industriales
etsii

Escuela Técnica Superior de Ingeniería Industrial

UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Escuela Técnica Superior de Ingeniería Industrial

Validación de una plataforma en la Nube con integración de robot móvil para asistir a personas mayores en residencias.

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

Autor: **Gabriel García López**
Director: Dra. Nieves Pavón Pulido
Codirector: Dr. Juan Antonio López Riquelme

Cartagena, a 9 de septiembre de 2022

Universidad Politécnica de Cartagena

AGRADECIMIENTOS

Quiero agradecer a mi familia, en especial a mis padres y mi hermano por siempre apoyarme día a día en todos los aspectos de la vida por muchos errores que haya podido cometer.

También quiero dar las gracias a mis amigos y en especial a mis compañeros Marta, Rodrigo y Sergio, quienes me han ayudado y apoyado siempre que lo he necesitado.

Por último, quisiera dedicar este Trabajo Fin de Grado a mi abuelo Norberto, a mi abuelo Cristóbal, a mi abuela Joaquina y a mi tía María Luisa, que en paz descansen, los cuales me hubiera gustado que hubiesen podido ver el resultado de este proyecto con sus propios ojos, sobre todo mi abuela Joaquina, quien vivió en una residencia de mayores en la última etapa de su vida.

RESUMEN

Este Trabajo Fin de Grado consiste en el diseño, desarrollo y puesta en marcha de un sistema asistencial, basado en el uso de un robot móvil junto con el apoyo de una plataforma en la nube, mediante el cual se le proporcione una ayuda a los profesionales que trabajan en residencias de mayores o en hospitales a la hora de desempeñar sus tareas.

Principalmente los objetivos de este proyecto abarcan actividades como la detección y acompañamiento de una persona, almacenamiento y extracción de información de la base de datos establecida en la nube, así como de la esquiva de diferentes obstáculos que puedan suponer un riesgo para la integridad del robot y de las personas que le rodean.

Proyectos como este son ahora más necesarios que nunca para paliar la gran cantidad de problemas que el paso de la pandemia del COVID19 dejó tras de sí en el ámbito residencial y sanitario, siendo esta también responsable de agravar muchas de las dificultades que ya padecían estos sectores imprescindibles para una sociedad actual cada vez más envejecida. Por desgracia, no hay demasiadas entidades dispuestas a invertir en proyectos como estos, pues muchas veces quedan estancados por falta de personas cualificadas para poder llevarlos a cabo o por el simple hecho del elevado coste económico que suponen, sumado a tiempos de desarrollo relativamente largos. Todo esto se debe a que diseñar un sistema basado en robótica no es algo trivial y mucho menos ponerlo en marcha en un entorno real de forma precisa, siendo esta tarea sumamente complicada. Aunque existen sistemas robóticos, con capacidades similares a las desarrolladas en este Trabajo Fin de Grado, destinados al uso en residencias de mayores u hospitales, ninguno ha conseguido implantarse de forma definitiva en un entorno real, denotando una gran demanda de investigación e investigadores en este ámbito.

Para la creación de este Sistema Robótico Asistencial se han empleado herramientas mundialmente reconocidas en el ámbito de la robótica y programación en la nube, siendo estas ROS (Robot Operating System), MediaPipe y Google Cloud junto con Eclipse. Todas ellas con la capacidad de poder habilitar la incorporación de nuevas mejoras tanto software como hardware en este sistema de cara a futuro.

Por último, cabe remarcar el correcto funcionamiento y el cumplimiento de todas funcionalidades recabadas en los objetivos propuestos a la hora de diseñar este sistema robótico, quedando esto constatado en los ensayos y pruebas realizados.

Contenido

Figuras

Tablas

1 CAPITULO 1

1.1 Introducción

La Robótica de Servicios está cada vez más integrada en nuestro día a día, desde las aspiradoras robóticas como el ampliamente conocido modelo Roomba, hasta otros capaces de interactuar con las personas, uno de los más recientes en este ámbito es Astro, creado por Amazon. Dentro de este gran marco encontramos la Robótica Asistencial, dedicada a diseñar y crear soluciones basadas en robots para ayudar, e incluso mejorar, la vida cotidiana de personas que se encuentren en residencias, hospitales o cualquier tipo de institución de esta índole. Este Trabajo de Fin de Grado (TFG), se centrará en este último punto, ya que como se ha visto durante la pandemia de COVID19, se requieren muchos más medios, a parte de los que ya existentes, para tratar a personas en situaciones de alerta sanitaria, por lo tanto, el desarrollo de proyectos que integren la Robótica para apoyar a este sector ayudará a que sea más robusto y fiable ante cualquier tipo de situación inesperada, además de mejorarlo notoriamente de cara a su funcionamiento diario.

1.2 Antecedentes

La pandemia de COVID19 ha dejado en evidencia las carencias tanto de nuestro sistema sanitario como de nuestra red de residencias, pero esto no es un caso aislado, sino que se ha podido apreciar en prácticamente todos los países afectados (1). Es por esto por lo que se necesita más que nunca invertir en mejorar los servicios tanto en residencias como en el sistema sanitario, siendo la Robótica Asistencial una de las posibles mejoras que se han de normalizar en estos sectores, debido a que ésta podría ayudar en uno de los problemas más recurrentes y comunes en dichos sectores, la falta de personal. Este problema es uno de los más importantes, ya que es el principal causante de una peor calidad en la asistencia realizada, derivando en situaciones donde el paciente puede llegar a quedar desatendido, tal y como se ha podido denunciar en multitud de residencias durante la pandemia.

La Robótica Asistencial no solventaría estos problemas de raíz, al menos de momento, pero ayudaría a quitar cargas de trabajo innecesarias gracias a robots que sean capaces de realizar

esas actividades, mejorando las condiciones laborales de los profesionales de estos ámbitos, repercutiendo positivamente en las personas que son atendidas por estos.

Otro problema que está muy presente en una sociedad cada vez más envejecida, y al cual no se le da la atención necesaria, es la soledad a la que se enfrentan multitud de personas mayores tanto en residencias, como en sus propias casas, pudiendo desembocar en graves alteraciones del ánimo y la salud emocional, exponiéndolas a trastornos incapacitantes como la depresión o la ansiedad, suponiendo un grave problema para su salud. Aunque ya existen entidades que se encargan de visitarlos, siguen sin tener medios suficientes para abarcar a todas las personas que lo necesitan. En este contexto, la Robótica Asistencial puede ser de gran ayuda, pues cada vez están surgiendo más y más robots con capacidades afectivas, los cuales puedan paliar parcialmente dicha soledad.

En cuanto al problema que surge a la hora de poder asistir a una persona mayor que se pueda encontrar sola en su casa o que tenga alguna patología que le impida comunicarse con normalidad, existen los sistemas de teleasistencia, capaces de ponerse en contacto con el usuario cuando este lo requiera para poder actuar en consecuencia. En este punto la Robótica Asistencial tiene un papel importante, pues mejora la calidad de este servicio al poder contar con un mayor abanico de posibilidades.

Por otro lado, se encuentran las personas con algún tipo de discapacidad, siendo alrededor del 15% de la población mundial según la Organización Mundial de la Salud (2), las cuales, debido a su situación, tienen barreras a la hora de interactuar con su entorno o cuando intentan acceder a algún tipo de servicio. Es por esto por lo que la Robótica Asistencial es una herramienta muy valiosa para estas personas, ya que les ayudaría a mejorar su calidad de vida.

Una de las mayores dificultades a la hora de implementar un sistema asistencial basado en un robot es la interacción que este tenga con su entorno, debido a que, por norma general, tanto las residencias, como los centros hospitalarios son lugares muy concurridos donde la variable ambiental no es constante, dado que en esos entornos hay un gran trasiego de personas, pudiendo estas modificar el entorno de trabajo del robot, haciendo que éste tenga que adaptarse ante posibles escenarios no previstos de una forma segura, eficiente y robusta.

En relación con soluciones ya existentes a la hora de desarrollar un proyecto basado en Robótica Asistencial, nos encontramos con un amplio abanico de posibilidades. Uno de los entornos más utilizados para llevar a cabo estos proyectos es sin lugar a duda ROS (Robot Operating System) (3). Este software libre y gratuito que contiene tanto las herramientas como las interfaces y componentes necesarios para facilitar la programación, validación y despliegue de software para robots, siendo capaz de unificar y comunicar entre si los principales elementos hardware

que lo componen; entre los cuales encontramos sensores, actuadores y sistemas de control. Esta herramienta permite realizar estas comunicaciones gracias a su sistema de nodos, pudiendo estos comunicarse entre sí gracias a mensajes contenidos en puertos o Topics, donde los diferentes nodos pueden publicar información y a los cuales pueden suscribirse para obtener datos compartidos por otros nodos. Otra de sus ventajas es su comunidad, ya que, al ser un entorno de software libre, permite que todos los usuarios que lo utilizan compartan entre sí sus proyectos e ideas, mejorando así de forma más rápida este ecosistema de desarrollo de software para robots.

Otra herramienta muy útil es MediaPipe (4), un software libre y gratuito al igual que ROS, el cual permite crear soluciones de Machine Learning (ML), para trabajar con vídeos e imágenes en tiempo real, con el fin de detectar objetos y elementos con contenido semántico (detección de la cara, skeleton tracking o estimación de la pose de objetos y personas, entre otros). Este software es muy útil, debido a que permite mediante una cámara RGB convencional reconocer y diferenciar personas de objetos, gracias a su sistema de Landmarks o puntos de referencia. Entre sus soluciones encontramos algunas capaces de seguir y reconocer el movimiento de nuestras manos, cara o directamente el cuerpo entero, siendo esto muy útil a la hora de querer implementar reconocimiento de personas en un robot. Este programa tiene también la ventaja de que se puede interconectar con ROS, pudiendo desarrollar proyectos de forma más eficiente.

1.3 Esquema de capítulos

- **Capítulo 1: Introducción**

  En este capítulo se detallan todos los antecedentes al desarrollo de este proyecto y que desembocan en la creación de este, así como de la motivación y objetivos a la hora de su diseño y desarrollo.

- **Capítulo 2: Estado del arte**

  En este apartado se documentan y describen qué sistemas robóticos residenciales o relacionados con la teleasistencia a personas mayores existen actualmente, detallando sus funcionalidades.

- **Capítulo 3: Diseño del sistema**

En esta parte de la documentación se abordan con más detenimiento los objetivos finales de este proyecto explicando, a su vez, la composición y diseño de la arquitectura hardware y de la arquitectura software del sistema robótico desarrollado.

- **Capítulo 4: Pruebas y resultados**

Aquí se describen todos los ensayos realizados para verificar el correcto funcionamiento de todos los componentes que conforman el robot, como la parte relacionada con la Nube, detallando posteriormente los resultados obtenidos en dichas pruebas.

- **Capítulo 5: Trabajo futuro y conclusiones**

En este apartado se muestran posibles avances y mejoras que podrían implementarse en futuros proyectos relacionados con este sistema robótico. Además, se incluye un resumen de las conclusiones obtenidas tras la finalización de este proyecto.

1.4 Motivación y objetivos

Una de las principales motivaciones para llevar a cabo este proyecto ha sido la situación actual generada por la pandemia del COVID19 porque, como se ha comentado anteriormente, ha arrojado luz sobre los problemas a los que se enfrentan los sectores sanitarios y residenciales, viendo como la Robótica Asistencial podría paliar sus efectos o incluso solucionarlos.

Se ha visto como la principal complicación a la que se enfrentan los profesionales de dichos ámbitos es la falta de personal, produciéndose una desatención en los pacientes y residentes de los cuales están a cargo. Este problema se podría solucionar gracias a la Robótica Asistencial, entrando de lleno en el principal objetivo de este proyecto, la implantación de un sistema de navegación, el cual sea capaz de hacer que un robot pueda reconocer y seguir a una persona, para que acompañe por ejemplo a un sanitario a visitar a los pacientes y de esta manera el robot podría cargar con todo lo que dicho trabajador necesitase.

Otro objetivo sería que el robot pudiese ir a un punto determinado sin ayuda de nadie gracias a un paquete de navegación, para que una vez en dicho punto, si es la habitación de un paciente, que reconozca a la persona para poder acercarse a ella y una vez lo suficientemente cerca, este pueda interactuar mediante una pantalla táctil con ella a través de un juego cognitivo o haciéndole preguntas sencillas, tales como: ¿Cómo te encuentras hoy? o ¿A qué juego quieres jugar hoy?

También se integrará un servicio en la Nube capaz de almacenar datos tanto de los profesionales sanitarios, como de los pacientes, para que luego esos datos puedan ser utilizados por el mismo robot.

Todo esto enmarcado en una sociedad cada vez más envejecida, lo cual hará que estos servicios sean cada vez más demandados.

2 CAPITULO 2

2.1 Estado del arte

2.1.1 Sistemas de Teleasistencia a mayores

Los sistemas de Teleasistencia son un servicio que proporcionan una atención las 24 horas al día a personas mayores o con algún tipo de discapacidad a través de un grupo de profesionales especializados, para que puedan ser atendidos a la mayor brevedad posible ante cualquier tipo de crisis, problema o, simplemente, para poder ponerse en contacto con sus familiares o amigos. El funcionamiento de estos sistemas puede variar dependiendo de los dispositivos empleados para que el usuario se ponga en contacto con el operador especialista, de los servicios que se vayan a proporcionar o de la respuesta que se genere (5).

Según los dispositivos utilizados podemos encontrar desde un simple botón que el usuario accionaría cuando necesitara algún tipo de asistencia, hasta sensores capaces de anticiparse a este hecho para poder intervenir de forma más rápida, sobre todo ante problemas relacionados con algún tipo de enfermedad. Dentro de este grupo disponemos de tres tipos de sistemas, los cuales son los activos, semiactivos y pasivos, los primeros son aquellos que funcionan mediante el accionamiento de un dispositivo, normalmente un botón, por parte del usuario.

En cuanto a los sistemas semiactivos, son aquellos que la central de la entidad que proporciona estos servicios se pone en contacto con el usuario a ciertas horas preestablecidas, ya sea para recordatorio de ciertas actividades o pautas de medicación o comprobación de su estado, entre otros.

También existen sistemas pasivos, capaces de alertar cuando no se ha llevado a cabo una acción por el usuario, ya sea, por ejemplo, levantarse de la cama o comprobar la ausencia de otras pautas típicas. Estos sistemas trabajan con multitud de sensores capaces de detectar cuando no se han realizado las actividades que han sido previamente registradas.

En relación con el tipo de servicio a realizar, hay una gran disparidad de casos, dividiéndose principalmente en: recordatorios; avisos no urgentes, los cuales no necesiten de la intervención de los servicios pertinentes de forma inmediata; avisos urgentes o críticos, estos sí necesitando la intervención inmediata de los servicios solicitados, como por ejemplo caídas, mareos o

infartos, entre otras eventualidades; resolución de alguna duda por el usuario o requerimiento de servicio de mantenimiento de los dispositivos del sistema de Teleasistencia.

Por último, nos encontramos con el tipo de respuesta que se genera, pudiéndose clasificar en dos tipos:

- Necesidad de desplazarse hacia la ubicación del cliente para solventar la situación o problema pertinente, en este caso podrán asistir los operarios especializados de la entidad encargada del sistema de teleasistencia o en caso de una emergencia crítica o sanitaria, estos se pondrán en contacto con las autoridades pertinentes para que estas acudan en su lugar.
- Sin necesidad de desplazamiento, en este caso se engloban todo lo relacionado con recordatorios o resolución de cualquier duda.

2.1.2 Sistemas de Asistencia a mayores mediante el uso de robots

La Robótica Asistencial o los sistemas de Asistencia a mayores mediante el uso de robots, se pueden definir como la combinación de características asistenciales, interactivas y afectivas integradas en un robot con el fin de comunicarse con una persona de la forma más "humana" posible. Es por eso por lo que este tipo de robots deben trabajar de forma autónoma, procurando que la acción ejercida por un operador humano sea la mínima, así como también deben ser capaces de poder actuar de forma intuitiva y rápida ante situaciones imprevistas, ya que estarán funcionando en entornos con una gran afluencia de personas.

Otro factor que se ha de tener muy en cuenta a la hora de integrar un robot en un entorno asistencial ya sea social, residencial u hospitalaria, es la interacción de este con las personas, pues éste será su primer cometido. Para ello, se han desarrollado diferentes tipos de metodologías o estudios para poder llevar a cabo dicha tarea (6), siendo uno de los más comunes el estudio etnográfico, consistiendo en el despliegue del robot en entornos reales desde fases tempranas de su desarrollo, poniendo a prueba sus aptitudes. Este estudio es muy útil pues permite corregir o pulir diferentes errores o fallos del prototipo en cuestión. Luego tendríamos el diseño o estudio centrado en el usuario, el cual es clave a la hora de llevar a cabo un proyecto de la envergadura de un sistema robótico, siendo su objetivo el entender e incorporar las diferentes necesidades y soluciones que el usuario final del producto requiere, como, por ejemplo: la comunicación del robot con una persona, la detección de la caída de una persona o el seguimiento del usuario. Por otra parte, nos encontraríamos el Diseño Participativo, basado

en el intercambio de ideas entre diseñadores y usuarios finales a la hora de realizar el proyecto, pero este método es más difícil de ejecutar, debido a que no siempre se va a poder establecer una comunicación fluida entre ambas partes.

Dentro de este amplio campo de estudio podemos encontrar muchos ejemplos de robots cuya finalidad es la asistencia a personas mayores o con alguna discapacidad, pudiendo destacar entre los siguientes:

2.1.2.1 PHAROS (PHysical Assistant RObot System)

PHAROS (7) es un robot destinado a ayudar e interactuar con las personas mayores a realizar diferentes ejercicios y actividades físicas en sus casas. El diseño hardware de este robot es el denominado Pepper (Figura 1), ya que PHAROS es la evolución de este modelo de robot, el cual es un robot con forma humanoide que incorpora una pantalla táctil para que la persona pueda interactuar con este de forma sencilla.

Figura 1. Robot Pepper.

Su arquitectura software se divide en dos partes, una destinada a recomendar ciertas actividades (Recomendador), adecuadas a la capacidad física del usuario que vaya a realizarlas,

a ciertas horas al día, previamente planificadas, mientras que el otro módulo software se encarga de reconocer a la persona, así como su posición y movimientos, para verificar si está realizando dichos ejercicios de forma correcta.

El módulo Recomendador se divide en 4 submódulos, tal y como se puede ver en la Figura 2:

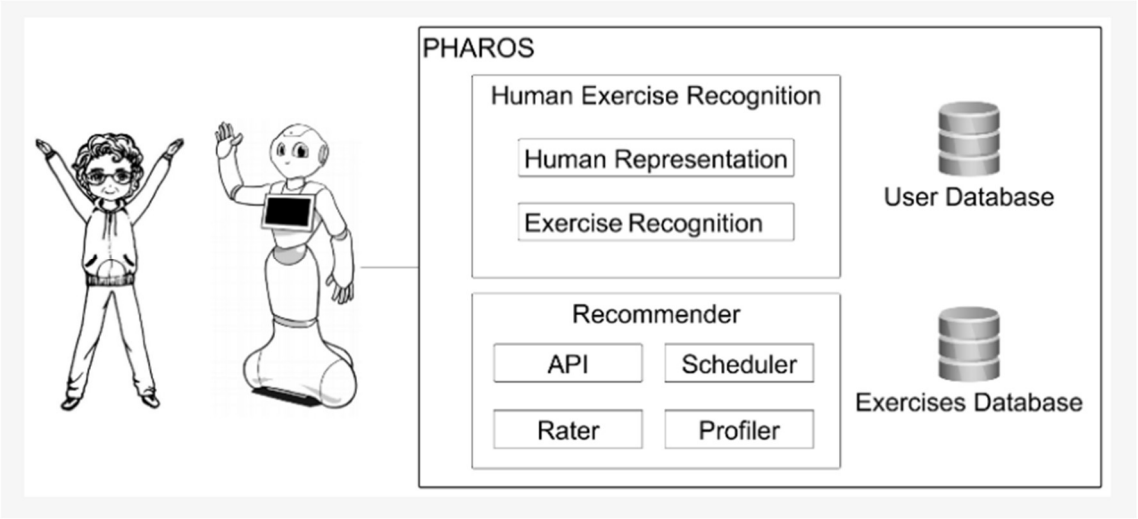


Figura 2. Esquema general de PHAROS.

El módulo API es, simplemente, una aplicación Android, mediante la cual, el usuario se puede registrar a través de la pantalla táctil del robot para que éste almacene su información en su base de datos y pueda recomendar ejercicios personalizados a cada usuario que este registrado.

El módulo Profiler o Perfilador se encarga de ver si hay algún ejercicio recomendado para el usuario registrado, gracias a un proceso de valoración que sugiere qué ejercicios debería realizar la persona mayor.

El módulo Scheduler o Programador se encarga de planificar cuántas veces al día y a qué horas deberá realizar los ejercicios recomendados por el módulo anterior.

Por último, el módulo más importante, Rater o Valorador, es el encargado de comparar cómo de bien ha realizado los ejercicios el usuario para mediante un proceso de valoración, mencionado anteriormente, comprobar si esas actividades son mejores o peores que otras para tal usuario. Esto lo consigue mediante el algoritmo de valoración Glicko2 (8), usado comúnmente en campeonatos de ajedrez para valorar los movimientos de ambos jugadores. Este algoritmo destina a cada jugador, o en este caso la persona mayor que realiza el ejercicio, un valor r, un valor de desviación RD y una volatilidad σ. Esta última mide o indica el nivel de fluctuación esperada en el valor de un jugador, siendo esta elevada cuando este hace jugadas

erráticas, es decir sin sentido o erróneas, mientras que es baja cuando el jugador hace movimientos correctos. Por lo tanto, gracias a este algoritmo el robot puede saber gracias a la cámara y al algoritmo que permite reconocer a la persona, si las actividades las está realizando de forma correcta o no. Esto es algo crucial, debido a que si no los ejecuta de forma correcta puede revelar posibles lesiones o problemas musculares u óseos de la persona, haciendo que ese ejercicio no sea recomendado para ella.

En cuanto al medio usado en este robot para reconocer a la persona, así como sus movimientos, se ha usado OpenPose (9). Este es un sistema que es capaz de estimar en tiempo real puntos clave tanto del cuerpo, cara, manos y pies de una o varias personas a la vez gracias a una cámara RGB. En este proyecto se ha usado la Red Neuronal Convolucional capaz de detectar 18 puntos de referencia en un cuerpo, pudiendo de esta manera el robot detectar e interpretar los movimientos realizados por la persona mayor. Véase el funcionamiento de este software en la Figura 3:

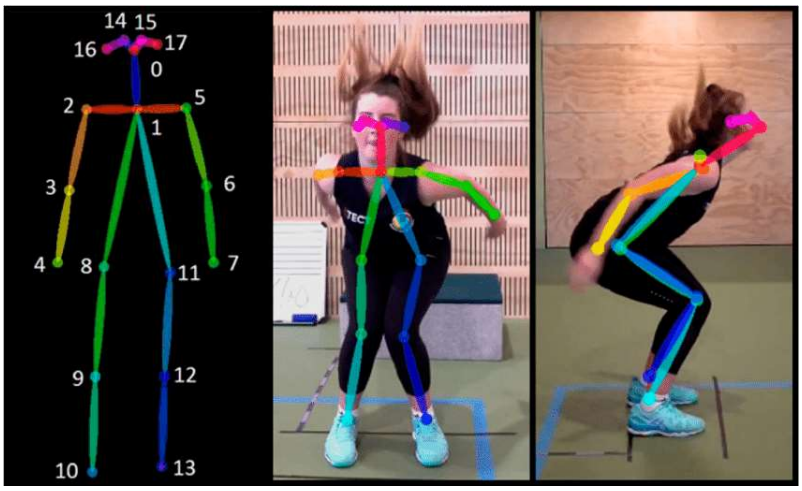


Figura 3. Detección cuerpo humano por OpenPose.

2.1.2.2 Hobbit PT2 platform

El sistema Hobbit PT2 platform, es una versión mejorada del Hobbit (10). Este robot utiliza componentes de bajo coste para que sea lo más asequible posible, ya que su objetivo es que sea instalado en el mismo hogar del usuario, haciendo que de esta manera este no necesite recurrir a ser ingresado en una residencia.

Sus características principales son las siguientes:

- **Detección de emergencia**

Esta función se puede dividir en dos subfunciones, una mediante la cual el robot es capaz de patrullar o vigilar la vivienda donde se encuentra, para que, si en un intervalo de 3 horas no ha detectado ningún signo de actividad por parte del usuario, comprobará si este se encuentra bien o si tiene algún problema, en caso de que así sea, el robot será capaz de tomar las medidas pertinentes. La otra subfunción sería la detección de caída de una persona (Figura 4), dividiéndose su funcionamiento en tres casos: detección de que la persona se está cayendo, esto lo realiza mediane el reconocimiento de diferentes puntos de referencia, así como sus velocidades y aceleraciones, reconocidos por una interfaz como la mencionada anteriormente; luego tendría el modo de identificar que la persona se encuentra en el suelo y que no se puede levantar, consiguiéndolo gracias a dos cámaras RGB, específicamente Asus Xtion Pro RGB-D, capaces de medir la profundidad, una situada en la cabeza del robot y otra en el cuerpo de este, haciendo que mediante comparación de los datos de ambas imágenes sepa cuando se encuentra una persona en el suelo. Por último, el reconocimiento de una señal de emergencia previamente registrada, realizada mediante el gesto con una o ambas manos, esto gracias al mismo motor software que reconoce cuando una persona se está cayendo. Cabe destacar que estos tres modos están siempre ejecutándose en segundo plano dentro del sistema, debido a que son cruciales para la seguridad e integridad del usuario de este tipo de robot.

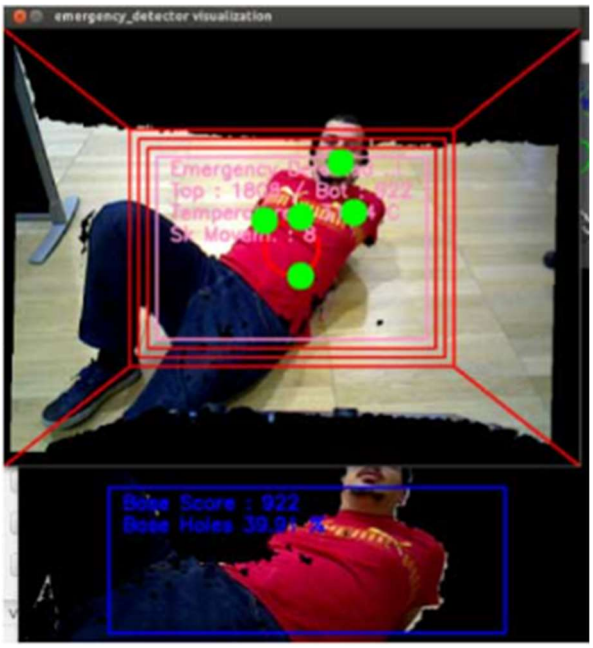


Figura 4. Detección de caída realizada por Hobbit PT2 platform.

- **Manejo de emergencia o emergency handling**

  Mediante esta función el robot es capaz de llamar tanto a familiares o a las autoridades pertinentes en caso de emergencia.

