



Escuela
Politécnica
Superior

Teleoperación de un robot mediante reconocimiento de gestos



Grado en Ingeniería Robótica

Trabajo Fin de Grado

Autor:

José Carlos Penalva Maciá

Tutor/es:

Miguel Ángel Cazorla Quevedo

Félix Escalona Moncholí

Mayo 2024



Universitat d'Alacant
Universidad de Alicante

Teleoperación de un robot mediante reconocimiento de gestos

Autor

José Carlos Penalva Maciá

Tutor/es

Miguel Ángel Cazorla Quevedo

Departamento de Ciencia de la Computación e Inteligencia Artificial

Félix Escalona Moncholí

Departamento de Ciencia de la Computación e Inteligencia Artificial



Grado en Ingeniería Robótica



Escuela
Politécnica
Superior



Universitat d'Alacant
Universidad de Alicante

ALICANTE, Mayo 2024

Preámbulo

La impulsión para llevar a cabo este proyecto se origina en el deseo de poner en práctica y ampliar los conocimientos obtenidos durante mi formación en Ingeniería Robótica, complementados con las enseñanzas aportadas por mis tutores en este proyecto. De esta manera, se pretende afrontar el desafío de integrar conocimientos básicos con tecnologías más avanzadas para facilitar el trabajo de futuras ampliaciones de proyectos relacionados en este ámbito.

Agradecimientos

Este proyecto no habría visto la luz sin el respaldo inquebrantable de mi familia, -padres, hermanos y abuelos-, que desde mi infancia me enseñaron la importancia de no renunciar a nuestras pasiones y me brindaron su apoyo incondicional frente a todos los desafíos que la vida me ha presentado. Les agradezco también por su comprensión en los momentos difíciles, siendo un pilar esencial en el desarrollo de este proyecto.

En segundo lugar, quiero expresar mi gratitud tanto a Félix Escalona como a Miguel Ángel Cazorla, del Departamento de Ciencia de la Computación e Inteligencia Artificial, quienes han sido fundamentales al proporcionarme las tecnologías y conocimientos necesarios para resolver cualquier duda a lo largo de todo el proyecto.

Finalmente, quiero expresar mi agradecimiento a todos mis amigos y compañeros que han demostrado interés y brindado su apoyo, contribuyendo de manera significativa para que este proyecto se llevara a cabo de la mejor forma posible.

*A mis padres, María Asunción Maciá y Jesús Carlos Penalva,
a mis hermanos, Jesús Penalva y María Penalva,
y a mis abuelos, María Asunción, María Teresa y José Antonio,
por ayudarme a mejorar cada día un poco más.*

*Lo que conocemos es una gota,
lo que no conocemos es el océano.*

Isaac Newton.

Índice general

1	Introducción	1
2	Marco Teórico	3
3	Objetivos	9
4	Metodología	11
4.1	Python	11
4.2	Math	11
4.3	OpenCV	11
4.4	Mediapipe	12
4.5	Tkinter	13
4.6	ROS	13
4.6.1	Rospy	14
4.7	Gazebo	14
4.8	SLAM	15
4.8.1	Gmapping	16
4.9	TurtleBot 3	16
5	Desarrollo	19
5.1	Primera versión	20
5.2	Segunda versión	22
5.3	Tercera versión	23
5.4	Mapeado	27
5.5	Cámara Turtlebot3	27
5.6	Interfaz	28
6	Resultados	31
6.1	Entornos de simulación	31
6.2	Experimentación	32
6.3	Resultados obtenidos	33
7	Conclusiones	39
	Bibliografía	41
	Lista de Acrónimos y Abreviaturas	43

Índice de figuras

1.1	Visión general de un sistema telerobotizado	2
2.1	Pipeline detección de gestos en "Conduct-a-Bot"	4
2.2	Guante para detección de gestos	5
2.3	Visión por computador	6
2.4	Detección de mano	7
4.1	Puntos de las manos detectables por mediapipe	12
4.2	Logo oficial ROS	14
4.3	Simulación en Gazebo	15
4.4	Mapeado con Gmapping	16
4.5	Turtlebot3 Burger	17
5.1	Esquema visual del proyecto	19
5.2	Estructura general control del robot	20
5.3	Gestos para una mano	21
5.4	Gestos Mano Izquierda	22
5.5	Puntos utilizados para el cálculo de ángulos	24
5.6	Rango láser TutleBot3	25
5.7	Ejemplo Detección de obstáculos	26
5.8	Cámara Kinect	28
5.9	Árbol de transformaciones	29
5.10	Selección de mapa	29
5.11	Cuadro de Texto interfaz	30
6.1	Entorno House	31
6.2	Vista Superior House	32
6.3	Cámara del TurtleBot3	36
6.4	Mapeado Completo	37
6.5	Interfaz Final	38

Índice de cuadros

6.1 Gestos detectados en la mano derecha	34
6.2 Gestos detectados en la mano izquierda	35

1 Introducción

El proyecto ejecutado tiene como objetivo integrar diversas tecnologías de vanguardia. Por consiguiente, es fundamental mencionar los conceptos de "Telerrobótica" y "Visión por Computador" para comprender el funcionamiento de las distintas aplicaciones diseñadas para cumplir nuestro objetivo.

La telerrobótica fusiona los principios de "teleoperación" y "robótica". La teleoperación comprende los elementos esenciales para manipular dispositivos a distancia por el operador. De esta manera, la telerrobótica se concentra en el control remoto de robots que, debido a diversas barreras, están distantes del operador. Este campo posibilita la ejecución de tareas en entornos peligrosos, inaccesibles o desafiantes para los humanos.

Dado que este proyecto se centra en el desarrollo de sistemas para teleoperar robots, es relevante destacar los componentes fundamentales que debe incluir cualquier sistema de teleoperación, y por ende, cualquier sistema de telerrobótica. El primero de estos componentes es el sistema maestro, responsable de controlar un sistema esclavo al enviarle las acciones ordenadas por un operador. En este proyecto, el sistema maestro consiste en una aplicación que detecta diversos gestos de las manos del operador para enviar la información deseada al sistema esclavo. El segundo elemento es el sistema esclavo, un dispositivo controlado remotamente que ejecuta las acciones indicadas por el sistema maestro y el operador. Por último, encontramos las barreras, elementos que separan el entorno donde opera el maestro del entorno donde el esclavo lleva a cabo las tareas asignadas. Estas barreras serán de un tipo u otro en función del entorno donde el sistema esclavo esté llevando a cabo sus tareas. En la Figura 1.1 podemos ver la estructura general de un sistema teleoperado.

Respecto a la visión por computador podemos decir que hace referencia a un grupo de tecnologías o herramientas que permiten a los equipos captar imágenes del mundo real, procesarlas y generar información a través de ellas. Esta tecnología se puede utilizar para tareas como detección de objetos, análisis de imágenes o videos y comparación estadística. A lo largo de este Trabajo Final de Grado (TFG) se hará uso de esta tecnología para la detección de las manos y la posterior clasificación de los gestos.

Como se ha mencionado uno de los usos principales de la visión por computador es la detección y clasificación de objetos en imágenes o en tiempo real. Actualmente, con la mejora de las diferentes tecnologías como librerías para la clasificación o la Inteligencia Artificial (IA), se ha conseguido mejorar la utilidad de la clasificación de imágenes en la visión por computador, reduciendo drásticamente el tiempo de procesamiento necesario. Estas mejoras han permitido reducir significativamente las imprecisiones que aparecían al utilizar esta tecnología, además de reducir el coste computacional de la visión por computador, permitiendo

a todos los operadores hacer uso de ella sin recibir retrasos en la ejecución.

Finalmente, a lo largo de este TFG, se tratará de informar acerca del proceso de investigación y desarrollo de los diferentes objetivos. En primer lugar, en el Capítulo 2 se mostrarán diferentes trabajos ya realizados que se relacionan con el tema principal. El Capítulo 3 expondrá los objetivos a cumplir durante la realización de este proyecto. En el Capítulo 4 se apuntarán todas las herramientas necesarias para cumplir los objetivos del capítulo anterior. A continuación, en el Capítulo 5, describirá la utilidad de cada herramienta para las aplicaciones creadas, además de mostrar las características más concretas del proyecto. En el Capítulo 6, se expondrán los resultados obtenidos y las diferentes pruebas que se han realizado con el objetivo de mejorar estos resultados. Más tarde, en el Capítulo 7, se extraerán conclusiones basadas en los resultados obtenidos a lo largo del estudio. Finalmente, se mostrará la bibliografía necesaria para realizar el proyecto junto con los acrónimos y abreviaturas.

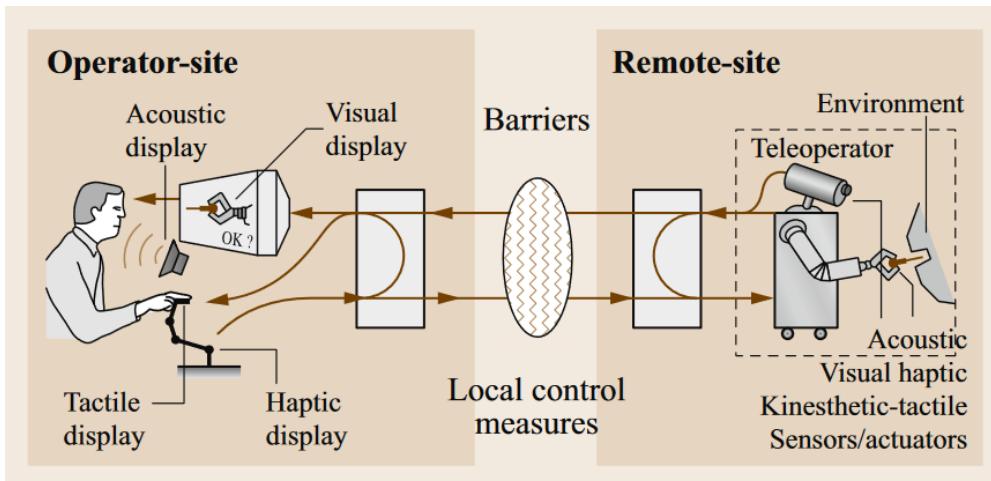


Figura 1.1: Visión general de un sistema telerobotizado
Obtenida de [1]

2 Marco Teórico

A lo largo de los años, el avance en las tecnologías de detección de gestos, la comunicación con diversos tipos de robots y la interacción humano-robot ha transformado profundamente el campo de la teleoperación. Este proceso de evolución ha convertido a la teleoperación en un área de investigación cada vez más atractiva en el ámbito de la ingeniería robótica. La capacidad de controlar robots a distancia utilizando gestos naturales y comandos intuitivos representa un avance significativo en la interacción entre humanos y máquinas, con importantes implicaciones en una variedad de aplicaciones prácticas, desde la asistencia en tareas industriales hasta la exploración espacial y la atención médica. En esta sección del marco teórico, exploraremos las tecnologías y conceptos clave que han impulsado el desarrollo de sistemas de teleoperación basados en gestos, así como las investigaciones previas y los avances en este campo que han sentado las bases para el presente proyecto.

El primer proyecto que encontramos relacionado con nuestro proyecto es el creado por Joseph DelPreto y Daniela Rus, [2], miembros del equipo del Laboratorio de Informática e Inteligencia Artificial del MIT (CSAIL). En este proyecto se busca crear un sistema capaz de teleoperar una variedad de robots, desde drones hasta Roombas, mediante el estímulo muscular sin necesidad de utilizar mandos a distancias, simplemente con el movimiento de los brazos.

Este sistema, llamado "Conduct-a-Bot", [3], hace uso de sensores de músculos wearables situados en los bíceps, tríceps y antebrazo del usuario. El sistema actual detecta 8 gestos de navegación predefinidos sin necesidad de calibración o entrenamiento sin conexión. Esto permite a cualquier usuario empezar a controlar drones nada más colocarse los distintos sensores simplemente moviendo los distintos músculos del brazo.

Los sensores musculares, llamados sensores de Electromiografía (EMG), se usan en el bíceps y el tríceps para detectar cuándo están tensos los músculos de la parte superior del brazo. También se lleva en el antebrazo un dispositivo inalámbrico con EMG y sensores de movimiento.

En los experimentos actuales, se utilizaron placas de procesamiento MyoWare con electrodos Covidien y un dispositivo de adquisición de datos de NI para transmitir la actividad de bíceps y tríceps. Se utilizó el brazalete Myo Gesture Control para monitorear la actividad del antebrazo. En el futuro, podrían sustituirse sensores y dispositivos de adquisición alternativos.

Los canales de aprendizaje automático procesan las señales musculares y de movimiento para clasificar 8 gestos posibles en cualquier momento. Para la mayoría de los gestos, los clasificadores no supervisados procesan los datos de los músculos y el movimiento para aprender

a separar los gestos de otros movimientos en tiempo real. Los Modelo de Mezcla Gaussiana (GMM) se actualizan continuamente para agrupar los datos de transmisión y crear umbrales adaptativos. Esto permite que el sistema se calibre a las señales de cada persona mientras realiza gestos que controlan el robot. Dado que no necesita ningún dato de calibración previo, esto puede ayudar a los usuarios a empezar a interactuar con el robot rápidamente.

Paralelamente a estos procesos de clasificación, una red neuronal predice la flexión o extensión de la muñeca a partir de las señales de los músculos del antebrazo. La red se entrena con datos de usuarios anteriores en lugar de requerir nuevos datos de entrenamiento de cada usuario. En la Figura 2.1 se puede observar el pipeline utilizado para la detección de los gestos.

