GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

CREACIÓN DE UN SISTEMA DE CONDUCCIÓN ASISTIDO CON INTELIGENCIA ARTIFICIAL EN TIEMPO REAL

Autor: Esther Usera Ciriza

Director: Nuria Oyaga de Frutos

Madrid

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Creación de un sistema de conducción asistido con Inteligencia Artificial en tiempo real en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico 2023/2024 es de mi autoría, original e inédito y no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

Fdo.: Esther Usera Ciriza Fecha: 14/06/2024

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Nuria Oyaga de Frutos Fecha: 14/ 06/ 2024

GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

CREACIÓN DE UN SISTEMA DE CONDUCCIÓN ASISTIDO CON INTELIGENCIA ARTIFICIAL EN TIEMPO REAL

Autor: Esther Usera Ciriza

Director: Nuria Oyaga de Frutos

Madrid

Agradecimientos

A toda mi familia, en especial a mis padres, a mis hermanos y a mis abuelas por darme todas las oportunidades y apoyarme a lo largo de todos estos años.

A mis amigas y amigos, por motivarme y ayudarme a creer en mí.

A Nokia, por facilitarme todos los recursos para el desarrollo de este proyecto, y por haberme ayudado a ganar nuevos conocimientos de ingeniería.

A mi compañera de trabajo, Amaya, por ofrecer su ayuda siempre que lo he necesitado.

A mi tutora, Nuria, por ayudarme y guiarme a lo largo del proyecto.

CREACIÓN DE UN SISTEMA DE CONDUCCIÓN ASISTIDO CON INTELIGENCIA ARTIFICIAL EN TIEMPO REAL.

Autor: Usera Ciriza, Esther.
Director: Oyaga de Frutos, Nuria.
Entidad Colaboradora: Nokia

RESUMEN DEL PROYECTO

Este proyecto consiste en el desarrollo y programación de un sistema de software utilizando el Robot Operating System (ROS) que permite la aplicación de técnicas avanzadas de inteligencia artificial (IA), en particular, el algoritmo YOLOv5, en la conducción asistida de un robot. La capacidad de percepción, gracias al software desarrollado, permite al robot tomar decisiones de forma adecuada y responder de manera efectiva a la situación en la que se encuentre.

Palabras clave: Robot, Conducción asistida o autónoma, inteligencia artificial, algoritmos, tiempo real

1. Introducción

Las áreas de la robótica y la conducción autónoma están en constante evolución dentro del mundo de las tecnologías modernas. La tecnología de la conducción autónoma ha avanzado considerablemente con herramientas como pueden ser sensores o cámaras.

Uno de los mayores desafíos es crear sistemas de software que permitan al robot ser consciente de su entorno. Para ello, ROS se ha consolidado como una herramienta clave para el desarrollo de software en robótica. En concreto ROS permite el movimiento del robot y puede recibir la información generada por las herramientas de IA para que el robot pueda tomar decisiones ante determinadas situaciones del entorno. Un ejemplo de la integración de las herramientas de IA es el uso del algoritmo YOLO que permite la detección de objetos en tiempo real. En concreto, se utiliza el YOLOv5 que destaca por su velocidad de procesamiento, simplicidad en la implementación, eficiencia en el uso de recursos y, al ser un código abierto, está respaldado por varios desarrolladores y por tanto en constante actualización.[1]

2. Definición del proyecto

Este proyecto consiste en desarrollar un sistema asistido de conducción mediante el uso del sistema operativo ROS y la implementación de IA, utilizando en concreto el algoritmo YOLOv5 para el reconocimiento de objetos y personas que le rodean al robot. El objetivo principal es lograr una conducción autónoma eficiente de manera que el robot pueda navegar y responder de manera correcta a las condiciones del entorno en las que se encuentre.

En primer lugar, se ha llevado a cabo un estudio exhaustivo sobre el sistema operativo ROS para el área de robótica, y sobre el uso del algoritmo YOLOv5 para la IA. Para el análisis de YOLOv5 se ha utilizado un código abierto en Python y se ha incorporado en Docker para manejar la ejecución del código y la obtención de los metadatos obtenidos a partir de las imágenes recibidas.

En segundo lugar, tras entender el funcionamiento de ROS, se ha trabajado en el desarrollo de sistemas de conducción, programando en Python los códigos para gestionar el movimiento del robot. Inicialmente, los códigos se lanzaron en el entorno de simulación Gazebo y una vez conseguido el correcto funcionamiento del robot en dicho entorno, se trasladaron al robot físico. Además, se han realizado los ajustes necesarios para asegurar que el código se adaptaba correctamente a las necesidades reales del entorno operativo.

Finalmente, se ha implementado la comunicación entre el PC servidor y el robot mediante el protocolo UDP para facilitar el intercambio en tiempo real de las imágenes y los metadatos analizados a partir de estas imágenes. El intercambio de esta información se ha conseguido implementando un router 5G configurado para evitar congestiones de red y asegurar una comunicación eficiente y continúa entre ambos.

3. Descripción del modelo/sistema/herramienta

Inicialmente, se desarrolla la parte de IA. Se comienza descargando todas las herramientas necesarias como la nueva versión de Nvidia, confirmar que se tiene la versión de Python adecuada, y el Docker para la parte de Yolo y el Docker de Polyp para permitir el uso de Gstreamer que se encarga de trabajar con aplicaciones multimedia utilizando el framework adecuado, en este caso, para video[2].

Posteriormente, se crean los códigos de ROS asociados a la navegación del robot. Al principio, se desarrolló un código con el objetivo de lanzarlo en el simulador Gazebo. Este código sigue una serie de requisitos como el poder arrancar el robot, avanzar en el eje linear, girar en el eje angular, aumentar la velocidad, y disminuir la velocidad. Una vez comprobado que el robot sigue los requerimientos de forma eficaz en la simulación se trasladó el código al robot físico. A la hora, de trasladarlo, se tuvieron que hacer cambios conforme a las condiciones del robot, como por ejemplo su capacidad de giro en comparación con la del robot en la simulación, ya que no eran el mismo modelo.