- **Prevención de caídas**

  Esta es una de sus características más importantes, pues se divide en dos modos, uno que permite al robot identificar objetos en el suelo, los cuales son inalcanzables para el usuario, pues este podría caerse al intentar alcanzarlo, y ser capaz de recogerlos y dárselos a la persona pertinente. Por otro lado, tendría la función de recomendar diferentes ejercicios físicos al usuario para mejorar su coordinación y estado físico en general. Además, también dispondría de un sistema de verificación encargado de valorar si el entorno de una sala es seguro para que el usuario pueda acceder a ella, ya sea porque hay agua en el suelo de dicha habitación o hay muchos obstáculos por medio, entre otras posibles circunstancias.

Por último, el robot cuenta con una interfaz capaz de reproducir diferentes expresiones para transmitir diferentes emociones (Figura 5), haciendo que su apariencia sea más afable. Este punto es muy importante, sobre todo en la robótica asistencial, pues este tipo de sistemas están pensados principalmente para relacionarse con persona mayores, las cuales puede que no estén acostumbrados a tratar con este tipo de dispositivos, así pues, la apariencia de estos es un factor crítico.

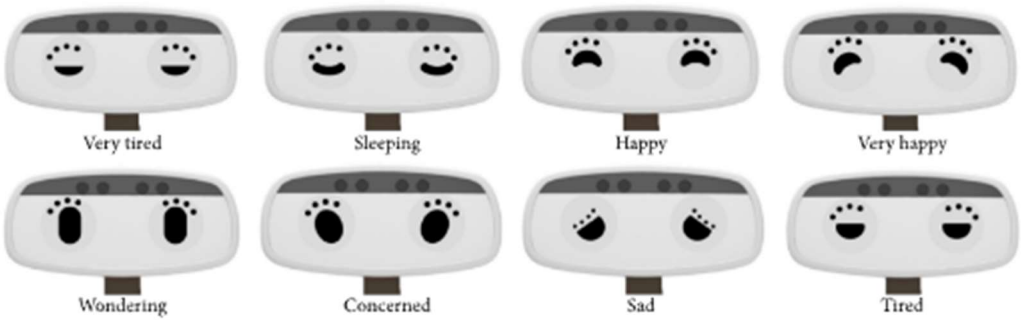


Figura 5. Interfaz de reproducción de expresiones faciales de Hobbit PT2 platform.

Este robot dispone de diferentes elementos hardware que le permiten realizar las funciones previamente mencionadas, siendo dichos elementos los detallados en la Figura 6:

Figura 6. Elementos hardware de Hobbit PT2 platform.

2.1.2.3 RAMCIP Robot

Este robot ha sido diseñado bajo el ámbito del proyecto europeo "Robotic Assistant for MCI Patients at Home" (11), estando principalmente enfocado a tratar a personas en fases iniciales de demencia y Alzheimer.

RAMCIP (12), al igual que los anteriores basa su funcionamiento en diferentes funciones, siendo estas las siguientes:

El robot está diseñado, al igual que el sistema anterior, para poder detectar si se ha caído una persona y actuar en consecuencia, ayudándole a levantarse o también a llamar a una segunda persona para que le asista al usuario si el robot es incapaz de realizar dicha actividad.

Además, dispone de la capacidad, por medio de una serie de sensores, de detectar ambientes con poca iluminación para avisar al usuario de que sería conveniente encender la luz para prevenir un posible tropiezo o caída, pero si es necesario el robot es capaz de encender la luz por sí mismo mediante un sistema de manipulación de objetos integrado en este. A su vez, mientras una persona cocina, puede detectar si se ha caído algún objeto para notificárselo a dicha persona o si el objeto está a su alcance, recogerlo directamente detectar si una persona se ha dejado un fogón encendido tras cocinar, avisándole y si fuera necesario actuar en consecuencia y apagar dicho fogón. También es capaz de verificar si la puerta del frigorífico se ha quedado abierta, cerrándola si el usuario no lo hace.

Otra de las funcionalidades más relevantes de las que este robot puede prestar, es la detección del estado de ánimo del usuario, para que en caso de que detecte que este se encuentra decaído o desanimado, actúe en consecuencia mediante la sugerencia de realizar una videollamada con algún familiar o amigo. Siendo esto posible gracias a la Tablet que integra.

Los componentes que conforman el hardware de este sistema se detallan en la Figura 7:

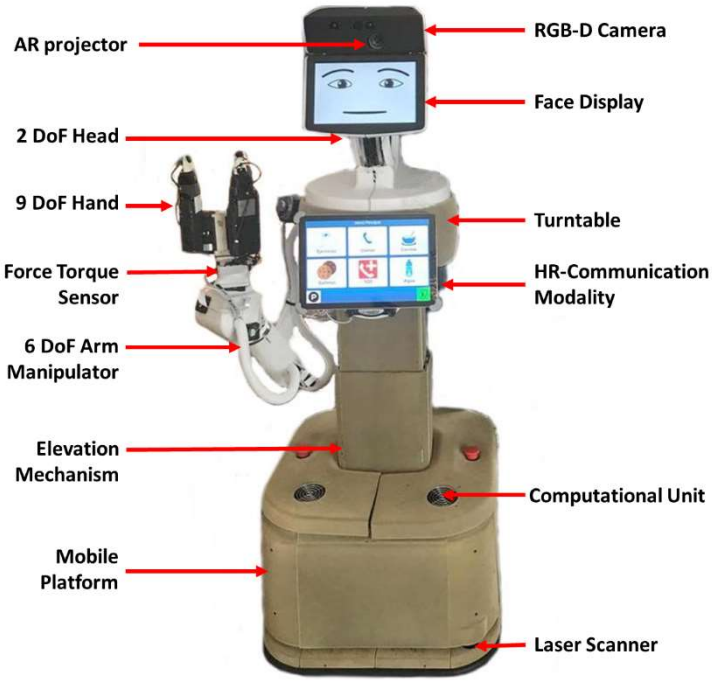


Figura 7. Componentes hardware de RAMCIP.

2.1.2.4 Kompaï R&D

Este robot (13) diseñado y desarrollado por Kompaï Robotics. Dispone de una arquitectura modular que le permite desempeñar multitud de tareas, así como integrar software libre haciendo que gracias a estas dos características sea un sistema muy personalizable, facilitando el desempeño del objetivo o actividad que se le designe.

Debido a su diseño modular, este robot permite realizar una amplia variedad de actividades, siendo estas las siguientes:

Empuje o transporte de material: gracias a sus dos motores de 80w sin escobillas y su módulo formado por dos barras rotatorias que extienden el alcance del robot, este es capaz de empujar

desde una silla de ruedas hasta carritos o esterilizadores basados en rayos ultravioleta, siendo esta una de sus cualidades más remarcables, pues al poder realizar este tipo de tareas puede llegar a ser de gran ayuda para el personal sanitario.

Asistente de movilidad: al igual que en el caso anterior, debido a las dos barras rotatorias, este sistema es capaz de ayudar a desplazarse a personas que padezcan de alguna discapacidad o limitación motora, de forma similar al funcionamiento de un andador, pero con la ventaja de que es el robot el que se mueve y por lo tanto, la persona no tendrá que realizar un gran esfuerzo para poder desplazarse con él, otra ventaja es la gran estabilidad que ofrece el robot frente a un andador convencional.

Este robot, tal y como se ha mencionado anteriormente, también permite la personalización de su software, esto es posible gracias a que su diseño está basado en ROS, siendo esta una plataforma de software libre para poder desarrollar sistemas basados en robótica. Todo esto integrado en su unidad de desarrollo y programación, una Jetson TX2 de Nvidia.

Uno de los puntos débiles de este robot es su autonomía, pues tal y como se indica en la página web del fabricante, dura hasta un máximo de 6 horas, pudiendo variar dependiendo del uso que se le esté dando.

En relación con su arquitectura hardware, está compuesto por los elementos que aparecen en la Figura 8:

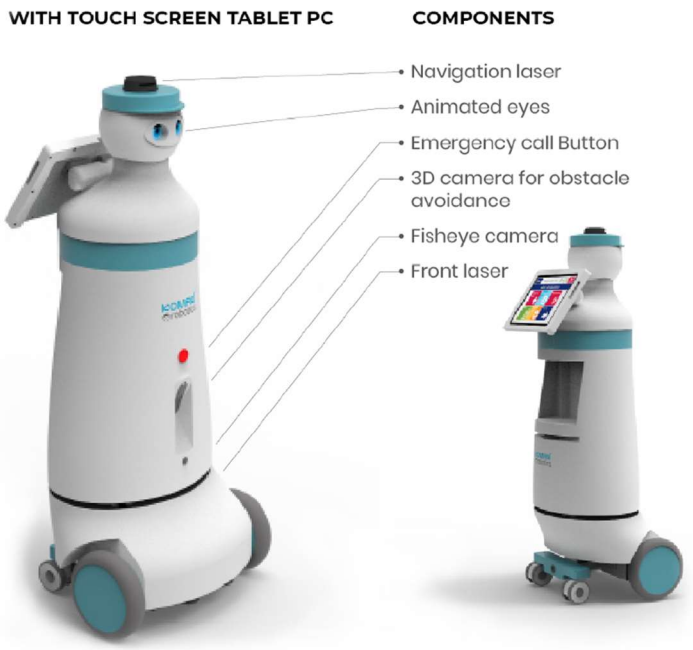


Figura 8. Componentes hardware de Kompaï R&D.

2.1.2.5 LARES

LARES (14) es un sistema robótico desarrollado por KAIST (Instituto Avanzado de Ciencia y Tecnología de Korea) en Korea del Sur por Z. Zenn Bien, Kwang-Hyun Park, Won-Chul Bang y Dimitar H. Stefanov, diseñado con la intención de alcanzar una interacción entre robot y persona lo más cercana posible, haciendo que puedan coexistir y colaborar estas dos partes de la forma cotidiana. Al contrario de los sistemas robóticos mencionados anteriormente, este contiene diferentes elementos domóticos, así como sillas de ruedas y camas inteligentes, pues está pensado para ser instalado directamente en la habitación de la persona mayor o discapacitada.

El desarrollo de este proyecto surgió de la gran demanda de asistencia por parte de personas mayores, pues esta parte de la población es cada vez mayor, debido a la elevada esperanza de vida de países desarrollados, llegando a ser de más de 80 años, así como de la necesidad de que personas con algún tipo de limitación o discapacidad puedan llevar una vida más segura y fácil.

Este proyecto se basa en la integración de tres elementos fundamentales, la integración de una cama inteligente, un sistema de reconocimiento capaz de detectar diferentes gestos o posturas llevadas a cabo por el usuario, y por último una red capaz de unificar y comunicar los dos elementos anteriores. Pudiéndose observar todo esto en la Figura 9.

En relación con la cama inteligente esta será capaz de reclinarse en diferentes posiciones, también dispondrá de un brazo robótico MANUS, véase en la Figura 10, el cual será capaz de acercar o mover objetos a petición del usuario, pudiendo realizar estas actividades gracias al sistema de reconocimiento instalado en la habitación.

Figura 9. Instalación cama inteligente.

Figura 10. Brazo robótico MANUS.

En las etapas iniciales del proyecto se diseñó un sistema del control basado en un control remoto como forma de manejar todo lo que relaciona a la cama inteligente, pero esta idea se desechó rápidamente pues este sistema no sirve para personas mayores o discapacitadas pues pueden perder el control remoto fácilmente o simplemente no acordarse de donde lo dejaron, no siendo capaces de encontrarlo, por lo tanto se optó por un sistema de reconocimiento basado en una serie de sensores y cámaras RGB instaladas por la habitación, mediante lo cual se puedan reconocer diferentes gestos como el señalar algún objeto con la mano, siendo esta una mejor manera de controlar todo este sistema robótico por parte del usuario. Este método de control se observa en la Figura 11.

Figura 11. Sistema de control remoto por gestos.

2.1.2.6 PARO robot terapéutico

PARO (15) fue desarrollado por el AIST (Instituto Nacional de Ciencia y Tecnología Avanzada), una institución líder en Japón en el ámbito de la robótica y automatización. Este es un robot terapéutico utilizado para tratar a personas con algún tipo de desorden o enfermedad psicológica, como lo puede ser la demencia o el Alzheimer, habiéndose usado en Europa y Japón en este tipo de tratamientos desde el 2003. En relación con su diseño, su aspecto está basado en el aspecto de una cría de foca para hacerlo más amigable y cercano, pues se hizo un estudio donde se concluyó que una foca es un animal "neutro", esto quiere decir que en relación con otros animales, muy poca gente ha tenido malas experiencias con este tipo de animal, haciéndolo idóneo para este propósito (16); a su vez cuenta con diferentes tipos de sensores tanto de percepción de luz, presión, sonido, temperatura y postura, haciendo que la interacción que pueda tener un paciente con él sea lo más amplia posible.

PARO es capaz de distinguir, gracias a sus diversos sensores, la luz de la oscuridad también puede reconocer cuando se le está tocando o por ejemplo se le ha tomado en brazos, e incluso es capaz de reconocer patrones de voz y distinguir de qué lugar le han llegado dichos estímulos.

Otra de las características que hacen a este robot tan útil en su uso en la medicina terapéutica, es la capacidad de poder expresar "emociones", tales como enfado, alegría o tristeza, en función del trato que le brinde él usuario, además de emitir sonidos y moverse imitando a una cría de foca, tal y como se puede apreciar en la Figura 12.

Figura 12. PARO robot terapéutico.

3 CAPITULO 3

3.1 Objetivo y condiciones del sistema

El objetivo de este proyecto es el diseño y desarrollo de un conjunto de componentes software distribuidos compatibles con el ecosistema ROS (Robotic Operating System), que conforman una arquitectura software capaz de controlar la arquitectura hardware (sensores, actuadores y control), de un robot asistencial real para que sea capaz de realizar las siguientes tareas:

- **Seguimiento y reconocimiento de personas**: el robot deberá ser capaz de seguir a una persona y detenerse a una distancia especificada por el usuario, cuando la persona a la que sigue se haya parado. Si no detecta a la persona objetivo iniciará un proceso de reconocimiento del lugar, por medio de rotaciones sobre su propio eje, hasta volver a encontrar al sujeto.
- **Evitación de obstáculos**: esta es una característica esencial que debe incorporar el robot, pues estará operando en entornos donde la presencia humana es muy elevada, por lo que hay una gran probabilidad de que se encuentre otras personas u objetos por su camino mientras sigue a la persona indicada.
- **Utilización, tanto descarga como carga, de datos de la Nube**: se deberá crear una plataforma en la Nube que contenga tanto un backend como un frontend, de tal forma que el robot pueda tanto almacenar como consultar la información que se encuentre en ella.
- **Distinción de la persona objetivo**: el robot tendrá que ser capaz de discernir la persona a la que está siguiendo de entre otros posibles sujetos que haya a su alrededor para que la función del seguimiento funcione de forma eficaz.
- **Navegación autónoma**: el robot deberá ser capaz de orientarse y dirigirse a un lugar determinado arbitrariamente por el usuario gracias a un mapa del lugar de trabajo del robot, que previamente ha sido escaneado por el mismo. Esta función no se contempla en este Trabajo Fin de Grado, pues no es uno de sus objetivos, pero es una funcionalidad necesaria de cara a proyectos futuros y que ha sido implementada.

La elección de los diferentes elementos que conformarán la estructura hardware y software del robot ha sido enormemente condicionada tanto por los objetivos mencionados previamente como por las siguientes condiciones:

- Versatilidad y escalabilidad del sistema, pues el fin de este proyecto no es diseñar un robot asistencial incapaz de implementar nuevos elementos o características, ya que de esa forma se estaría acortando enormemente sus capacidades y vida útil, debido a que al estar pensado para funcionar en un entorno real no controlado, lo contrario a un ambiente automatizado, donde pueden ocurrir imprevistos o cambios, el robot debe estar preparado para poder llevar a cabo diferentes actividades dependiendo de las circunstancias en las que opere.
- Robustez, puesto que, al trabajar en un ambiente con tanta entropía, como puede ser un hospital o una residencia, el robot debe de poder actuar de forma segura, integrando diferentes planes de contingencia que se pongan en marcha ante cualquier emergencia o imprevisto que pueda surgir en el entorno.
- Facilidad de implementación, este es un requisito importante, pues como se ha mencionado anteriormente, este sistema debe poder ser escalable, por lo tanto, todos los componentes que lo conformen deben ser compatibles entre sí.

3.2 Arquitectura hardware

La estructura hardware ha estado muy condicionada por el tipo de software que se iba a utilizar, pues tanto la detección como el reconocimiento de personas precisan de un tipo de programación de alto nivel que no cualquier computador consigue ejecutar con fluidez. En etapas iniciales del proyecto se intentó utilizar una Raspberry Pi 4 (17) como unidad central de proceso, donde se ejecutarían los diferentes programas que dictarían el comportamiento del robot, pero tras numerosas pruebas se desechó esta idea, pues este dispositivo carecía de la capacidad operacional suficiente para poder ejecutarlos, debido a que introducía una latencia demasiado alta para un sistema que debe trabajar prácticamente a tiempo real. Como solución a esta problemática se optó por utilizar un miniPC el cual sí era apto para cumplir con nuestros requisitos. Los demás componentes hardware utilizados han sido una cámara RGB-D, en concreto una Asus Xtrion Live Pro (18), un dispositivo de barrido Láser 2D y el propio robot.

3.2.1 Asus Xtion Live Pro

Se ha optado por el uso de esta cámara RGB-D (Figura 13) debido a su compatibilidad con el sistema operativo Ubuntu Linux, así como su facilidad de uso en el entorno ROS, pues solo se tiene que descargar el paquete OpenNI (19) para emplearla.

Esta cámara cuenta con las especificaciones mostradas en la Tabla 1:

Tabla 1. Características Asus Xtrion Live Pro

Asus Xtrion Live Pro	
Consumo energético	Menor de 2,5 W
Distancia de uso	Entre 0,8 m y 3,5 m
Campo de visión	58° Horizontalmente 45° Verticalmente 70° Diagonalmente
Tipo de sensor	RGB, profundidad y micrófono.
Resolución	640x480 píxeles a 30 fps
Sistema operativo compatible	Windows y Ubuntu Linux
Dimensiones	18 cm x 3,5 cm x 5 cm

Figura 13. Cámara RGB Asus Xtrion Live Pro.

3.2.2 Hokuyo UTM-30LX

En relación con el láser utilizado, se ha usado el modelo UTM-30LX de Hokuyo (Figura 14) (20), el cual nos brinda unas prestaciones más que suficientes para que se cumplimenten de forma

más que óptima nuestros requisitos a la hora de desarrollar este proyecto. Este laser se caracteriza por ser apropiado para su implementación en robots que operen a altas velocidades, debido a su alta velocidad de respuesta, constando con un arco o rango de detección de unos 270°, así como de una distancia máxima de detección de 30 m y una distancia mínima de detección de 0,1 m. Otras de sus características más relevantes, como también las mencionadas previamente se encuentran detalladas en la Tabla 2:

Tabla 2. Especificaciones Hokuyo UTM-30LX.

Hokuyo UTM-30LX	
Fuente de alimentación	12VCD±10%
Fuente luminosa	*Diodo semiconductor laser*: λ=905nm *Clase de seguridad del láser*: 1 FDA
Consumo de corriente	*Max*: 1 A *Normal*: 0,7 A
Distancia de detección	0,1 m hasta 30 m
Rango de detección	270°
Precisión	*De 0,1 m a 10 mm*: ± 30 mm *De 10 m a 30 m*: ± 50 mm
Precisión angular	0,25° (360°/1.440)
Tiempo de escaneo	25 msec/scan
Nivel de sonido	Menor de 25 dB
Salida síncrona	NPN colector abierto
Interfaz	USB2.0
Tipo de conexión	Cable USB con conector tipo-A

Figura 14. Láser Hokuyo UTM-30LX.

3.2.3 Pioneer 3-DX

El robot utilizado en el diseño y desarrollo de este sistema robótico ha sido el modelo Pioneer 3-DX (Figura 15) (21) (22), el cual es un robot ligero compuesto por una rueda loca giratoria y dos ruedas propulsadas por un par motores diferenciales, haciendo que sea ideal para su uso en interiores. Su pequeño tamaño, junto con su gran autonomía, gracias a que se le pueden conectar hasta tres baterías al mismo tiempo, lo hace idóneo para su utilización en el proyecto que nos acontece, es más, la gama de robots Pioneer está diseñada precisamente para su uso en proyectos de investigación, así como para su uso en el ámbito educativo, puesto a la facilidad que presenta a la hora de implementarle mejoras tanto a su software como a su hardware. Este modelo trae consigo un codificador rotatorio o encoder en cada una de sus ruedas motrices, una batería, un microcontrolador con el firmware de ARCOS y un panel frontal con SONAR. Las características técnicas de este robot se detallan en la Tabla 3:

Tabla 3. Características Pioneer 3-DX

Pioneer 3-DX	
Construcción	Chasis de aluminio de 1.6 mm de grosor Ruedas rellenas de goma
Operatividad	*Peso del robot*: 9 kg *Peso máximo con el que puede cargar*: 17 kg

Movimiento de accionamiento diferencial	*Máxima velocidad alcanzable*: 1,2 m/s *Velocidad de rotación máxima*: 300°/s *Terreno transitable*: interiores
Potencia	*Tiempo de uso*: 8 a 10 horas dependiendo del número de baterías *Tiempo de carga*: 10 horas o 2,5 horas con el uso de un cargador de gran capacidad *Fuentes de alimentación disponibles*: 5 V a 1,5 A conmutados 12 V a 2,5 A conmutados
Baterías	Pueden utilizarse hasta 3 a la vez *Voltaje de salida*: 12 V *Capacidad*: 7,2 Ah (cada una) *Tipo*: Plomo ácido Se pueden intercambiar las baterías en caliente
Microcontrolador Entradas/Salidas	32 entradas digitales 8 salidas digitales 7 salidas analógicas 3 puertos de expansión serie
Panel de control del usuario	Zumbador piezoeléctrico programable MIDI Indicador luminoso de alimentación Indicador de carga de baterías Dos interruptores de alimentación auxiliar Interruptor para reinicio del sistema Pulsador de habilitación de los motores

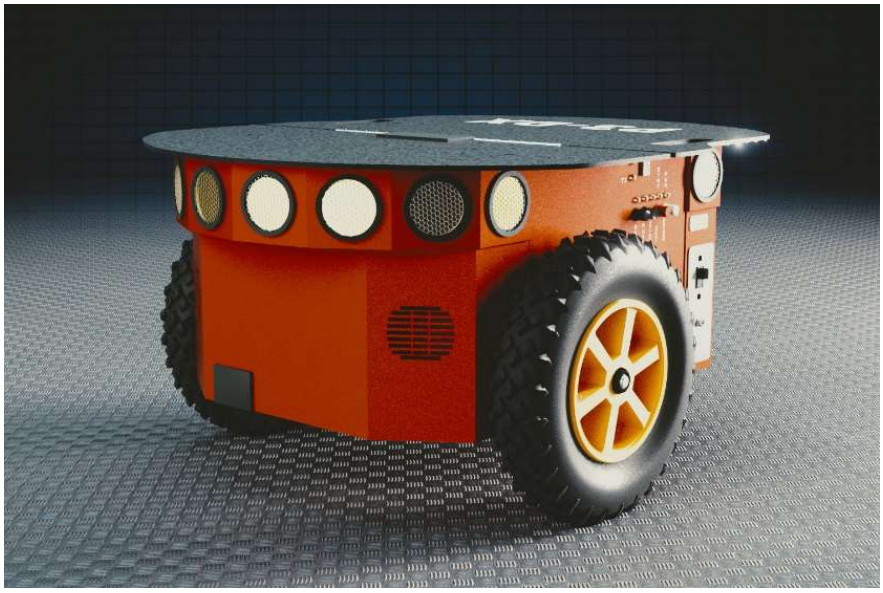


Figura 15. Pioneer 3-DX.

3.2.4 MiniPC MSI

Debido al uso de lenguaje de alto nivel como MediaPipe a la hora de programar ciertas partes de este proyecto, se ha requerido de una capacidad computacional muy superior de la que pueden proporcionar dispositivos como una Raspberry Pi 4, por lo tanto, se ha optado por el uso de un minipc (Figura 16) para ejecutar todos los programas desarrollados, sin la preocupación de que se puedan dar posibles fallos o errores a causa de que todos ellos se estén reproduciendo de forma simultánea. En concreto se ha utilizado un minipc con un procesador Intel Celeron con dos núcleos y unos 8 GB DDR4 de RAM.

Figura 16. MiniPc MSI.

3.3 Arquitectura software

En cuanto a las herramientas software utilizadas, podemos destacar ROS (Robot Operating System) y MediaPipe, usadas para la creación de la estructura software del robot la cual permite el correcto su funcionamiento, mientras que, en relación con el diseño de la plataforma en la Nube, se ha optado por el uso de los software de programación Eclipse y Visual Studio Code (23), este último para programar de forma más sencilla y óptima el frontend de nuestra plataforma, junto con el servicio Google Cloud.

3.3.1 ROS

ROS es un framework de desarrollo específicamente diseñado para desarrollar software para Robótica, de código abierto, mediante el cual cualquier desarrollador software puede programar todo tipo de aplicaciones para robots gracias a su extenso conjunto de librerías, paquetes y herramientas.

3.3.1.1 Modelo Publicador/Subscriptor

ROS se basa en un modelo de nodos, estos serían cada uno de los programas que constituyen la arquitectura software distribuida, los cuales definen el comportamiento del robot. Dichos nodos se comunican entre sí por medio de mensajes, pudiendo ser estos desde simples números enteros hasta imágenes completas. Cada nodo desconoce la existencia de los otros, pues su manera de comunicarse radica en el modelo de publicación/subscripción (24) (25), donde cada mensaje se publica en un cierto topic, funcionando este como un depósito donde se almacenan ciertos mensajes de un tipo en específico, haciendo que a la hora de que un nodo quiera acceder a la información a un mensaje en concreto, este deberá suscribirse al topic donde ese mensaje ha sido almacenado. Otro factor que hace de ROS un método tan efectivo a la hora de diseñar y programar sistemas basados en robótica es el hecho de que cada nodo puede publicar y subscribirse a más de un topic a la vez. A continuación, se puede apreciar en la Figura 17 las relaciones entre nodos y cómo funciona el sistema publicador/subscriptor a forma de resumen:

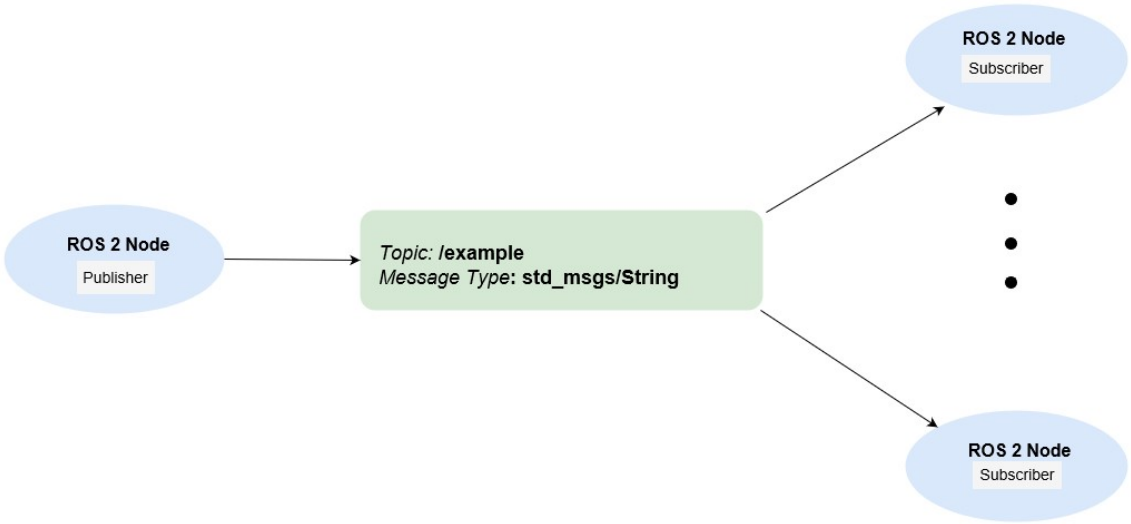


Figura 17. Sistema publicador/subscriptor ROS.