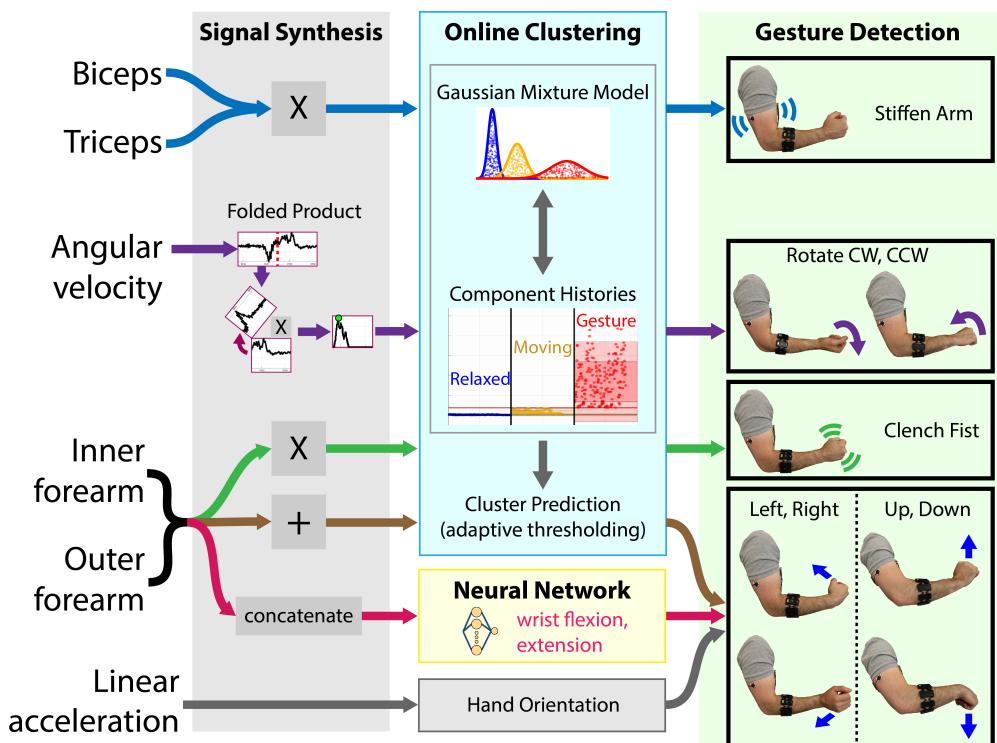


Figura 2.1: Pipeline detección de gestos en "Conduct-a-Bot"
Obtenida de [3]

Los gestos con las manos son una forma de comunicación no verbal que se puede utilizar en varios campos, como la comunicación entre personas sordomudas, el control de robots, la Interacción Humano-Computadora (HCI), la domótica y las aplicaciones médicas. Los trabajos de investigación basados en gestos con las manos han adoptado muchas técnicas diferentes, incluidas aquellas basadas en tecnología de sensores instrumentados y visión por computadora. En otras palabras, el signo de la mano se puede clasificar en muchos títulos, como postura y gesto, así como dinámico y estático, o un híbrido de ambos.

El artículo escrito por Munir Oudah, Ali Al-Naji y Javaan Chahl, [4], se centra en una

revisión de la literatura sobre técnicas de gestos con las manos e introduce sus ventajas y limitaciones en diferentes circunstancias. Además, tabula el rendimiento de estos métodos, centrándose en las técnicas de visión por computadora que se ocupan de los puntos de similitud y diferencia, la técnica de segmentación de la mano utilizada, los algoritmos y desventajas de clasificación, el número y tipos de gestos, el conjunto de datos utilizado, el rango de detección (distancia). y tipo de cámara utilizada. Este artículo es una descripción general exhaustiva de los métodos de gestos con las manos con una breve discusión de algunas posibles aplicaciones.

El objetivo principal de este estudio es introducir un sistema que pueda detectar gestos humanos específicos y utilizarlos para transmitir información o con fines de mando y control. Por lo tanto, incluye no sólo el seguimiento del movimiento humano, sino también la interpretación de ese movimiento como órdenes significativas. Generalmente se utilizan dos enfoques para interpretar gestos para aplicaciones HCI. El primer enfoque se basa en guantes de datos (usables o de contacto directo) y el segundo enfoque se basa en visión por computadora sin necesidad de usar ningún sensor.

Los sensores portátiles basados en guantes se pueden utilizar para capturar el movimiento y la posición de la mano. Además, pueden proporcionar fácilmente las coordenadas exactas de las ubicaciones, orientación y configuraciones de la palma y los dedos mediante el uso de sensores conectados a los guantes. Sin embargo, este enfoque requiere que el usuario esté conectado físicamente a la computadora, lo que bloquea la facilidad de interacción entre el usuario y la computadora. Sin embargo, el enfoque moderno basado en guantes utiliza la tecnología táctil, que es una tecnología más prometedora y se considera tecnología háptica de grado industrial. Donde el guante brinda retroalimentación háptica que hace que el usuario sienta la forma, la textura, el movimiento y el peso de un objeto virtual mediante el uso de tecnología de microfluidos.

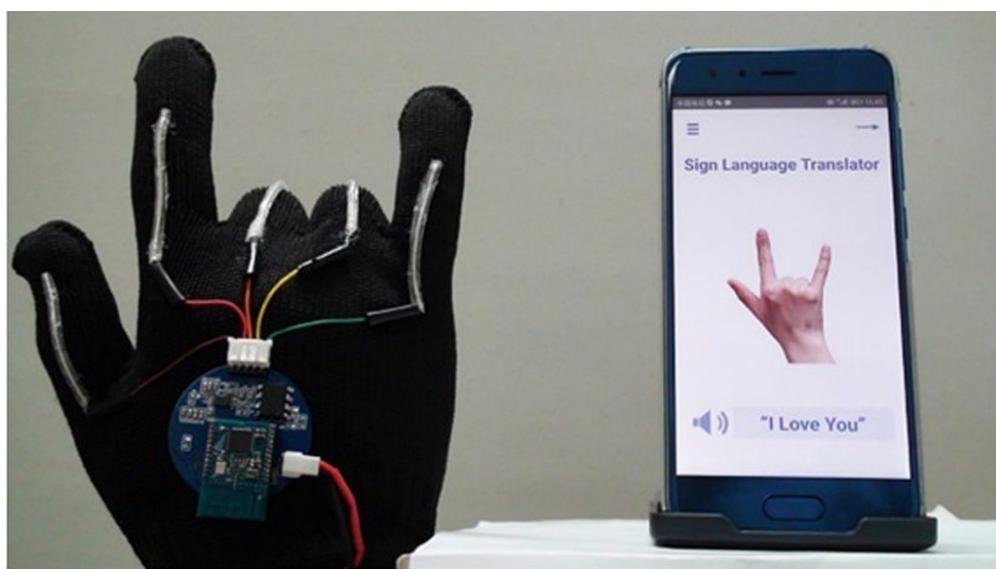


Figura 2.2: Guante para detección de gestos
Obtenida de [5]

El sensor basado en visión por computador es una técnica común, adecuada y aplicable porque proporciona comunicación sin contacto entre humanos y computadoras. Se pueden utilizar diferentes configuraciones de cámaras, como monocular, ojo de pez e IR. Sin embargo, esta técnica implica varios desafíos, incluida la variación de la iluminación, problemas de fondo, el efecto de las occlusiones, fondos complejos, tiempo de procesamiento intercambiado con la resolución y la velocidad de fotogramas y objetos de primer plano o de fondo que presentan el mismo tono de color de piel o que aparecen como manos.

Dentro de la detección de gestos utilizando visión por computadora, existen varios enfoques comunes. Uno de ellos es el reconocimiento basado en colores, que implica el uso de guantes u otros objetos con colores específicos para facilitar la detección de puntos característicos en las manos. Otro enfoque es el reconocimiento basado en la apariencia, que consiste en extraer características de la imagen, como la forma de la mano, y compararlas con modelos predefinidos. Este método se basa en la intensidad de los píxeles en la imagen. Por otro lado, el reconocimiento basado en el movimiento puede ser utilizado para la detección de gestos al analizar una secuencia de imágenes para extraer el movimiento del objeto de interés. Otros métodos incluyen el reconocimiento basado en esqueletos, el reconocimiento basado en profundidad, el reconocimiento basado en modelos 3D y el reconocimiento basado en deep learning.

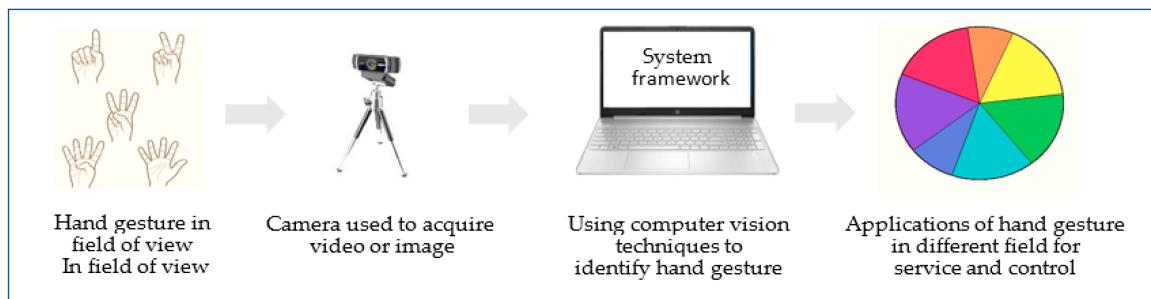


Figura 2.3: Visión por computador
Obtenida de [4]

A medida que los robots se han vuelto más presentes en nuestra vida diaria, la Interacción Humano-Robot (HRI) ha tenido un impacto positivo en el desarrollo de la robótica. Por lo tanto, ha habido un interés creciente en el desarrollo del reconocimiento de gestos de las manos basado en la visión para que la Interacción Humano-Computadora (HCI) supere las barreras entre humanos y robots. El objetivo es que la interacción con los robots sea tan natural como la que existe entre individuos. Los gestos con las manos pueden proporcionar información natural, intuitiva y creativa para comunicarse con robots.

En el artículo escrito por Jing Qi, Li Ma, Zhenchao Cui y Yushu Yu, [6], se realiza un análisis del reconocimiento de gestos con las manos utilizando tanto cámaras monoculares como cámaras RGB-D para este fin. En concreto, el proceso principal de reconocimiento de gestos visuales incluye adquisición de datos, detección y segmentación de gestos con las manos, extracción de características y clasificación de gestos, que son discutidos en este artículo.

Además, se revisan evaluaciones experimentales y los algoritmos de reconocimiento de gestos para la interacción humano-robot.

Como se expone en este artículo, el primer paso crucial en la detección de gestos es separar el gesto del fondo de la escena para permitir su reconocimiento por parte del ordenador. Esto se debe a que la computadora captura tanto los detalles del gesto como los de la escena. La segmentación de la mano implica dividir la colección de coordenadas de puntos de píxeles obtenidas en la fase de detección de gestos, lo que reduce el cálculo de puntos de píxeles y facilita las operaciones posteriores. Es esencial completar con éxito esta etapa para lograr una identificación precisa de los gestos. Aunque existen diversas técnicas de segmentación de gestos, muchas de ellas enfrentan dificultades en términos de precisión, estabilidad y velocidad, especialmente en entornos con fondos complejos o variaciones en la distancia entre la cámara y la persona.

Tras la segmentación, se extrae información relevante de la imagen mediante la extracción de características, y el tipo de gesto se reconoce utilizando estas características. La clasificación de gestos implica la categorización de las características espaciotemporales extraídas de los gestos, y representa la etapa final del proceso de reconocimiento de gestos. Entre los principales métodos de clasificación destacados en este artículo se encuentran la comparación de plantillas, la clasificación basada en información geométrica, la deformación dinámica del tiempo, los modelos ocultos de Markov, machine learning y deep learning.

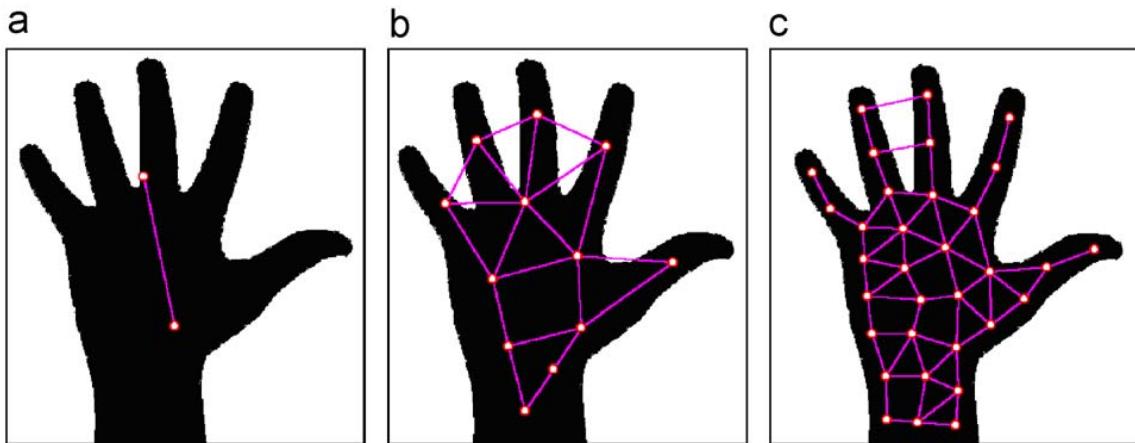


Figura 2.4: Detección de manos
Obtenida de [7]

3 Objetivos

Para realizar este Trabajo Final de Grado (TFG) se propone el desarrollo de un sistema de control teleoperado del robot TurtleBot3 mediante una detección de gestos del operario. Para esto, se coordinarán todos los movimientos que pueda realizar el robot con unos gestos predeterminados y captados por una WebCam cualquiera. Así mismo, se desarrollará una opción de mapeado de forma que el robot permita crear un mapa del entorno por el que está trabajando, de esta forma al tener un mapa creado facilitará al usuario saber cuando encontrará un obstáculo en su camino. Por lo que los objetivos de este proyecto vienen definidos por:

- Explorar los actuales métodos de detección de manos y buscar como aplicarlos a la teleoperación de un robot móvil.
- Lograr una sensación lo más fiel posible a la realidad en el movimiento del robot.
- Descubrir distintos métodos de mapeado de entornos y así poder aplicarlo a nuestro robot móvil.
- La creación de los códigos que nos permitan realizar la detección de gestos, movimiento del robot y observación de la propia cámara del robot.