Una vez comprobado que el robot físico funciona de manera correcta, se procede a conectar la comunicación entre el ordenador servidor y el robot. Para ello, se integra la comunicación UDP en ambos. Por un lado, el ordenador se integra en la parte de inteligencia artificial para enviar los metadatos analizados de las imágenes enviadas, y, por otro lado, se crea un código nuevo en el robot para recibir los metadatos y en función de ellos analizar si hay una persona o no en la visión del robot. En el caso de que haya una persona, se envía un mensaje al código principal que gestiona la conducción del

robot, para evitar su movimiento del robot. Seguidamente, se realizó una versión más avanzada del código anterior, en la que el robot se frena en caso de que la persona ocupe más de un 60% de la visión del robot. Todos los códigos se lanzaron en un mismo código launch para mantener la organización y eficiencia del proyecto.

A continuación, se muestra un esquema explicando el desarrollo del proyecto:

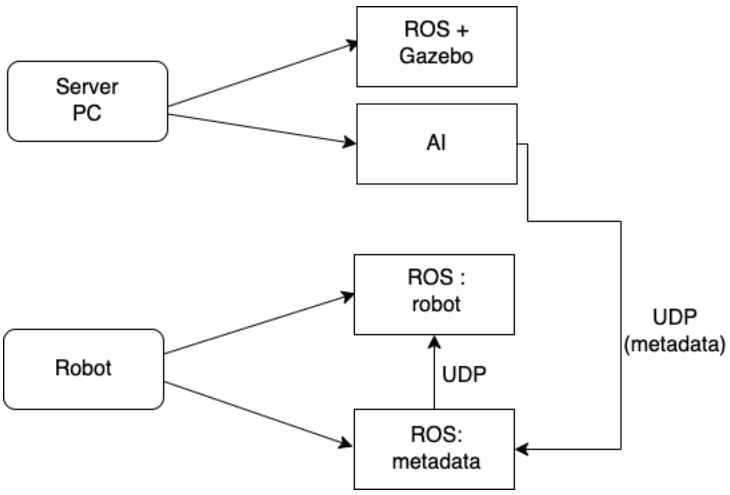

Figura 1 Esquema del proyecto

En el ordenador servidor se gestionó la parte del código ROS ejecutado en el simulador Gazebo, y a su vez se gestionó los códigos que se encargan de lanzar la parte de IA.

En el robot se han utilizado dos códigos ROS. El primero se encargó de la navegación del robot y el segundo se encargó de recibir los metadatos analizados por el algoritmo YOLOv5. Estos se consiguieron por comunicación UDP desde el ordenador servidor. Posteriormente, se programó que, si a través de los metadatos se detecta una persona, se mandará un mensaje UDP al primer código para detener el movimiento del robot.

4. Resultados

Tras conseguir los objetivos propuestos en la parte del software del proyecto, los resultados deben de estar alineados con todo lo descrito en el apartado anterior. En este caso, solo se va a mostrar los resultados de la parte final y la más importante, en la que, si el robot se encuentra a una distancia específica (peligrosa) de una persona, se detiene su movimiento.

```
('OBJ. AREA:', 0.58693359375)
Person detected, still safe
('Received stop message:', 'Person detected, still safe')
```

Figura 2 Resultado del robot a una distancia segura

En el primer caso, la persona ocupa un 58,69% de la visión del robot, por lo que se manda un mensaje al robot que muestra que se detecta a la persona, pero a una distancia segura.

```
('OBJ. AREA:', 0.601015625)
Person detected is too close, stopping robot.
('Received stop message:', 'Person detected is too close, stopping robot')
```

Figura 3 Resultado del robot a una distancia peligrosa

En el segundo caso, la persona ya ocupa más del 60% de la visión del robot, por lo que se manda un mensaje de aviso para detener al robot.

```
---
linear:
  x: 0.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0
---
```

Figura 4 Resultado de la velocidad del robot

Al ejecutar el comando para analizar la velocidad que lleva el robot, se puede observar que tanto en el eje linear como en el eje angular su velocidad es 0.0, por lo que no se detecta ningún tipo de movimiento.

5. Conclusiones

Se han alcanzado los objetivos propuestos al inicio del proyecto, mediante la implementación de un plan estructurado especificando las tareas a realizar en cada mes. Se obtuvo un funcionamiento óptimo en la simulación del robot en Gazebo, lo que permitió el desempeño correcto a la hora de trasladar el código al robot físico. Se ha desarrollado un sistema que no solo permite la navegación del robot, sino que, le habilita reaccionar adecuadamente en respuesta a su entorno por medio de algoritmos de inteligencia artificial. La combinación del sistema operativo ROS y el algoritmo de inteligencia artificial YOLOv5 destaca el potencial del proyecto para su uso en aplicaciones robóticas futuras gestionadas en entornos dinámicos y en tiempo real.

6. Referencias

[1] H. N. Hernández, «Ampliando las capacidades de YOLOv5: desde la detección de objetos hasta la clasificación de audio,» Open Sistemas, 09 10 2023. [En línea]. Available: https://opensistemas.com/ampliando-las-capacidades-de-

yolov5/#:~:text=la%20inteligencia%20artificial.-,YOLOv5%20y%20la%20clasificación%20de%20audio,objetos%20en%20imágenes%20y%20videos. [Último acceso: 10 06 2024].

[2] «¿Por qué GStreamer?,» Fluendo, [En línea]. Available: https://fluendo.com/es/gstreamer/. [Último acceso: 10 06 2024].

CREATION OF A REAL-TIME ARTIFICIAL INTELLIGENCE ASSISTED DRIVING SYSTEM.

