; // Puerto para enviar comandos de servo

    // Inicializa datos, conecta con la Raspberry Pi y comienza la rutina de movimiento
    public override void SetTonyData(string newIP, UITextUpdate newTextUpdate, bool
isEnable)
    {
        base.SetTonyData(newIP, newTextUpdate, isEnable);

        textUpdate.UpdateText("Intenta Conectar Servo Control");

        if (enableComponent)
        {
            client = new TcpClient(raspberryPiIP, port);
            stream = client.GetStream();
            StartMoveArms();
            textUpdate.UpdateText("Servo Control Conectado");
        }
        else
        {
            textUpdate.UpdateText("Vr Servo Control activado");
        }
    }

    // Inicia la rutina de movimiento automático de brazos
    public void StartMoveArms()
    {
        StartCoroutine(MoveArms());
    }

    // Cierra la conexión al cerrar la aplicación
    void OnApplicationQuit()
    {
        if (stream != null)
            stream.Close();
        if (client != null)
            client.Close();
    }

    // Envía un comando de movimiento de servo en el formato: ID,Ángulo,Tiempo
    public void MoverServo(int servoId, int angulo, int tiempo)
    {
        if (stream != null && client.Connected && enableComponent)
        {
            string command = $"{servoId},{angulo},{tiempo}";
            byte[] message = Encoding.ASCII.GetBytes(command);
```

```
            stream.Write(message, 0, message.Length);
        }
    }

    // Rutina que mueve los brazos horizontal y verticalmente con espera entre
movimientos
    IEnumerator MoveArms()
    {
        foreach (MovementRobotArms arms in movementRobotArms)
        {
            if (enableMovement)
            {
                arms.MoveHorizontal();
                yield return new WaitForSeconds(delayTime);
                arms.MoveVertical();
                yield return new WaitForSeconds(delayTime);
            }
        }

        // Control de manos al finalizar los brazos
        gripTesting.RightHand();
        yield return new WaitForSeconds(delayTime);
        gripTesting.LeftHand();
        yield return new WaitForSeconds(delayTime);

        // Repite continuamente el movimiento
        StartCoroutine(MoveArms());
    }

    // Cambia el estado de activación del movimiento automático
    public void SwitchEnableArms()
    {
        enableMovement = !enableMovement;
    }
}
```

6.2.5 Código de C# que permite actualizar la IP de todos los códigos anteriores (TonyManager)

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class TonyManager : TonyData
{
    public bool waiting = false; // Manejar estados de espera

    public List<TonyInfoData> tonyInfoData; // Lista de todos los módulos del robot

    public UITextUpdate onlyVrUpdate;

    private void Awake()
    {
        // Obtiene la IP guardada previamente
        raspberryPiIP = PlayerPrefs.GetString("IP", "");
```

```
        // Actualiza el texto en pantalla según si está activado el componente
principal
        onlyVrUpdate.UpdateText(enableComponent ? "Only Vr Activado" : "Only Vr
Desactivado");

        foreach (TonyInfoData currentTonyData in tonyInfoData)
        {
            currentTonyData.tonyData.SetTonyData(
                raspberryPiIP,                      // IP de la Raspberry Pi
                currentTonyData.tonyDisplayText,
                currentTonyData.isEnable
            );
        }
    }
}

[Serializable]
public class TonyInfoData
{
    public string nameComponent;
    public TonyData tonyData;
    public bool isEnable;
    public UITextUpdate tonyDisplayText;
}

public class TonyData : MonoBehaviour
{
    public string raspberryPiIP;
    public bool enableComponent = false;
    public UITextUpdate textUpdate;
    [HideInInspector] public bool isReady;

    // Permite configurar la IP, el texto de estado y si se habilita o no el componente
    public virtual void SetTonyData(string newIP, UITextUpdate newTextUpdate, bool
isEnable)
    {
        enableComponent = isEnable;
        raspberryPiIP = newIP;

        if (newTextUpdate != null)
        {
            textUpdate = newTextUpdate;
        }
    }
}
```