Estos objetivos se pueden concretar en:

- Desarrollar un sistema capaz de detectar gestos realizados por las manos del operario y así conseguir la teleoperación de un robot móvil.
- Implementar una interfaz gráfica que permita una conexión más sencilla entre el operario y el robot.

Finalmente, como objetivo personal, este proyecto ha permitido profundizar en conceptos proporcionados a lo largo del grado en Ingeniería Robótica en la especialidad de computación:

- Conocer un poco más el término de teleoperación y cómo funciona.
- Hacer uso de técnicas de visión por computador vistas en el grado y profundizar en ellas.

4 Metodología

La ejecución de este proyecto nos ha permitido tener la oportunidad de explorar diversas tecnologías y metodologías previamente desconocidas. Es por eso que, en este capítulo, se detallarán todas las herramientas requeridas para la elaboración de este TFG.

4.1 Python

Entre todos los lenguajes de programación que existen en la actualidad, se ha decidido utilizar Python para elaborar los códigos principales de este proyecto ya que es un lenguaje de programación ampliamente utilizado en las aplicaciones web, el desarrollo de software, la ciencia de datos y el machine learning.

Se trata de un lenguaje de programación de código abierto, orientado a objetos, fácil de interpretar y que destaca por su claridad respecto a su sintaxis. Además, Python incluye una biblioteca estándar, que contiene una gran cantidad de librerías diferentes, de forma predeterminada con una gran cantidad de funciones reutilizables. A día de hoy es uno de los lenguajes de programación más utilizados alrededor del mundo. [8]

4.2 Math

La biblioteca *math* en Python es un conjunto de funciones y constantes matemáticas predefinidas que están integradas en el lenguaje Python. Proporciona acceso a operaciones matemáticas comunes y funciones avanzadas que son útiles para cálculos numéricos y científicos. Además, incluye funciones para operaciones básicas como la raíz cuadrada, el valor absoluto, funciones trigonométricas, exponenciales y logaritmos, entre otras. La librería *math* es ampliamente utilizada en programación científica, ingeniería, análisis de datos y otras áreas donde se requieren operaciones matemáticas precisas y eficientes.

4.3 OpenCV

OpenCV es una biblioteca de visión por computador de *software* de aprendizaje automático y código abierto. OpenCV fue desarrollada por la compañía Intel para proporcionar una infraestructura común para aplicaciones de visión por computadora y acelerar el uso de la percepción artificial en los productos comerciales. En la actualidad, ha ganado gran popularidad debido a su extenso conjunto de funciones y por contar con un soporte multiplataforma para Linux, Windows, IOS, entre otros.

La biblioteca tiene más de 2500 algoritmos optimizados, que incluyen un conjunto completo de algoritmos de aprendizaje automático y visión por computadora, tanto clásicos como de

última generación. Estos algoritmos se pueden utilizar para detectar y reconocer rostros, identificar objetos, clasificar acciones humanas en videos, rastrear movimientos de cámara, rastrear objetos en movimiento, extraer modelos 3D de objetos, producir nubes de puntos 3D a partir de cámaras estéreo, unir imágenes para producir una alta resolución. Imagen de una escena completa, buscar imágenes similares en una base de datos de imágenes, eliminar los ojos rojos de imágenes tomadas con flash, seguir los movimientos oculares, reconocer paisajes y establecer marcadores para superponerlos con realidad aumentada, etc. [9]

4.4 Mediapipe

Mediapipe es una biblioteca de código abierto creado por la compañía Google que proporciona herramientas y componentes para facilitar el uso de técnicas de IA y Machine Learning (ML). Esta plataforma facilita el desarrollo de aplicaciones de visión por computadora, seguimiento de manos, detección de rostros, detección de objetos, entre otras tareas relacionadas con la percepción y el análisis de imágenes y video.

La principal ventaja que proporciona Mediapipe es la gran variedad de ejemplos y soluciones básicas a partir de las que podemos realizar soluciones más complejas y avanzadas. Uno de sus usos más destacados es el procesamiento de imágenes en tiempo real, tanto audio, imágenes y lecturas de diferentes sensores. Esto facilita su uso para el seguimiento de personas, detección de rostros o objetos o la detección de puntos característicos de las manos. [10]

Para nuestro proyecto necesitamos realizar un seguimiento de manos por lo que haremos uso de la herramienta Mediapipe Hands que es una solución que permite la detección de manos, con 21 puntos de referencias 3D. Permitiendo identificar cada uno de los dedos y palma de la mano. Para ello MediaPipe emplea Machine Learning, de donde han obtenido múltiples modelos que trabajan juntos. El primero de ellos es un modelo que simplemente detecta la palma de la mano y, una vez obtenida el área en donde se encuentra la mano, la imagen pasará por un hand landmark model, que es el modelo que permitirá ubicar los 21 puntos de referencia en la imagen dada por la detección de palma. En la Figura 4.1 se pueden observar los 21 puntos característicos de una mano. [11].

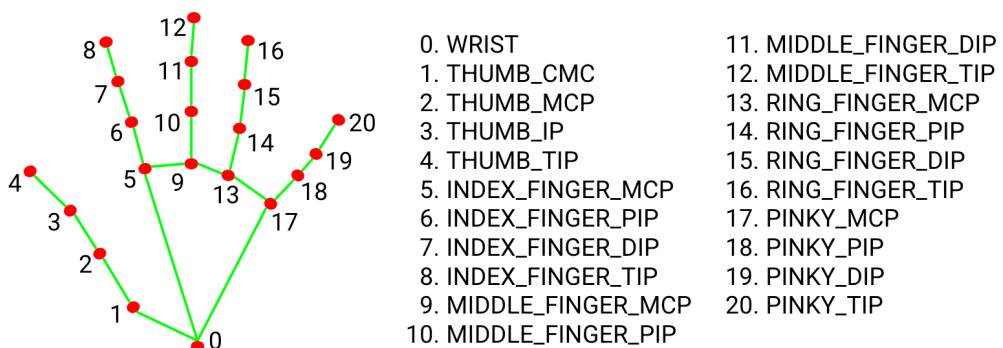


Figura 4.1: Puntos de las manos detectables por mediapipe
Obtenida de [10]

4.5 Tkinter

Tkinter es una librería del lenguaje de programación Python y funciona para la creación y el desarrollo de aplicaciones de escritorio. Esta librería facilita el posicionamiento y desarrollo de una interfaz gráfica de escritorio con Python. Tkinter es el paquete estándar de Python para interactuar con Tk.

De acuerdo a la documentación de Python, TK se describe a sí mismo como el único toolkit o kit de herramientas para el desarrollo de una Interfaz Gráfica de Usuario (GUI) que funciona en todos los sistemas operativos, es decir, funciona en Windows, Mac OS y Linux. Además, está diseñado y preparado para lenguajes dinámicos con un alto nivel, como pueden ser Tcl, Ruby o Perl, entre otros. [12].

En nuestro proyecto se ha decidido hacer uso de la librería Tkinter para poder crear una interfaz gráfica que permita a cualquier usuario ejecutar todos los procesos que necesite, como lanzar el simulador Gazebo, observar el árbol de transformaciones, crear un mapeado, guardarlo o cargar un mapa ya creado, así como ejecutar los códigos de detección de gestos, movimiento del turtlebot o poder abrir la cámara del robot de una forma visual y rápida.

4.6 ROS

Robot Operating System (ROS) es un conjunto de bibliotecas de software y herramientas de código abierto que le ayudan a crear aplicaciones para robots. En el uso de ROS podemos encontrar desde controladores hasta algoritmos de última generación y con potentes herramientas de desarrollo. Además, proporciona abstracción de hardware, visualizadores, transferencia de mensajes, administración de paquetes y más. Por esto, ROS tiene lo que necesita para su próximo proyecto de robótica. [13]

A pesar de no ser un sistema operativo, ROS provee los servicios estándar de uno de estos tales como la abstracción del hardware, el control de dispositivos de bajo nivel, la implementación de funcionalidad de uso común, el paso de mensajes entre procesos y el mantenimiento de paquetes. Está basado en una arquitectura de grafos donde el procesamiento toma lugar en los nodos que pueden recibir, mandar y multiplexar mensajes de sensores, control, estados, planificaciones y actuadores, entre otros.

ROS tiene dos partes básicas: la parte del sistema operativo, ros, y ros-pkg. Esta última consiste en un conjunto de paquetes aportados por la contribución de usuarios, organizados en pilas, que implementan las funcionalidades tales como localización y mapeo simultáneo, planificación, percepción, simulación, etc. [14]

Por todo esto, podemos decir que ROS ofrece una plataforma de software estándar a los desarrolladores a través de industrias que los llevarán desde la investigación y el prototipado hasta el despliegue y producción.



Figura 4.2: Logo oficial ROS
Obtenida de [13]

4.6.1 Rospy

Rospy es una biblioteca cliente pura de Python para ROS. La API del cliente ros.py permite a los programadores de Python interactuar rápidamente con temas, servicios y parámetros de ROS. El diseño de ros.py favorece la velocidad de implementación sobre el rendimiento del tiempo de ejecución, de modo que los algoritmos puedan crear prototipos y probarse rápidamente dentro de ROS. También es ideal para código de ruta no crítica, como código de configuración e inicialización. Muchas de las herramientas ROS están escritas en ros.py para aprovechar las capacidades de introspección de tipos. Muchas de las herramientas ROS, como rostopic y rosservice, están construidas sobre ros.py.

Durante este proyecto, se empleará la librería ros.py para desarrollar los códigos encargados de controlar el movimiento del robot TurtleBot en Python. Dado que el control del robot TurtleBot requiere el uso de ROS, es esencial utilizar ros.py, la biblioteca que nos permite crear los códigos en Python compatibles con ROS. [15]

4.7 Gazebo

Gazebo es una colección de bibliotecas de software de código abierto diseñadas para simplificar el desarrollo de aplicaciones de alto rendimiento. La audiencia principal de Gazebo son los desarrolladores, diseñadores y educadores de robots. Sin embargo, Gazebo se ha estructurado para adaptarse a muchos casos de uso diferentes. Cada biblioteca dentro de Gazebo tiene dependencias mínimas, lo que permite su uso en tareas que van desde la resolución de transformaciones matemáticas hasta la codificación de video y hasta la simulación y la gestión de procesos.[16]

A lo largo de este proyecto se ha hecho uso del simulador de Gazebo para simular distintos mundos con obstáculos totalmente diferentes y nuestro TurtleBot3, de forma que en esta

plataforma podamos observar como reacciona el robot y así poder determinar los cambios que debemos llevar a cabo. Un ejemplo de mundo es el que se observa en la Figura 4.3, donde se ven todos los obstáculos y el propio TurtleBot.

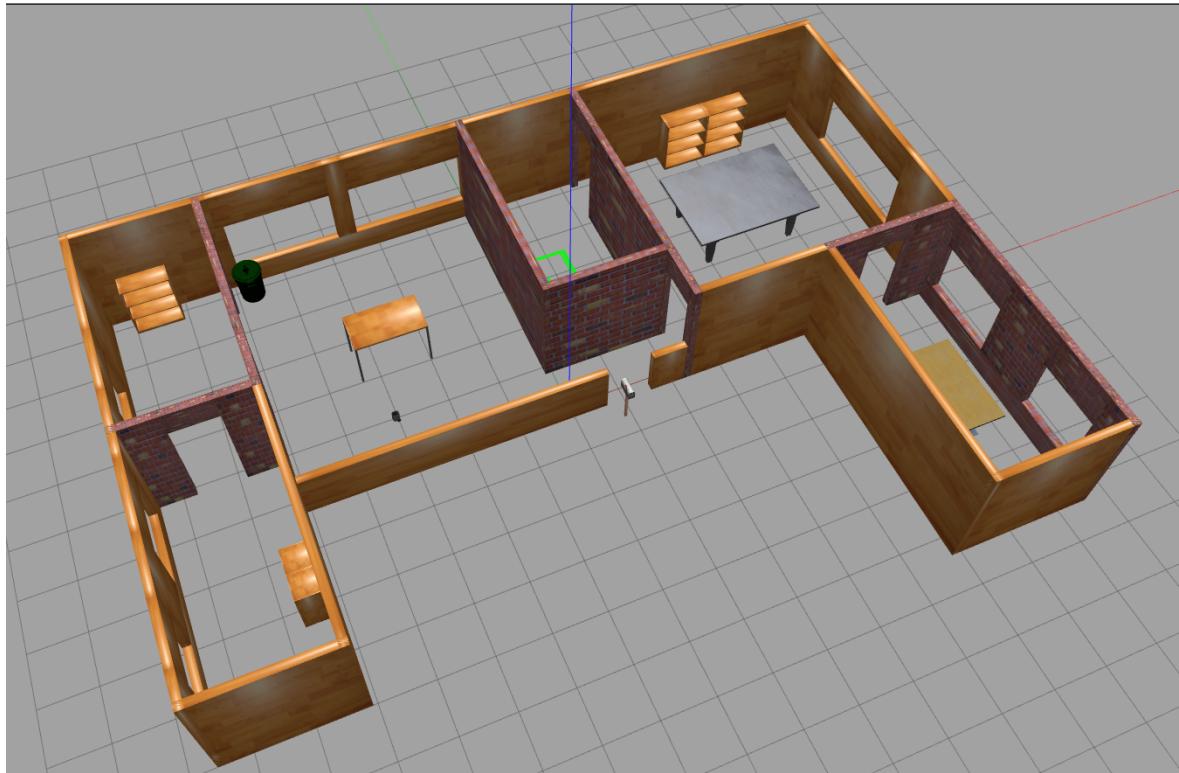


Figura 4.3: Simulación en Gazebo

4.8 SLAM

Simultaneous Localization and Mapping (SLAM) es un problema fundamental en robótica, donde el robot necesita construir simultáneamente un mapa de su entorno y localizarse dentro de él. SLAM es un desafío porque el robot tiene que lidiar con datos de sensores ruidosos e incompletos, obstáculos dinámicos, cierres de bucles y restricciones computacionales. SLAM se puede clasificar en dos tipos: en línea y fuera de línea. SLAM en línea realiza el mapeo y la localización en tiempo real, mientras que SLAM fuera de línea procesa los datos del sensor después de que el robot ha terminado su exploración. SLAM también puede utilizar diferentes tipos de sensores, como escáneres láser, cámaras, unidades de medición inercial. [17].