Author: Usera Ciriza, Esther.
Supervisor: Oyaga de Frutos, Nuria.
Collaborating Entity: Nokia

ABSTRACT

This project consists of the development and programming of a software system using the Robot Operating System (ROS) that allows the application of advanced artificial intelligence techniques, particularly the YOLOv5 algorithm, in the driving of a robot. The ability to perceive, thanks to the software developed, allows the robot to make decisions appropriately and respond effectively to the situation in which it finds itself.

Keywords: Robot, Assisted or Autonomous Driving, Artificial Intelligence, Algorithms, Real Time

1. Introduction

The areas of robotics and autonomous driving are constantly evolving within the world of modern technologies. Autonomous driving technology has advanced considerably with tools such as sensors or cameras.

One of the biggest challenges is to create software systems that allow the robot to be aware of its surroundings. To this end, the Robot Operating System (ROS) has established itself as a tool for the development of software in robotics, specifically to allow the movement of the robot and to be able to receive the information generated by Artificial Intelligence (AI) tools so that the robot can respond to it. An example of the integration of AI tools is the use of the YOLO algorithm that allows the detection of objects in real time. Specifically, the version YOLOv5 is used, which stands out for its processing speed, simplicity in implementation, efficiency in the use of resources and being an open source, it is supported by several developers and therefore constantly updated. [1]

2. Definition of the project

This project consists of developing an assisted driving system using the ROS operating system and the implementation of artificial intelligence, specifically using the YOLOv5 algorithm for the recognition of objects and people around the robot. The main objective is to achieve efficient autonomous driving so that the robot can navigate and respond correctly to the conditions in which it finds itself.

First, an exhaustive study has been carried out on the ROS operating system for the robotics part, and for AI part, the YOLOv5 algorithm. For the analysis of YOLOv5, an open-source code in Python has been used and incorporated into Docker to handle the execution of the code and the obtaining of the metadata obtained from the images received.

Secondly, after understanding how ROS works, development focused on the driving systems. Python was used to program the codes managing the robot's movement. Initially, the codes were launched in the Gazebo simulation environment. Once the robot operated correctly in the simulation, the codes were transferred to the physical robot. Necessary adjustments were made to ensure that the code is correctly adapted to the real needs of the operating environment.

Lastly, communication between the server PC and the robot will be implemented using the UDP protocol to facilitate the real-time exchange of the images and their analyzed metadata. The exchange of this information will be achieved by implementing a 5G router configured to avoid network congestion and ensure efficient and continuous communication between the two.

3. Description of the model

To begin, the AI part is developed. You start by downloading all the necessary tools such as the new version of Nvidia, the appropriate version of Python, and the Docker for the Yolo part and the Polyp Docker to allow the use of GStreamer that is responsible for the creation of multimedia applications [2], in this case, video.

Subsequently, the ROS codes associated with the robot's navigation are created. At first, a code was developed with the aim of releasing it in the Gazebo simulator. This code follows a series of requirements such as being able to start the robot, advance on the linear axis, rotate on the angular axis, increase speed, and decrease speed. Once the requirements were met effectively in the simulation, the code was transferred to the physical robot. Adjustments had to be made for the code to adjust to the conditions of the robot, such as its turning capacity since the robot of the simulation is a different model.

Once the physical robot is verified to be functioning correctly, UDP communication is established between the server computer and the robot. The server computer integrates AI to send the analyzed metadata of the images. Meanwhile, a new code is created in the robot to receive this metadata and analyze whether there is a person within robot's vision. If there is a person, a message is sent to the main code that manages the robot's driving, to prevent its movement from the robot.

An advanced version of the previous code was made, in which the robot brakes in case the person occupies more than 60% of the robot's vision. All codes are launched in the same launch code, to maintain the organization and efficiency of the project.

Below is an outline explaining the development of the project:

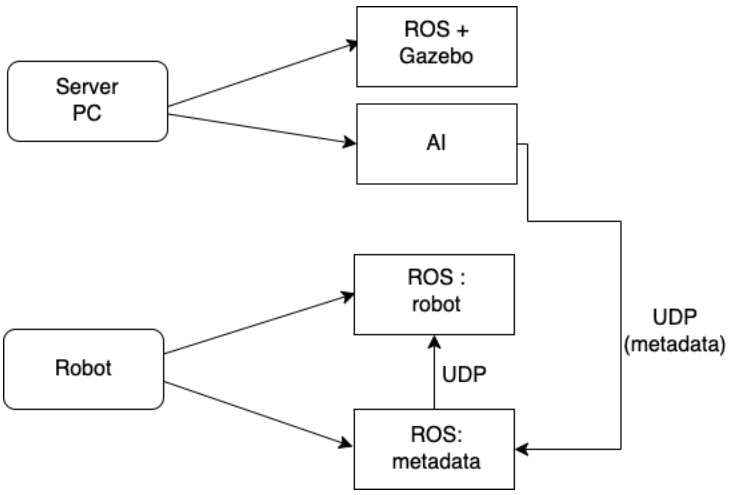

Figure 1 Outline of the project

The server computer manages the part of the ROS code executed in the Gazebo simulator and most importantly, it manages the codes that are responsible for launching the artificial intelligence part.

Two ROS codes are used in the robot. The first code manages the robot's navigation, while the second receives the metadata analyzed by the YOLOv5 algorithm. This is achieved by UDP communication from the server computer. Subsequently, if a person is detected in the metadata, a UDP message will be sent to the first code to stop the robot's movement.

4. Results

After achieving the goals proposed in the software part of the project, the results must be aligned with everything described in the previous section. In this case, only the results of the final and most important part will be shown, in which, if the robot is at a dangerous/ specific distance from a person, its movement is consequently stopped.

```
('OBJ. AREA:', 0.58693359375)
Person detected, still safe
('Received stop message:', 'Person detected, still safe')
```

Figure 2 Result of a safe distance detection

In the first case, the person occupies 58.69% of the robot's vision. A message is sent to the robot showing that the person is detected, but at a safe distance.