6.3. Códigos de Python para establecer conexión con unity

6.3.1 Código server_manager.py

(Código encargado de conectar todos los códigos)

```
import threading
from camera_server import CameraServer
from servo_controller_server import ServoControllerServer
from servo_control_server import ServoControlServer
```

```
# Encargado de ejecutar todos los códigos
class ServerManager:
    def __init__(self):
        self.camera_server = CameraServer()# Encargado de enviar imágenes
        self.servo_controller_server = ServoControllerServer()# Encargado de
mover los servo
        self.servo_control_server = ServoControlServer()# Encargado de los
movimientos
# Para que se ejecuten en paralelo los servidores
    def start_servers(self):
        camera_thread = threading.Thread(target=self.camera_server.start)
        servo_controller_thread =
threading.Thread(target=self.servo_controller_server.start)
        servo_control_thread =
threading.Thread(target=self.servo_control_server.start)

        camera_thread.start()
        servo_controller_thread.start()
        servo_control_thread.start()

        camera_thread.join()
        servo_controller_thread.join()
        servo_control_thread.join()

if __name__ == "__main__":
    manager = ServerManager()
    manager.start_servers()
```

6.3.2 Servo_controller_server.py

(Código encargado de mover la cámara y activar acciones)

```
import socket
import time
import sys
import importlib.util
import hiwonder.ActionGroupControl as AGC  # Control de acciones del robot

# Cargar módulos externos
lab_tool_path = '/home/pi/LAB_Tool/ServoCmd.py'
spec_lab_tool = importlib.util.spec_from_file_location("ServoCmd_LAB",
lab_tool_path)
ServoCmd_LAB = importlib.util.module_from_spec(spec_lab_tool)
sys.modules["ServoCmd_LAB"] = ServoCmd_LAB
spec_lab_tool.loader.exec_module(ServoCmd_LAB)

tony_pi_path = '/home/pi/TonyPi_PC_Software/ServoCmd.py'
spec_tony_pi = importlib.util.spec_from_file_location("ServoCmd_Tony",
tony_pi_path)
ServoCmd_Tony = importlib.util.module_from_spec(spec_tony_pi)
sys.modules["ServoCmd_Tony"] = ServoCmd_Tony
spec_tony_pi.loader.exec_module(ServoCmd_Tony)

# Configuración de IDs y tiempo de movimiento para servos cámara
```

```
servo_id_vertical = 1
servo_id_horizontal = 2
use_time = 500 # Tiempo en milisegundos para el movimiento del servo

angulo_vertical = 0
angulo_horizontal = 0

def mover_servomotor_vertical(incremento):
    global angulo_vertical
    angulo_vertical += incremento
    angulo_vertical = max(-45, min(45, angulo_vertical)) # Limite de angulo
    pwm_value = int(1500 + (angulo_vertical * 1000 / 90))
    ServoCmd_LAB.setServoPulse(servo_id_vertical, pwm_value, use_time)
    print(f"Servo vertical movido a {angulo_vertical} grados (PWM={pwm_value})")

def mover_servomotor_horizontal(incremento):
    global angulo_horizontal
    angulo_horizontal += incremento
    angulo_horizontal = max(-45, min(45, angulo_horizontal))
    pwm_value = int(1500 + (angulo_horizontal * 1000 / 90))
    ServoCmd_LAB.setServoPulse(servo_id_horizontal, pwm_value, use_time)
    print(f"Servo horizontal movido a {angulo_horizontal} grados
(PWM={pwm_value})")

class ServoControllerServer:
    def __init__(self, host='0.0.0.0', port=8002):
        self.host = host
        self.port = port

    def start(self):
        while True:
            try:
                print(f"Iniciando servidor combinado de servos y movimiento en
el puerto {self.port}...")
                server_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
                server_socket.bind((self.host, self.port))
                server_socket.listen(1)
                print(f"Esperando conexión...")

                connection, addr = server_socket.accept()
                print(f"Conectado desde {addr}")
# Comandos
                try:
                    while True:
                        data = connection.recv(1024).decode('utf-8').strip()
                        if not data:
                            break

                        if data.strip() == 'W':
                            mover_servomotor_vertical(10)
                        elif data.strip() == 'S':
                            mover_servomotor_vertical(-10)
                        elif data.strip() == 'D':
```

```
                        mover_servomotor_horizontal(-10)
                    elif data.strip() == 'A':
                        mover_servomotor_horizontal(10)
                    else:
                        try:
                            AGC.runActionGroup(data, 1)
                        except Exception as e:
                            print(f"Error ejecutando acción: {e}")

                    time.sleep(0.1)

            except (ConnectionResetError, BrokenPipeError) as e:
                print(f"Conexión perdida: {e}")
            finally:
                connection.close()

        except Exception as e:
            print(f"Error en el servidor combinado: {e}")
        finally:
            if 'server_socket' in locals():
                server_socket.close()
            print("Reiniciando servidor en 5 segundos...")
            time.sleep(5)
```