3.3.1.2 Servicio Cliente/Servidor

ROS también brinda la posibilidad de que los nodos se puedan intercambiar información mediante un tipo de comunicación denominado servicio (26). Un servicio se basa en la relación cliente-servidor entre dos o más nodos, siempre teniendo en cuenta que solo puede existir un servidor para cada tipo de servicio que se cree, pero cada uno de estos puede contener más de un cliente. La comunicación entre ambas partes se lleva a cabo de forma síncrona, pues cada vez que un cliente solicite una petición al servidor para acceder a un determinado tipo de mensaje, los cuales seguirán estando enrutados en sus topic correspondientes, este se quedará bloqueado o parado hasta que reciba la respuesta del servidor, esto puede tener su parte positiva y negativa, ya que si el servidor no responde o la respuesta es demasiado lenta el cliente se podrá quedar bloqueado un tiempo indefinido, siendo esto fatal para el correcto funcionamiento de nuestro sistema, mientras que por otra parte al quedarse el cliente esperando la información del servidor, nos estamos cerciorando de que el nodo que actúa como cliente no ejecutará ninguna operación hasta que le llegue el mensaje requerido. La forma de definir un servicio es mediante un nombre, el cual funcionará como su identificador, y luego se deberán especificar dos tipos de datos, uno será el mensaje o el conjunto de mensajes que el nodo cliente le enviará en modo de petición al nodo servidor, mientras que el otro tipo de dato estará formado por el conjunto de mensajes que el servidor enviará al cliente en forma de respuesta. En la Figura 18 se puede ver de forma gráfica cómo se realiza el intercambio de información entre nodos cliente y servidor:

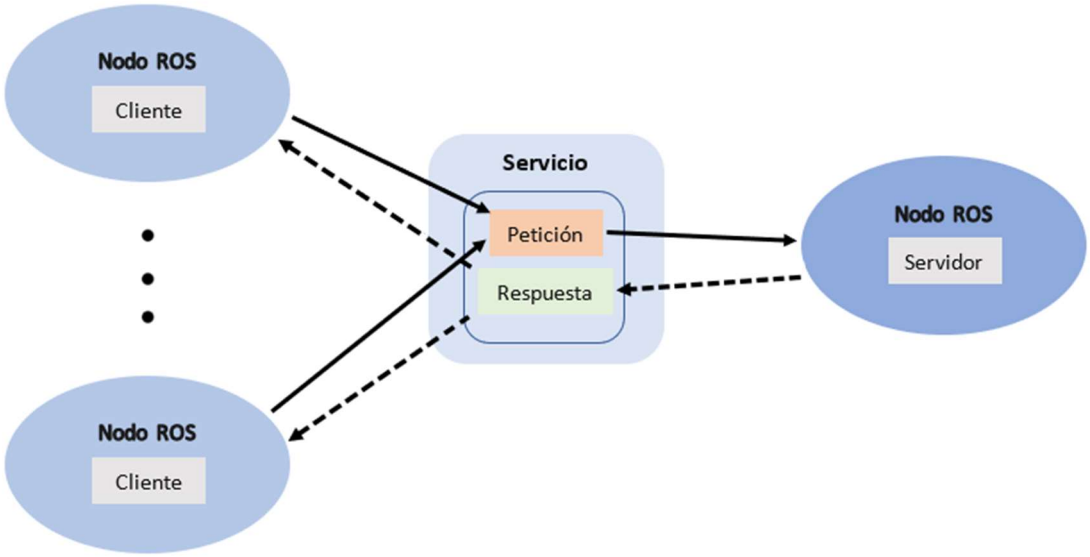


Figura 18. Servicio Cliente/Servidor ROS.

3.3.1.3 Lenguajes de programación

ROS permite trabajar con diversos lenguajes de programación, gracias a esto se pueden crear multitud de programas con objetivos y funciones muy diversas. En este proyecto se ha necesitado emplear tanto C++ como Python a la hora de desarrollar los programas que conformarán los nodos de nuestro robot que dictarán su funcionamiento. ROS al trabajar por defecto con el lenguaje C necesitaremos descargar una librería extra, al igual que cuando se quiere trabajar utilizando Python, ya que se necesita para poder utilizar MediaPipe dentro del entorno de ROS, siendo estas librerías “Roscpp” (27) y “Rospy” (28) respectivamente, ambas totalmente documentadas, respaldadas y mantenidas desde la misma web de ROS.

Roscpp y Rospy son unas librerías que implementan el lenguaje de programación C++ y Python respectivamente a ROS, esto se debe a que implementan una client library o librería cliente, la cual consiste en una recopilación de código mediante el cual permite al programador crear la mayoría de los elementos básicos de ROS, ya sean nodos, publicar mensajes en topics o subscribirse a ellos, crear servicios, etc., mediante dichos lenguajes de programación.

3.3.1.4 OpenNI

OpenNI es un framework de código libre capaz de interconectar una cámara RGB cuyo modelo no sea Kinect, debido a que no es compatible con ese tipo de dispositivo, con el entorno ROS. Gracias a esto se pueden generar imágenes que muestren profundidad, formadas por una nube de puntos donde cada uno almacena su posición respecto a los tres ejes de coordenadas X, Y y Z, luego dichas imágenes generadas se almacenarán en topics para que los nodos puedan acceder a dicha información fácilmente.

Existe también OpenNI 2.0 (29) actualmente, por lo que se deberá utilizar la versión que mejor funcione con la versión de ROS que se tenga instalada en nuestro ordenador.

3.3.2 MediaPipe

MediaPipe (MP) es un software libre multiplataforma mediante el cual se pueden construir diferentes soluciones basadas en Machine Learning capaces de procesar de forma automática datos de archivos de video o imagen, además de poder trabajar con video en tiempo real.

El kit de herramientas que ofrece MediaPipe comprende desde el marco de trabajo o Framework hasta los diferentes modelos que ofrece, pudiéndose apreciar en la Figura 19 cómo está estructurado dicho kit:

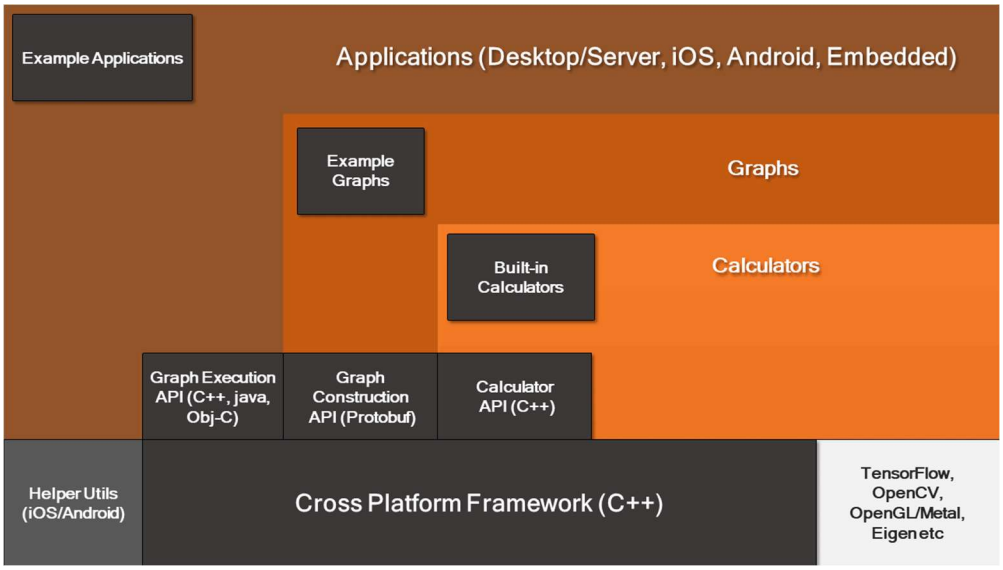


Figura 19. Estructura MediaPipe.

La percepción de la información que MediaPipe realiza de un archivo se basa en una serie de grafos formados por una determinada cantidad de puntos de referencia o landmarks, dependiendo de la solución por la que se haya optado, unidos mediante un número de líneas. En concreto, MediaPipe asigna unos valores a cada uno de esos landmarks, los cuales son los que se utilizarán para las diferentes aplicaciones que se les quisiera dar. En la Figura 20 se observa una imagen antes y después de ser procesada por una de las soluciones:

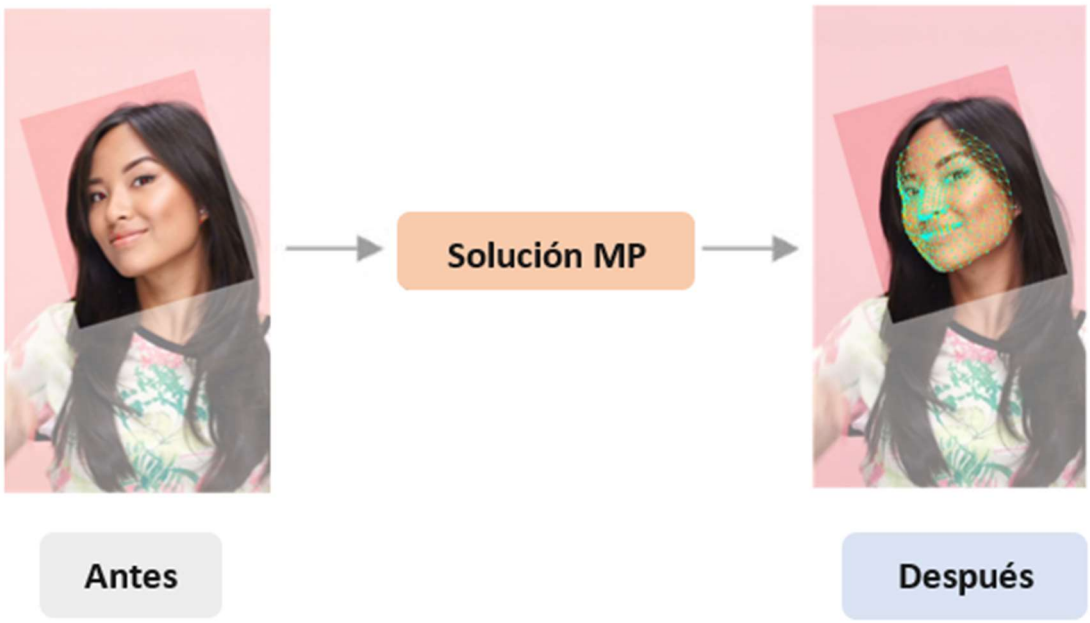


Figura 20. Proceso MediaPipe.

Actualmente MediaPipe cuenta con un total de dieciséis modelos basados en Machine Learning (30), así como la capacidad de implementarse en hasta seis lenguajes de programación, pero, a la hora de utilizar este software se deberá tener muy en cuenta esto último, pues dependiendo del lenguaje escogido, se podrán usar unas soluciones u otras, tal y como se aprecia en la Tabla 4.

Tabla 4. Lenguajes de programación compatibles con las soluciones de MediaPipe.

	Android	IOS	C++	Python	JS	Coral
Face Detection	✓	✓	✓	✓	✓	✓
Face Mesh	✓	✓	✓	✓	✓	
Iris	✓	✓	✓			
Hands	✓	✓	✓	✓	✓	
Pose	✓	✓	✓	✓	✓	
Holistic	✓	✓	✓	✓	✓	
Selfie Segmentation	✓	✓	✓	✓	✓	
Hair Segmentation	✓		✓			
Object Detection	✓	✓	✓			✓
Box Tracking	✓	✓	✓			
Instant Motion Tracking	✓					
Objetron	✓		✓	✓	✓	
KNIFT	✓					
AutoFlip			✓			
MediaSequence			✓			
Youtube 8M			✓			

Para este proyecto se necesitará realizar un seguimiento y reconocimiento de una persona, por ende, la solución que se ha decidido utilizar ha sido la denominada Pose, la cual se detallará más adelante. Luego se ha tenido que escoger el lenguaje de programación que se ha de emplear para crear nuestro programa, por lo que, echando un vistazo a la Tabla 4 se puede apreciar cómo podemos utilizar tanto Android, IOS, C++, Python o JavaScript (JS). Finalmente se optó por el uso de Python debido a su mayor facilidad de uso con respecto a las otras opciones, además de su alta compatibilidad con ROS y Ubuntu Linux, permitiendo la creación de nodos ROS basados en Python haciendo leves modificaciones en los que nos encontramos por defecto en dicho entorno y sin la necesidad de instalarse ningún compilador en específico, tal y como sucede al utilizar C++, ya que este requiere de Bazelisk para poder ejecutarse el modelo de forma correcta, puesto que el uso de compiladores externos a ROS puede perturbar o entorpecer el correcto funcionamiento del sistema, junto con la complejidad que eso conlleva.

3.3.2.1 Pose

Pose (31) es una solución proporcionada por MediaPipe que permite reconocer y realizar un seguimiento con gran exactitud de un cuerpo humano. Este modelo cuenta con 33 landmarks o puntos de referencia representados en el espacio tridimensional, junto con la capacidad de realizar una máscara de segmentación del fondo del video o imagen gracias al uso de BlazePose (32) y a una cámara RGB, este último siendo un requisito necesario.

Uno de los principales problemas a la hora de usar modelos similares a Pose, donde se necesita llevar a cabo un seguimiento en tiempo real del cuerpo de una persona, como es nuestro caso, es el requerimiento de dispositivos con hardware muy potente con el gran coste económico que supone. Sin embargo, Pose se puede ejecutar en prácticamente cualquier dispositivo móvil, ordenador o incluso desde el propio navegador.

La obtención de los 33 puntos de referencia mencionados anteriormente se efectúa de la siguiente forma:

1. Primero se emplea un modelo mediante el cual se ubica a la persona y se establece la región de interés alrededor de la misma. Este modelo tal y como se aprecia en la Figura 21 haya a la persona en la imagen y recorta el mínimo área de la imagen donde esta se encuentre.

Figura 21. Ajuste a la región de interés.

2. Una vez realizado el proceso anterior, se procederá a aplicar el modelo de puntos de referencia de la solución Pose en la imagen resultante de la aplicación del modelo previo, pudiendo observarse el resultado en la Figura 22:

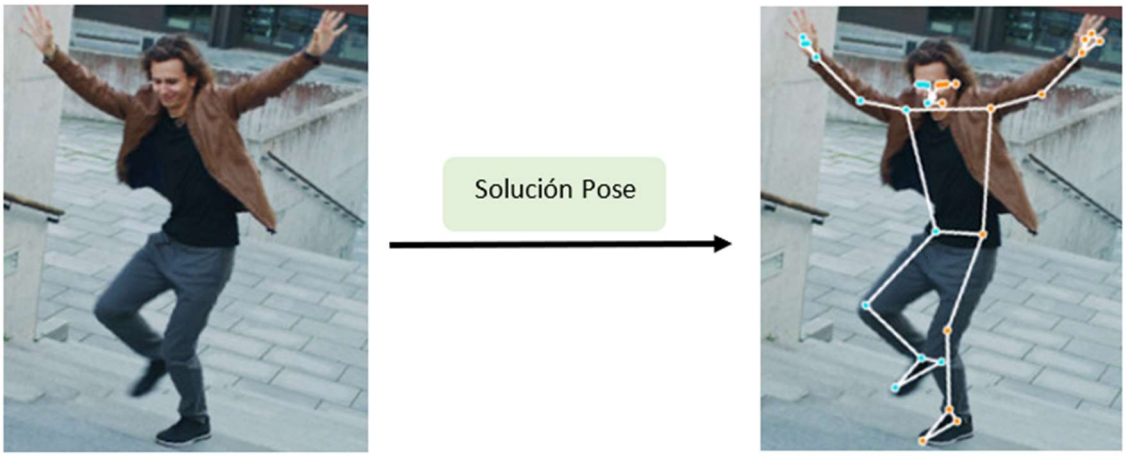


Figura 22. Solución del modelo Pose.

3. Por último, si la entrada a la solución Pose es un vídeo los pasos anteriores solo se aplicarán al primer fotograma por defecto, ya que para los siguientes el modelo solo realizará un seguimiento de los landmarks o puntos clave, no obstante, si dichos puntos se pierden o el programa no es capaz de localizarlos se volverán a invocar las dos funciones anteriores. Este proceso se puede ver de forma clara en el diagrama de la Figura 23:

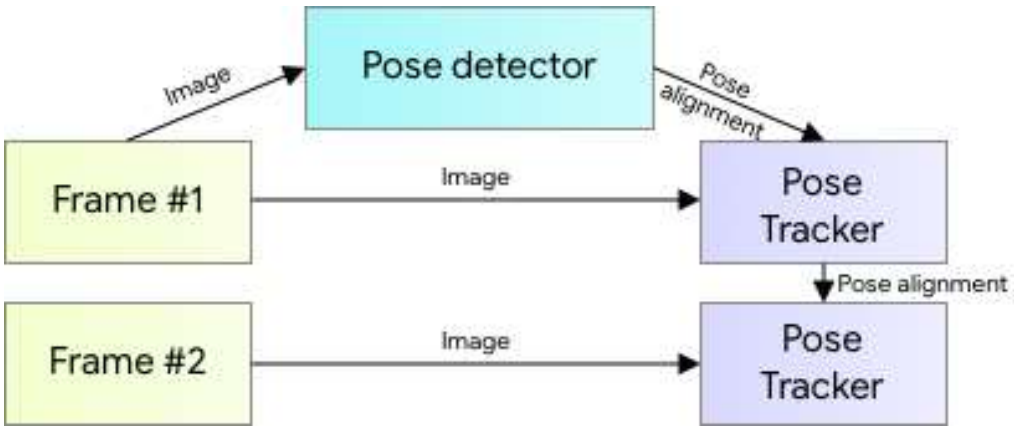
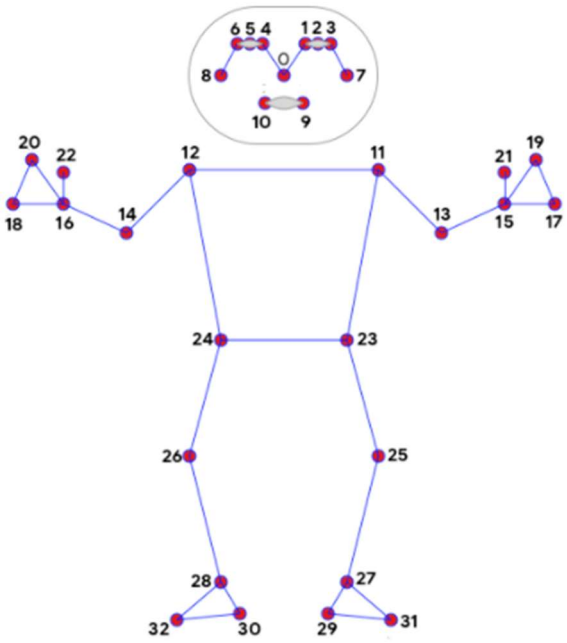


Figura 23. Proceso de la solución Pose.

En la Figura 23 se puede distinguir como solo en el primer fotograma se aplica el proceso de detección de los puntos de referencia, y que en los siguientes fotogramas solamente realiza un seguimiento de estos.

3.3.2.2 Landmarks

Los 33 puntos de referencia o puntos clave que utiliza Pose para determinar la posición exacta en la que se encuentra el cuerpo de una persona en el espacio tridimensional, los cuales han sido mencionados previamente, están numerados y asociados a un nombre, tal y como se puede apreciar en la Figura 24.



0. nose
1. left_eye_inner
2. left_eye
3. left_eye_outer
4. right_eye_inner
5. right_eye
6. right_eye_outer
7. left_ear
8. right_ear
9. mouth_left
10. mouth_right
11. left_shoulder
12. right_shoulder
13. left_elbow
14. right_elbow
15. left_wrist
16. right_wrist
17. left_pinky
18. right_pinky
19. left_index
20. right_index
21. left_thumb
22. right_thumb
23. left_hip
24. right_hip
25. left_knee
26. right_knee
27. left_ankle
28. right_ankle
29. left_heel
30. right_heel
31. left_foot_index
32. right_foot_index

Figura 24. Landmarks de la solución Pose.

3.3.2.3 Composición de los landmarks

Cada landmark o punto de referencia/clave está compuesto por una serie de datos que brindan la información necesaria para poder realizar el seguimiento y detección correcto del cuerpo de una persona, siendo estos los siguientes:

- **x**: coordenada cuyo valor se encuentra normalizado entre 0,0 y 1,0 obtenida del ancho de la imagen.

- **y**: coordenada cuyo valor se encuentra normalizado entre 0,0 y 1,0 obtenida del alto de la imagen.
- **z**: coordenada que representa la profundidad a la que se encuentra un landmark, medido desde la referencia del punto medio de los puntos clave de la cintura, siendo su valor cada vez menor si el punto de referencia se encuentra cada vez más cerca del dispositivo que capte la imagen. Su escala es aproximadamente igual a la coordenada x.
- **Visibility o visibilidad**: indica en un rango de valores desde el 0,0 al 1,0, el nivel de visibilidad del punto de referencia, siendo 0,0 totalmente ocluido y 1,0 perfectamente visible.

3.3.2.4 Parámetros de ajuste

En cuanto a la configuración de la solución Pose, disponemos de 5 parámetros para ajustar el modelo a nuestras preferencias y requisitos, siendo estos:

- **STATIC_IMAGE_MODE:** esta variable permite dos modos, false y true, de manera que si se le asigna el valor false tratará la entrada al modelo Pose como un video en tiempo real, de esta forma detectará y localizará a la persona más visible en los primeros fotogramas y tras esto solo realizará un seguimiento de los landmarks en los demás fotogramas. Sin embargo, si se le atribuye el valor true, realizará tanto la detección y localización de la persona junto con la asignación de los landmarks correspondientes en cada uno de los fotogramas que se le suministren como entrada, esta opción es recomendable utilizarla cuando se quieren tratar con este modelo imágenes estáticas y sin relación entre ellas. El valor por defecto de este parámetro es false.
- **MODEL_COMPLEXITY**: esta variable determina la complejidad del modelo de puntos de referencia que se esté usando, pudiendo obtener un valor desde 0 al 2, siendo el 0 el de menor complejidad y el 2 el de mayor. El valor por defecto es 1.
- **SMOOTH_LANDMARKS**: al asignarle el valor true este parámetro filtra los puntos clave del cuerpo de una persona a través de varias imágenes para reducir la vibración de estos en la respuesta final del modelo. Se ha de tener en cuenta que este parámetro solo funciona si STATIC_IMAGE_MODE es false. Su valor por defecto es true.
- **ENABLE_SEGMENTATION**: este parámetro genera una máscara de segmentación del cuerpo de la persona si su valor es true. Por defecto tiene asignado el valor false.
- **SMOOTH_SEGMENTATION**: al igual que la variable SMOOTH_LANDMARKS, si su valor es igual a true, evita la vibración o temblequeo, a través del filtro varias imágenes de

entrada o fotogramas, de la máscara de segmentación creada si ENABLE_SEGMENTATION tiene un valor true, por lo tanto, solo realizará su función si ENABLE_SEGMENTATION es true y STATIC_IMAGE_MODE es false. Siendo por defecto igual a true.

- **MIN_DETECTION_CONFIDENCE**: este parámetro determina el valor mínimo de confianza del modelo de rastreo de personas para considerarlo un resultado exitoso, pudiéndose variar dicho valor desde 0,0 a 1,0. Su valor por defecto es 0,5.
- **MIN_TRACKING_CONFIDENCE**: determina el valor mínimo de confianza que se debe tolerar para que el modelo de puntos de referencia de la solución Pose sea aceptable. A mayor valor, conllevará una menor precisión de los resultados logrados. Cabe destacar que si STATIC_IMAGE_MODE tiene como valor true, esta variable será ignorada. Puede obtener valores desde 0,0 a 1,0, siendo su valor por defecto igual a 0,5.

3.3.3 Eclipse

Se trata de un entorno de desarrollo integrado o IDE (integrated development environment) gratuito (Figura 25) (33), mediante el cual se pueden programar aplicaciones relacionadas con la nube en diversos lenguajes de programación, siendo el más utilizado Java, para luego ser desplegadas tanto de forma local como directamente en la nube.

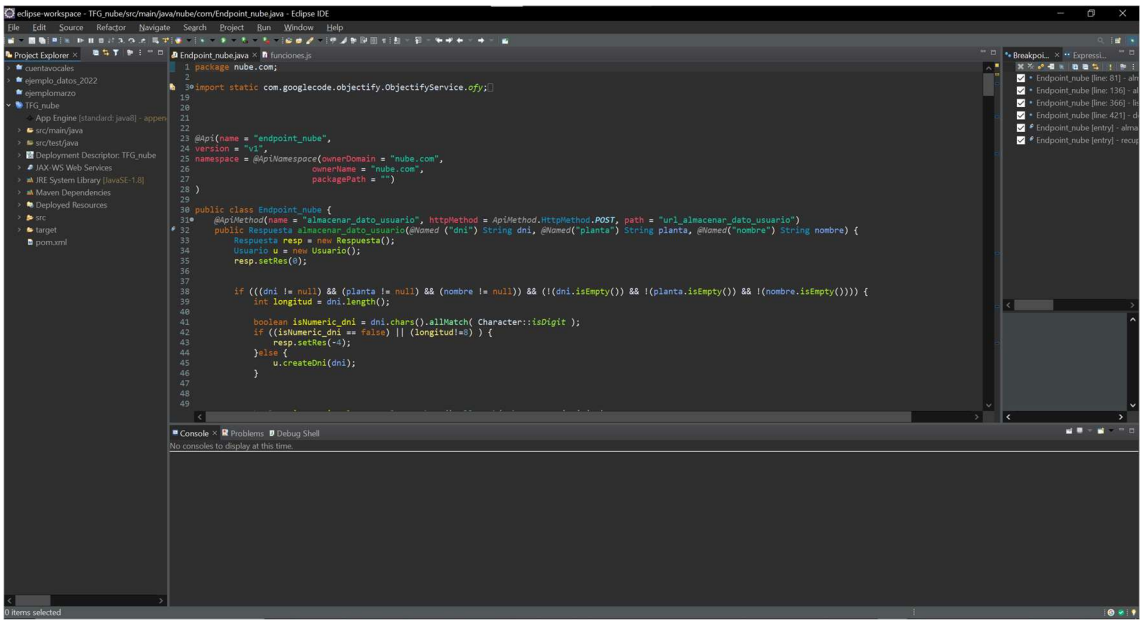


Figura 25. Entorno de trabajo de Eclipse.

3.3.4 Google Cloud

Google Cloud (34) engloba un vasto conjunto de recursos tales como ordenadores, servidores, unidades de disco y medios virtuales, que se ubican en los centros de datos de Google. Estos centros se localizan en diversas partes del mundo, como Asia, Europa, Australia o América, con el fin de proporcionar una mayor estabilidad en sus servicios ante cualquier posible fallo del sistema, así como de una menor latencia en el intercambio de datos.