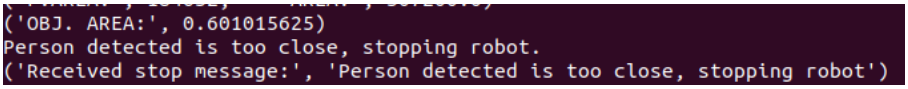

Figure 3 Result of a dangeorus distance detection

In the second case, the person occupies more than 60% of the robot's vision. Consequently, a warning message is sent to stop the robot.

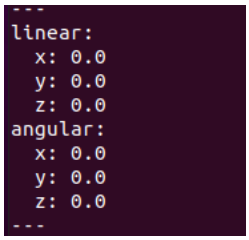

Figure 4 Result of the robot's velocit

When executing the command to analyze the robot´s speed, it is observed in Figure 4 that both the linear and angular axes have a speed of 0.0, so no movement is detected.

5. Conclusions

The objectives proposed at the beginning of the project have been achieved through the implementation of a structured plan specifying the tasks for each month. Optimal performance was obtained in the simulation of the robot in Gazebo, which allowed the correct performance when transferring the code to the physical robot. A system has been developed that not only allows the robot to navigate, but also enables it to react appropriately in response to its environment through artificial intelligence algorithms. The combination of the ROS operating system and the YOLOv5 artificial intelligence algorithm highlights the project's potential for future robotic applications in dynamic and real-time environments.

6. References

[1] H. N. Hernández, «Ampliando las capacidades de YOLOv5: desde la detección de objetos hasta la clasificación de audio,» Open Sistemas, 09 10 2023. [En línea]. Available: https://opensistemas.com/ampliando-las-capacidades-de-yolov5/#:~:text=la%20inteligencia%20artificial.-

,YOLOv5%20y%20la%20clasificación%20de%20audio,objetos%20en%20imágenes%20y%20videos. [Último acceso: 10 06 2024].
[2] «¿Por qué GStreamer?,» Fluendo, [En línea]. Available: https://fluendo.com/es/gstreamer/. [Último acceso: 10 06 2024].

Índice de la memoria

Índice de figuras

Índice de Tablas

Capítulo 1. INTRODUCCIÓN

La robótica y la inteligencia artificial (IA) han emergido como fuerzas transformadoras en la ingeniería y en la tecnología informática. La combinación de ambas está redefiniendo nuestro mundo a un ritmo acelerado y abriendo una puerta hacia nuevas posibilidades.

Una de las áreas donde la robótica y la IA están presentes es en el ámbito de la conducción asistida. Gracias a la evolución tecnológica, se están desafiando los límites del transporte en términos de seguridad, eficiencia y comodidad. En este sentido la conducción asistida representará un gran avance en la movilidad del futuro.

La combinación de programas gestionados a través de algoritmos de IA con sistemas de procesamiento de datos permite poder desarrollar sistemas robóticos avanzados que disponen de conducción asistida en tiempo real. Estos sistemas permiten la toma de decisiones autónoma mejorando la seguridad y eficacia en la conducción. A su vez, poder integrarlo en tiempo real asegura una respuesta rápida y precisa con respecto a los cambios en el entorno. La conducción autónoma gracias a estas tecnologías está avanzando en la capacidad de prescindir del mismo conductor, con el con siguiente salto cualitativo que esto supone en el transporte en un futuro.

Este proyecto proporciona investigación e información adicional sobre el futuro de la conducción autónoma. Los resultados permiten su aplicación en diversos ámbitos de la ingeniería, desde el área de exploración espacial hasta el área de medicina asistida.

Capítulo 2. ESTADO DE LA CUESTIÓN.

Este capítulo se divide en dos partes. La primera hace referencia a la historia de la robótica, la inteligencia artificial y el metaverso hasta la actualidad. La segunda parte se centra en los códigos extraídos y utilizados de internet para el desarrollo del proyecto. En primer lugar, se explicarán los códigos asociados al algoritmo YOLO de Inteligencia Artificial y, posteriormente, a aquellos relacionados con el sistema operativo de código abierto, ROS. [1]

2.1 EVOLUCIÓN DE LA ROBÓTICA, IA Y EL METAVERSO.

2.1.1 LA ROBÓTICA.

La historia de la robótica se remonta a miles de años atrás, con los primeros indicios de las antiguas civilizaciones replicando modelos en los que se plasman los movimientos y comportamientos de los seres humanos.

En la antigüedad, civilizaciones como la egipcia y la griega desarrollaron artefactos mecánicos con variedad de propósitos. Por ejemplo, en el caso de la Antigua Grecia, el famoso ingeniero y matemático Arquímedes descubrió el poder de las palancas, las poleas y los planos inclinados en el siglo III a.C.[2] Estos fueron de los primeros mecanismos que simulaban en cierta medida movimientos del ser humano, entre ellos, mecanismos que actuaban como brazos y manos humanas.

Durante la Edad Media y el Renacimiento la ingeniería experimentó cambios y avances significativos. En el siglo XIII se construyó lo que se conoce como el primer androide de la historia, el hombre de hierro de Alejandro Magno. Construido con materiales de hierro, cristal y cuero, el autómata tenía la capacidad de realizar tareas como atender la puerta del monasterio o entretener a los visitantes. [2]

En el año 1495, Leonardo da Vinci revolucionó la historia de los autómatas al diseñar el "*Automa Cavaliere*". Esta creación, rodeado de una armadura completa, tenía la capacidad

de desarrollar una amplia gama de movimientos, desde sentarse y levantarse hasta mover los brazos y las piernas. [2]

Ambas creaciones de seres autómatas, capturaron la creatividad y la imaginación para el desarrollo futuro de la robótica actual.