6.3.3 Camera_server.py

(Código encargado de conectar con la cámara)

```
import socket
import struct
import time
import cv2
import threading

class CameraServer:
    def __init__(self, host='0.0.0.0', port=8000):
        self.host = host
        self.port = port # Puerto TCP

    def start(self):
        while True:
            try:
                print("Iniciando servidor de cámara...")
# Inicia la captura de la camara
                camera = cv2.VideoCapture(0)
                camera.set(cv2.CAP_PROP_FRAME_WIDTH, 320)  # Reducir resolución
                camera.set(cv2.CAP_PROP_FRAME_HEIGHT, 240)
                camera.set(cv2.CAP_PROP_FPS, 15)  # Reducir FPS
# Si no detecta la cámara espera y reintenta
                if not camera.isOpened():
                    print("Error: No se pudo acceder a la cámara.")
                    time.sleep(5)
```

```
                    continue

                server_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
                server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR,
1)
                server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_RCVBUF,
1024 * 64)  # Aumentar buffer de recepción
                server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_SNDBUF,
1024 * 64)  # Aumentar buffer de envío
                server_socket.bind((self.host, self.port))
                server_socket.listen(1)
                print("Esperando conexión de la cámara...")

                connection, client_address = server_socket.accept()
                print("Conectado a la cámara desde:", client_address)
# Función para capturar y enviar imágenes por TCP
                def capture_video():
                    while True:
                        ret, frame = camera.read()
                        if not ret:
                            print("Error al capturar la imagen")
                            break

                        encode_param = [int(cv2.IMWRITE_JPEG_QUALITY), 50]  #
Reducir calidad de compresión
                        result, image = cv2.imencode('.jpg', frame,
encode_param)
                        image_data = image.tobytes()
                        image_size = len(image_data)

                        try:
                            connection.sendall(struct.pack('<I', image_size))
                            connection.sendall(image_data)
                        except BrokenPipeError:
                            print("Conexión de cámara perdida: Broken pipe.
Cliente desconectado.")
                            break

                        time.sleep(0.1)

                capture_thread = threading.Thread(target=capture_video)
                capture_thread.start()
                capture_thread.join()
# Manejo de errores y reinicio automático
            except Exception as e:
                print(f"Error en el servidor de cámara: {e}")
            finally:
                if 'camera' in locals() and camera.isOpened():
                    camera.release()
                if 'server_socket' in locals():
                    server_socket.close()
                print("Reiniciando servidor de cámara en 5 segundos...")
                time.sleep(5)
```

6.3.4 Servo_control_server.py

(Código encargado del movimiento de los brazos)

```
import socket
import time
import sys
import importlib.util

# Cargar módulo desde TonyPi_PC_Software
tony_pi_path = '/home/pi/TonyPi_PC_Software/ServoCmd.py'
spec_tony_pi = importlib.util.spec_from_file_location("ServoCmd_Tony",
tony_pi_path)
ServoCmd_Tony = importlib.util.module_from_spec(spec_tony_pi)
sys.modules["ServoCmd_Tony"] = ServoCmd_Tony
spec_tony_pi.loader.exec_module(ServoCmd_Tony)

def mover_servo(servo_id, angulo, tiempo):
    pwm_value = int(1500 + (angulo * 1000/90))
    ServoCmd_Tony.setServoPulse(servo_id, pwm_value, tiempo)

class ServoControlServer:
    def __init__(self, host='0.0.0.0', port=8003):
        self.host = host
        self.port = port

    def start(self):
        while True:
            try:
                print(f"Iniciando servidor de control específico de servos en el
puerto {self.port}...")
                server_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
                server_socket.bind((self.host, self.port))
                server_socket.listen(1)
                print(f"Esperando conexión para control específico de
servos...")
# Conexión con unity

                connection, addr = server_socket.accept()
                print(f"Conectado desde {addr}")
# Recibir y procesar comandos
                try:
                    while True:
                        data = connection.recv(1024).decode('utf-8').strip()
                        if not data:
                            break

                        try:
                            servo_id, angulo, tiempo = map(int, data.split(','))
                            mover_servo(servo_id, angulo, tiempo)
                        except ValueError:
```

```
                            print("Error en el formato del comando. Formato
esperado: id,angulo,tiempo")

                        time.sleep(0.1)

                except (ConnectionResetError, BrokenPipeError) as e:
                    print(f"Conexión de servos perdida: {e}")
                finally:
                    connection.close()
# Reinicio
            except Exception as e:
                print(f"Error en el servidor de control específico de servos:
{e}")
            finally:
                if 'server_socket' in locals():
                    server_socket.close()
                print("Reiniciando servidor de control específico de servos en 5
segundos...")
                time.sleep(5)
```