Este medio permite el despliegue de aplicaciones que se hayan programado en entornos como Eclipse IDE, de manera que se puedan acceder a ellas desde cualquier dispositivo mediante una plataforma frontend. Todo esto es posible debido a que en la computación en la nube tanto el software como los hardware tradicionales pasan a ser servicios, donde por medio de estos se pueden desarrollar infinidad de páginas y recursos web.

Los servicios ofertados por esta plataforma abarcan una gran cantidad de opciones, desde aplicaciones basadas en IoT (Internet de las Cosas), hasta ML (Machine Learning), o incluso Big Data. En este proyecto se han utilizado los servicios llamados App Engine y Cloud Storage.

3.3.4.1 App Engine

App Engine es un servicio que permite desplegar páginas web que cualquier desarrollador haya programado de forma sencilla y ágil, ya que gracias a su infraestructura todo es gestionado por el propio servicio. Acepta un amplio abanico de lenguajes de programación mediante los cuales se haya creado la aplicación, siendo algunos ejemplos los siguientes: Java, PHP, C#, Go, Python, Ruby, Java o Node.js.

3.3.4.2 Cloud Storage

Este servicio (35) permite el almacenamiento de los objetos relacionados con algún proyecto ubicado en Google Cloud, siendo estos objetos datos inmutables que pueden estar formados por cualquier tipo de archivo, quedando almacenados en contenedores denominados buckets, los cuales son únicos para cada proyecto. En la Figura 26 se puede apreciar de forma gráfica cómo quedaría la estructura jerárquica que compone este servicio:

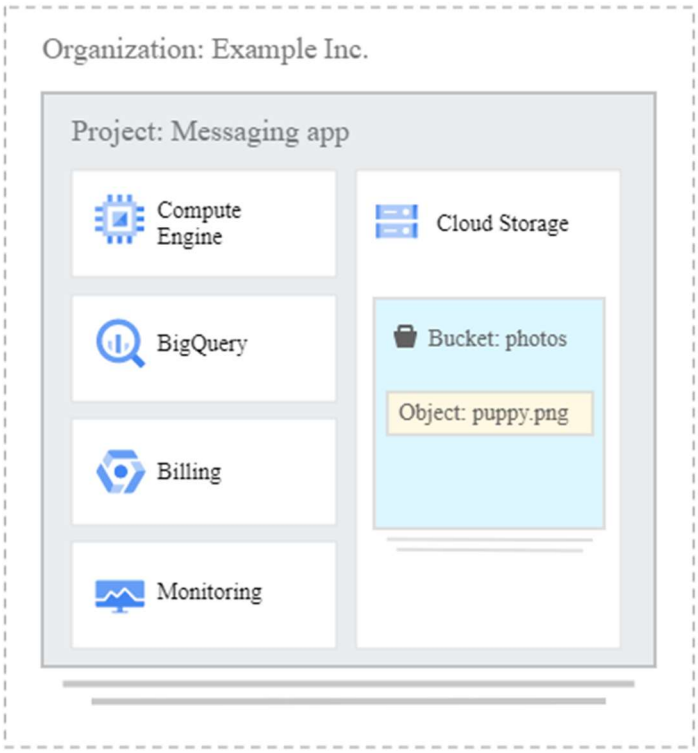


Figura 26. Estructura jerárquica Cloud Storage.

3.3.5 Componentes del sistema

Una vez han sido explicados los diferentes componentes, servicios y soluciones que se han utilizado durante el desarrollo de este sistema robotizado, estamos en situación de comenzar a explicar tanto las funcionalidades como el diseño de los diferentes elementos que permiten que el robot pueda desempeñar sus actividades objetivo correctamente.

En este apartado se detallará tanto el diseño como las funciones de cada uno de los programas y aplicaciones que conforman nuestro proyecto.

3.3.5.1 Servicio

Para el correcto funcionamiento de nuestro sistema robótico, se ha tenido que crear un servicio (36) mediante el cual podamos intercambiar información entre dos nodos, siendo estos los correspondientes al reconocimiento de la persona mediante el uso de MediaPipe y al cálculo de la distancia en milímetros de la cámara hacia la persona. Se ha optado por este tipo de comunicación bilateral debido a que para poder obtener la distancia mencionada se ha tenido que utilizar el lenguaje de programación C++, mientras que para la detección de la persona se

ha utilizado Python ya que es más sencillo e intuitivo usar este lenguaje para crear un nodo que contenga Mediapipie en ROS, sin embargo, no se ha podido incluir el cálculo de la profundidad en este último debido a la gran complejidad que esto conllevaría.

Este servicio contiene cuatro datos diferentes que el cliente, en este caso el nodo que contiene el programa de reconocimiento de la persona, enviará al servidor, nodo que calcula la profundidad, como petición a través de este servicio. Esos cuatro datos son los siguientes:

- **x**: coordenada del eje horizontal en píxeles obtenida de los fotogramas extraídos por la cámara RGB.
- **y**: coordenada del eje vertical en píxeles obtenida de los fotogramas extraídos por la cámara RGB.

Estas dos coordenadas provienen exclusivamente de uno de los landmarks generados por el modelo Pose de MediaPipe, pues solo nos hace falta uno de estos para poder hacer el seguimiento de una persona.

- **Image depth**: Imagen generada por la cámara RGB gracias al software OpenNI, la cual contiene los valores de profundidad almacenados gracias a la nube de puntos.
- **Image depth_rect**: Imagen generada por la cámara RGB gracias al software OpenNI, conteniendo esta los valores de profundidad rectificados, es decir, proyectados en el eje z de la cámara, almacenados gracias a la nube de puntos.

Como respuesta, el nodo servidor enviará dos datos como respuesta, a través del servicio, al nodo cliente. Esa información es la siguiente:

- **Distancia**: distancia en milímetros a uno de los landmarks calculada a partir de los datos proporcionados por el nodo cliente.
- **Distancia rectificada**: distancia o profundidad rectificada, a uno de los puntos de referencia, proyectada sobre el eje z de la cámara RGB, obtenida por medio de los datos suministrados por el nodo cliente.

Para una mejor comprensión se muestra en la Figura 27 un gráfico que sintetiza el funcionamiento de este servicio cliente-servidor:

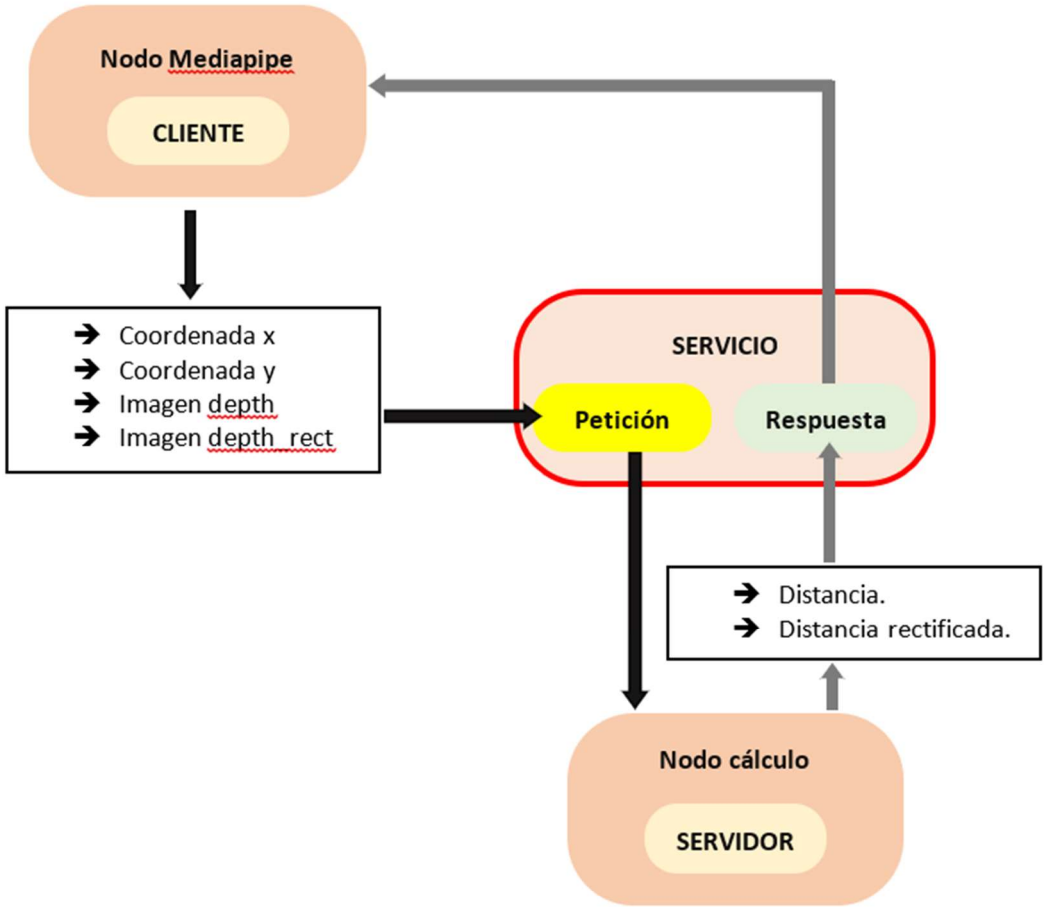


Figura 27. Funcionamiento servicio cliente/servidor.

3.3.5.2 Reconocimiento de la persona

Para la detección y localización de una persona se ha utilizado MediaPipe, concretamente su solución denominada Pose usando el lenguaje de programación Python para poder llevar a cabo esta operación, tal y como se ha comentado en apartados anteriores.

Al trabajar en el entorno ROS, se ha tenido que programar un nodo el cual pueda desempeñar esta acción, estando formado por 3 funciones, siendo estas las siguientes:

- **Cliente del servicio**

  Hace de cliente para el servicio creado al cual se le pasarán cuatro datos, estos son la posición en píxeles en coordenadas “x” e “y” del landmark a usar como punto de referencia para localizar a la persona, además de tanto la imagen generada por la nube de puntos, mediante la cual podemos obtener la distancia desde el eje de referencia de la cámara a

dicha posición, como de la imagen que contiene esa distancia rectificada o proyectada en el eje de referencia z de la cámara.

El cliente de un servicio desempeña un papel crucial para la correcta ejecución de todos los nodos del sistema, pues una vez el programa entra en ella, esta se bloqueará hasta que reciba la respuesta correcta del servidor. Sin embargo, para que no se nos pueda quedar un tiempo indefinido parado el programa, cosa que podría ser fatal para el sistema, se le ha implementado un método try/catch por el que, si no recibimos la información apropiada del servidor, nos aparecerá un mensaje de error y se saldrá de esta función, pudiendo intentar establecer la conexión con el servidor de nuevo.

Si todo se ha ejecutado de forma correcta y se han recibido los datos deseados por parte del servidor, la función devolverá una de las dos respuestas posibles de nuestro servicio, ya sea la distancia de la cámara al punto de referencia o esa misma distancia rectificada, dependiendo de cuál sea la que se necesite.

- **Callback**

El callback es una de las funciones más básicas de un nodo ROS, puesto que será llamada en cada una de las iteraciones del programa desde el listener, el cual se explicará más adelante. En ella se ha implementado la solución Pose de MediaPipe, así como se ha calculado tanto la coordenada “x” como la “y” reales desde el sistema de referencia de la cámara, el cual funcionará como nuestro sistema de referencia global, esto es así debido a lo siguiente:

Tanto el láser que implementa el robot, como la cámara poseen un sistema de referencia propio, por lo que se ha tenido que establecer un sistema de referencia global para poder referir todas las medidas a un mismo. Por ello se ha hecho coincidir el eje “y” del sistema de referencia de la cámara con el láser, pero con un desfase debido a la colocación de los mismos elementos hardware, aunque esto no es un problema ya que las medidas realizadas se han tomado respecto al eje “z”. En la Figura 28 se representa la disposición de ambos sistemas de referencia:

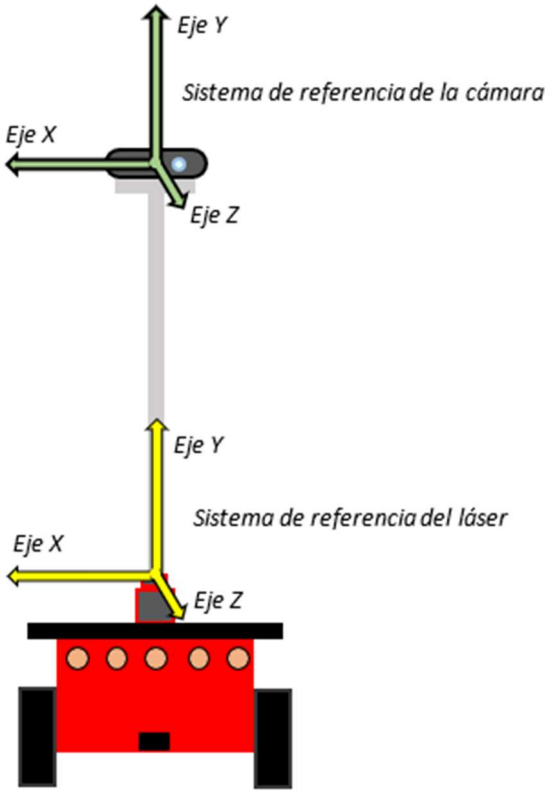


Figura 28. Sistemas de referencia del láser y cámara RGB.

Como se puede ver en la Figura 28 ambos sistemas de referencia coinciden en el eje y, haciendo que se pueda considerar como sistema global o de referencia el de la cámara RGB, puesto que la orientación de los demás ejes es la misma en ambos.

A esta función se le han de pasar todos los datos, obtenidos desde sus correspondientes topic, a los cuales se haya subscrito este nodo, siendo cada uno de los fotogramas recogidos por la cámara, junto con sus correspondientes imágenes de nubes de puntos que almacenen la medida de la profundidad y la misma pero rectificada. Por ende, serían un total de tres datos: imagen, imagen de profundidad e imagen de profundidad rectificada.

Lo primero que se realizará en el callback será la conversión de los datos de la imagen y la imagen de profundidad. Esto es así debido a que la información que se recoge mediante OpenNI y se almacena en el determinado topic, no puede ser usada directamente por MediaPipe, por lo que se deberá hacer una conversión a un tipo de dato válido. Realizado lo anterior se procederá a configurar los parámetros de configuración del modelo Pose, en nuestro caso solo se ha establecido el min_detection_confidence y min_tracking_confidence a 0,5, puesto que es una confianza más que aceptable para

nuestro proyecto, mientras que el resto de los parámetros se han dejado con su valor por defecto.

Luego se ha aplicado el modelo Pose especificando que solo realice el seguimiento y detección del hombro izquierdo de la persona, obteniendo de esta manera las coordenadas x e y del susodicho, pero todo esto sin la opción de que al ejecutarse el programa se pueda ver en una ventana el landmark señalado en el vídeo, puesto que no ha sido necesario para este sistema robótico.

Una vez obtenidas las coordenadas deseadas, se procederá a llamar a la función del cliente del servicio, de la que obtendremos como respuesta la distancia de la profundidad rectificada en milímetros, para que una vez obtenida esta, se realizará la conversión de las coordenadas de píxeles a milímetros, puesto que MediaPipe trabaja mediante coordenadas basadas en los píxeles de la imagen que se le introduce como entrada, por lo tanto si queremos darles un uso real a estas, se debe realizar dicha conversión. La Figura 29 muestra una imagen que desglosa ambos tipos de coordenadas:

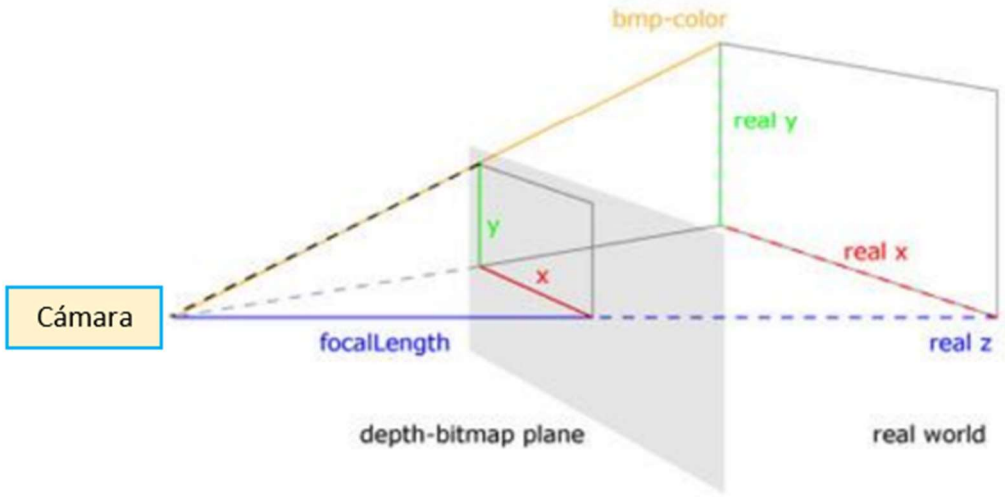


Figura 29. Ejes de coordenadas de la cámara RGB.

Como se puede observar en la Figura 29, podemos ver una diferenciación entre coordenadas generadas en el plano de profundidad de bits o depth-bitmap plane, y el plano referente al mundo real, por lo que se ha de realizar una conversión de las coordenadas del plano de

profundidad de bits a sus correspondientes en el plano real, ya que no disponemos de estas últimas de una forma directa.

Para poder realizar la conversión previamente mencionada se ha de utilizar la siguiente ecuación:

$$coord_{real} = \frac{distancia}{distancia_{focal}} \cdot \left(coord_{pixel} - c_{coord}\right)$$

Donde cada una de sus variables se corresponde con lo siguiente:

- $\boldsymbol{Coord_{real}}$: esta sería la solución de la ecuación, correspondiéndose con la coordenada en el plano real.
- $\boldsymbol{Distancia}$: esta es la distancia de la profundidad rectificada del punto que se quisiera obtener el valor de una de sus coordenadas en el plano real.
- $\boldsymbol{Distancia_{focal}}$: este sería un parámetro propio de la cámara RGB que se esté utilizando, correspondiéndose con la distancia focal de la misma. Para poder obtenerlo podemos simplemente ver desde la terminal la respuesta del mensaje CameraInfo.msg ubicado en el topic sensor_msgs (37), donde encontraremos su valor en una matriz con K como identificador, teniendo la siguiente forma:

  $$K = \begin{pmatrix} fx & 0 & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{pmatrix}$$

  Siendo la distancia focal los parámetros fx y fy, ya que ambos son el mismo valor.

  - $\boldsymbol{Coord_{pixel}}$: es la coordenada obtenida mediante el modelo Pose de MediaPipe, cuya unidad viene dada en píxeles.
  - $\boldsymbol{c_{coord}}$: este parámetro también se ubica en la matriz K mencionada previamente, y se corresponde con los puntos principales del eje “x” o “y”, dependiendo de la coordenada que se quiera calcular. Su valor se da en píxeles y suele corresponderse con el punto medio de su respectivo eje.

Por último, una vez se han realizado de forma correcta las operaciones anteriores, se llevará a cabo la publicación de dos mensajes ubicados en sus correspondientes topics, uno será la distancia rectificada, mientras que el otro contendrá las coordenadas “x” e “y” referidas al plano real, esto se realizará debido a que nos será útil tener esas mediciones en unos topics a los cuales otros nodos se puedan subscribir.

- **Listener**

Constituye otra de las funciones básicas de un nodo ROS, siendo en ella donde el nodo se subscribirá a los topics correspondientes, además de definir cuantas veces se llamará al callback para que este se ejecute, es otras palabras, dicta el número de veces que el nodo estará activo.

En nuestro caso se quiere que el nodo no se pare o se bloquee mientras el usuario no lo decida, por lo que se ha configurado de manera que una vez el nodo está en funcionamiento, este trabaje de forma indefinida. Otra de las acciones que se tienen que efectuar en esta función, es la toma de datos de las imágenes a usar de forma síncrona, puesto que en todo momento se deberá trabajar con un fotograma y con sus correspondientes nubes de puntos, por lo que la existencia un desfase en la recepción de estos datos puede conllevar a una gran bajada de precisión en el seguimiento y detección de la persona. Para solventar esta problemática se ha empleado la función message_filters.ApproximateTimeSynchronizer (38), asegurándonos de esta manera que la recepción de los mensajes deseados sea simultánea.

3.3.5.3 Nodo Servidor

Este programa se ha diseñado para que trabaje exclusivamente como servidor del servicio explicado anteriormente. Por ello su estructura difiere de un nodo ROS convencional, pues no se subscribirá a ningún topic ni tampoco publicará ningún mensaje, solo intercambiará información con el nodo que incluya una función cliente.

Constará de dos funciones, las cuales se detallarán a continuación:

- **Obtención de la distancia de profundidad**

Gracias a esta función el servidor podrá comunicarse con el cliente, publicando la respuesta a su petición. Para que esto pueda ejecutarse de forma correcta, se ha programado de manera que la función reciba como variables de entrada los datos que el cliente envía como petición y las variables donde se almacenarán los datos publicados como respuesta a dicha petición. Una vez se le ha pasado la información mencionada, se realizará una operación try/catch para convertir adecuadamente las imágenes, conformadas por nubes de puntos y recibidas en forma de petición por parte del cliente, de esta manera se cerciorará de que se

ha podido realizar la conversión de esas imágenes a un formato con el que se pueda trabajar en C++, siendo este el lenguaje de programación utilizado en este nodo.

Realizadas las acciones anteriores, se procederá a obtener las distancias de profundidad de las coordenadas que han sido enviadas por el cliente (rectificada y por defecto), a la cámara RGB, en milímetros, esto es posible gracias a una función propia de OpenCV (39) en C++, debido a que este sería un proceso mucho más complicado de realizar en un lenguaje como Python.

Obtenidas las distancias, serán almacenadas en el servicio en las variables destinadas a la respuesta del servidor, haciendo posible que el cliente tenga acceso a ellas.

Por último, se procederá a calcular de nuevo las coordenadas reales, tal y como se hizo en el nodo destinado al reconocimiento de la persona, para poder ser visualizadas por la terminal, de manera que podamos comprobar que todo está funcionando correctamente. La única diferencia que se puede apreciar, respecto a la ecuación utilizada en el nodo de reconocimiento de la persona, es que la distancia ha sido dividida por 1000 para poder ver el resultado final en metros. Quedándose la ecuación de la siguiente forma:

$$coord_{real} = \frac{distancia}{distancia_{focal} \cdot 1000} \cdot \left(coord_{pixel} - c_{coord}\right)$$

- **Función main**

Es el núcleo de nuestro servidor, pues es de donde se llamará a la función explicada anteriormente tantas veces como sea necesario. En ella también se ha de publicar el servicio, al igual que se haría con la publicación de un topic, mediante un identificador que se le asigne arbitrariamente de manera que el cliente pueda acceder a él para comunicarse con el servidor, además se debe asignar la función que realizará las operaciones del servidor, en este caso será la mencionada previamente.

Como no queremos que el servidor deje de funcionar en ningún momento, se ha programado de manera que siempre que reciba una petición, este envíe una respuesta, por lo que solo se detendrá si un usuario lo detiene manualmente.

3.3.5.4 Seguimiento de la persona

Este nodo es uno de los más importantes del sistema robótico, pues se encarga del correcto funcionamiento y operabilidad de los motores del robot, permitiendo de esta manera que este pueda moverse con libertad y como consecuencia pueda seguir a la persona localizada mediante los nodos anteriores.

Al igual que los programas explicados previamente, el nodo ha sido subdividido en una serie de funciones, las cuales formarán la estructura del nodo. En este caso se ha dividido al nodo en cinco funciones, entre las que nos encontraremos 2 callback, cosa que no ocurría en ninguno de los programas anteriores. Seguidamente se detallará qué acción realiza cada una de las funciones, así como su estructura si procede:

- **Localizar**

  La función localizar desempeña un papel primordial es este programa, pues su tarea será la de verificar que las coordenadas a las que se subscribe este nodo, esto lo realizará en uno de los callback, son válidas para poder verificar si se está detectando de forma correcta a la persona o por si el contrario se ha perdido su rastro.

  Primero se han de crear dos variables globales donde se almacenen tanto la coordenada “x” como la “y”, para luego validar gracias a esta función que ambas coordenadas nos son útiles. El control de dichos datos se llevará a cabo mediante la comprobación de que ni la “x” ni la “y” sean igual a cero o a un valor no numérico, puesto que si esto es así la función devolverá un true haciendo saber que se ha de buscar a la persona, mientras que si no se da ningún caso de los anteriores, devolverá un false indicando que la persona está siendo detectada correctamente.

- **Parar**

  Esta función es imprescindible en un programa como este, pues es donde se controla el movimiento del robot, ya que determina cuando se ha de detener o cuando es posible que este avance el robot. Para poder realizar esta tarea se ha llegado a la conclusión de que el robot se detenga cuando la distancia entre él y cualquier obstáculo detectado por el láser, pudiendo ser un objeto o persona, sea menor a un metro, siendo esta distancia más que suficiente tras los ensayos realizados.

  Por lo tanto, a esta función se le deberá pasar como dato la información recibida a través del topic que almacena los datos recabados por el láser, siendo estos la distancia de

profundidad medida en cada uno de sus haces de luz infrarroja (este concepto se detallará más a fondo en el siguiente apartado), devolviendo como respuesta un valor true si tras recorrer cada uno de los puntos del láser, hay alguno cuya medición esté por debajo de un metro indicando que el robot debe pararse, aunque esta medida se puede modificar dependiendo del ambiente donde opere el robot, mientras que, si no se cumple esa premisa, devolverá un false posibilitando la libre movilidad del robot.

- **Esquivar obstáculo**

Uno de los objetivos de este proyecto era la capacidad de esquivar obstáculos que se le interpusiesen en el camino al robot mientras se realiza el seguimiento de una persona, por ello se ha creado esta función, la cual permite ejecutar esa tarea.