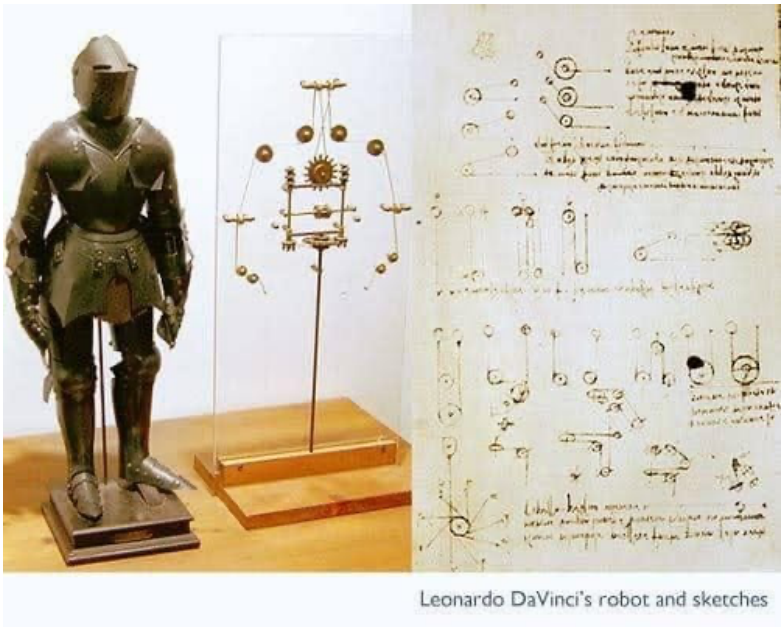

Figura 5 Automa Cavaliere [3]

En el siglo XVII, gracias a la contribución del filósofo y matemático alemán Gottfried Von Leibniz, se tuvo un avance crucial en el mundo de la robótica y programación. Este avance fue la creación del sistema binario, estableciendo fundamentos de cálculo automático y estableciendo las bases para el desarrollo posterior de la computación y de los lenguajes programación basados en sistemas binarios.[2]

Tras estos acontecimientos, se desencadenó la Revolución Industrial, un período clave de la historia humana para el mundo de la ciencia y la ingeniería en sí. En este periodo surgieron máquinas con la capacidad de realizar tareas humanas con velocidad y exactitud, revolucionando los procesos de producción y los niveles de eficiencia.

La Revolución Industrial marcó un impulso hacia la creación de máquinas cada vez más autónomas, mejorando y aumentando los niveles de producción considerablemente. Además, no solo contribuyeron en cuanto a eficiencia, sino que también aliviaron la carga laboral de los seres humanos. [4] Un ejemplo de estas máquinas es el Telar de Jacquard, considerado como uno de los inicios de los ordenadores modernos[5].

Figura 6 Telar de Jacquard [5]

La robótica, como se entiende hoy en día, comienza en los años cincuenta del siglo XX. Este periodo marca un punto de inflexión para el surgimiento de los primeros robots, impulsados por avances en electrónica y computación. En paralelo, la investigación comienza a avanzar en la búsqueda de desarrollar IA para procesar información con una velocidad y eficacia similar al ser humano. [4]

En 1950 Alan Turing publica "*Computing Machinery and Intelligence*", donde establece una prueba, conocida como la prueba de Turing. En esta prueba se evalúa si una máquina es capaz de imitar el comportamiento inteligente de un ser humano. Consiste en un intercambio de mensajes donde si una persona no puede distinguir si está hablando con otra persona o con una máquina, entonces significa que la máquina ha pasado la prueba. Esta prueba, se

utiliza como estándar para evaluar el futuro de la IA y la capacidad de las máquinas a igualar las capacidades humanas.[6]

Anterior a este suceso el escritor de ciencia de origen ruso Isaac Asimov dictó las tres famosas leyes de la robótica anticipando el futuro[4][7]:

1. Ética y seguridad del robot: un robot tiene prohibido dañar a un ser humano, ya sea mediante acción directa o inacción que permita el sufrimiento del ser humano.
2. Responsabilidad del robot: un robot debe de obedecer las órdenes impartidas por el ser humano, salvo que estén asociadas con la primera ley.
3. Autonomía y Protección: un robot debe de proteger su propia existencia siempre y cuando se cumplan las dos primeras leyes.

A lo largo del siglo XX, se logró el diseño del primer robot programable. George Devol fue el ingeniero detrás de este logro. Según documentos, a partir de Devol es cuando puede denominarse robot a una máquina.[4] El primer robot programable se conoce como el PUMA (Programmable Universal Machine for Assembly), capaz de manipular objectos y ubicarlos en la posición deseada.[8]

Figura 7 George Devol junto al robot Puma [9]

Se observó un continuo desarrollo de diversidad de robots, cada robot con una amplia gama de aplicaciones. Entre ellos, destacan el primer brazo robótico creado en 1971. Durante la misma década, se lograron significantes avances en la robótica espacial, incluyendo el lanzamiento de las sondas Voyager I y II.[4]

A partir de la década de los 80 y 90, el progreso tecnológico marcó una revolución al superar las limitaciones del robot industrial por medio de la integración de la visión artificial. Esto marcó un hito en el mundo de la robótica ya que los robots además de simular movimientos del ser humano iban a ser capaces de comprender y procesar información sobre los objetos presentes en su entorno.[4]

2.1.1.1 La robótica en la conducción.

Los primeros robots en ganar reconocimiento en la mayoría de la población van a ser los coches autónomos, destinados para circular sin intervención humana. A lo largo del siglo XXI, la tecnología en los coches ha avanzado desde poder aparcar solos hasta frenar a cierta distancia para poder evitar un atropello. Aun así, quedan grandes avances en este mundo ya que, hoy en día no existen coches completamente autónomos, que puedan circular en todo tipo de carreteras y situaciones meteorológicas. [10]

Para avanzar con la tecnología de conducción autónoma se va a necesitar trabajar con sistemas como sensores ultrasónicos, radares, sensores infrarrojos para visión nocturna, posicionamiento y navegación por satélite para la precisión de la localización, LiDAR para analizar su entorno. El mayor desafío actualmente es desarrollar software que ayude al coche interpretar su entorno. Mediante los algoritmos de aprendizaje automático que componen la IA, se interpretarán correctamente las imágenes observadas por el coche y de esta forma, se podrán tomar decisiones adecuadas a la situación. [10]

2.1.2 LA INTELIGENCIA ARTIFICIAL

En el siglo XXI, nos encontramos ante una era dominada por la Inteligencia Artificial (IA o AI), la cual ha transformado radicalmente nuestra interacción con la tecnología. Los robots

se encuentran presentes en nuestro día a día, desde robots cirujanos en hospitales hasta vehículos sin conductor.