ROS ofrece varios paquetes y herramientas para ayudar a implementar y probar algoritmos SLAM en su robot, como gmapping, cartographer, rtabmap y slam_toolbox. En nuestro caso, hemos hecho uso del paquete Gmapping para realizar nuestros mapeados.

4.8.1 Gmapping

Gmapping es un paquete que contiene ROS basado en SLAM para realizar el mapeado de una zona desconocida con el uso de un robot móvil. Gmapping utiliza un escáner láser y datos de odometría para hacer SLAM en línea basado en un filtro de partículas. Crea un mapa de cuadrícula de ocupación 2 Dimensiones (2D) que muestra la probabilidad de que cada celda esté ocupada por un obstáculo. En la Figura 4.4 se puede observar como se crea el mapa con las medidas captadas por el láser. [18].

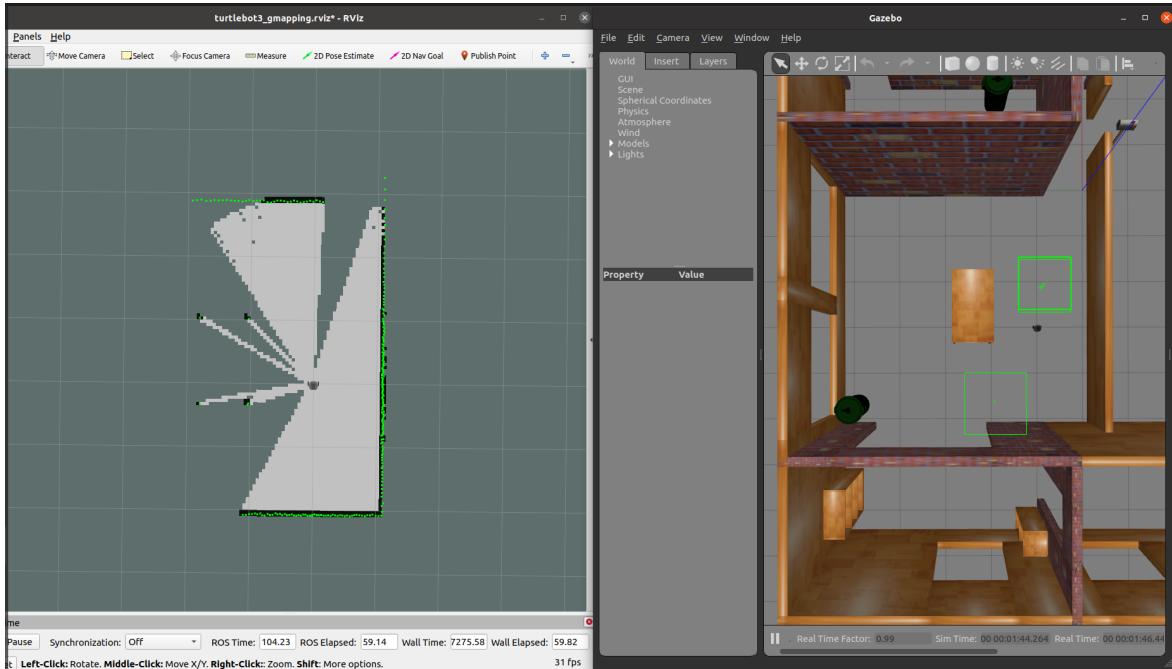


Figura 4.4: Mapeado con Gmapping

4.9 TurtleBot 3

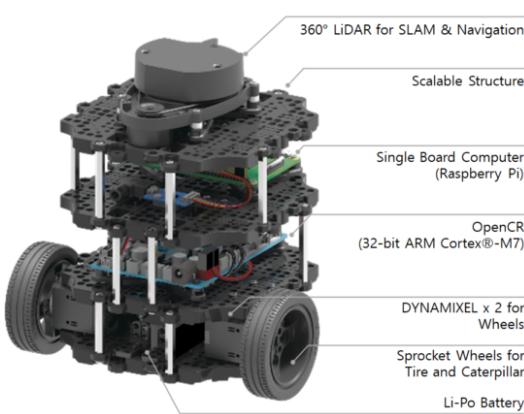
TurtleBot es un kit de robot personal de bajo costo con software de código abierto. Con TurtleBot, podrás construir un robot que pueda conducir por tu casa, ver en 3D y tener suficiente potencia para crear aplicaciones interesantes ya que el kit TurtleBot consta de una base móvil, un sensor de distancia 2D/3D, una computadora portátil o una computadora de placa única y el kit de hardware de montaje TurtleBot.

TurtleBot está diseñado para ser fácil de comprar, construir y ensamblar, utilizando productos de consumo disponibles en el mercado y piezas que se pueden crear fácilmente a partir de materiales estándar. Como plataforma de robótica móvil de nivel básico, TurtleBot tiene muchas de las mismas capacidades que las plataformas de robótica más grandes de la empresa. [19].

El principal motivo para utilizar este robot es porque está diseñado para utilizarse principalmente con ROS, lo que hace que posible el uso de todas las bibliotecas necesarias para utilizar el TurtleBot con total control. Además, el uso de ROS y del TurtleBot nos permite utilizar el simulador Gazebo mencionado en el apartado anterior para simular todos los comportamientos del robot.

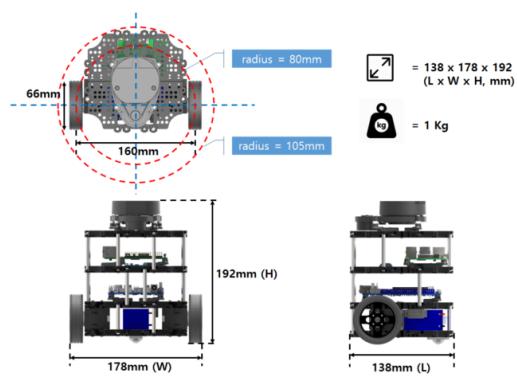
Por todo esto y teniendo en cuenta que la versión de ROS utilizada es ROS Noetic, el único turtlebot que funciona en simulación para esta versión es el TurtleBot3.

TurtleBot3 Burger



(a) Características TurtleBot3 Burger

TurtleBot3 Burger



(b) Dimensiones TurtleBot3 Burger

Figura 4.5: Turtlebot3 Burger
Obtenidas de [20]

El TurtleBot3 es un robot móvil pequeño, asequible y programable basado en ROS para uso en educación, investigación, pasatiempos y creación de prototipos de productos. La tecnología principal de TurtleBot3 es SLAM, Navegación y Manipulación, lo que lo hace adecuado para robots de servicio a domicilio. El TurtleBot puede ejecutar algoritmos SLAM para construir un mapa y puede conducir por su habitación. Además, se puede controlar de forma remota desde una computadora portátil, un joypad o un teléfono inteligente con Android. El TurtleBot también puede seguir las piernas de una persona mientras camina por una habitación.

Por último, debemos mencionar que como TurtleBot3 encontramos dos modelos de robot, el modelo *Burger* y el modelo *Waffle Pi*. Respecto a las principales características de ambos modelos, podemos decir que la principal diferencia entre ambos modelos es que el modelo *Burger* dispone del uso de una Raspberry Pi, mientras que el modelo *Waffle Pi* dispone de una cámara para captar percepciones. [20]. Dado que en nuestro proyecto haremos uso del modelo *Burger*, en las Figuras 4.5a y 4.5b, observamos las principales características y medidas de este modelo, donde se puede comprobar lo que mencionamos anteriormente, que el robot TurtleBot3 es un robot de pequeñas dimensiones perfecto para su fácil transporte.

5 Desarrollo

A lo largo de esta sección, se expondrán los distintos pasos que se han llevado a cabo para llegar a nuestro objetivo. En la Figura 5.1, se puede observar, de manera global, cada una de las etapas que se han tenido que realizar.

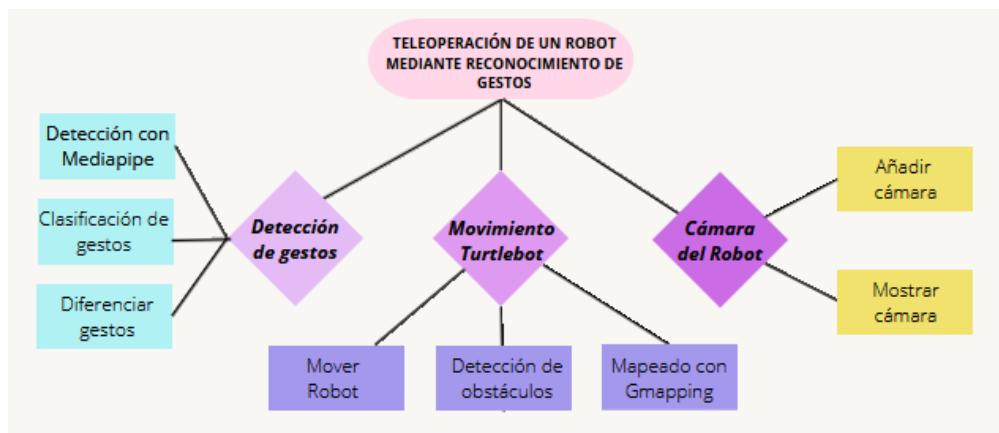


Figura 5.1: Esquema visual del proyecto

Debemos mencionar que para poder realizar este proyecto se han ido creando diferentes versiones que cada vez mejoraban a la anterior en algún concepto. Además, en todas las versiones se ha utilizado el esquema que se puede observar en la Figura 5.2 para realizar el control del robot.

En este esquema se pueden observar los siguientes elementos. *Script Publicador*, se trata de un script programado en Python que se encarga de realizar la detección de gestos y, una vez detectado y clasificado, se encargará de publicar en un topic, creado por un nodo de ROS, el número correspondiente al gesto que realice el usuario. En este script, nuestro nodo de ROS se llamará "*nodo_publicador*" y publicará en los topics "/mano_derecha" y "/mano_izquierda" los gestos detectados de la mano derecha e izquierda respectivamente.

Script Suscriptor, se trata de un script también escrito en Python que tendrá varios objetivos a realizar, el primero de ellos será leer el topic en el que se ha publicado el resultado del gesto; otro objetivo será utilizar este valor leído y aplicar ciertas velocidades a los motores del robot. Además, este script tendrá que realizar una detección de obstáculos utilizando el láser que está implementado en el Turtlebot3, y en función del valor leído por el láser aplicar unas velocidades u otras. Para enviar las velocidades este script crea otro nodo de ROS, que además de estar suscrito al topic mencionado anteriormente, publicará en el topic correspondiente las velocidades.

En este segundo script, nuestro nodo de ROS se llamará *"nodo_suscriptor"* donde nos suscribiremos a los tópicos *"/mano_derecha* y *"/mano_izquierda"* para conocer el gesto que se está realizando y tras procesarlo y saber la acción que queremos hacer, se publicará en el topic */cmd_vel* la velocidad que llevará el TurtleBot3.

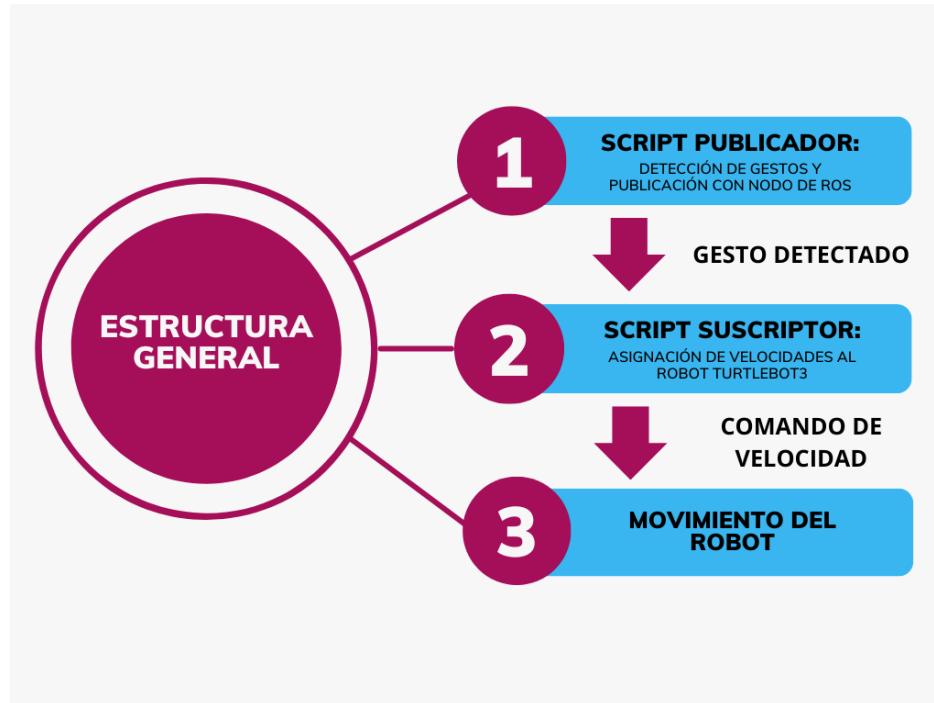


Figura 5.2: Estructura general control del robot

5.1 Primera versión

Para esta primera versión debemos decir que el objetivo principal es realizar una detección de gestos de una sola mano utilizando la librería Mediapipe y, una vez detectado, realizar una conexión sencilla con una simulación del TurtlBot3 en Gazebo.