Antes de proceder a la explicación de su funcionamiento, debemos entender cómo es procesada la información recibida por del láser infrarrojo que se ha usado en este proyecto. Este dispositivo opera mediante la proyección de un determinado número de haces de luz infrarroja a lo largo de un radio concreto, normalmente en forma de abanico, para luego rebotar en la superficie donde alcance dicho haz y ser detectado por el láser. En concreto, el radio de detección del láser utilizado para este sistema robótico es el que se muestra en la Figura 30:

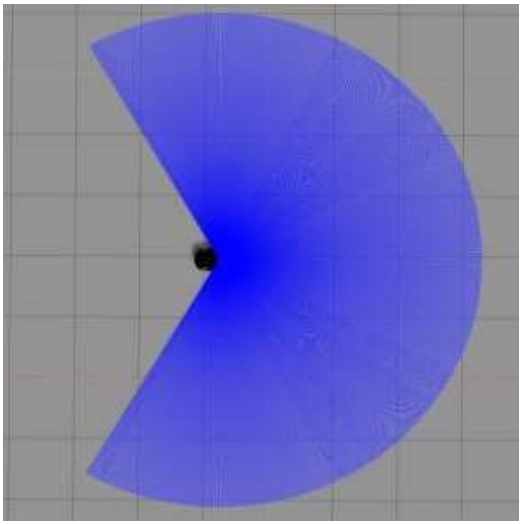
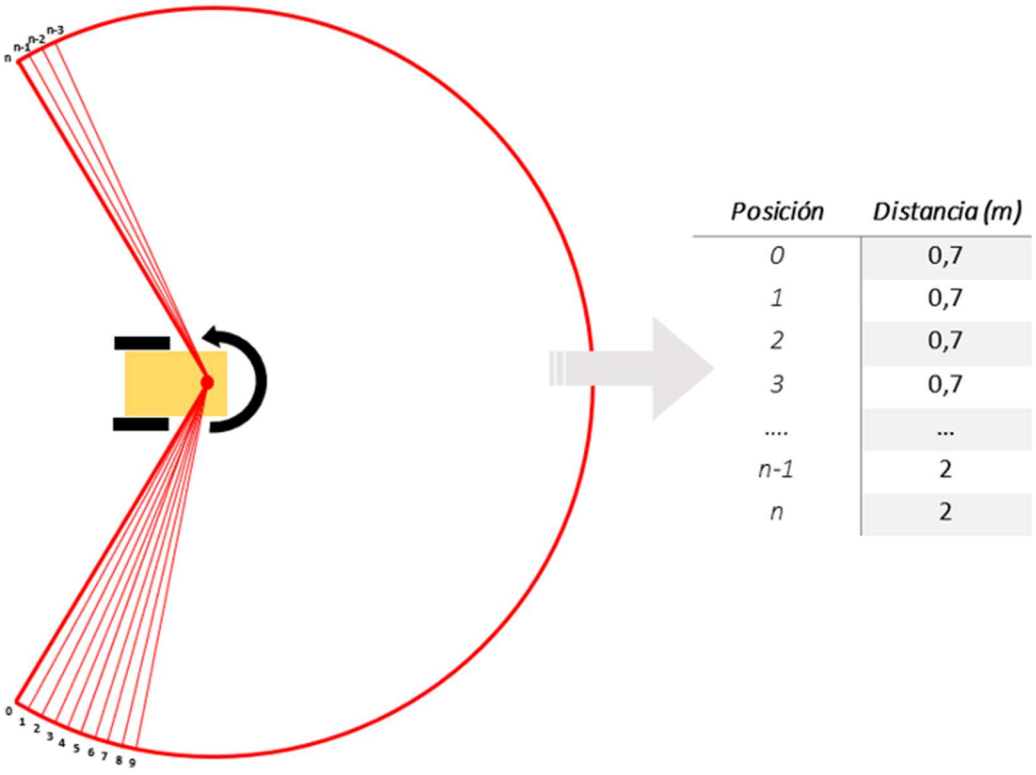


Figura 30. Radio de detección del láser.

Luego toda la información recibida por el láser es almacenada en un topic de manera que se hace corresponder a cada haz de luz infrarroja con un número y con una distancia, medida en metros, que se haya detectado. Todo esto para que cualquier nodo pueda subscribirse a

dicho topic y acceder a estos datos, como es el caso de este programa. De forma gráfica, en la Figura 31 se muestra la forma de recolección de los datos recibidos (40).



Posición	Distancia (m)
0	0,7
1	0,7
2	0,7
3	0,7
....	...
n-1	2
n	2

Figura 31. Método del láser para la recolección de medidas.

Como se puede apreciar en la Figura 31, a cada una de las posiciones que el láser es capaz de alcanzar se le asigna la medida recolectada por el haz infrarrojo correspondiente a dicha posición, desde la 0 hasta la "n", donde el valor "n" variará dependiendo del modelo de láser infrarrojo empleado. Una vez se ha comprendido el funcionamiento y cómo se almacenan los datos recibidos por el dispositivo, pasaremos a la explicación de esta parte del programa.

A la hora de esquivar obstáculos cuando el robot está siguiendo a una persona se deberán ubicar los objetos u otras personas que se localicen por delante del mismo, puesto que, si se encuentran por detrás no haría falta tenerlos en cuenta, ya que el robot nunca se desplazará marcha atrás, siempre lo hará hacia delante o girando cuando quiera cambiar de dirección. Teniendo este concepto en cuenta, podemos establecer el rango de detección del láser en unos 180°, siendo estos más que suficientes para poder desempeñar esta actividad.

Determinado el rango con el que trabajaremos, se ha optado por la partición de este en cinco áreas, las cuales precisarán en qué lugar se haya el obstáculo. Las áreas definidas son las siguientes:

- Derecha.
- Izquierda.
- Centro Izquierda.
- Centro Derecha.
- Centro.

Mostrándose en la Figura 32 la representación gráfica de las mismas, junto con el rango de operabilidad:

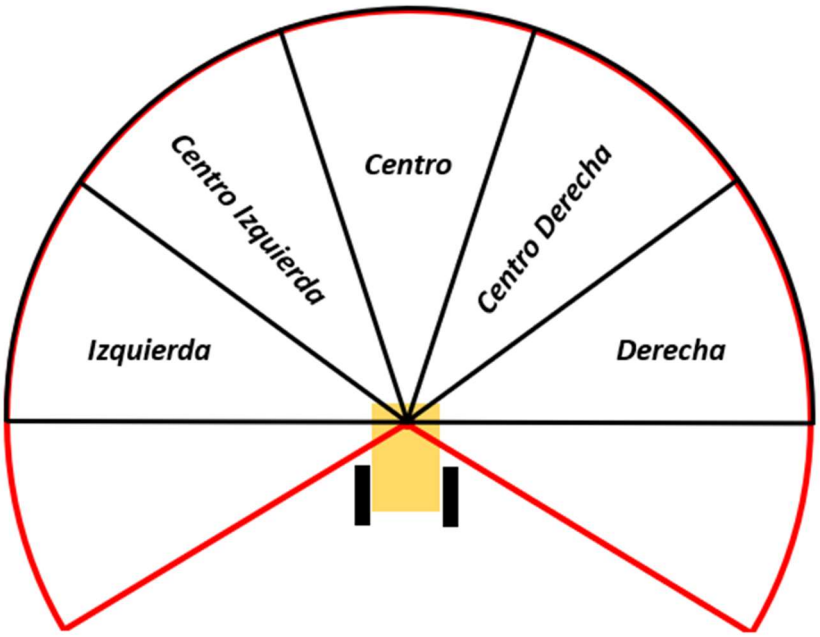


Figura 32. Áreas de detección definidas.

Definidas las zonas de detección de objetos o personas, se han de estudiar todos los casos posibles que se pueden dar a la hora de que el robot se encuentre con uno o varios de esos obstáculos, así como si es necesario emplear todas las áreas creadas, a lo cual tras varios ensayos se ha llegado a la conclusión que, con tan solo utilizar tres zonas, siendo estas la central, la centro derecha y la centro izquierda, el robot es capaz de evitar los obstáculos de una forma óptima. Por lo tanto, tendríamos un total de 8 casos posibles representados en la Tabla 5:

Tabla 5. Casos posibles al encontrar obstáculos.

	Centro izquierda	**Centro**	**Centro derecha**
Caso 1			
Caso 2		✘	
Caso 3			✘
Caso 4	✘		
Caso 5		✘	✘
Caso 6	✘	✘	
Caso 7	✘	✘	✘
Caso 8	✘		✘

Identificados todos los casos necesarios para que el robot pueda desempeñar esta tarea, procederemos a la explicación del funcionamiento de esta función, a la cual se le deberán asignar todos los datos almacenados por el topic que almacena la información recabada por el láser infrarrojo. Luego se establecerá una serie de condiciones las cuales determinarán en cuál de las tres posiciones (central, centro izquierda o centro derecha) se encuentran los obstáculos, esto se realizará estableciendo un umbral mínimo de detección, mediante el cual, si un objeto o persona se encuentra por debajo de ese umbral o rango en una de las zonas, estableciéndose en unos 0,3 metros, una de las condiciones se cumplirá, haciendo que la función devuelva su número asignado. Por ejemplo, si un objeto se encuentra a menos de 0,3 metros del robot y se ubica en la zona central, la función devolverá como valor entero un 2. De forma gráfica ocurriría se puede apreciar en la Figura 33:

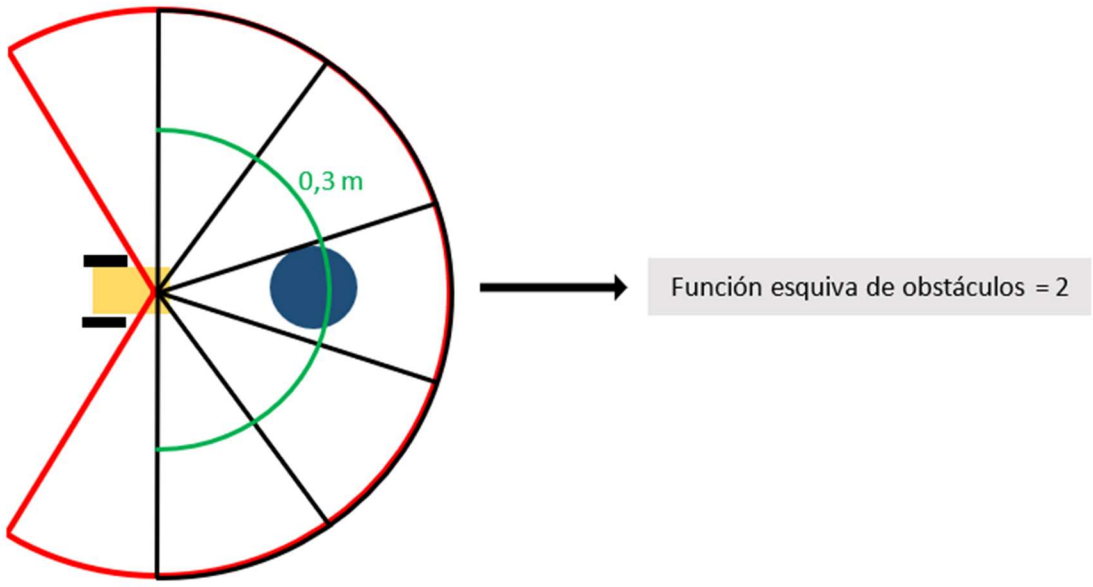


Figura 33. Detección de obstáculo.

- **Control velocidad**

Al estar destinado nuestro robot a trabajar en un entorno cerrado donde se encuentran muchas personas, así como obstáculos de diversa índole, se ha de controlar muy detenidamente la velocidad que pueda alcanzar. Para ello se ha creado esta función, la cual determina a qué velocidad puede moverse nuestro robot, midiéndose esta en m/s.

Se llegó a la conclusión de que el robot no debe sobrepasar los 0,2 m/s, puesto que si se utiliza un valor mayor aparecían diversos fallos y errores a la hora de localizar a la persona, a causa de que al existir una latencia considerable pero no excesiva a la hora de detectar y ubicar a la persona para realizar el seguimiento, el robot debe ir a una velocidad no muy elevada para que pueda procesar toda la información necesaria para ejecutar de forma correcta esa tarea. Otra condición que se tuvo en cuenta era la operatividad de los motores utilizados, ya que si estos reducían su velocidad por debajo de los 0,1 m/s podían llegar a pararse como consecuencia de cómo se realizan los intercambios de datos entre estos dispositivos y la placa base del robot.

Teniendo en mente estos requisitos, se estableció una función cuyo valor de entrada es la variable que almacena la coordenada “x” del sistema de referencia establecido en el robot, la cual se corresponde con la distancia de profundidad calculada en el nodo que ejecuta la solución Pose de MediaPipe, pues es ese nodo quien publica ese dato. Con este valor de entrada, se ha establecido la condición de que, si la variable x es menor a dos metros, la

velocidad que debe llevar el robot es de 0,1 m/s, pero si esta no se cumple, el robot deberá ir a una velocidad de 0,2 m/s.

- **Callback**

Tal y como se ha comentado al inicio de este apartado, este programa contiene dos callback, que son invocadas por el listener o función main, siendo esta la primera. Esta función es la responsable de asignar los valores de las coordenadas "x" e "y" respecto al sistema de referencia del robot, apreciándose en la Figura 34:

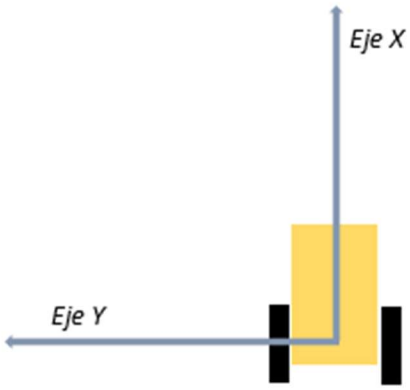


Figura 34. Sistema de referencia del robot.

Donde las variables que se le deberán pasar a esta función son las coordenadas publicadas por el nodo que contiene el modelo Pose de MediaPipe, correspondiéndose la coordenada "x" con la medida de la profundidad calculada y la coordenada "y" con la "x" del sistema de referencia de la cámara. Para asegurarnos que ambos valores son válidos, se ha establecido una condición que comprueba si ambos valores son numéricos, por lo que si no la cumplen se les asignará arbitrariamente el valor 0 a ambos.

- **Láser callback**

Esta función constituye el segundo callback de programa, siendo en este dónde se realicen la mayoría de las operaciones referentes al seguimiento de la persona previamente detectada. Como variables de entrada, se le atribuirán a esta función la información recibida por el láser, al igual que ocurría en la función encargada de indicar cuando debía detenerse el robot o la encargada de determinar en qué posición se encuentra un obstáculo.

Lo primero que se realizará en esta función será llamar a la función Parar para comprobar si el robot puede moverse, en caso de que deba detenerse se procederá a asignarle tanto a su velocidad lineal y angular un valor igual a cero, publicándose estos valores en el listener o función main para que los motores puedan seguir esta consigna. Por el contrario, si el robot puede moverse, se aplicará el algoritmo denominado Pure Pursuit o Persecución Pura, puesto que este se encargará de que el robot pueda realizar un seguimiento correcto de una persona gracias a la información suministrada por los demás nodos. Para aplicar este algoritmo establecemos un sistema de coordenadas en el robot como el mencionado en el primer callback (Figura 35).

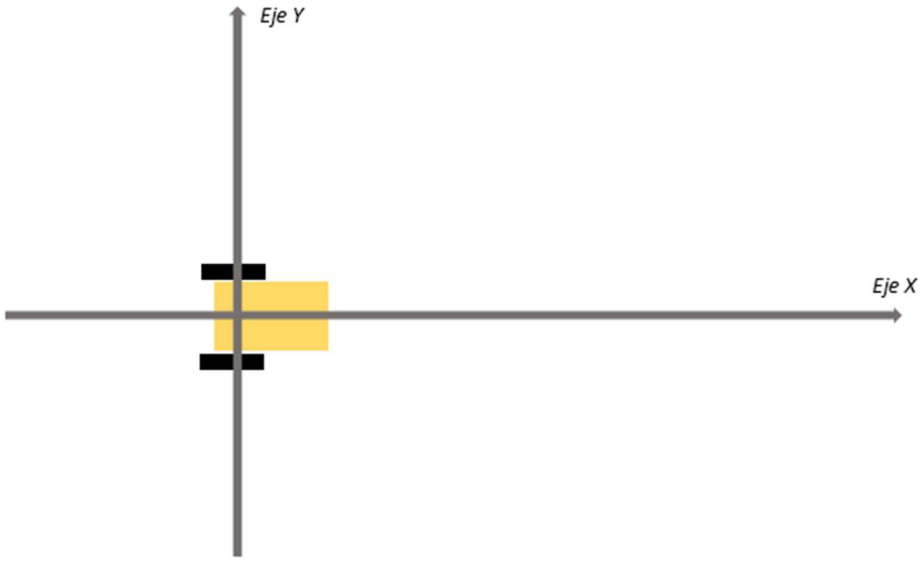


Figura 35. Sistema de referencia establecido.

Establecidos los ejes de coordenadas, designamos un punto objetivo al que debe desplazarse el robot, donde la distancia desde el centro del sistema de referencia y el objetivo se denomina L, en nuestro caso ese punto se corresponderá con el landmark obtenido gracias a la solución Pose de MediaPipe (Figura 36).

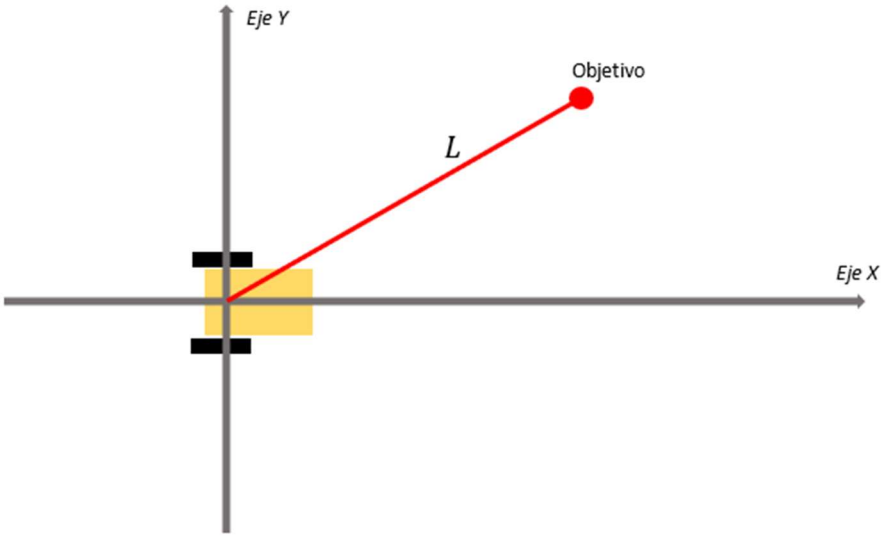
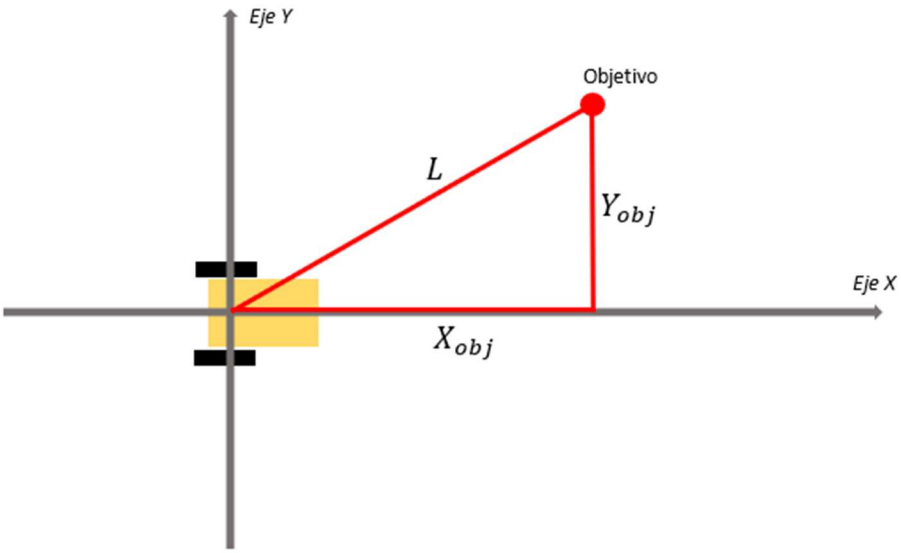


Figura 36. Objetivo establecido.

Por lo que la distancia L se puede calcular de la siguiente forma (Figura 37).



$$L^2 = X_{obj}^2 + Y_{obj}^2$$

Figura 37. Cálculo distancia L.

Donde la "X" y la "Y" del punto objetivo, son la medida de profundidad y la coordenada "x" respectivamente, obtenidas en el nodo que ejecuta la solución Pose, tal y como se ha explicado en el primer callback.

Una vez hallada la distancia entre el punto al que se debe llegar, necesitaremos el ángulo de giro, puesto que al definir nosotros arbitrariamente la velocidad lineal dependiendo de la distancia del robot a la persona, explicado en la función velocidad, es la única variable que nos faltaría para poder calcular la velocidad angular. Para obtener dicha variable se ha aplicado la siguiente ecuación:

$$R = D + Y_{obj}$$

La cual se obtiene si definimos una circunferencia de radio "R" la cual pase por el eje del sistema de coordenadas, así como por el punto objetivo, y cuyo centro se encuentre en el eje "Y" del sistema de referencia, véase la Figura 38.

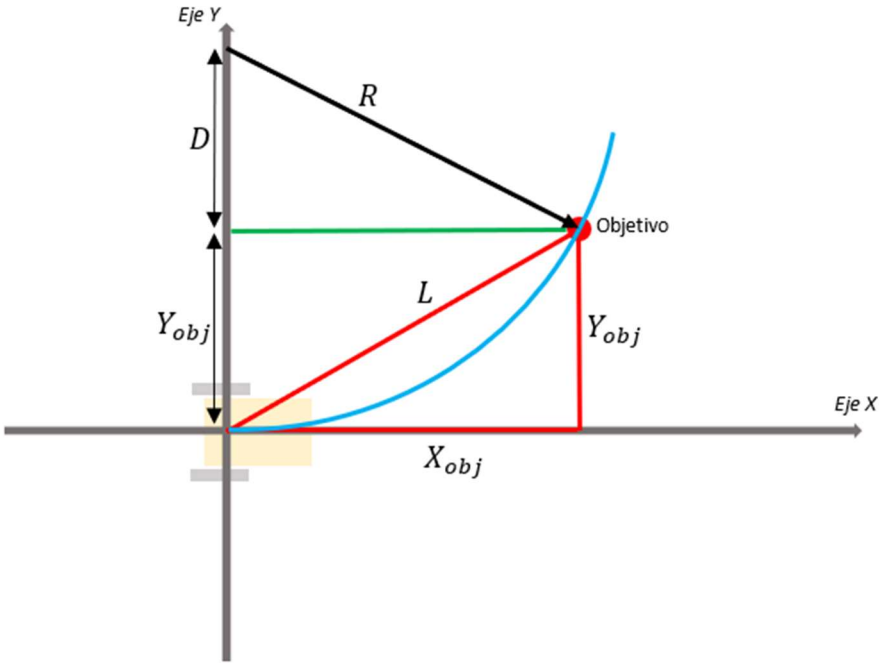


Figura 38. Trazo de la circunferencia con radio R.

Tras esto podemos hallar el ángulo de giro que debe realizar el robot, viéndose este proceso en la Figura 39:

$$R = D + Y_{obj} \Longrightarrow D = R - Y_{obj}$$

$$R^2 = D^2 + X_{obj}^2 \Longrightarrow R^2 = (R - Y_{obj})^2 + X_{obj}^2$$

$$\cancel{R^2} = \cancel{R^2} + Y_{obj}^2 - 2 \cdot R \cdot Y_{obj} + X_{obj}^2 \Longrightarrow 0 = Y_{obj}^2 - 2 \cdot R \cdot Y_{obj} + X_{obj}^2$$

$$Y_{obj}^2 + X_{obj}^2 = 2 \cdot R \cdot Y_{obj} \Longrightarrow L^2 = 2 \cdot R \cdot Y_{obj}$$

$$\Downarrow$$

$$\frac{1}{R} = \frac{2 \cdot Y_{obj}}{L^2} \Longrightarrow \gamma = \frac{2 \cdot Y_{obj}}{L^2}$$

Figura 39. Cálculo del ángulo de giro del robot.

Obtenido el ángulo de giro se puede calcular la velocidad angular de la siguiente manera:

$$\omega = v \cdot \gamma$$

Siendo ω la velocidad angular, v la velocidad lineal obtenida desde la función velocidad y γ el ángulo calculado previamente. Sin embargo, para el correcto funcionamiento del seguimiento, se ha tenido que multiplicar la velocidad angular por "-1", haciendo que esta velocidad se quede referenciada correctamente al sistema de ejes del robot.

Calculado todo lo anterior se comprobará mediante la función de esquiva de obstáculos si se encuentra algún objeto o persona delante del robot, debido a que, si se da uno de los casos mencionados en el apartado de dicha función, el robot deberá pararse, haciendo que su velocidad lineal sea igual a 0 m/s, sin embargo, su velocidad angular será igual a 0,3 rad/s o -0,3 rad/s, para poder evitar ese obstáculo y poder continuar el seguimiento, reemplazando de esta forma a las velocidades previamente calculadas.

Por último, en el caso de que la coordenada "y" obtenida del callback anterior sea igual a cero, y que la función localizar tenga un valor true, se empezará a buscar a la persona girando el robot sobre sí mismo a una velocidad angular de 0,1 rad/s o -0,1 rad/s, dependiendo de si el último valor de la coordenada y hubiese sido positivo o negativo respectivamente.

- **Listener o main**

Esta función se encargará de subscribirse a los topic necesarios para el funcionamiento del nodo, siendo uno de estos el publicado por el nodo donde se ejecuta el programa de MediaPipe, el cual contiene tanto la distancia hacia la persona, como la coordenada "x" del landmark definido, estableciéndose como su callback el primero de los explicados anteriormente, mientras que el otro subscriptor obtendrá la información del topic encargado de almacenar los datos recabados por el láser infrarrojo, y asignándole la función laser callback como su callback .

Otra de sus tareas será publicar los valores calculados o asignados tanto de la velocidad lineal como la angular, permitiendo de esta manera el funcionamiento de los motores, sin embargo, se deberá establecer en esta función como valor por defecto de dichas velocidades respecto a cada uno de los ejes 0 rad/s o 0 m/s según corresponda. Aunque a las únicas que se les modificará su valor gracias a los procesos de las funciones previamente detalladas, serán la velocidad lineal respecto al eje "x" del robot, y la velocidad angular respecto al eje "z" del robot. Esta publicación se realizará con una frecuencia de 10 Hz mientras todo funcione de forma correcta, estando este nodo operando de forma ininterrumpida a menos que un usuario lo detenga.

3.3.6 Componentes de la plataforma de la nube

La plataforma en la Nube permitirá almacenar y acceder a la información de tanto los residentes del hospital o residencia en la que se instale este sistema robótico, así como de los auxiliares que están a su cuidado y también de las diferentes actividades que podrán realizar dichos residentes gracias al robot. Para su creación se ha tenido que desarrollar un entorno frontend y backend, los cuales permitirán que esta plataforma opere de forma correcta.

3.3.6.1 Backend

Esta parte constituye el núcleo de la plataforma en la Nube, puesto que es donde se programarán todas las funcionalidades de las que se quiera dotar a la aplicación que se vaya a desplegar. En este caso, se ha utilizado como lenguaje de programación Java, debido a que se ha usado la API Objectify (41) para desarrollar el backend. La estructura de este programa

contiene diferentes funciones que permiten la correcta ejecución de todas las tareas que es capaz de ejecutar la plataforma, las cuales son las que se detallan a continuación:

3.3.6.1.1 Endpoint en la Nube

Esta constituirá la función principal del programa, donde se ubican todas las clases que permiten que se realicen las actividades de las que se encargará la aplicación en la nube, y desde donde se llamarán al resto de funciones que se han creado. En ella se encuentran un total de ocho clases públicas encargadas de desempeñar cada una de las tareas que se han implantado en la plataforma en la nube, siendo estas las siguientes:

- **Almacenar dato usuario**

  Esta clase recibirá como datos tres string, uno será el DNI (Documento Nacional de Identidad) de un residente, otro ha de ser el nombre de este y el último deberá ser la planta de la residencia u hospital donde se ubica la habitación de este. Una vez han sido proporcionadas estas tres variables en forma de string o frase, se comprobará que ninguna tiene un valor nulo o, en otras palabras, que no se le haya asignado ningún valor, luego se verificará que la variable referente a la planta sea un valor numérico, así como que también lo sea la que almacena el DNI, añadiéndole a esta última la constatación de que está formada por ocho dígitos, pues no es necesaria la letra del DNI a la hora de recoger los datos de los residentes puesto que con los valores numéricos es información más que suficiente para poder identificar y diferenciar a cada uno de los usuarios almacenados en la plataforma en la nube. Tras todo lo anterior se procederá a guardar la información del residente o usuario en la base de datos de la plataforma.