Según el periódico, La Vanguardia, se dice que estamos en la Cuarta Revolución Industrial, donde dos de los componentes clave son el desarrollo de la IA y el funcionamiento autónomo de las máquinas.[11] Esta revolución destaca una relación de colaboración ente robots y personas para potenciar la humanidad del futuro.

No fue hasta 1950 cuando Alan Turing planteó la pregunta fundamental "¿Pueden pensar las máquinas?" en el famoso artículo de "*Computing Machinery and Intelligence*", mencionado anteriormente.[12] La prueba de Turing tenía como objetivo fundamental determinar si una máquina podía exhibir un comportamiento inteligente similar al del humano. A posteriori, John McCarthy desarrolló el primer lenguaje de programación de IA, LISP. A lo largo de las décadas siguientes, hubo avances significativos en algoritmos y técnicas de aprendizaje automático, apoyando la financiación y el continuo desarrollo de sistemas más complejos de IA.[12]

En la década de 1990, gracias al avance en la informática y en la capacidad de procesar de forma cada vez más eficiente grandes cantidades de datos, se alcanzaron algoritmos de aprendizaje que sentaron las bases de la IA contemporánea. Estos avances han provocado un crecimiento exponencial de esta tecnología, generando aplicaciones como el reconocimiento de imágenes y de voz, el procesamiento del lenguaje natural y los sistemas autónomos. [12]

En la actualidad, la IA no se enfoca en generar nuevo conocimiento, sino en recolectar y analizar los datos obtenidos para poder tomar decisiones a partir de ellos.

La compañía española Iberdrola, índice que la IA se apoya en tres elementos básicos[12]:

1. Los datos representan la información recopilada y estructurada.
2. El hardware constituye el procesamiento que nos permite trabajar con una gran cantidad de datos a una velocidad y precisión superiores.

3. El software compone un conjunto de instrucciones y algoritmos que permiten el entrenamiento de sistemas para procesar datos, analizar patrones y generar nueva información.

La evolución tecnológica de estos tres elementos está permitiendo el crecimiento exponencial de la IA.

2.1.3 EL METAVERSO.

El metaverso es una experiencia digital en red, parcial o totalmente inmersiva, que reúne a personas, lugares, objetos y/o información en tiempo real, trascendiendo lo posible en el mundo físico.[15] El objetivo del metaverso es mejorar y optimizar la calidad de vida del ser humano.

Según un artículo de la empresa Nokia, se hace referencia a múltiples metaversos. Se definen tres tipos de metaverso cada uno con propósitos y prioridades diferentes[15].

1. Metaverso del consumidor.
2. Metaverso industrial.
3. Metaverso empresarial.

Cada metaverso se conecta en diferentes grados, pero, en cualquier caso, se comparten tecnologías, dispositivos e interfaces. Por un lado, el metaverso del consumidor se enfocará en actividades de entretenimiento y ocio, mientras que, por el otro lado, los metaversos industriales y empresariales se centrarán en aplicaciones más orientadas a los negocios. El metaverso empresarial centrará sus objetivos en aplicaciones de tecnología de la información, y el metaverso industrial abordará aplicaciones de tecnología operativa[15].

Un ejemplo destacado del metaverso es la realidad virtual (RV). Durante las décadas de los años 80 y 90, la RV comenzó a formarse como una tecnología concreta. Jaron Lanier, conocido como el "padre de la realidad virtual", estableció la empresa VPL Research creando mecanismos tecnológicos como guantes y gafas de RV que acercaban al usuario con la interacción de la RV.[13]

En el año 2010, se revolucionó el mundo de la realidad virtual con la empresa Oculus VR fundada por Palmer Luckey. Se presentó modelo Oculus Rift, un dispositivo de realidad virtual que proporciona a sus usuarios unq visión 360 en un mundo virtual en 3D. [13]

En la actualidad el producto contiene 3 dispositivos. Se incluye el casco de RV rift, el Oculus touch, y dos sensores Oculus. De esta forma se puede observar, manipular objetos, y experimentar sensaciones de manera inmersiva. Para poder acceder al metaverso los usuarios necesitaran los dispositivos que se muestran en la Figura 8. El objetivo es obtener una conectividad adecuada para vincular los dispositivos con la plataforma y la aplicación que se ejecutan en el entorno del metaverso.[14]

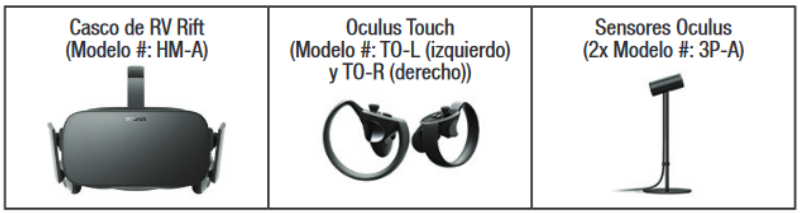

Figura 8 Productos asociados al Oculus Rift [14]

2.2 CÓDIGOS EXTERNOS UTILIZADOS.

En el presente apartado se abordará el uso de códigos desarrollados por terceros a lo largo del proyecto. En primer lugar, se detallarán los códigos relacionados con la inteligencia artificial que han sido utilizados. En segundo lugar, se expondrán los códigos relacionados con el área de robótica, abarcando tanto aquellos empleados en la simulación como en la implementación del robot físico.