En primer lugar, debemos decir que la detección de los diferentes gestos se realizará en todas las versiones utilizando la librería Mediapipe, explicada en el apartado anterior. Sin embargo, en este punto, solo se realizará la detección de una de las manos con el objetivo de conocer como funciona exactamente el uso de esta librería. Además, hay que destacar que la librería mediapipe nos proporciona las coordenadas 'X' e 'Y' de cada uno de los 21 puntos característicos, por lo que en esta versión, se realizará una clasificación de los gestos utilizando simplemente estas coordenadas de los extremos de cada dedo.

En la Figura 5.3, se pueden ver todos los gestos que se detectarán tanto en esta versión como en las siguientes y el movimiento al que corresponde cada uno de esos gestos. Dado

que aquí solo se pueden realizar detecciones de una mano se muestran los gestos realizados siempre por la misma mano.

Respecto al movimiento del robot en esta primera versión, hay que mencionar que el robot podrá realizar cuatro movimientos muy simples, hacia delante, atrás, giro a la derecha y por último giro a la izquierda. Además, hay que destacar que al realizar simplemente la detección de una de las dos manos el robot solo podrá realizar un movimiento, es decir, no podrá avanzar mientras está girando, solo podrá avanzar o girar. Por último, destacar que en todas las versiones tanto la velocidad lineal como la angular será siempre constante.



Figura 5.3: Gestos para una mano

5.2 Segunda versión

Como se mencionaba al inicio de este apartado, se han ido realizando varios cambios a la versión anterior para así conseguir una mejora que nos permita llegar a nuestro objetivo. En este caso, el principal cambio que se puede observar, se encuentra en la detección de los gestos.

En este punto, se ha incluido la detección de la segunda mano, por lo que ahora nuestro programa permite al usuario realizar el movimiento deseado utilizando cualquiera de las dos manos. Otro cambio realizado es, que como ya tenemos ambas manos disponibles para el movimiento del robot, publicaremos el gesto detectado por la mano derecha en el topic `"/mano_derecha` y el gesto de la mano izquierda en el topic `"/mano_izquierda`, de esta forma podemos diferenciar que gesto se ha realizado en cada mano. Respecto a la clasificación de los gestos nada cambia, se siguen diferenciando gestos utilizando las coordenadas 'X' e 'Y' de los extremos de los dedos.

En el movimiento del TurtleBot3 también se produce algún cambio. Dado que en este punto ya tenemos ambas manos, vamos a utilizar cada una de las manos para realizar unos movimientos concretos. En la versión anterior, todos los movimientos partían de la misma mano, sin embargo, ahora vamos a utilizar el gesto de la mano derecha para mover el robot únicamente hacia delante o atrás y el gesto de la mano izquierda para realizar los giros hacia un lado u otro.

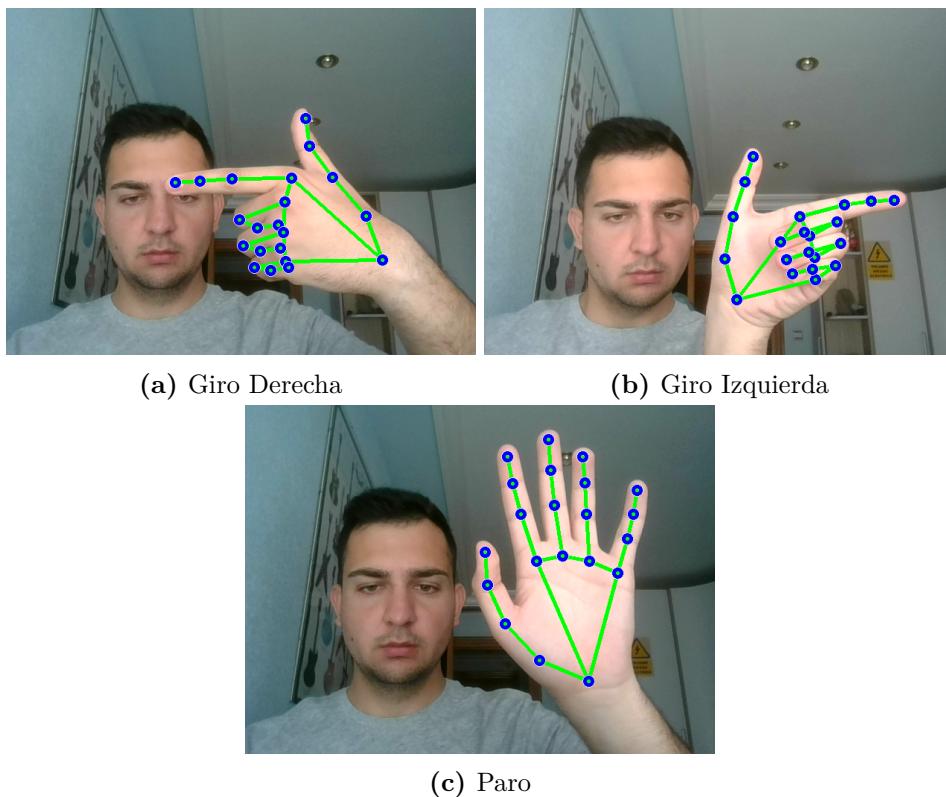


Figura 5.4: Gestos Mano Izquierda

Los gestos detectados por la mano derecha serán los que se pueden observar en las Figuras 5.3a y 5.3b, mientras que los detectados por la mano izquierda son los mostrados en la Figura 5.4.

5.3 Tercera versión

En esta última versión, se modifican los dos códigos que tenemos hasta el momento, tanto el detector de gestos como el encargado de mover el TurtleBot3. Como se ha mencionado anteriormente, hasta este momento se diferencian los gestos utilizando las coordenadas 'X' e 'Y' de los extremos de los dedos, sin embargo, este método de clasificación era muy impreciso. Y respecto al movimiento del TurtleBot3, no existía ningún tipo de restricción al encontrarse con algún obstáculo en la trayectoria a realizar.

En primer lugar, nos ocupamos de mejorar la clasificación de los gestos, para ello la decisión tomada para lograrlo fue obtener el ángulo que aparece entre los puntos 0, 4 y 8 que, como podemos ver en la Figura 5.5 se corresponden con los extremos de los dedos pulgar (punto 4) e índice (punto 8) y el punto correspondiente a la articulación de la muñeca (punto 0). Para obtener el ángulo formado, nos hemos creado la función *calcular_angulo* que está formada con los siguientes pasos:

1. Cálculo de vectores entre los puntos: se calcula el vector desde el punto 4 al punto 0 y desde el punto 0 al punto 8.

$$\text{vector}_{ab} = [X_b - X_a, Y_b - Y_a] \quad (5.1)$$

$$\text{vector}_{bc} = [X_c - X_b, Y_c - Y_b] \quad (5.2)$$

2. Cálculo del producto punto o producto escalar entre los vectores:

$$\text{producto_punto} = \text{vector}_{ab}[0] \times \text{vector}_{bc}[0] + \text{vector}_{ab}[1] \times \text{vector}_{bc}[1] \quad (5.3)$$

3. Cálculo de las magnitudes de los vectores para conocer la longitud de los vectores:

$$\text{magnitud}_{ab} = \sqrt{(\text{vector}_{ab}[0])^2 + (\text{vector}_{ab}[1])^2} \quad (5.4)$$

$$\text{magnitud}_{bc} = \sqrt{(\text{vector}_{bc}[0])^2 + (\text{vector}_{bc}[1])^2} \quad (5.5)$$

4. Uso de estas magnitudes para el cálculo del coseno del ángulo entre los vectores:

$$\text{coseno_theta} = \frac{\text{producto_punto}}{\text{magnitud}_{ab} \cdot \text{magnitud}_{bc}} \quad (5.6)$$

5. Nos aseguramos de que el coseno se encuentre en el rango [-1, 1].

$$\text{coseno_theta} = \max(-1, \min(1, \text{coseno_theta})) \quad (5.7)$$

6. Calculamos el ángulo deseado en radianes y lo pasamos a grados. Para pasarlo a grados

utilizamos la función *degrees* de la librería *math*:

$$\theta_radianes = \arccos(\coseno_\thetaeta) \quad (5.8)$$

$$\theta_grados = degrees(\theta_radianes) \quad (5.9)$$

7. Determinación del signo del ángulo utilizando el producto vectorial, se hace uso de la función *copysign* que se puede encontrar en la librería *math*:

$$signo = copysign(1, vector_ab[0] \times vector_bc[1] - vector_ab[1] \times vector_bc[0]) \quad (5.10)$$

8. Aplicación del signo al ángulo en grados. Se aplica el signo obtenido en el paso anterior al ángulo calculado en grados para obtener el ángulo final.

$$angulo_final = \theta_grados \cdot signo \quad (5.11)$$

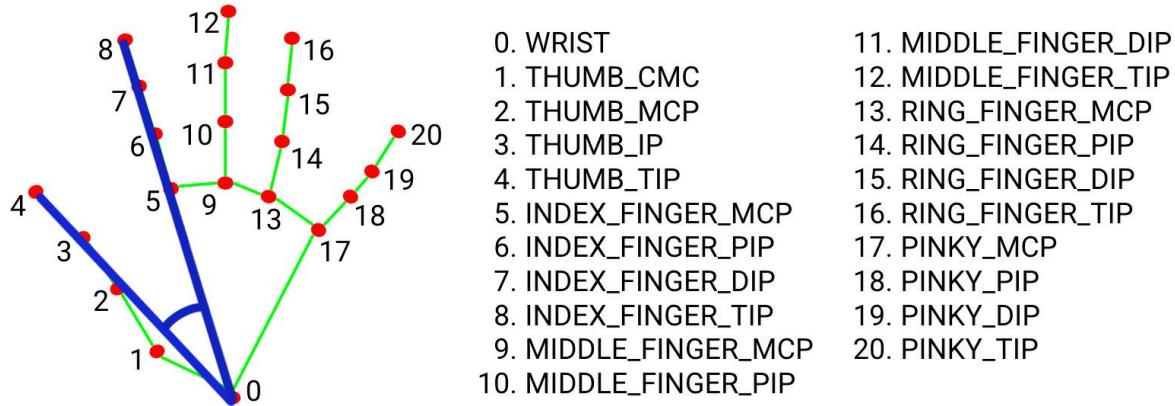


Figura 5.5: Puntos utilizados para el cálculo de ángulos

Una vez finalizado todos estos pasos, llegamos al punto de determinar que ángulos forman cada gesto. Para ello, utilizando esta función, comprobamos todo el rango de movimiento para cada uno de los gestos asignados. En el caso de los gestos de la mano derecha para mover el robot hacia delante o atrás, sabremos que se está haciendo el gesto de la Figura 5.3a para avanzar cuando el ángulo esté entre 10 y 40 grados. Sin embargo, cuando se detecten entre 120 y 180 grados o entre -120 y -180 grados se estará realizando el gesto de la Figura 5.3b, y se enviará el mensaje de retroceder. Por último, en caso de detectar cualquier otro ángulo se estará haciendo el gesto de parar el robot como el de la Figura 5.3e.

Respecto a la mano izquierda vamos a calcular el ángulo igual que en el caso de la otra mano, sin embargo, cuando se detecte un ángulo entre 65 y 95 grados sabremos que el usuario estará haciendo el gesto de la Figura 5.4a, para girar a la derecha. Por otro lado, si se obtiene un ángulo entre -80 y -115 grados nos encontraremos ante el gesto de la Figura 5.4b. Por último, para detectar el gesto de paro, correspondiente a la Figura 5.4c, sabremos que será cuando no se detecte ningún ángulo entre los establecidos anteriormente.

Es importante destacar que estos valores se determinaron mediante un amplio rango de movimiento para cada gesto, permitiendo que la mano tenga cierto grado de giro durante el gesto en lugar de mantenerse en una posición exacta durante todo el movimiento.

Por otro lado, respecto al movimiento también se han producido cambios, en esta versión se introduce la detección de objetos. Con esta detección buscamos evitar que el robot golpee algún obstáculo que pueda aparecer mientras está realizando la trayectoria enviada por el operario. En este caso, haremos uso del láser que posee el modelo del robot capaz de detectar 360 grados en 2D y recopilar un conjunto de datos alrededor del robot.

Para poder detectar obstáculos deberemos modificar el nodo suscriptor mencionado al inicio de este apartado. En primer lugar, tendremos que suscribirnos al topic ”/scan” de donde se obtendrá un vector con los 360 valores captados por el láser, la forma de procesar los valores de este vector será la siguiente. Hay que destacar que los valores que se obtienen del vector están captados en todo momento de izquierda a derecha, por lo que los primeros valores serán los que capta el robot por el lado izquierdo.