- **Almacenar dato auxiliar**

  Esta clase tiene la misma funcionalidad que la anteriormente explicada, salvo que esta recopilará la información del auxiliar, sin embargo, se le atribuirán cuatro variables, siendo estas el DNI del auxiliar o enfermero en cuestión, la planta a la que está asignado, su nombre y su número de teléfono para poder contactar con él en caso de una emergencia. Las comprobaciones serán las mismas que en la clase previa en lo referente al DNI y a la planta, sin embargo, en relación con el número de teléfono se comprobará que todos los dígitos son valores numéricos y que hay un total de nueve. Por último, se almacenará toda la información en la base de datos.

- **Almacenar dato actividad**

A diferencia de las clases previas, esta recibirá siete variables en forma se string. A la primera variable se le ha asignado la definición resumida del contenido de la actividad a almacenar en el sistema, la segunda se referirá al enlace mediante el cual se podrá acceder a ella en caso de que se deba realizar a través de internet, la tercera indicará el estado de la actividad, pudiendo ser este "realizado" o "no realizado", la cuarta ha de contener el DNI del residente o usuario al que está destinada dicha actividad, la quinta ha de almacenar el DNI del auxiliar responsable de dicho residente, la sexta indicará la fecha de inicio de la actividad y por último la séptima tendrá como valor la fecha de finalización de esta.

Una vez se le ha pasado toda la información anterior, se procederá a comprobar que ninguna variable no contenga ningún valor, siendo este paso esencial en todas las clases, para luego verificar que las fechas que se le han atribuido como inicio y final de la actividad se encuentren en el siguiente formato: "uuuu-MM-dd HH:mm:ss", o en otras palabras "año-mes-día hora:minutos:segundos", puesto que si no es de este modo, no se podrá hacer correctamente la conversión de la fecha a timestamp o marca temporal de tipo epoch (segundos transcurridos desde el 1 de enero de 1904 hasta una fecha determinada), puesto que este es el formato mediante el cual se podrán realizar diferentes operaciones con estas fechas. Luego se comprobará la validez de los DNI al igual que se ha hecho en clases previas, para después verificar que esta actividad no se encuentre ya asignada al residente o auxiliar cuyos DNI han sido recibidos por esta clase, si no es el caso, y se ha corroborado de que toda la información percibida es correcta, se procederá a almacenar la actividad en la base de datos.

Para poder identificar y diferenciar cada actividad, esta clase generará automáticamente un identificador para cada una, estando formado por los siguientes elementos: "act-" + "descripción introducida" + "-" + "primeros tres dígitos del usuario que va a realizar esta actividad".

- **Recuperar dato**

Esta clase se encargará de mostrar la información asignada al identificador de un usuario o residente, siendo este su DNI sin letra. Para ello se le deberá pasar el DNI del usuario, como un string, del que se quiera consultar sus datos, los cuales serán mostrados si el identificador que se ha introducido es válido.

- **Recuperar dato auxiliar**

Al igual que la clase anterior, esta clase devolverá la información almacenada en la base de datos, pero esta vez siendo la correspondiente a un auxiliar. Se le deberá pasar como variable el DNI en forma de string del auxiliar en cuestión, ya que este es su identificador, y si esta información es válida la clase devolverá como respuesta los datos asignados a dicho DNI.

- **Recuperar dato actividad**

La función de esta clase es idéntica a las dos detalladas previamente, pues devolverá como respuesta toda la información asignada al identificador de la actividad, el cual se le haya pasado como una variable de tipo string a esta clase. Pudiendo realizar la operación si el identificador es correcto.

- **Listar usuario**

A esta clase se le pasarán como variables tipo string tanto el DNI de un usuario o residente, como el de un auxiliar, pues su finalidad es la de asignar un residente a dicho auxiliar. Para ello, primero se deberá comprobar que ninguna de las variables es nula, tal y como se realiza en todas las clases, luego se examinará que el auxiliar no sea responsable de ese usuario actualmente, por lo que, si no se da el caso y todos los datos provistos son válidos, se procederá a asignar al residente a dicho auxiliar.

- **Delete o eliminar**

Gracias a esta clase se puede eliminar la información de un usuario, auxiliar o actividad de la base de datos. Se le atribuirán dos variables de tipo string a la clase, siendo una el identificador correspondiente de la entidad a eliminar, así como de un valor que permitirá discernir el tipo de entidad que se quiera borrar, pudiendo ser este uno de los siguientes: “usuario”, “actividad” o “auxiliar”. Tras esto se comprobará si el identificador y el tipo de entidad que al que se refiere son válidos, para seguidamente proceder a su eliminación de la base de datos, donde cabe remarcar que si ese dato estaba también ubicado en otra entidad, como puede ser el caso de que se quisiera borrar la información de un usuario que está asignado o enlistado a un auxiliar, tras eliminar a dicho residente de la plataforma en la nube, también desaparecerá su referencia en los datos relacionados con dicho auxiliar. Por último, en el caso de que se eliminen tanto un usuario como un auxiliar que tengan en relación una actividad, esta también será eliminada de la base de datos, puesto que cada actividad solo puede ser destinada a no más un residente y un auxiliar a la vez.

El resto de las funciones que componen el backend de la plataforma en la Nube sirven como almacenamiento temporal de la información recibida antes de ser guardada definitivamente en la base de datos, puesto que este paso intermedio es esencial a la hora de poder trabajar con este tipo de servicio, pues se necesitan reunir todos los datos recibidos por las clases a través del endpoint antes de poder publicar la entidad correspondiente en la nube. También se han desarrollado una serie de funciones cuya finalidad es la de mostrar por pantalla la información relacionada con una entidad, estas funciones poseen el sobrenombre de "Respuesta", teniendo también la tarea de devolver la referencia a cualquier error que haya podido surgir durante la ejecución de este programa. En la Figura 40 se muestra el árbol de proyecto donde se puede apreciar todas las funciones creadas:

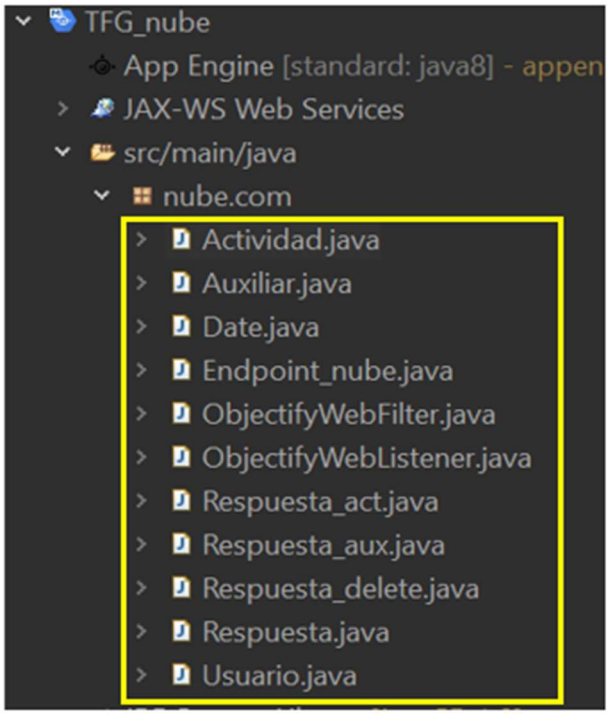


Figura 40. Árbol de proyecto en Eclipse.

Cabe destacar que la función denominada "Date" (Figura 40) no sirve como almacenamiento temporal de ningún tipo de entidad, antes de que esta se guarde en la nube, sino que su propósito es el de comprobar si las fechas que se puedan introducir en forma de variables en alguna de las clases de la función "endpoint" están escritas en el formato correcto, el cual ha sido detallado en el apartado que hace referencia a la clase "Almacenar dato actividad".

3.3.6.2 Frontend

El frontend de una plataforma en la nube comprende tanto la interfaz gráfica, como las diferentes funciones creadas que constituyen la página web mediante la cual una persona puede interactuar con dicha plataforma a través de todas las funcionalidades desarrolladas en el backend de esta. Para programar esta parte de nuestra aplicación o plataforma en la nube se ha utilizado el lenguaje de programación JavaScript, pues se ha optado por el uso de Bootstrap, en concreto su versión 4, a la hora de diseñar la interfaz gráfica de la página web que se ha creado.

Bootstrap es un CSS (Cascading Style Sheets) framework que permite el diseño, personalización y creación de páginas web de forma sencilla utilizando el método CSS, convirtiéndolo en uno de los framework más utilizados hoy en día a la hora de desarrollar entornos frontend.

A la hora de escoger el diseño de nuestra página web se ha tenido muy en cuenta su sencillez, puesto que debe ser lo más transparente posible para que no se puedan dar confusiones cuando se introduzcan datos a la nube o se realice alguna petición a las funcionalidades de las cuales consta la misma. Para ello se ha optado por un diseño mediante el cual se subdivide la página web en ocho apartados bien diferenciados, uno para cada tarea desarrollada en el backend, pudiendo acceder solo a uno de estos a la vez mientras se esté utilizando, impidiendo que se pueda dar el caso de que dos apartados estuvieran abiertos al mismo tiempo, previniendo posibles confusiones a la hora de utilizar este servicio. En la Figura 41 se muestra el diseño de la web:

Figura 41. Interfaz de la página web.

A continuación, se explicarán cada una de las opciones que presentan los apartados de los que se compone la página web:

- **Almacenar datos usuario**

Este apartado consta de tres campos donde se podrá introducir la información necesaria para almacenar en la base de datos de la nube a un residente. Para ello se deberán rellenar dichos campos y pulsar el botón "almacenar usuario", y si todo se ha realizado correctamente, debajo de "Datos del usuario", aparecerá como respuesta de la nube la información almacenada del residente en forma de JSON, tal y como se puede apreciar en la Figura 42:

Figura 42. Almacenar datos usuario.

- **Recuperar datos usuario**

Como su nombre indica, mediante este apartado se podrá acceder a la funcionalidad creada en el backend de la plataforma en la nube que tiene el mismo nombre, mediante la cual se podrá acceder a los datos de los residentes almacenados en la base de datos introduciendo su identificador, siendo este su DNI sin letra, y pulsar el botón "recuperar datos usuario" para recibir como respuesta debajo de "Datos del usuario:", si el DNI es correcto, la información a la que se quiere acceder en forma de JSON. En la Figura 43 se muestra su funcionamiento:

Figura 43. Recuperar datos usuario.

- **Almacenar datos auxiliar**

Este apartado tendrá la misma función que el referente a "Almacenar datos usuario", con la única diferencia de que la información almacenada en este caso será la de un auxiliar. Para realizar este proceso se deberán rellenar los campos establecidos y pulsar el botón "Almacenar Datos Auxiliar", al igual que en el apartado mencionado previamente. Una vez realizado lo anterior, si todos los datos son válidos, se recibirá como respuesta la información almacenada debajo de "Datos del auxiliar:" en forma de JSON, tal y como se muestra en la Figura 44:

Figura 44. Almacenar datos auxiliar.

- **Recuperar datos auxiliar**

En este apartado se podrá acceder a la información guardada en la base de datos de la nube referente a un auxiliar mediante la entrada de su identificador en el campo designado, siendo este su DNI, al igual que ocurría con el apartado "Recuperar datos usuario". Si el DNI es correcto se devolverá como respuesta, debajo de "Datos del auxiliar:", los datos almacenados referentes a ese identificador en forma de JSON, pudiéndose ver esto en la Figura 45:

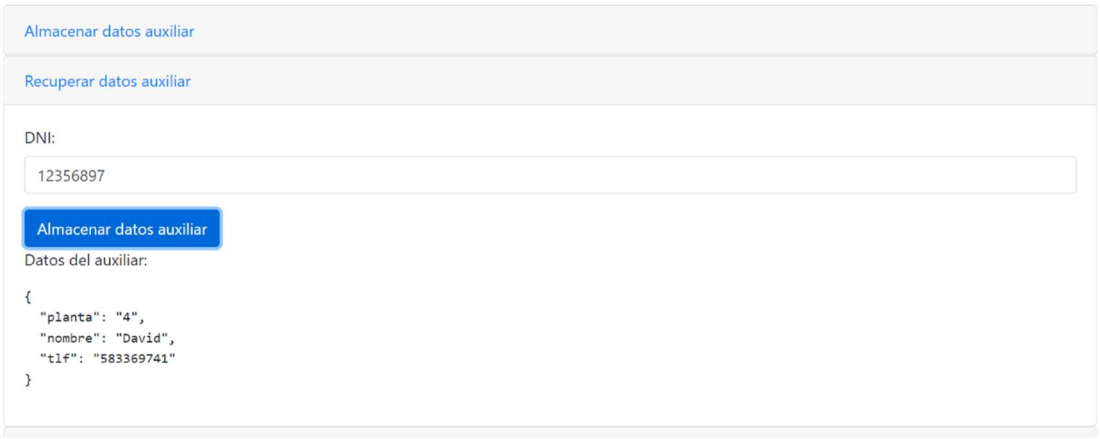


Figura 45. Recuperar datos auxiliar.

- **Almacenar datos actividad**

Tal y como ocurre en los apartados previos donde se almacenaban los datos de los usuarios y los auxiliares, en este se deberá introducir la información en los campos correspondientes referente a la actividad que se desee implementar. Sin embargo, a la hora de introducir las fechas de inicio y final de la actividad, se tendrán que ingresar en el formato que aparece indicado en su correspondiente campo puesto que, si no se realiza de esta manera, no se guardará la actividad en la base de datos, tal y como se detalló en la función "Almacenar dato actividad" del backend. Si todos los datos son válidos una vez pulsado el botón de "Almacenar datos actividad", aparecerá la información introducida en formato JSON debajo de "Datos de la actividad" como respuesta a nuestra petición, tal y como se contempla en la Figura 46:

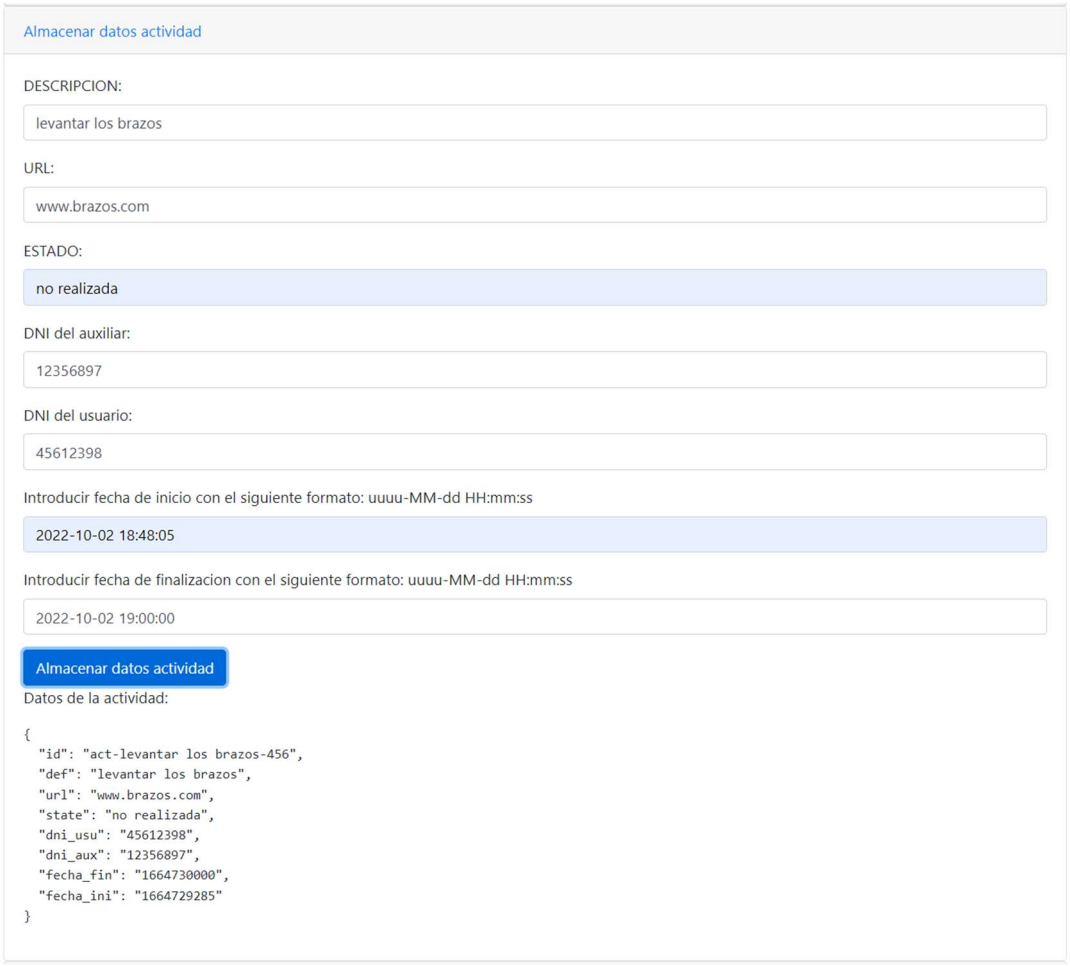


Figura 46. Almacenar datos actividad.

- **Recuperar datos actividad**

En este apartado se podrá acceder a los datos de una actividad almacenada en la plataforma en la nube, al igual que ocurría en los apartados “Recuperar datos usuario” y “Recuperar datos auxiliar” para la información referente a los usuarios y auxiliares respectivamente. En este caso se deberá introducir el identificador de la actividad, siendo este el generado automáticamente por la función “Almacenar datos actividad” desarrollada en el backend. Si el identificador es válido, aparecerá como respuesta a la petición en forma de JSON la actividad almacenada debajo de ”Datos de la actividad:”, tal y como se muestra en la Figura 47:

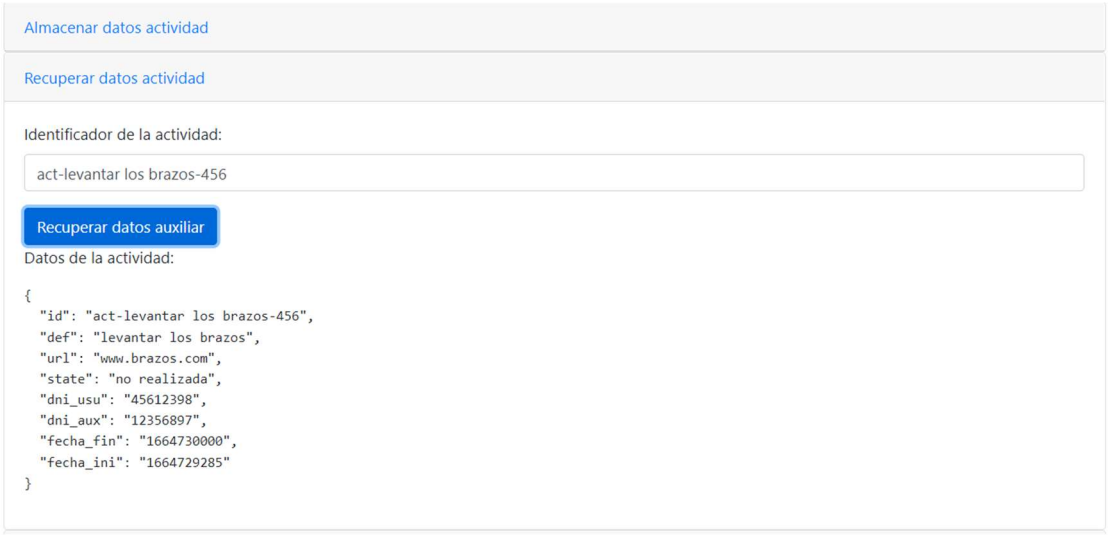


Figura 47. Recuperar datos actividad.

- **Listar usuario**

Este apartado será el encargado de enviar la petición a la nube referente a asignar un usuario o residente a un auxiliar determinado. Para ello se deberán introducir los DNI o identificadores de tanto el usuario como el auxiliar y presionar el botón "Listar usuario", donde si ambos datos son válidos, aparecerá como respuesta, debajo de "Datos del auxiliar actualizados:", la información del auxiliar actualizada, pudiéndose ver que el usuario se encontrará listado en su "lista de usuarios" o "lista_usu", tal y como se aprecia en la siguiente Figura 48:

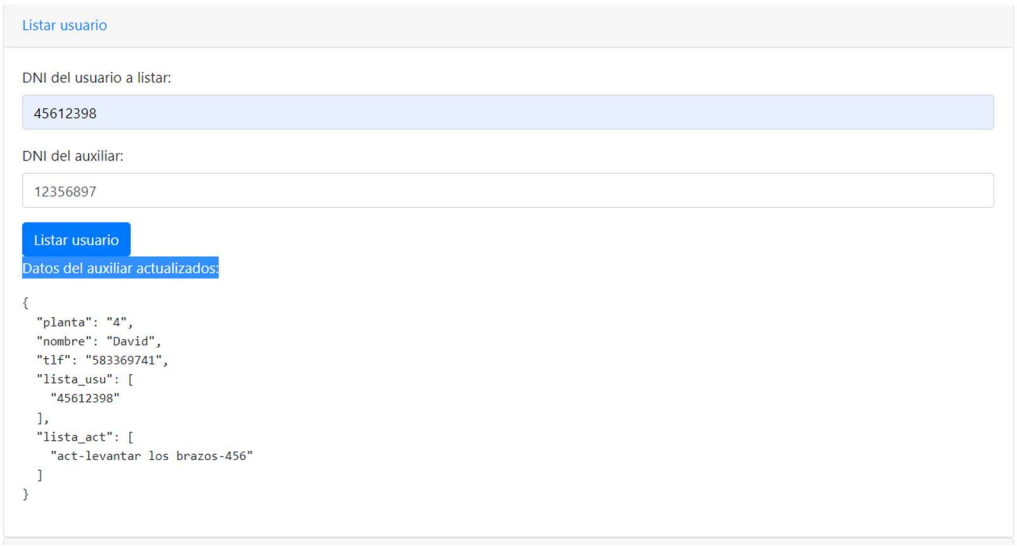


Figura 48. Listar usuario.

- **Borrar entidad**

Mediante este apartado se podrá acceder a la funcionalidad creada en el backend destinada a eliminar un usuario, actividad o auxiliar de la base de datos de la nube. Para realizar esta tarea se deberá introducir el identificador de la entidad que se quiera borrar en el campo correspondiente, así como de escoger en el desplegable "Seleccione el tipo de entidad" qué tipo de dato o entidad hace referencia ese identificador. Una vez realizado lo anterior se pulsará el botón "Borrar", y si los datos introducidos son válidos, aparecerá como respuesta un "0", indicando que se ha eliminado con éxito la información. En la Figura 49 se muestra cómo se borrar un usuario de la base de datos, mientras que en la Figura 50 se puede ver la eliminación de un auxiliar.

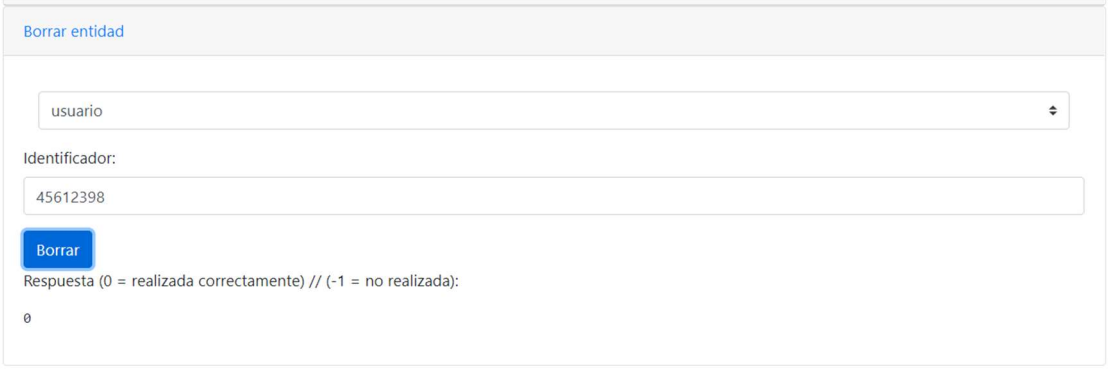


Figura 49. Borrado de la información de un usuario.

Figura 50. Borrado de los datos de un auxiliar.

4 CAPITULO 4

En este capítulo se detallarán tanto las pruebas realizadas para la verificación del correcto funcionamiento de todas las funcionalidades diseñadas y desarrolladas en este proyecto, como los resultados obtenidos tras su realización.

4.1 Pruebas iniciales de ROS y MediaPipe

Las primeras pruebas que se realizaron durante el diseño de este sistema robótico estuvieron relacionadas con la comprobación del correcto funcionamiento de todas las herramientas software explicadas en el Capítulo 3, debido a que todas ellas se utilizarán de forma conjunta, siendo indispensable verificar su funcionalidad de forma individual.

En el caso de ROS, se realizaron varias pruebas sencillas. Una de ellas consistió en que un nodo se subscribiera a un topic proporcionado por la cámara RGB-D gracias al software OpenNI. De esta manera se comprobó que la recepción de datos provenientes del sensor RGB-D mencionado eran correctos.

Otras de las pruebas relacionadas con el entorno de programación ROS, fue la creación de un servicio básico, siguiendo los tutoriales proporcionados por la web oficial de este meta sistema operativo, mediante el cual el cliente se programaría en Python, mientras que el servidor se programaría en C++. Esta prueba se realizó para verificar que las partes que componen un servicio basado en la estructura cliente/servidor en ROS pueden estar basadas en diferentes lenguajes de programación, resultando en la veracidad de este hecho.

Por último, en relación con el entorno ROS, se llevó a cabo una prueba para comprobar que la información publicada desde un nodo les llegara correctamente a los motores del robot, siendo este un chequeo crucial debido a que, si se hubiera producido algún conflicto relacionado con las consignas que reciben los motores, no se habría podido avanzar con el proyecto hasta solventar este problema.

Por otra parte, se realizaron diferentes comprobaciones a la hora de utilizar MediaPipe, donde la primera que se efectuó fue la confirmación de la posibilidad de utilizar este software dentro de un dispositivo cuyo sistema operativo sea Linux Ubuntu, puesto que, si no hubiera sido posible su uso en dicho entorno, se habría tenido que optar por algún otro tipo de software para

detectar a una persona mediante una cámara RGB. Tras verificar que MediaPipe es compatible con Linux Ubuntu, se procedió a realizar un nodo simple de ROS en el cual se incluyera una de las soluciones que ofrece MediaPipe, verificando de esta manera que este es capaz de ser ejecutado bajo el entorno ROS. En esta última prueba se nos presentaron dos complicaciones, las cuales son las siguientes:

- Si se quiere ejecutar un nodo que contenga una solución de MediaPipe, este debe ser programado en Python, dado que de esta manera se podrá compilar/ejecutar fácilmente a través de los comandos definidos en ROS, pudiendo ser parte a su vez de la red de nodos que construida. Al contrario de si se utiliza por ejemplo C++ como lenguaje de programación para dicho nodo, siendo este igualmente compatible dentro de un SO Linux Ubuntu tal y como se menciona en la página web de MediaPipe, pues será necesario instalarse un compilador ajeno a ROS, impidiendo o dificultando sobremanera que se pueda comunicar con el resto de los componentes del sistema.
- Se debe tener muy en cuenta el consumo de recursos por parte de MediaPipe, puesto que se trata de un lenguaje de alto nivel, lo que desemboca en una gran demanda de estos, sobre todo relacionados con la CPU.