2.2.1 CÓDIGOS IA.

2.2.1.1 YOLOv5.

Para poder implementar toda la parte de inteligencia artificial focalizada en detección de objetos se utilizará el algoritmo YOLO (You Only Look Once). Este algoritmo ofrece un avanzado sistema de detección de objetos en tiempo real de código abierto, haciendo uso de una única red neuronal convolucional. Su funcionamiento se basa en la división de la imagen en regiones, lo que se consigue por medio de predicción de cuadros de identificación y probabilidades por región. Este algoritmo tiene la capacidad de aprender representaciones generalizables de los objetos. Gracias a este funcionamiento se tiene una baja tasa de error en la detección de objetos para entradas nuevas y distintas al conjunto de datos de entrenamiento. [16]

En el algoritmo YOLO destacan principalmente las siguientes características: velocidad, precisión de detección, buena generalización y código abierto. [17]

YOLO se caracteriza por su notable velocidad, gracias a un enfoque simplificado que previene de complejas canalizaciones. De esta forma, el algoritmo puede procesar imágenes a una velocidad de 45 fotogramas por segundo. Adicionalmente, alcanza una precisión media (mAP) que supera a la de otros sistemas y lo posiciona como una opción destacada en el procesamiento en tiempo real. [17]

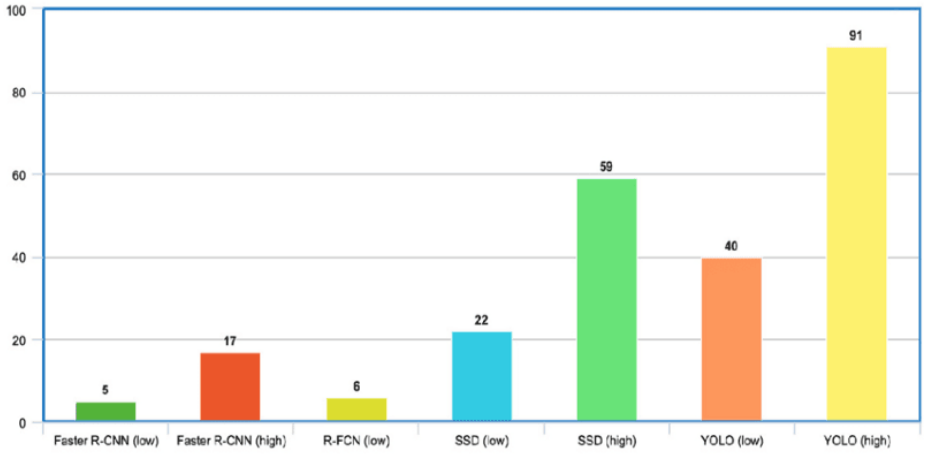

Figura 9 Velocidad de YOLO en comparación con otros detectores de objetos [17]

El algoritmo ofrece una alta precisión en la detección de objetos minimizando los errores de fondo, y a su vez destaca en términos de generalización. Esto es especialmente útil para aplicaciones que requieren una detección de objetos rápida y robusta.[17]

Por último, la decisión de convertir YOLO en código abierto ha facilitado un trabajo colaborativo, permitiendo así un progreso constante del modelo. Esta iniciativa ha optimizado de manera rápida y eficiente el avance y mejora del algoritmo. [17]

En cuanto a la arquitectura de la red neuronal, consta de 24 capas convolucionales, 4 de agrupamiento máximo y 2 completamente conectadas.[17]

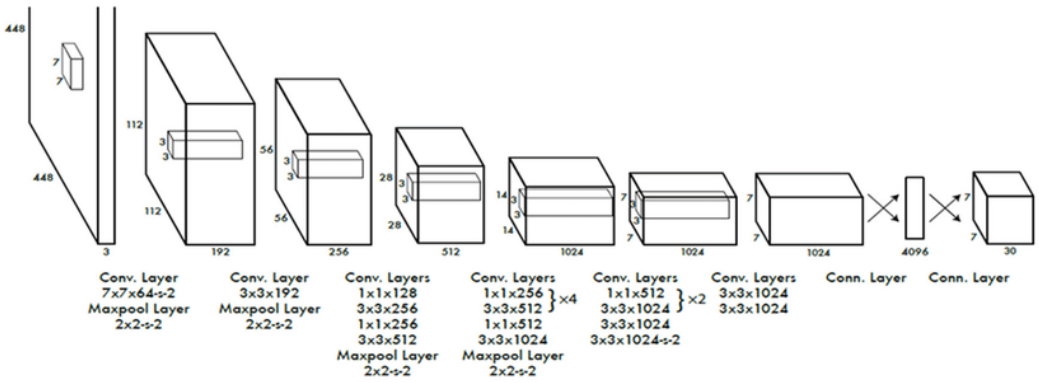

Figura 10 Arquitectura de YOLO [17]

El funcionamiento de la red se resume en los siguientes pasos: en primer lugar, se ajusta el tamaño de la imagen a 448x448. A posteriori, se aplican convoluciones de 1x1 y 3x3 que reducen los canales y generan una salida cuboidal. En todas las capas se utiliza la función de activiación ReLU, excepto en la última, que emplea una función lineal.[17]

YOLO se basa en varias etapas para la detección de objetos en imágenes. En primer lugar, divide la imagen en celdas de cuadrícula NxN, en la que cada una de ellas se responsabiliza de predecir la clase y confianza del objeto que abarca el área.[17] Posteriormente, se determinan las cajas delimitadoras que se encargan de distinguir a los objetos destacados.