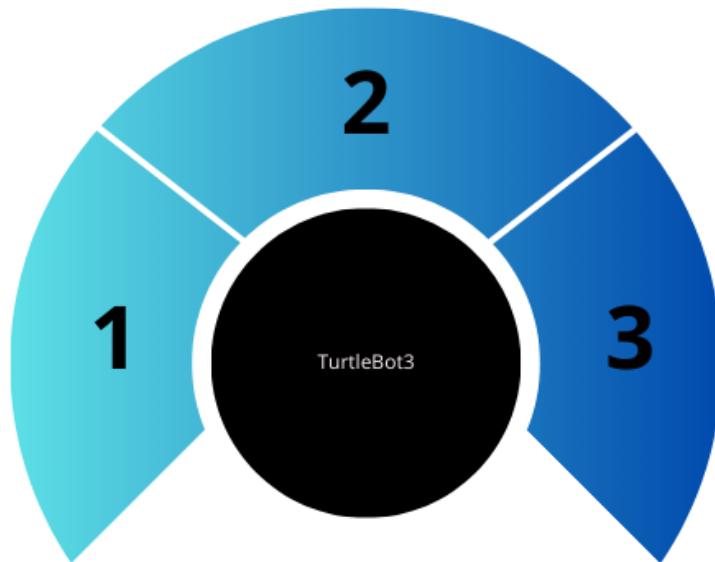


Figura 5.6: Rango láser TutleBot3

- Dividimos el tamaño total del vector en tres partes iguales, quedando de esta forma, los
-

primeros 120 valores se considerarán como objetos que aparecen por el lado izquierdo, los siguientes 120 datos serán objetos que aparecen en la parte delantera del robot y los últimos 120 valores del vector será todo lo captado por el robot en el lado derecho. Como se puede observar en la Figura 5.6.

- Para la detección del lado izquierdo y derecho se procederá de manera totalmente igual. El proceso realizado será recorrer todo el rango correspondiente y calcular la media de todos los valores de este rango. Una vez obtenida la media, si se obtiene una distancia menor a un valor concreto, se prohibirá el movimiento del robot hacia ese lado. En la Figura 5.6 se corresponden con las secciones 1 y 3.
 - Respecto a la parte delantera, se realizará el siguiente cálculo de la distancia. Se recorre todo el rango y si en algún momento se observa que una de las medidas es muy pequeña se obligará al robot a detenerse, ya que se considerará que el robot está excesivamente cerca de un obstáculo. Por otro lado, si ninguna medida es menor a este dato se realizará lo mismo que en los laterales, se calculará la media. En la Figura 5.6 se trata de la sección 2.

Durante la mejora del código, nos enfrentamos a un problema relacionado con la orientación del láser incorporado en el modelo del robot. Este estaba montado en la dirección opuesta, lo que provocaba que al retroceder, el robot detectara obstáculos acercándose, y al avanzar, indicara que se alejaba de ellos. Para resolver esto, invertimos la orientación del láser en el modelo del robot, girándolo 180 grados. Este ajuste nos permitió detectar correctamente una variedad de obstáculos, como paredes, mesas y sillas, cumpliendo así con nuestro objetivo.

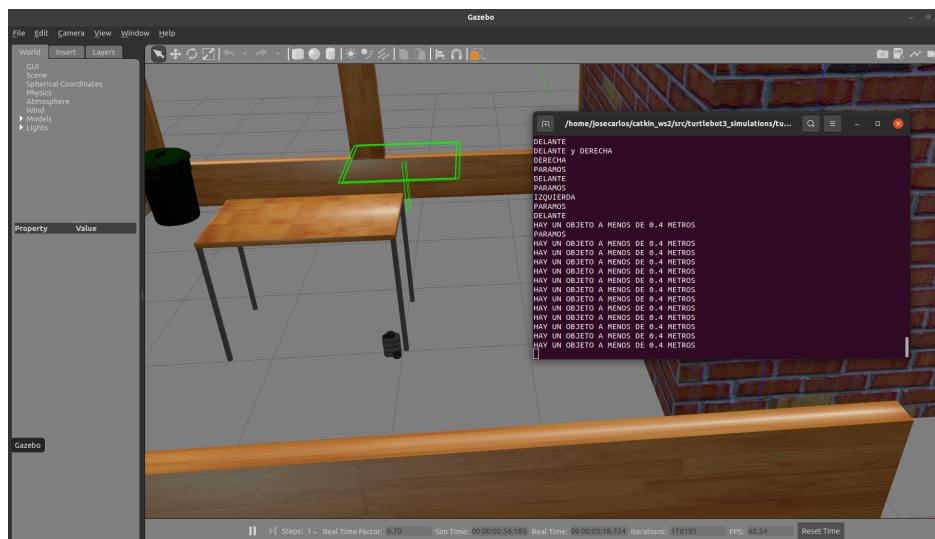


Figura 5.7: Ejemplo Detección de obstáculos

Una vez completado el proceso de ajuste del láser y determinar su funcionamiento, se ilustra un ejemplo de detección de objetos en la parte frontal del robot. La Figura 5.7 muestra este ejemplo, donde el robot se encuentra cerca de un objeto. En el terminal mostrado en la parte derecha de la figura, se indica la distancia entre el robot y el objeto detectado. En este punto,

el proyecto impide cualquier movimiento hacia adelante, ya que se ha implementado para restringir el movimiento hacia objetos detectados a una cierta distancia.

5.4 Mapeado

Para concluir el bloque relacionado con el movimiento del TurtleBot3, decidimos incorporar la opción de mapear el entorno durante la ejecución de una tarea. Para esto, investigamos métodos de mapeo utilizando robots móviles junto con ROS.

La razón detrás de esta adición se obtiene de la necesidad de conocer la ubicación del robot en entornos inicialmente desconocidos. El mapeo permite al operador del robot generar un mapa del área desconocida, lo que facilita el seguimiento preciso del robot y evitar colisiones con obstáculos en futuras operaciones.

Como mencionamos anteriormente, ROS ofrece varios paquetes y herramientas para implementar algoritmos SLAM en robots móviles, como gmapping, cartographer, rtabmap y slam_toolbox.

En nuestro caso, optamos por el algoritmo Gmapping debido a su familiaridad y compatibilidad con el TurtleBot3. Este algoritmo utiliza mediciones láser para calcular la probabilidad de presencia de obstáculos en la trayectoria del robot, permitiendo así la creación dinámica de mapas. A diferencia de otros algoritmos, como Cartographer y Rtabmap, que requieren datos adicionales como información de la IMU, Gmapping solo necesita mediciones láser para llevar a cabo el mapeo, aunque también puede hacer uso de datos de odometría para realizar una detección mucho más completa del entorno al que se enfrenta. Además, el algoritmo slam_toolbox ofrece herramientas adicionales como fusión, almacenamiento, edición y visualización de mapas, pero no fue seleccionado para nuestro caso debido a sus requisitos y características específicas. [21].

5.5 Cámara Turtlebot3

Después de completar los bloques de detección de gestos y movimiento del TurtleBot3, como se muestra en el diagrama de flujo de la Figura 5.1, avanzamos al bloque correspondiente a la cámara del robot. Este bloque se incorpora con la idea inicial de brindar una sensación de inmersión al usuario encargado del movimiento del robot, permitiéndole observar constantemente si el robot se está aproximando a algún obstáculo.

Como se indica en el diagrama, el primer paso dentro de este bloque es "Añadir cámara". Esto se debe a que el modelo utilizado para este proyecto, el TurtleBot3 Burger, no incluye una cámara en su diseño estándar. Para abordar esta carencia, optamos por agregar al modelo Burger la misma cámara presente en el modelo Waffle Pi, con el propósito inicial de evaluar la utilidad de esta adición. Dado que se trata de una cámara RGB, inicialmente solo podíamos visualizar en RVIZ la imagen capturada y los colores.

En este punto, decidimos explorar la posibilidad de cambiar la cámara de una RGB a una que nos permitiera capturar información de profundidad de los objetos observados. Esto nos permitiría crear una nube de puntos y ampliar las capacidades del proyecto en futuras evoluciones. Para lograr este objetivo, hicimos el cambio de una cámara RGB a una cámara Kinect, como la mostrada en la Figura 5.8.



Figura 5.8: Cámara Kinect
Obtenida de [22]

En cuanto al apartado "Mostrar cámara", se ha añadido al diagrama con el objetivo de facilitar al usuario el acceso a la visualización de la cámara de manera más directa, sin la necesidad de recurrir a RVIZ. Para lograr esto, se ha desarrollado un pequeño código en Python donde nos suscribimos al topic que corresponde a los datos de la cámara, lo que nos permite abrir una nueva ventana que muestra exactamente lo mismo que está captando el robot.

5.6 Interfaz

El último objetivo es crear una interfaz gráfica que facilite el uso del proyecto para cualquier operador de manera intuitiva y sin la necesidad de usar múltiples terminales para lanzar los procesos requeridos.

La primera versión de la interfaz, desarrollada con Tkinter, era bastante simple. Consistía en unos pocos botones que, en un principio, solo permitían abrir el simulador Gazebo, mostrar el árbol de transformaciones y ejecutar los códigos creados. Sin embargo, el código se ha evolucionado hasta obtener la interfaz final mejorando poco a poco los fallos encontrados. En la versión final de la interfaz ofrece una variedad de botones para permitir al usuario realizar diversas funciones según sus necesidades. Estos botones son los siguientes:

- "Lanzar Gazebo": Este botón abre el entorno de Gazebo mostrado en la Figura 6.1, utilizado en nuestro proyecto junto con el robot.

- "Árbol transformaciones": Permite al usuario visualizar el árbol de transformaciones para verificar la correcta unión de los componentes del robot. En el caso de nuestro proyecto se observará el árbol de la Figura 5.9.

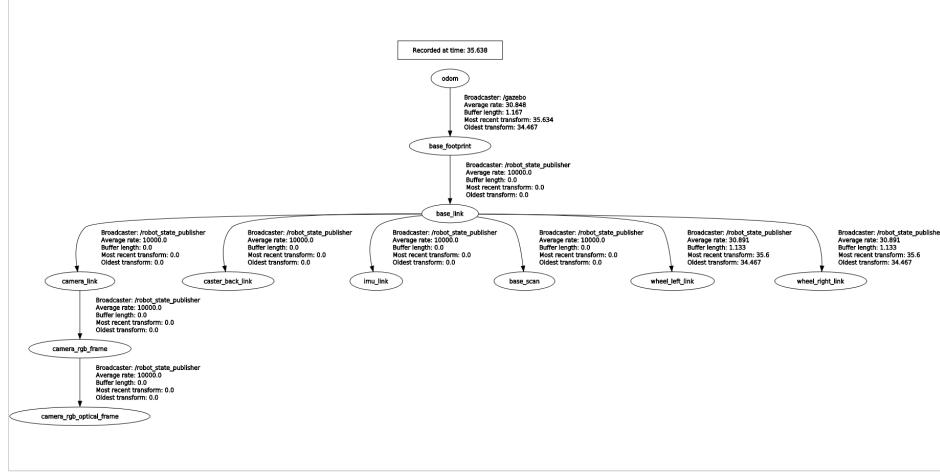


Figura 5.9: Árbol de transformaciones

- "Crear Mapeado": Inicia el proceso de mapeado utilizando Gmapping.
- "Guardar Mapa": Guarda en el dispositivo el mapa creado por el robot. Es importante utilizar este botón antes de cerrar el mapeado para evitar la pérdida del mapa.
- "Seleccionar Mapa": Abre un mapa en formato pgm, convirtiéndolo a imagen y mostrándolo en la interfaz en lugar de la imagen del TurtleBot3. Como en la Figura 5.10.



Figura 5.10: Selección de mapa

- "Detección Gestos": Inicia el código creado para la detección de gestos.
- "Turtlebot Movimiento": Lanza el código encargado de mover el TurtleBot3.
- "Turtlebot Cámara": Abre la cámara del Turtlebot3.
- Botones de cierre: Incluye botones individuales para cerrar cada uno de los procesos mencionados, así como un botón para cerrar la interfaz en su totalidad.
- Cuadro de texto: Aquí se muestra el proceso que se está realizando tras pulsar cada botón, por ejemplo, si se pulsa "Cerrar Transformaciones" se mostrará "Cerrando árbol de transformaciones", etc. Como en la Figura 5.11.



Figura 5.11: Cuadro de Texto interfaz

6 Resultados

A lo largo de este capítulo se presentarán tanto los experimentos realizados como los resultados obtenidos a lo largo de este proyecto. Al igual que en el apartado anterior, tanto la experimentación realizada como los resultados de cada experimento está separada en función de la versión en la que se encontraba el proceso.

Además, se mostrarán algunos de los entornos de simulación utilizados para llevar a cabo pruebas con el fin de verificar si el funcionamiento del proyecto cumplía con las expectativas iniciales.

6.1 Entornos de simulación

Para llevar a cabo todas las pruebas requeridas, se utilizaron los entornos de Gazebo mostrados en las Figuras 6.1 y 6.2. Este entorno de simulación proporciona escenarios variados con una amplia gama de obstáculos, que van desde objetos grandes y fijos como paredes, hasta objetos de menor tamaño como mesas o cubos de basura, como se puede observar en las imágenes.