4.2 Pruebas iniciales de Eclipse

Respecto a la creación de la plataforma en la Nube, se realizaron unas pruebas iniciales relacionadas con la utilización de Eclipse, donde el lenguaje de programación que se ha utilizado ha sido Java.

La primera prueba que se realizó se basó en el uso del programa Postman, el cual permite realizar peticiones de forma local a través de un servicio o aplicación que se haya creado con la capacidad de enviar o recibir datos desde la Nube. Para que este proceso funcione correctamente primero se debe inicializar desde el símbolo de sistema del ordenador un emulador que reproduzca el funcionamiento de la base de datos de Google Cloud, tal y como se puede apreciar en la Figura 51:

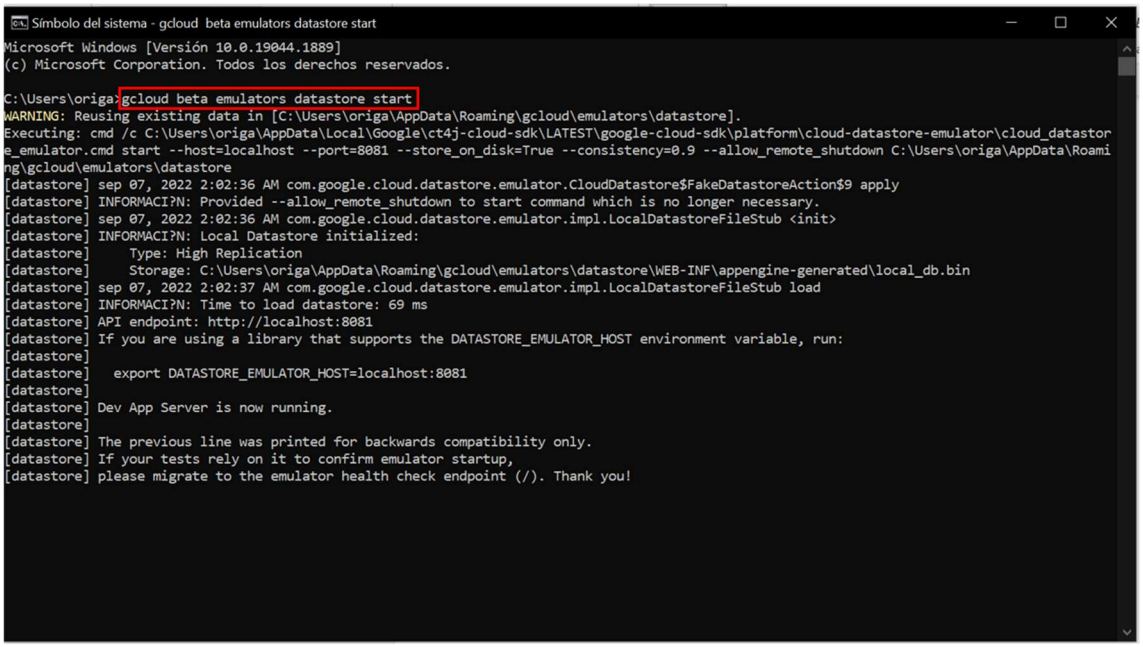


Figura 51. Emulador base de datos de Google Cloud.

Tras la ejecución del emulador, se realizaron diferentes peticiones desde Postman a una sencilla aplicación, creada previamente, para comprobar su correcto funcionamiento.

Una de las ventajas de utilizar Postman como método para realizar diferentes peticiones a la Nube es la nula necesidad de estar conectado a Google Cloud todo el tiempo, con los consiguientes costes económicos que esto puede conllevar debido a que es un servicio de pago, permitiendo de esta manera la verificación de la correcta ejecución de todas las funciones desarrolladas en el backend y frontend de la plataforma en la Nube de forma sencilla y rápida. Sin embargo, este programa puede fallar al intentar abrirlo desde su aplicación, quedándose este con la interfaz en blanco impidiendo realizar ninguna prueba, tal y como se puede ver a en la Figura 52:

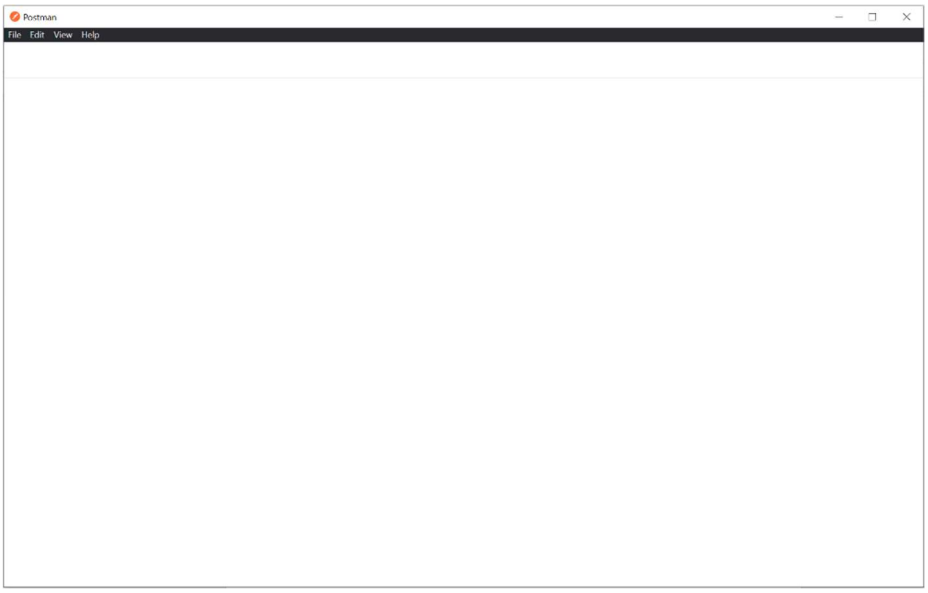


Figura 52. Error al iniciar Postman.

Existe una solución sencilla a este inconveniente, la cual es el uso de Postman (42) desde su página web, aunque se deberá instalar la aplicación Postman Agent para poder realizar peticiones de forma local de esta forma.

Otra de las pruebas que se han realizado referente al envío de peticiones a la misma aplicación anterior, ha sido el envío de información a la base de datos de Google Cloud, verificando de esta manera que también funciona la plataforma o aplicación desarrollada utilizando el servicio de Google, puesto que es un punto crucial, ya que la plataforma en la Nube final deberá poder enviar y recibir información de Google Cloud (Figura 53).

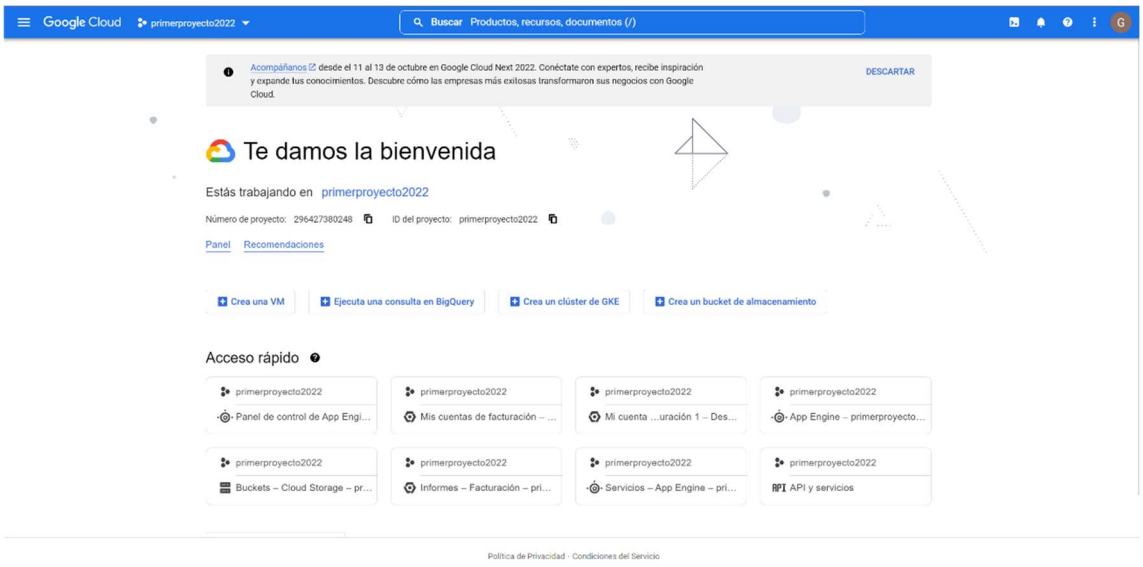


Figura 53. Google Cloud.

4.3 Pruebas integración de elementos hardware y su relación con la arquitectura software

Este es uno de los apartados que más peso ha tenido a la hora de diseñar y desarrollar el sistema, pues el objetivo de este es ser implementado en un robot real y que este realice todas las tareas que se han designado como objetivos de la forma más eficaz y óptima posible.

En un principio se optó por el uso de una Raspberry Pi 4 como dispositivo encargado de ejecutar todos los programas relacionados con la detección, localización y seguimiento de una persona, pero se presentaron diferentes dificultades, cuya procedencia desconcertó en un principio, siendo estos los siguientes:

- Al utilizar una cámara RGB-D, esta genera una gran cantidad de información referente a las imágenes que capta, pues es capaz de generar una nube de puntos de cada uno de los fotogramas tomados, incluso permite la visualización de la imagen de entrada en diferentes espectros y tonalidades, por lo que pese a su baja resolución (640x480 píxeles), es necesario que el dispositivo conectado a ella pueda procesar grandes cantidades de datos en poco tiempo, ya que las imágenes captadas por la cámara, como las Nubes de puntos que genere, tienen que procesarse a la mayor velocidad posible para poder ejecutar el seguimiento de la persona de manera precisa, siendo fatales los retrasos en la transmisión de la información. Por ende, tras numerosos ensayos y pruebas con la cámara conectada a la Raspberry Pi 4 se confirmó que esta introduce unas latencias demasiado elevadas a la hora de transmitir los datos enviados por la cámara.
- Otro de los problemas que se detectaron durante el uso de la Raspberry Pi 4, es su insuficiente potencia computacional a la hora de ejecutar varios nodos de ROS de forma simultánea, dado que tras numerosas pruebas ser cercioró que este dispositivo carece de una CPU suficiente para realizar esta tarea.
- Por último, y en relación con los errores anteriores, se comprobó tras varios ensayos, que la Raspberry Pi 4 es incapaz de ejecutar de forma fluida ninguna solución de MediaPipe a través de un nodo ROS, debido a que surgían retrasos de hasta 15 segundos entre la captación de la imagen por parte de la cámara RGB hasta el procesamiento de

esta mediante el programa que incluye la solución de MediaPipe, haciendo imposible la implementación de un sistema que permita el seguimiento de una persona.

Tras el intento fallido de implementar este sistema róbotico en una Raspberry Pi 4, se intentó poner en marcha en el ordenador que venía instalado en el robot Pionner 3-DX, pero tras varias pruebas se descartó esta idea, pues presentaba los mismos problemas que la Rapberry Pi 4, latencias muy elevadas a la hora de transmitir la información y escasez de potencia a la hora de ejecutar numerosos programas de forma simultánea, esto es altamente probable debido a su antigüedad, siendo esta de unos 10 años.

Finalmente se optó por realizar unas pruebas ejecutando todos los nodos de ROS, incluyendo el que ejecuta la solución Pose de MediaPipe, en un minipc, obteniendo unos resultados más que satisfactorios, puesto que la velocidad de envío y recepción de estos entre los nodos era bastante rápida, mientras que la latencia de transmisión de la información procedente de la cámara RGB a través de los topics generados por el software OpenNI era aceptable, por lo que se acabó tomando la decisión de utilizar este minipc como núcleo o dispositivo de ejecución de los programas encargados del correcto funcionamiento del sistema robótico.

4.4 Pruebas finales del sistema de seguimiento de una persona

Para la comprobación de la correcta ejecución del proceso de detección, localización y seguimiento de una persona mediante este sistema robótico desarrollado, se ha puesto a prueba la realización de estas tareas en dos entornos diferentes, siendo el primero en un pasillo, mientras que el otro se trata de una zona más abierta.

En cuanto a las pruebas realizadas en el pasillo se han dividido en dos subtipos, baja luminosidad (Figura 54) y luminosidad suficiente (Figura 55). En el primero de los casos se ha podido observar como el robot es capaz de detectar a una persona a una distancia no muy amplia, alrededor de unos 2 metros, debido a que, si la persona se aleja demasiado, el robot no será capaz de detectarla, entrando de esta manera en el modo de búsqueda. Aunque se ha comprobado que, si la persona se vuelve a introducir en ese rango de detección, el robot es capaz de retomar el seguimiento. En cambio, si la luminosidad aumenta, siendo este el segundo caso, el robot puede detectar a la persona a una distancia mayor que en la prueba anterior, haciendo que el seguimiento sea más fluido.

Cabe destacar que al utilizar MediaPipe como software mediante el cual localizar a la persona, no se está haciendo uso de la Nube de puntos a la hora de ejecutar esta tarea, provocando errores comprensibles por parte del sistema robótico al intentar discernir correctamente los landmark de la solución Pose, lo que desemboca en detecciones de personas falsas o no válidas, haciendo que momentáneamente se desplace hacia una dirección donde no se encuentra la persona objetivo, pudiendo proceder estos falsos positivos desde el propio reflejo de la persona en una superficie reflectante, hasta objetos o artefactos que según la iluminación MediaPipe pueda llegar a detectar como una persona.

Otro problema común que se ha podido observar es la facilidad con la que el robot puede llegar a perder la referencia de la persona que esté siguiendo si se realiza un cambio brusco en la intensidad de la luz del entorno donde esté operando, pues este fenómeno puede deberse desde la acción de encender o apagar las luces, hasta un cambio de intensidad luminosa como consecuencia de la entrada de luz solar por una puerta o ventana. Este error se debe principalmente a la cámara RGB utilizada, por ende, si se usara otro modelo de cámara RGB este problema podría solventarse.

Figura 54. Prueba pasillo baja luminosidad.

Figura 55. Prueba pasillo mayor luminosidad.

En cuanto a la prueba ejecutada en un entorno más amplio (Figura 56) se ha podido corroborar la mejora del seguimiento del robot al estar desplegado en un lugar más amplio, siendo las condiciones luminosas de este también mejores que las pruebas comentadas previamente. Se ha podido observar como el robot es capaz de seguir a una persona incluso si esta se desplaza rápidamente hacia los ángulos muertos del mismo, gracias a la implementación de una función que permite al robot girar sobre sí mismo para volver a encontrar a la persona, pudiendo percibir su buena capacidad de retomar la marcha ante una pérdida momentánea del objetivo.

Además, se ha podido comprobar cómo el robot es capaz de ignorar a otras personas que se introduzcan en su campo de visión cuando está realizando el seguimiento de otor individuo, siendo esto algo esencial, dado que el objetivo final es introducirlo en un ambiente con un alto flujo de personas. Cabe resaltar que en esta prueba se nos han presentado errores comunes a las realizadas en el pasillo, tales como fallos debidos a los cambios repentinos de luminosidad, así como de detecciones erróneas por parte de MediaPipe.

Figura 56. Prueba en entorno más amplio.

Otro ensayo que se ha efectuado es el referente a la parada del robot cuando se encuentra cerca de la persona, es decir, cuando la ha alcanzado. En esta prueba se ha constatado como este ejercicio lo realiza perfectamente, sin ningún error o fallo detectado, así como a la hora de esquivar obstáculos se ha podido observar como el robot ejecuta esta acción con bastante solvencia, produciéndose un choque con una pared u objeto de forma muy esporádica. En la Figura 57 se puede apreciar como el robot se acerca a la persona hasta detenerse a una distancia prudencial:

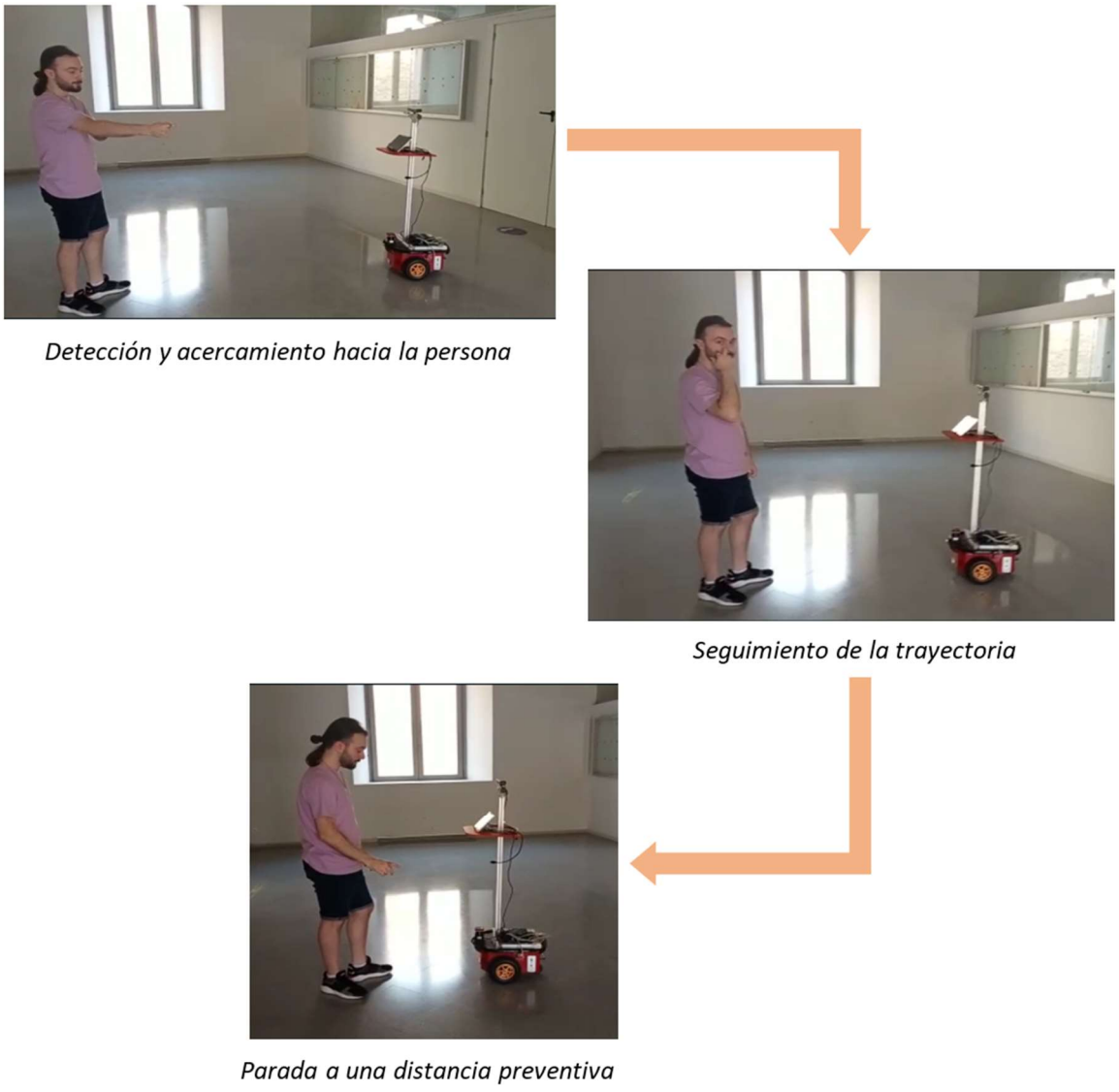


Figura 57. Ensayo de parada a distancia preventiva.

4.5 Pruebas finales de la plataforma en la Nube

Para verificar el funcionamiento apropiado de la plataforma en la Nube, se han realizado múltiples peticiones a esta, abarcando cada una de las opciones creadas, siendo estas las explicadas en el apartado Componentes de la plataforma de la nube. En las siguientes figuras se puede apreciar cómo se reciben las respuestas a las solicitudes a la Nube realizadas, siendo las correctas incluso cuando se trata de recibir el código referente a algún error con la petición.

Tal y como se aprecia en la Figura 58, se recibe un -5, el cual hace referencia a que un campo de la petición está vacío.

Figura 58. Introducción errónea de datos en la página web.

En la Figura 59 se puede observar cómo ante una petición válida de almacenamiento de datos del usuario, se recibe como respuesta la información guardada en la base de datos de la Nube.

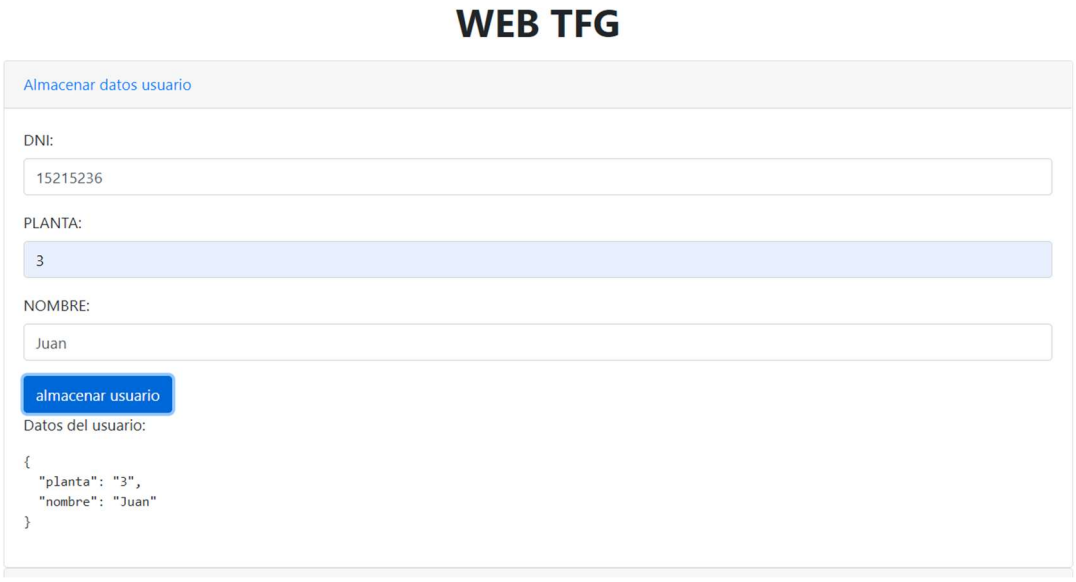


Figura 59. Introducción de datos de usuario.

A continuación, se puede ver en la Figura 60 como al enviar una petición a la Nube relacionada con la eliminación de una entidad existente, se recibe un 0 significando este la correcta ejecución de esta.

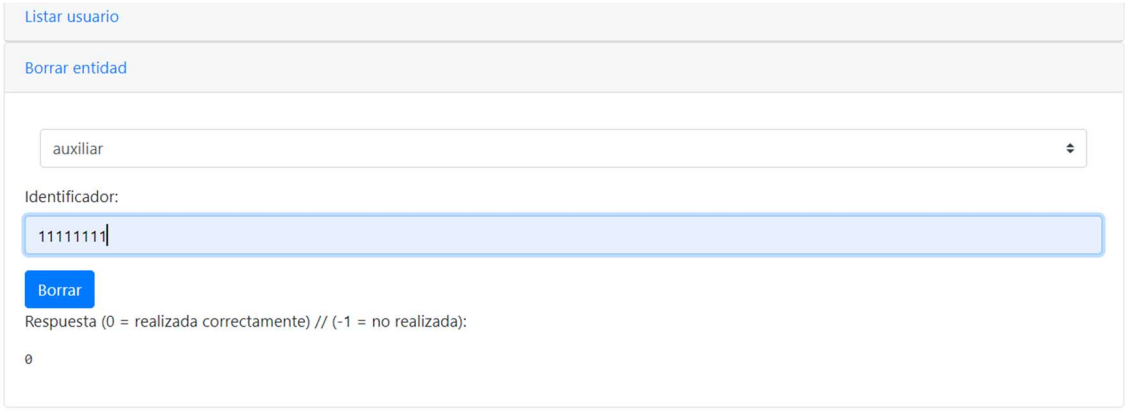


Figura 60. Petición correcta para eliminar entidad.

Pero si se intenta volver a enviar la misma petición, véase la Figura 61, al estar la entidad ya borrada, se recibirá un -1:

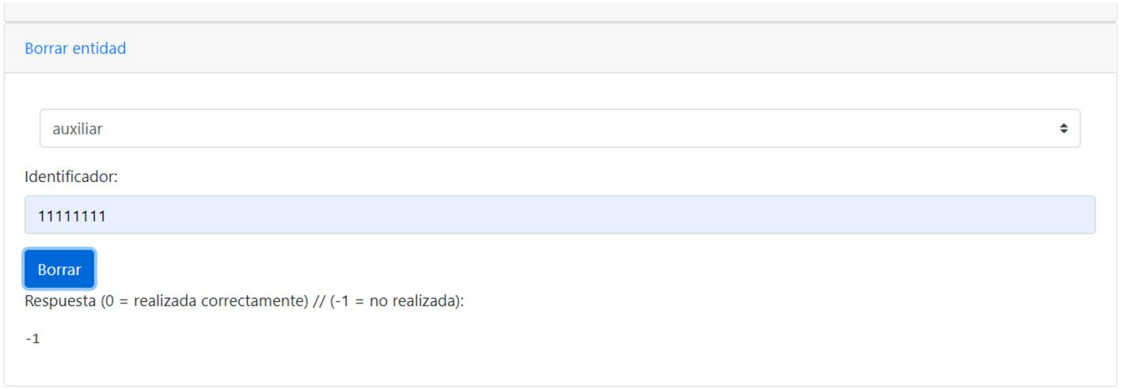


Figura 61. Petición errónea para eliminar una entidad inexistente.

Este proceso de comprobación, de cada una de las diferentes peticiones que se pueden efectuar, se ha realizado chequeando que se reciben las respuestas definidas en el backend a cada tipo petición. Pudiendo constatar el buen funcionamiento del programa referente tanto al backend como al frontend que conforman la plataforma en la Nube diseñada.

4.6 Discusión de las pruebas y resultados obtenidos

A lo largo de este capítulo se han expuesto los diferentes ensayos o pruebas realizadas para comprobar el adecuado funcionamiento de todas las partes que constituyen este proyecto, a su vez se han podido detectar una serie de errores o dificultades en algunos aspectos, algo dentro de lo normal en un sistema robótico, pues en este se deben introducir técnicas y elementos tanto software como hardware muy dispares, teniendo estos que trabajar de forma conjunta y sincronizada, con la capacidad añadida de poder intercambiarse diferentes tipos de archivos y datos.

Se ha podido constatar el buen funcionamiento de las principales tareas que debe ejecutar el robot, tanto la detección, localización como seguimiento de la persona por parte del robot, aunque se han producido algunos errores bajo ciertas circunstancias, las cuales podrían solventarse utilizando otro tipo de software relacionado con el reconocimiento de personas o usando otro tipo de cámara RGB-D con una mayor resolución y especificaciones. También se debe remarcar la buena resiliencia del proceso de seguimiento de una persona cuando se producen situaciones adversas como las explicadas anteriormente, tales como detecciones erróneas o pérdida del "landmark" obtenido por la solución Pose de MediaPipe, designado para la realización de esta tarea.

Además, cabe resaltar la correcta ejecución de todo lo relacionado con la plataforma en la Nube, no habiéndose detectado ningún fallo en lo referente al envío de peticiones, con la consiguiente recepción de la respuesta, ni con el almacenamiento o eliminación de información de la base de datos de la Nube, sea esta de forma local o proporcionada por Google Cloud.