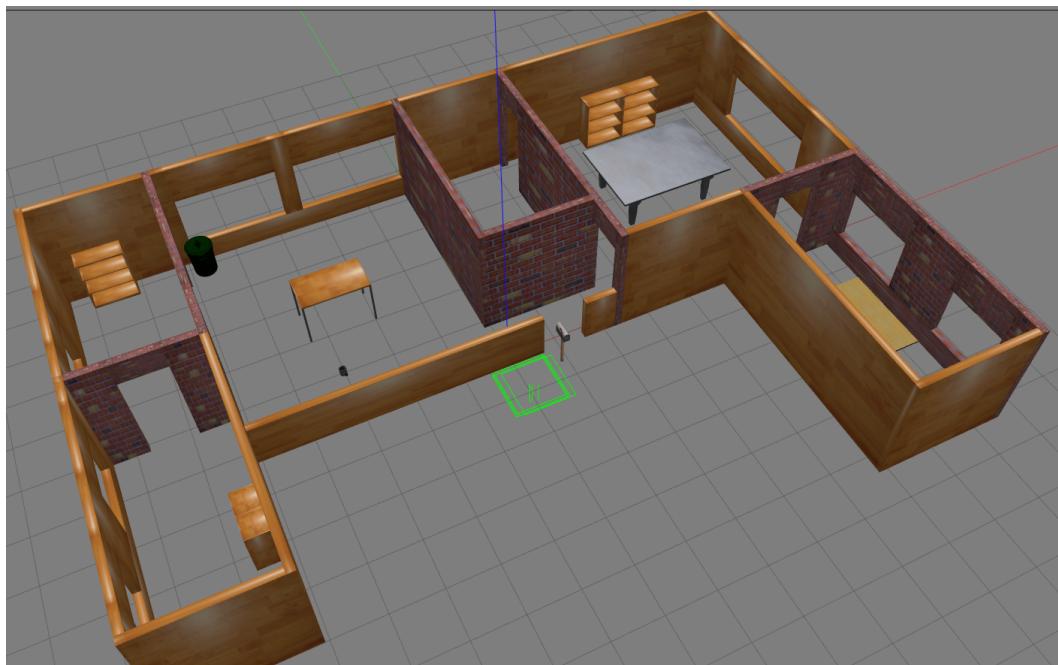


Figura 6.1: Entorno House

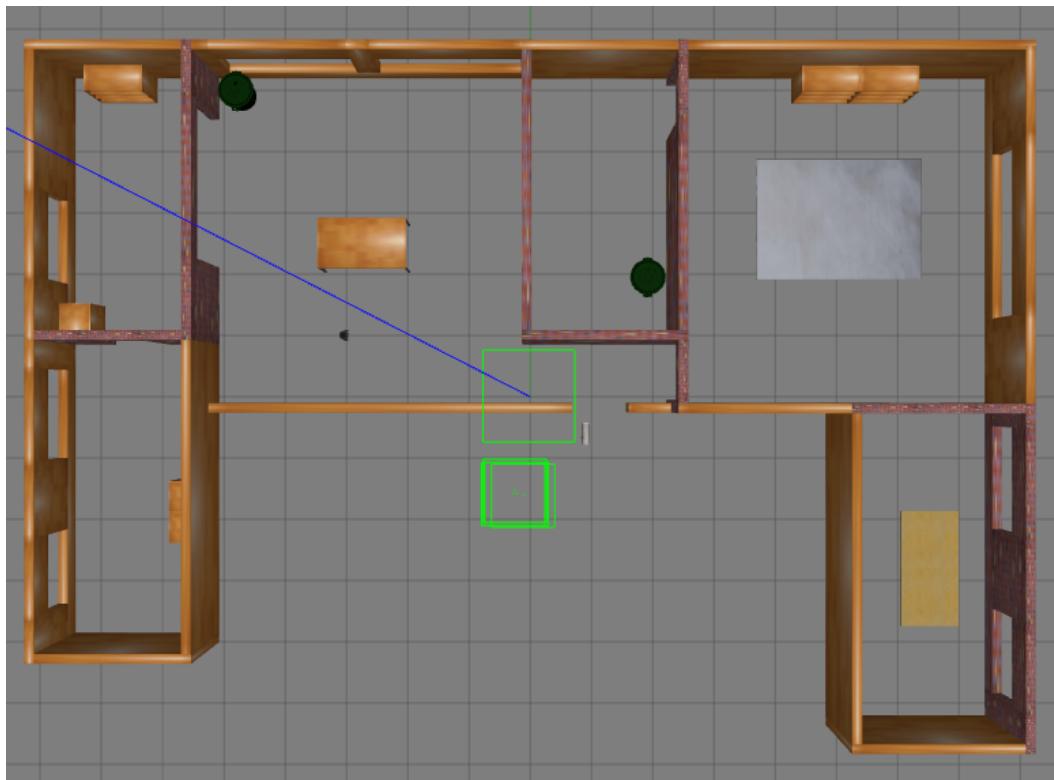


Figura 6.2: Vista Superior House

6.2 Experimentación

Como se ha mencionado previamente, se llevaron a cabo pruebas para cada una de las versiones desarrolladas. Sin embargo, todas estas versiones comparten objetivos comunes, como:

- ¿Es fácil de usar el sistema?
- ¿Se clasifican correctamente los gestos?
- ¿Son intuitivos los gestos aplicados para cada movimiento?
- ¿Es correcto el movimiento del robot?

Respecto al uso de la cámara incorporada en el TurtleBot3 y el mapeado que éste es capaz de hacer se comprobará:

- ¿Se abre correctamente la cámara?
- ¿Funciona la cámara como debería?
- ¿El proceso de mapeado se realiza de manera fluida?

Por último, respecto a la interfaz se busca comprobar los siguientes objetivos:

- ¿La interfaz funciona correctamente?
- ¿Es fácil de utilizar?

6.3 Resultados obtenidos

A lo largo de esta sección se han recogido y mostrado los resultados de comprobar los puntos que se encuentran en el apartado de experimentación. Además se irán mostrando las ventajas y desventajas de cada una de las versiones diseñadas a lo largo de este proyecto.

Durante las pruebas realizadas con la primera versión del sistema, se observó que este era fácil de usar, ya que el robot podía realizar movimientos captados con una sola mano. Sin embargo, la clasificación de gestos no cumplía de manera consistente con su objetivo. Esto se debía a que al clasificar los gestos basándose únicamente en las coordenadas 'X' e 'Y', el sistema a veces confundía los gestos captados, lo que resultaba en movimientos incorrectos por parte del robot. Además, este método de clasificación presentaba limitaciones, ya que al detectar gestos solo con una mano, el robot no podía realizar trayectorias circulares, lo que generaba una sensación de movimiento limitado, pues solo podía avanzar o girar, sin combinar ambos movimientos.

En cuanto a las ventajas identificadas, se destacaba que los gestos resultaban intuitivos para el usuario, lo que facilitaba el uso del sistema y permitía aprender rápidamente cómo controlar el movimiento del robot. Respecto a la detección de los puntos característicos de la mano podemos decir que en esta versión ya se posee un código capaz de detectar estos puntos correctamente.

Tras evaluar las ventajas y desventajas de la primera versión, se decidió implementar mejoras en el sistema para crear la segunda versión del código. En esta iteración, se introdujo la detección de la segunda mano, lo que permitió resolver uno de los problemas más significativos. Esta mejora posibilitó al usuario realizar trayectorias combinadas de avance y giro o retroceso y giro, lo que amplió considerablemente las posibilidades de control del robot. Además, al asignar el gesto de la mano derecha para avanzar o retroceder y el gesto de la mano izquierda para girar, se logró una sensación de movimiento más realista en el TurtleBot3 y facilitó el aprendizaje del sistema para el usuario.

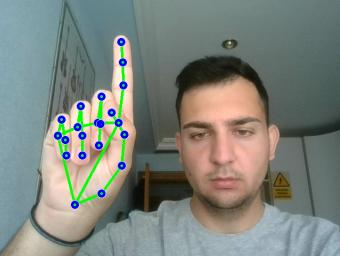
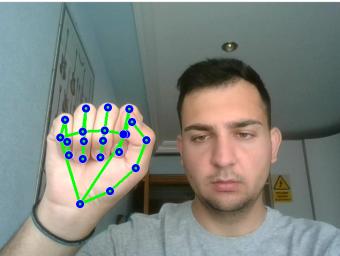
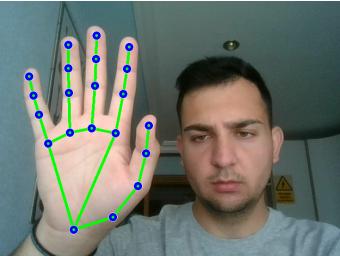
Sin embargo, a pesar de estas mejoras, la clasificación de gestos aún se basaba en las coordenadas 'X' e 'Y' de los extremos de los dedos, lo que ocasionaba errores en la clasificación de gestos. Como resultado, el TurtleBot3 a veces se movía de manera incorrecta al detectar un gesto que el usuario no estaba realizando.

Para la última versión del sistema, se abordó el problema persistente relacionado con la clasificación de gestos. Para resolverlo, se implementó una función para calcular el ángulo entre tres puntos, como se detalló en la sección anterior. Al aplicar esta función, se logró una clasificación de gestos totalmente precisa, lo que garantiza que el TurtleBot3 no realice movimientos incorrectos al detectar el gesto deseado por el operador.

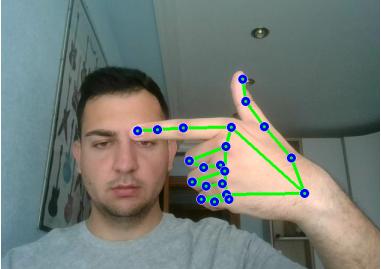
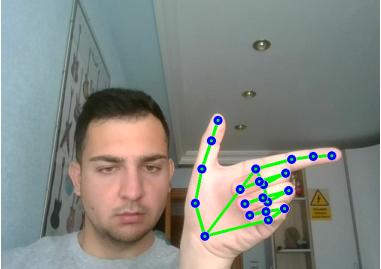
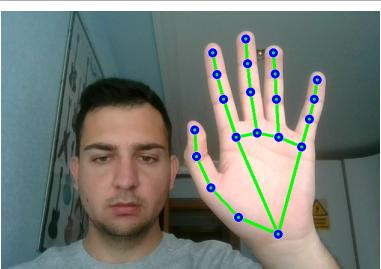
Además, en esta versión se incorporó la detección de obstáculos utilizando el láser. Ajustando la distancia máxima a los objetos, se logró el objetivo principal de detener automáticamente el robot ante la presencia de cualquier objeto a una distancia específica. Esto impide que el usuario pueda mover el robot hacia esa dirección, mejorando así la seguridad y la precisión del movimiento.

Con estas mejoras implementadas en la última versión, se logran los objetivos establecidos en la sección de experimentación. Se obtiene un sistema fácil de usar e intuitivo, con gestos clasificados de manera precisa y un robot que se mueve en la dirección deseada.

Respecto a los gestos utilizados para el movimiento del robot los podemos observar en las tablas 6.1 y 6.2, donde vemos los movimientos para avanzar y retroceder de la mano derecha, y los de giro con la mano izquierda.

Gestos Detectados Mano Derecha
 Movimiento hacia delante
 Movimiento hacia atrás
 Paramos el robot

Cuadro 6.1: Gestos detectados en la mano derecha

Gestos Detectados Mano Izquierda	
	Giro a la derecha
	Giro a la izquierda
	Paramos el robot

Cuadro 6.2: Gestos detectados en la mano izquierda

La inclusión de la cámara en el modelo del TurtleBot3 y su posterior ajuste tenían como objetivo permitir la apertura de la cámara según lo deseara el usuario, con el fin de crear una experiencia más inmersiva. Mostrar al operador lo que el robot está viendo proporciona una sensación de inmersión real en la teleoperación, ya que permite al usuario observar lo mismo que está observando el robot. Un ejemplo de este proceso se muestra en la Figura 6.3, donde en la parte izquierda se visualiza el robot y su dirección de mirada, mientras que en la parte derecha se muestra lo que captta la cámara y se presenta al usuario.

Después de llevar a cabo las pruebas durante la experimentación, nos planteamos si se cumplían los criterios establecidos en el subapartado anterior. En el caso de la cámara, nos preguntamos si se abría correctamente y si funcionaba según lo previsto. Es seguro afirmar que la cámara se abre perfectamente cuando el operador así lo desea, y además funciona correctamente, ya que muestra en tiempo real lo que está captando el TurtleBot3.

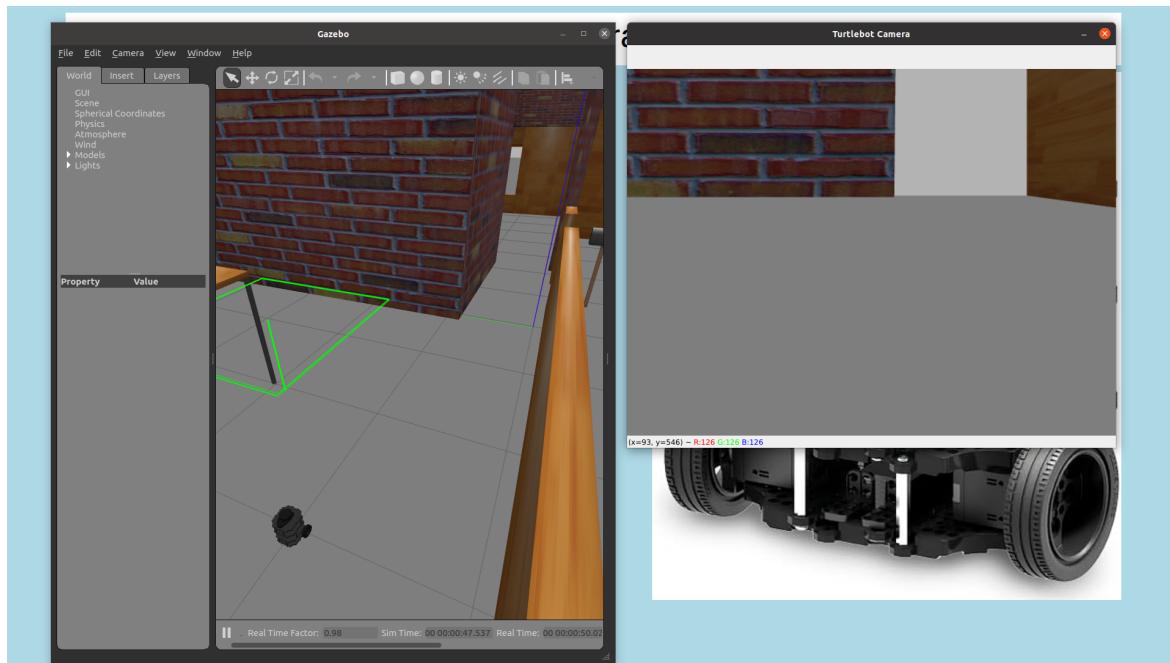


Figura 6.3: Cámara del TurtleBot3

En relación a las pruebas realizadas para el mapeado, nos planteamos si se llevaba a cabo de manera correcta, como se puede observar en el apartado de experimentación. Utilizando el algoritmo Gmapping, nuestro objetivo era determinar si el mapeado lograba detectar adecuadamente todos los objetos presentes en la trayectoria del robot. Además, cualquier mapeado generado podría ser utilizado posteriormente para conocer la posición y trayectoria del robot.