De modo que, a modo de resumen, estos serían los principales beneficios de las herramientas y servicios tanto software como hardware empleados en este proyecto:

- Aunque MediaPipe utilice un lenguaje de programación de alto nivel, esto no impide que se pueda ejecutar en dispositivos con unas prestaciones no muy elevadas, dentro de unos límites, ya que como se ha expuesto en este capítulo, dispositivos como una Raspberry Pi 4 o similares pueden tener problemas a la hora de ejecutar las soluciones que este software proporciona.
- Todos los elementos hardware, tanto la cámara, el dispositivo de barrido Láser 2D, el minipc o el mismo robot, pueden ser mejorados o intercambiados por modelos con

mejores prestaciones de cara a proyectos futuros, aumentando la durabilidad de este sistema robótico, así como su vida útil.

- Al utilizar el entorno de programación ROS para implementar y recoger todas las funciones que permiten el correcto manejo y funcionamiento del robot, posibilita la creación de forma sencilla de nuevos programas, los cuales pueden introducirse fácilmente en un nodo de ROS, añadiendo nuevas funcionalidades a este.
- El uso de Goolge Cloud como base para el despliegue de la plataforma en la Nube, facilita su implantación en entornos como hospitales o residencias, debido a que es muy fácil su uso y puesta en marcha, además, proporciona respuestas a las peticiones realizadas de forma casi inmediata.

Por otro lado, nos encontramos las limitaciones o los problemas que se han encontrado tras la realización de los diferentes ensayos o pruebas detallados anteriormente en este capítulo, estos son los siguientes:

- Uno de los problemas referentes al uso de Google Cloud como entorno donde desplegar nuestra plataforma, es el coste económico que podría conllevar, debido a que no es un servicio gratuito, sin embargo, el sistema de cuotas de Google Cloud permite establecer límites al intercambio de datos que podría facilitar la limitación del coste.
- Al tratarse de un sistema compuesto por numerosos programas que intercambian una gran cantidad de información en poco tiempo, se requiere de un dispositivo lo suficientemente potente para que no se produzcan latencias elevadas en la transmisión de esos datos.
- El uso de MediaPipe para la detección y reconocimiento de una persona conlleva una pérdida de precisión en la realización de este ejercicio, pues este software se basa a la hora de determinar los “landmarks” de las soluciones que proporciona en imágenes sin ninguna nube de puntos asociada, la cual dota de información referente a la profundidad y posición a la que se encuentra cada uno de los puntos detectados por una cámara RGB-D. Todo esto desemboca en fallos centrados en detecciones erróneas, como por ejemplo el no distinguir entre la imagen de una persona proyectada en una superficie y una persona real.
- La utilización de una cámara RGB-D con una baja resolución como la empleada en este proyecto, puede conllevar en la pérdida de la persona a la hora de estar realizando el seguimiento de ésta, pues un cambio de luz brusco o la operatividad del robot en entornos con baja iluminación puede desembocar en diferentes errores en relación con la localización de la persona. Tanto esta limitación, como la mencionada más arriba, pueden solventarse, fusionando la información de profundidad producida por el propio sensor RGB-D, enriqueciendo la solución de MediaPipe.

5 CAPITULO 5

En este capítulo se detallan las conclusiones de este proyecto, así como posibles mejoras que se le podrían implementar a este sistema robótico de cara a futuro.

5.1 Conclusiones

La pandemia provocada por el COVID-19 reveló muchas de las deficiencias a las que se enfrentaban los sistemas sanitarios y residenciales de prácticamente todo el mundo, añadiendo los nuevos problemas provocados por la propia pandemia. De entre los sectores mencionados previamente, el residencial fue uno de los más afectados, pues muchas residencias carecían de los medios y el personal necesario para afrontar las dificultades generadas por esta enfermedad, la cual es mucho más agresiva con personas mayores o aquellas que padezcan patologías previas.

Una de las carencias más repetidas fue la falta de personal en estas residencias, provocando en algunos casos una pobre asistencia a los residentes de dichas instituciones. A todo esto, se tiene que añadir el hecho de que nos encontramos ante una sociedad cada vez más envejecida, por lo que los lugares como las residencias o los servicios de Teleasistencia serán cada vez más comunes. Por lo tanto, este proyecto surge de las necesidades manifestadas ante estos problemas o situaciones y que no están cubiertas en su mayoría o simplemente disponen de un gran margen de mejora.

Durante el transcurso del desarrollo y diseño de este sistema robótico, se han utilizado numerosas herramientas tanto software y hardware, las cuales han permitido la creación de este proyecto, también han proporcionado la capacidad de que se puedan implementar nuevas mejoras de cara a futuras ampliaciones de este sistema, así como también del cumplimiento de los objetivos marcados, siendo estos la detección, localización y seguimiento de una persona, normalmente un auxiliar o enfermero, la comunicación con la nube y también la capacidad de esquivar de obstáculos.

En general, este proyecto ha supuesto un gran desafío tecnológico mediante el cual se espera que pueda contribuir a mejorar la calidad de vida en un futuro de las personas que viven en una residencia o se encuentran en un hospital, además, de intentar aliviar cierta carga de trabajo de los profesionales que trabajan en dichas instituciones.

5.2 Trabajo futuro

Como se ha comentado a lo largo del trabajo, este proyecto tiene un amplio abanico de posibilidades relacionadas con la implementación de nuevas mejoras en este, gracias principalmente a los recursos y servicios tanto software como hardware escogidos para el diseño y puesta en marcha de todo el sistema robótico. Entre las mejoras más relevantes se pueden destacar las siguientes:

- **Detección basada en la nube de puntos**

  Uno de los problemas más comunes que se ha presentado en relación con la localización de una persona, es la falta de información que recaba MediaPipe a la hora de determinar si un ente es una persona o no lo es, desembocando en problemas y dificultades en el momento de determinar los "landmarks" de esta. Por ello, se podría desarrollar un sistema con las herramientas software adecuadas que reconociera a una persona basándose en la nube de puntos asociada con la imagen o fotograma captado por la cámara RGB, solventando de esta forma errores relacionados con falsas detecciones, pues la nube de puntos generada por el cuerpo de una persona es muy particular, haciendo que sea difícilmente confundida con la generada por otro objeto o entidad.

- **Delimitación de una zona segura para el usuario**

  El objetivo final de este robot es su implantación en una residencia u hospital, los cuales son lugares con un gran tránsito de gente, la mayoría personas mayores o en situación delicada, por lo que se podría crear un algoritmo o programa, el cual puede estar basado en algún método relacionado con la Inteligencia Artificial, que sea capaz de recrear virtualmente una zona segura para cada persona, la cual sea interpretada por el robot como un área que no puede atravesar, evitándose de esta forma posibles accidentes provocados por choques entre el robot y una persona, pudiendo estos llegar a causar lesiones a los pacientes que residen es esos lugares.

- **Llamada a un auxiliar**

  Una función que se podría implementar en un futuro sería la capacidad de que un residente, o una persona que lo necesite, pudiera llamar a un auxiliar o enfermero del centro donde se encuentre operando el robot. De esta manera el robot podría actuar como un sistema de comunicación directa entre residentes y auxiliares en caso de una emergencia.

- **Capacidad de carga de objetos**

Algo bastante útil que se podría implementar como un complemento del robot sería un módulo que permita cargar algunos objetos que pueda necesitar el auxiliar o la persona a la que esté siguiendo, o simplemente que pueda trasportar algún tipo de carga no muy pesada de un lugar del edificio donde opere hasta otro punto de éste.

6 Bibliografía

1. Mediavilla, Manu. AMNISTÍA INTERNACIONAL. [En línea] 3 de Diciembre de 2020. https://www.es.amnesty.org/en-que-estamos/reportajes/residencias-en-tiempos-de-covid-personas-mayores-abandonadas-a-su-suerte/.

2. OMS. Discapacidad y salud. [En línea] 24 de Noviembre de 2021. https://www.who.int/es/news-room/fact-sheets/detail/disability-and-health.

3. ROS. [En línea] ROS.org. https://wiki.ros.org/es/ROS/Introduccion.

4. MediaPipe. MediaPipe. [En línea] https://mediapipe.dev/.

5. Teleasistencia Vital. [En línea] Teleasistencia Vital. https://teleasistenciavital.com/blog/tipos-de-teleasistencia/.

6. Katie Winkle, Praminda Caleb-Solly, Ailie Turton y Paul Bremner. Mutual Shaping in the Design of Socially Assistive Robots: A Case Study on Social Robots for Therapy. [En línea] 7 de Marzo de 2019. https://link.springer.com/article/10.1007/s12369-019-00536-9.

7. Angelo Costa, Ester Martinez-Martin ,Miguel Cazorla y Vicente Julian. PHAROS—PHysical Assistant RObot System. [En línea] 11 de Agosto de 2018. https://www.mdpi.com/1424-8220/18/8/2633/htm.

8. Glickman, Professor Mark E. Example of the Glicko-2 system. [En línea] 22 de Marzo de 2022. http://www.glicko.net/glicko/glicko2.pdf.

9. Ginés Hidalgo, Zhe Cao, Tomas Simon, Shih-En Wei, Yaadhav Raaj, Hanbyul Joo, y Yaser Sheikh. OpenPose. [En línea] https://github.com/CMU-Perceptual-Computing-Lab/openpose.

10. Bajones, Markus. Hobbit: Providing Fall Detection and Prevention for the Elderly in the Real World. [En línea] 2018. https://www.hindawi.com/journals/jr/2018/1754657/.

11. CORDIS. Robotic Assistant for MCI patients at home. [En línea] https://cordis.europa.eu/project/id/643433/es.

12. Ioannis Kostavelis, Dimitrios Giakoumis, Georgia Peleka, Andreas Kargakos, Evangelos Skartados, Manolis Vasileiadis y d Dimitrios Tzovaras. RAMCIP Robot: A Personal Robotic Assistant;. [En línea] 2018. https://openaccess.thecvf.com/content_ECCVW_2018/papers/11134/Kostavelis_RAMCIP_Robot_A_Personal_Robotic_Assistant_Demonstration_of_a_Complete_ECCVW_2018_paper.pdf.

13. Kompaï R&D. [En línea] Kompaï robotics. https://kompai.com/robot-for-research-and-development/.

14. Z. Zenn Bien, Kwang-Hyun Park, Won-Chul Bang y Dimitar H. Stefanov. LARES: An Intelligent Sweet Home for Assisting the Elderly and the Handicapped. [En línea] Diciembre de 2003. https://www.researchgate.net/profile/Zeungnam-Bien/publication/2946879_LARES_An_Intelligent_Sweet_Home_for_Assisting_the_Elderly_an

d_the_Handicapped/links/550139c10cf2aee14b58ee47/LARES-An-Intelligent-Sweet-Home-for-Assisting-the-Elderly-and-the-Handicapp.

15. Parorobots. Parorobots. [En línea] 2014. http://www.parorobots.com/.

16. Petersen, Sandraa, y otros. The Utilization of Robotic Pets in Dementia Care. [En línea] 18 de Agosto de 2016. https://content.iospress.com/articles/journal-of-alzheimers-disease/jad160703#jad-55-jad160703-t001.

17. Raspberry Pi 4. [En línea] Raspberry Pi. https://www.raspberrypi.com/products/raspberry-pi-4-model-b/.

18. Nock, Charles A. Asus Xtrion Pro Live Specifications. [En línea] 2013. https://www.researchgate.net/figure/Specifications-and-comparison-of-Asus-Xtion-Pro-Live-and-Microsoft-Kinect_tbl1_259002786.

19. OpenNI (ROS). [En línea] ROS.org. https://wiki.ros.org/openni_launch.

20. HOKUYO. HOKUYO UTM-30LX. [En línea] https://www.hokuyo-aut.jp/search/single.php?serial=169#spec.

21. Adept. Pionner 3-DX. [En línea] 2011. https://www.generationrobots.com/media/Pioneer3DX-P3DX-RevA.pdf.

22. Inc, MobileRobots. Pioneer 3 Operations Manual. [En línea] 2006. https://www.inf.ufrgs.br/~prestes/Courses/Robotics/manual_pioneer.pdf.

23. Visual Studio Code. [En línea] Microsoft . https://code.visualstudio.com/.

24. ROS Totorials Publisher and Subscriber (C++). [En línea] ROS.org. https://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber(c++).

25. ROS Tutorials Publisher and Subscriber (Python). [En línea] ROS.org. http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28python%29.

26. ROS Services. [En línea] ROS.org. http://wiki.ros.org/Services.

27. ROS roscpp. [En línea] ROS.org. http://wiki.ros.org/rospy/Overview/Services.

28. ROS rospy. [En línea] ROS.org. http://wiki.ros.org/rospy.

29. OpenNI 2. [En línea] Structure. https://structure.io/openni.

30. MediaPipe Solutions. [En línea] MediaPipe. https://google.github.io/mediapipe/solutions/solutions.

31. MediaPipe Pose. [En línea] MediaPipe. https://google.github.io/mediapipe/solutions/pose.html.

32. Grishchenko, Valentin Bazarevsky y Ivan. MediaPipe with BlazePose. [En línea] 13 de Agosto de 2020. https://ai.googleblog.com/2020/08/on-device-real-time-body-pose-tracking.html.

33. Eclipse . [En línea] Eclipse Foundation. https://www.eclipse.org/downloads/.

34. Google Cloud. [En línea] Google. https://cloud.google.com/gcp/?hl=es&utm_source=google&utm_medium=cpc&utm_campaign=emea-es-all-es-bkws-all-all-trial-e-gcp-1011340&utm_content=text-ad-none-any-DEV_c-CRE_593880918212-ADGP_Hybrid%20%7C%20BKWS%20-%20EXA%20%7C%20Txt%20~%20GCP%20~%20General%23.

35. Cloud Storage. [En línea] Google. https://cloud.google.com/storage/docs/introduction?hl=es-419.

36. ROS Tutorials Service and Client. [En línea] ROS.org. https://wiki.ros.org/ROS/Tutorials/WritingServiceClient%28c%2B%2B%29#roscpp_tutorials.2FTutorials.2FWritingServiceClient.Building_your_nodes.

37. sensor_msgs/CameraInfo Message. [En línea] https://docs.ros.org/en/noetic/api/sensor_msgs/html/msg/CameraInfo.html.

38. ROS message_filters. [En línea] ROS.org. https://wiki.ros.org/message_filters.

39. OpenCV. [En línea] OpenCV.org. https://opencv.org/.

40. sensor_msgs/LaserScan Message. [En línea] https://docs.ros.org/en/lunar/api/sensor_msgs/html/msg/LaserScan.html.

41. Objectify. [En línea] Google App Engine . https://github.com/objectify/objectify/wiki.

42. Postman. [En línea] Postman, Inc. https://www.postman.com/.

43. Dimas, Alicia Martínez. Fundación Alzheimer España. [En línea] 22 de Septiembre de 2015. http://www.alzfae.org/fundacion/459/teleasistencia-que-es-en-que-consiste-como-contratarlo#:~:text=La%20Teleasistencia%20es%20un%20servicio,cualquier%20situaci%C3%B3n%20de%20necesidad%20o.

7 Anexos

7.1 Anexo I. Instalación de ROS

Para poder instalarse este servicio primero se deberá escoger el sistema operativo en el que se vaya a instalar, puesto que si se usa Windows o MacOS se tendrá que trabajar a través de una máquina virtual, debido a que ROS opera bajo el sistema operativo Linux Ubuntu. Por lo tanto, lo más recomendable es trabajar bajo Linux Ubuntu, a la hora de utilizar ROS, ahorrándonos de esta manera errores y fallos de compatibilidad.

A continuación, se detallarán los pasos a realizar para instalar correctamente ROS, todo este proceso se llevará a cabo desde la terminal de Linux Ubuntu:

Primero tendremos que actualizar nuestras "sources.list", mediante el siguiente comando:

```
Gabriel@nombrePC:~$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu
$(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```

Luego se deberá instalar Curl de la siguiente manera:

```
Gabriel@nombrePC:~$ sudo apt install curl
```

Instalado Curl, se tendrán que configurar sus claves:

```
Gabriel@nombrePC:~$ curl -s
https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo apt-
key add -
```

Después de haber realizado lo anterior, se necesitarán actualizar los repositorios a través del siguiente comando:

```
Gabriel@nombrePC:~$ sudo apt update
```

Por último, se procederá a instalar la versión completa de ROS:

```
Gabriel@nombrePC:~$ sudo apt install ros-noetic-desktop-full
```

Tras realizar los pasos anteriores ya tendremos ROS instalado en nuestro sistema operativo Linux Ubuntu.

7.2 Anexo II. Instalación de MediaPipe

En este apartado se detallará cómo realizar la instalación de MediaPipe en el sistema operativo Linux Ubuntu utilizando Python como lenguaje de programación para MediaPipe. Para ello se realizarán los siguientes pasos:

Primero se deberá instalar Python, en concreto su versión 3, en nuestro entorno Linux Ubuntu:

```
Gabriel@nombrePC:~$ sudo apt install python3
```

Para comprobar que se ha instalado correctamente, introduciremos el siguiente comando:

```
Gabriel@nombrePC:~$ python3 --version
```

Tras esto, se actualizarán los repositorios antes de proseguir con la instalación:

```
Gabriel@nombrePC:~$ sudo apt update
```

Luego se necesitará instalar OpenCV para que MediaPipe funcione correctamente, esto se realizará con el siguiente comando:

```
Gabriel@nombrePC:~$ sudo apt install python3-opencv
```

Después se procederá a instalar MediaPipe con el siguiente comando:

```
Gabriel@nombrePC:~$ pip3 install mediapipe
```

Por último, como el objetivo es utilizar MediaPipe en conjunto con ROS, se tendrá que instalar el servicio de ROS denominado “rospy”, debido a que gracias a este se podrán programar nodos

de ROS utilizando Python como lenguaje de programación. La instalación se realizará a través del siguiente comando:

```
Gabriel@nombrePC:~$ sudo apt-get install python-rospy
```

Con esto ya estaría instalado Mediapipe en nuestro entorno de trabajo, listo para poder ser implementado en un nodo de ROS.

7.3 Anexo III. Instalación de Eclipse

Para instalarse el software Eclipse en nuestro equipo se deberá visitar la siguiente página web:

https://cloud.google.com/eclipse/docs

Una vez en ella, en el apartado Comenzar se seleccionará la opción Guía de inicio rápido, tal y como puede verse en la Figura 62:

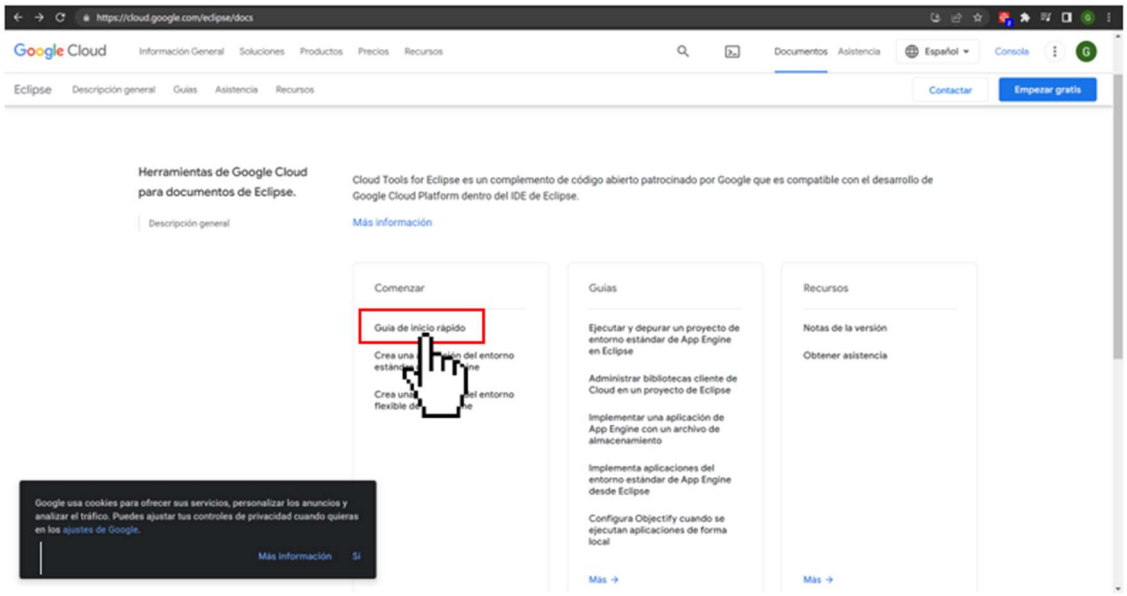


Figura 62. Guía de Inicio rápido.

Dentro de Guía de inicio rápido se encontrarán los pasos a seguir para instalar Eclipse en nuestro ordenador. Lo primero que se realizará será pulsar en la opción Descargar Eclipse para descargar el programa, véase la Figura 63.

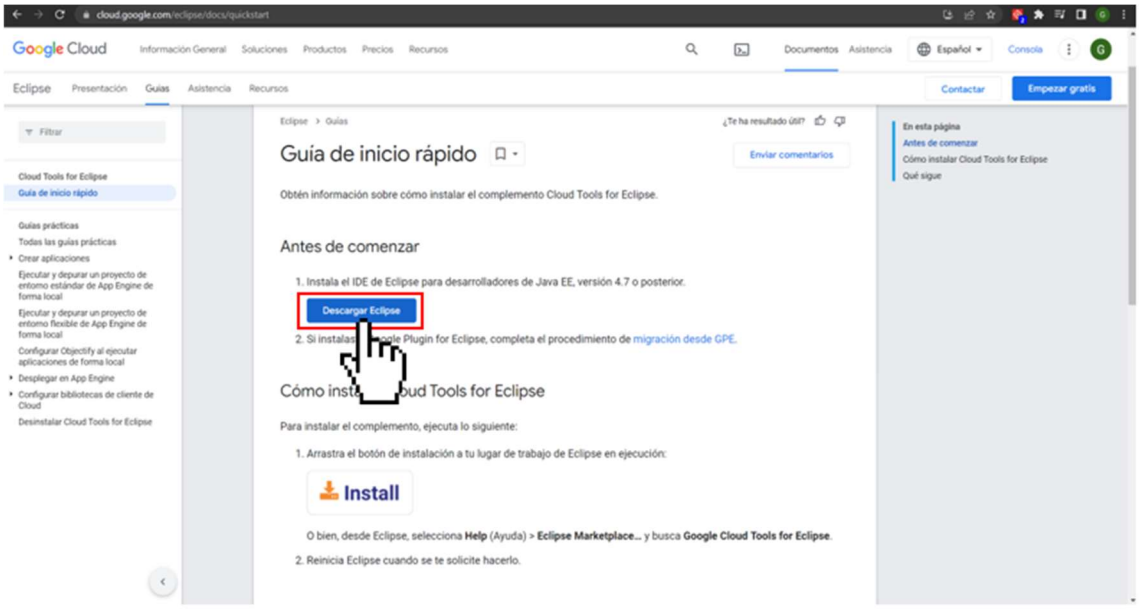


Figura 63. Botón descarga Eclipse.

Una vez descargado e instalado el programa, se deberá instalar el software JDK 8 (Java Development kit 8), tal y como se indica en la Guía de inicio rápido. Cuando todos los pasos anteriores han sido realizados, se tendrán que instalar las herramientas para la nube o Cloud Tools para Eclipse, para realizar este proceso se arrastrará el botón Install que aparece en el apartado Cómo instalar Cloud Tools for Eclipse hacia nuestro lugar de trabajo de Eclipse, tal y como se puede apreciar en la Figura 64:

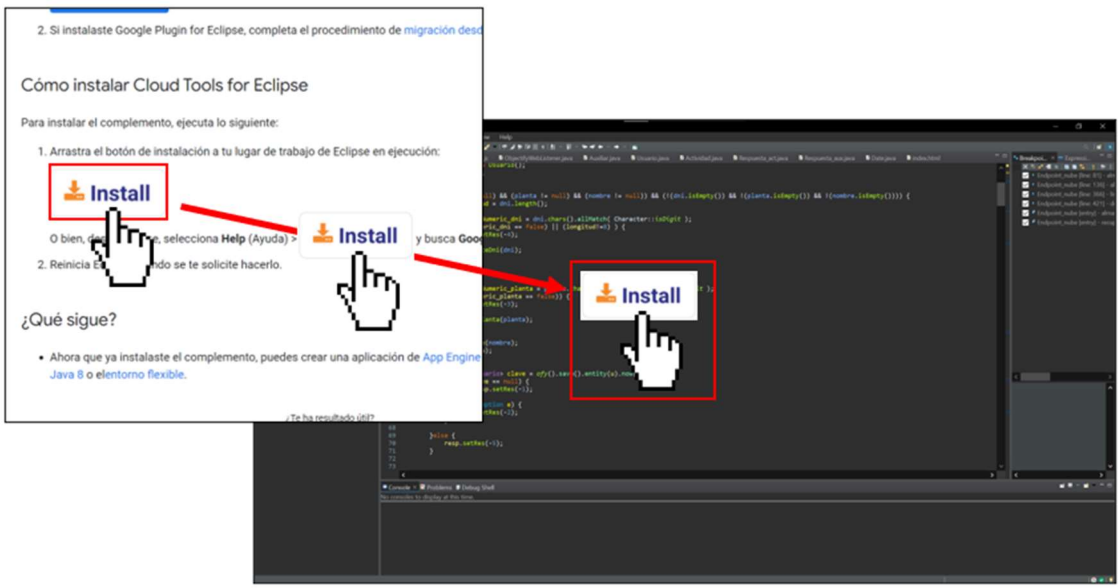


Figura 64. Instalación Cloud Tools.

Tras el paso previo, se añadirá el comando gcloud a las variables de entorno de nuestro equipo, este se encuentra en la siguiente ubicación:

```
C:\Users\NOMBRE DE USUARIO\AppData\Local\Google\ct4j-cloud-
sdk\LATEST\google-cloud-sdk\bin\gcloud
```

Por último, si todo está correctamente instalado debería aparecer el icono de Google Cloud en la barra de herramientas de nuestro entorno de trabajo de eclipse, pudiéndose ver esto en la Figura 65:

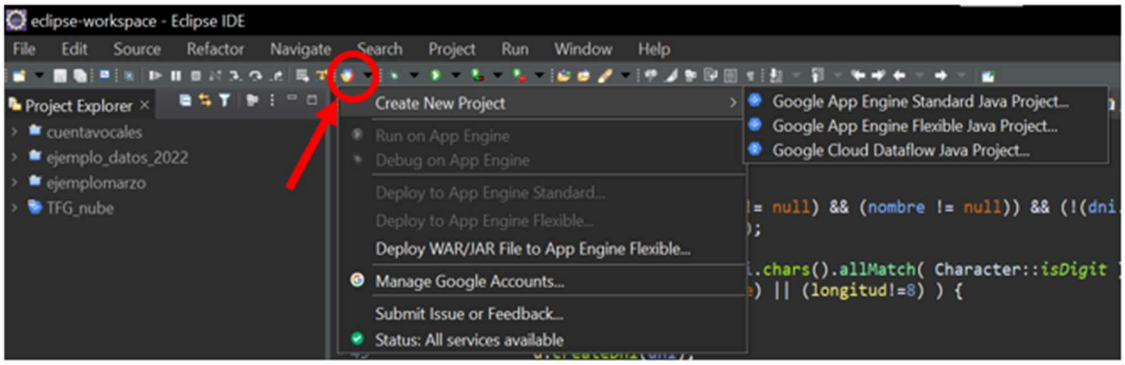


Figura 65. Comprobación de la correcta instalación.