En la Figura 6.4, se muestra el resultado de aplicar Gmapping para mapear una zona desconocida. En el lado izquierdo se observa el mapa creado por Gmapping, mientras que en el lado derecho se muestra el entorno de simulación en Gazebo.

En este caso, se llevó a cabo el mapeado del entorno mostrado en la Figura 6.2, lo que permite comparar el resultado obtenido con el entorno real. Como se puede observar, el mapeado detecta adecuadamente los objetos presentes en el entorno. Los puntos negros en el centro de algunas salas representan las patas de las mesas, los círculos vacíos indican la presencia de cubos de basura, y las líneas presentes en algunas paredes corresponden a las estanterías situadas en esas paredes.

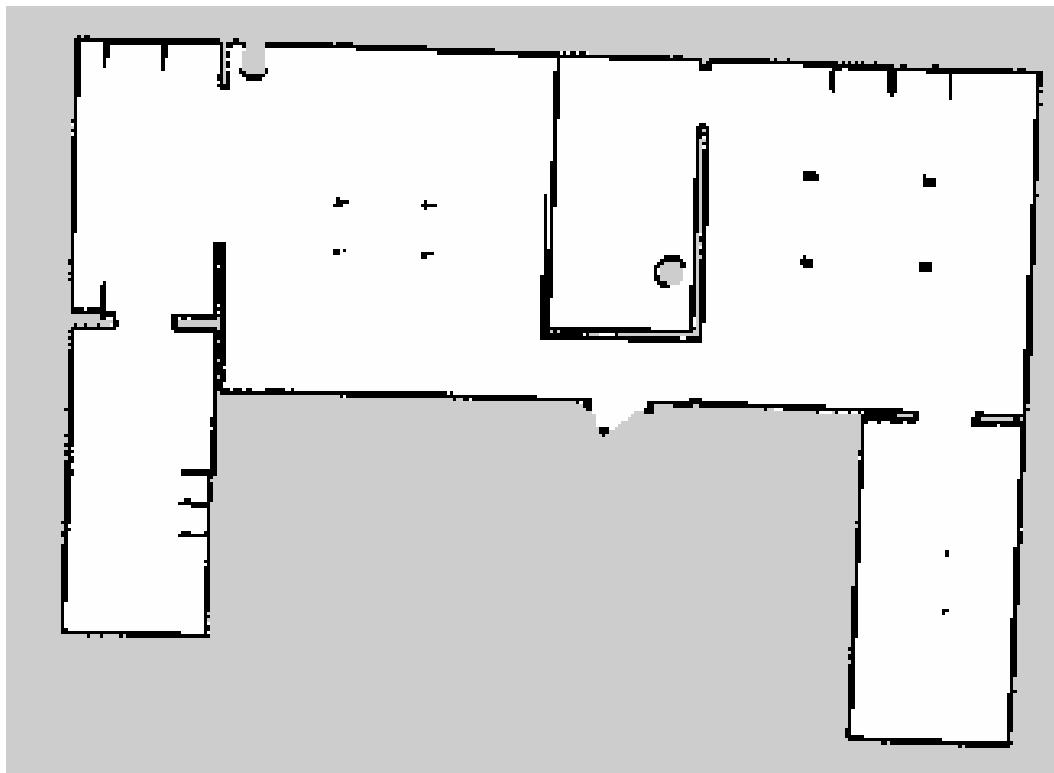


Figura 6.4: Mapeado Completo

Para evaluar los resultados obtenidos de la experimentación con la interfaz, nos centramos en determinar si esta funciona correctamente y si es fácil de utilizar. La interfaz desarrollada se muestra en la Figura 6.5.

En la imagen, se puede apreciar el diseño final de la interfaz, que incluye un cuadro de texto para mostrar la última acción realizada. Además, cuenta con botones para llevar a cabo diversas acciones, como lanzar el mundo de Gazebo (Figura 6.1), mostrar el árbol de transformaciones del robot, realizar el mapeado y guardararlo. También permite abrir un mapa ya creado, reemplazando la imagen del TurtleBot3 por la del mapa correspondiente. Asimismo, se incluyen botones para ejecutar los códigos de detección de gestos, movimiento del TurtleBot3 y apertura de la cámara del robot. Se han agregado botones para cerrar todos los procesos mencionados, así como uno para cerrar la interfaz misma.

Después de evaluar los resultados obtenidos, nos preguntamos si se cumplen los objetivos establecidos para la interfaz. Es evidente que la interfaz es muy fácil de usar, ya que su diseño intuitivo permite al usuario comprender su funcionamiento con solo mirarla. Además, tras realizar varias pruebas, podemos confirmar que la interfaz funciona correctamente. Es decir, al pulsar un botón específico, se ejecuta la función correspondiente tal como se describe en el texto del botón.

En la Figura 6.5 se puede observar los botones explicados en el apartado anterior, con

el cuadro de textos, imagen, etc. Como podemos ver están todos los botones para realizar cualquier proceso mencionado, botones para cerrar cada proceso, el cuadro de texto y el lugar donde se mostrará el mapa o la imagen del TurtleBot3.



Figura 6.5: Interfaz Final

El resultado final se puede observar en el video que aparece en el siguiente enlace de drive, en el que se muestra el funcionamiento general del proyecto: <https://youtu.be/s25wEALuHlk>

7 Conclusiones

En el desarrollo de este proyecto, se ha explorado la implementación de un sistema capaz de teleoperar el TurtleBot3 mediante el reconocimiento de gestos, junto con la integración de funciones adicionales como la detección de obstáculos, la apertura de una cámara y el mapeado del entorno. A lo largo de este trabajo, se han diseñado y probado múltiples versiones del sistema, con el objetivo de mejorar su precisión, funcionalidad y facilidad de uso. En este apartado de conclusiones, se resumirá el proceso de desarrollo, se discutirán los resultados obtenidos y se plantearán posibles vías para futuras investigaciones y mejoras en el sistema.

Tras revisar los objetivos establecidos al inicio de este proyecto, podemos afirmar que cada uno de ellos se ha cumplido con creces. Como se ha demostrado en el apartado anterior, todos los experimentos realizados han permitido detectar gestos y utilizarlos para controlar el movimiento del robot, logrando así el objetivo principal de teleoperar el robot.

En cuanto a la detección y clasificación de gestos, las tecnologías aplicadas en las diferentes versiones del sistema han cumplido completamente con los objetivos establecidos ya que se detectan perfectamente los 21 puntos característicos de las manos además de clasificar los gestos correctamente en la última versión. En relación con las funciones adicionales, como la detección de obstáculos, la apertura de una cámara y el mapeado, podemos afirmar que los conocimientos adquiridos a lo largo del grado nos han permitido cumplir con estos objetivos sin encontrar adversidades.

En cuanto a futuros trabajos relacionados con este proyecto, es importante destacar que los estudios sobre teleoperación continúan evolucionando constantemente, con el objetivo de lograr una teleoperación de robots cada vez más realista. En este contexto, para este proyecto se sugieren varias mejoras para futuras versiones. En primer lugar, se propone la implementación de una red neuronal capaz de clasificar los gestos realizados por el usuario, lo que permitirá una clasificación más precisa que la obtenida en la versión actual del sistema. Además, se plantea la posibilidad de generar una nube de puntos a partir de la información captada por la cámara de profundidad incluida en el modelo del robot. Esta nube de puntos podría ser enviada a unas gafas de Realidad Virtual (VR), lo que proporcionaría una experiencia mucho más inmersiva que la actual, la cual se basa únicamente en la imagen captada por la cámara.

Bibliografía

- [1] C. Preusche, D. Reintsema, T. Ortmaier, and G. Hirzinger, “The dlr telepresence experience in space and surgery,” pp. 35–40, 10 2005.
- [2] J. DelPreto and D. Rus, “Plug-and-play gesture control using muscle and motion sensors,” in *Proceedings of the 2020 ACM/IEEE International Conference on Human-Robot Interaction*, HRI ’20, (New York, NY, USA), p. 439–448, Association for Computing Machinery, 2020.
- [3] “Controlling drones and other robots with gestures.” <https://www.csail.mit.edu/research/controlling-drones-and-other-robots-gestures>, 8 2020.
- [4] M. Oudah, A. Al-Naji, and J. Chahl, “Hand gesture recognition based on computer vision: A review of techniques,” *Journal of Imaging*, vol. 6, no. 8, 2020.
- [5] “Guantes para traducción de lenguaje de signos.” <https://www.forbes.com.mx/tecnologia-guantes-que-traducen-el-lenguaje-de-signos/>.
- [6] M. L. C. Z. Qi, J., “Computer vision-based hand gesture recognition for human-robot interaction: a review.,” 2024.
- [7] E. Stergiopoulou and N. Papamarkos, “Hand gesture recognition using a neural network shape fitting technique,” *Eng. Appl. Artif. Intell.*, vol. 22, pp. 1141–1158, 2009.
- [8] Python, “Página oficial para descarga de python.” <https://www.python.org/downloads/>, 2024.
- [9] OpenCV, “Página oficial para descarga de opencv.” <https://opencv.org/>, 2024.
- [10] Mediapipe, “Página oficial para uso del ejemplo de mediapipe.” https://developers.google.com/mediapipe/solutions/vision/hand_landmarker, 2024.
- [11] U. de Mediapipe, “Explicación mediapipe hands.” <https://omes-va.com/mediapipe-hands-python/>, 2024.
- [12] Tkinter, “Información general tkinter.” <https://keepcoding.io/blog/que-es-tkinter/>, 2024.
- [13] ROS, “Página oficial de ros.” <https://www.ros.org/>, 2024.
- [14] “Información sobre ros.” <https://openwebinars.net/blog/que-es-ros/>, 2024.
- [15] Rospy, “Información sobre rosp.” <http://wiki.ros.org/rospy>, 2024.
- [16] Gazebo, “Página oficial de gazebo.” <https://gazebosim.org/home>, 2024.

- [17] SLAM, “Información general slam.” <https://es.linkedin.com/advice/0/how-can-you-use-ros-slam-skills-ros?lang=es>, 2024.
- [18] Gmapping, “Gmapping: Ros wiki.” <http://wiki.ros.org/gmapping>, 2024.
- [19] Turtlebot, “Página oficial turtlebot.” <https://www.turtlebot.com/>, 2024.
- [20] T. Información, “Robotis e-manual: Turtlebot3.” <https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/>, 2024.
- [21] M. Gmapping, “¿como mapear con gmapping?” https://emanual.robotis.com/docs/en/platform/turtlebot3/slam_simulation/.
- [22] C. Kinect, “Información cámara kinect.” <https://www.vidaextra.com/hardware/kinect-esta-oficialmente-muerto>, 2024.
- [23] M. . G. Developers, “Explicación mediapipe hands.” <https://developers.google.com/mediapipe/solutions>, 2024.
- [24] M. P. Gómez, “Reconocimiento de gestos aplicado al control de sistemas robóticos,” *Repositorio Institucional de la Universidad de Alicante*, 2023.
- [25] Álvaro Martínez Martínez, “Teleoperación de robots humanoides mediante visión artificial y realidad virtual,” *Repositorio Institucional de la Universidad de Alicante*, 2023.
- [26] L. B. V. Emmanuel Nuño Ortega, “Teleoperación: técnicas, aplicaciones, entorno sensorial y teleoperación inteligente.” <https://upcommons.upc.edu/bitstream/handle/2117/570/IOC-DT-P-2004-05.pdf>, 2004.
- [27] L. G. Herrero, “Detección de gestos con las manos orientada a la interacción persona-robot,” *Archivo Digital Universidad Politécnica de Madrid*, Julio 2022.

Lista de Acrónimos y Abreviaturas

2D	2 Dimensiones.
3D	3 Dimensiones.
EMG	Electromiografía.
GMM	Modelo de Mezcla Gaussiana.
GUI	Interfaz Gráfica de Usuario.
HCI	Interacción Humano-Computadora.
HRI	Interacción Humano-Robot.
IA	Inteligencia Artificial.
IR	Infrarrojos.
ML	Machine Learning.
NI	National Instruments.
RGB	Red, Green and Blue.
ROS	Robot Operating System.
SLAM	Simultaneous Localization and Mapping.
TFG	Trabajo Final de Grado.
TFM	Trabajo Final de Máster.
VR	Realidad Virtual.