Este proceso se consigue mediante un módulo de regresión que asigna atributos como la posición, la altura o el tamaño de cada caja.[17] En el caso de este proyecto, el cuadrado delimitador va a ser de la siguiente forma:

$$Y= [bx,by,bh,bw]$$

- bx, by corresponden con las coordenadas x e y de la caja delimitadora.
- bh, bw corresponden con la altura y la anchura de la caja delimitadora.

En la Figura 11, se muestra el proceso de cómo calcular las coordenadas de caja de una imagen de 448x448:

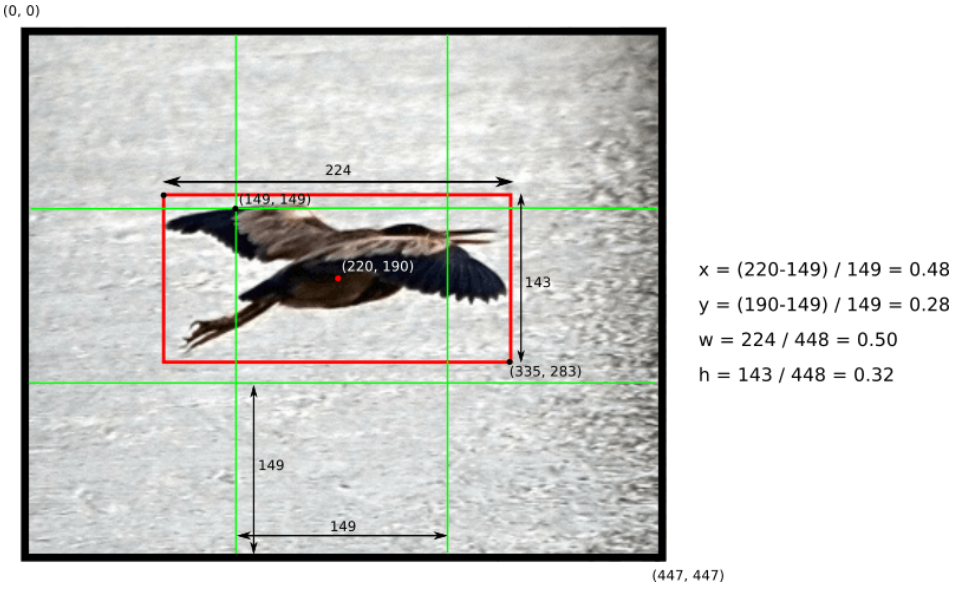

Figura 11 Cálculo de coordenadas de una imagen 448x448. [18]

Además, se utiliza la intersección sobre la unión, también conocido como IOU. Este valor que varía entre 0 y 1, tiene el objetivo de filtrar cajas redundantes para conservar sólo aquellas que sean relevantes. El usuario debe de definir un umbral y considerar solo aquellas celdas que contengan un IOU>umbral. [17]

Finalmente, se utiliza la Supresión No Máxima (NMS) que se encarga de eliminar cajas redundantes y conservar aquellas que contengan la mayor puntuación de probabilidad de detección. El NMS se aplica ya que hay casos en los que el IOU no es suficiente y de esta forma se elimina el ruido de las casillas redundantes de detección de objetos.[17]

2.2.1.2 Código YOLOv5

En la siguiente sección, se destacarán los códigos implementados en relación con el algoritmo YOLOv5, así como su funcionamiento.

El código empleado para la implementación utilización del algoritmo YOLOv5 que se ha usado en este trabajo ha sido desarrollado por el autor Glenn Jocher y está disponible en su repositorio de GitHub. [19]

Para descargar el proyecto, en el propio GitHub el autor incluye una serie de pasos a seguir en el apartado "*Documentation*":

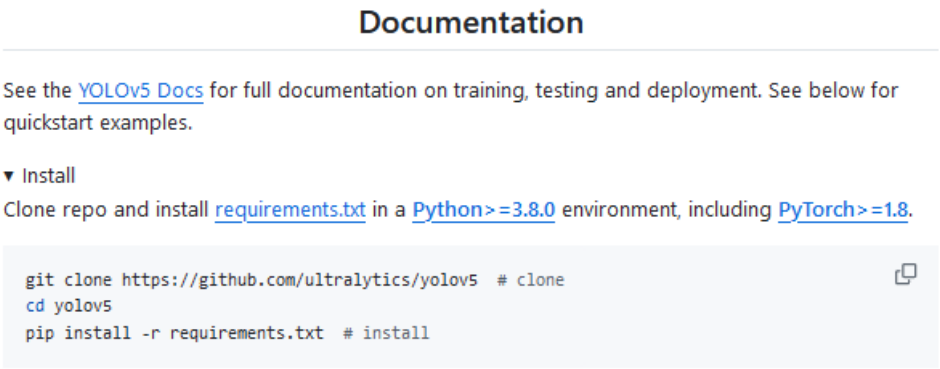

Figura 12 Documentación de proceso de instalación YOLOv5 [19]

Se necesita instalar *requirements.txt* para tener los paquetes fundamentales necesarios en el funcionamiento de YOLOv5 funcione. Entre estos paquetes están:

- gitpython: permite la interacción de Git con Python. [20]

- Matplotlib: es una biblioteca que se encarga de la generación de gráficos en dos dimensiones.[21]
- Numpy: biblioteca en Python que proporciona funciones matemáticas.[22]
- Opencv-python: biblioteca de software de visión por computadora y aprendizaje automático de código abierto.[23]
- Pillow: permite abrir, manipular y guardar imágenes.[24]
- Psutil: se encarga de monitorear procesos del sistema. (cpu, memoria, etc) [25]
- PyYAML: sirve para manejar archivos YAML, un lenguaje que se encarga de serializar datos. [26]
- Requests: permite realizar solicitudes HTTP.
- Scipy: contiene herramientas y algoritmos matemáticos. [27]
- Torch: contiene el PyTorch requerido 1.8.0.
- Tqdm: utiliza algoritmos para predecir tiempos. [28]
- Ultralytics: herramientas específicas para crear y entrenar modelos de aprendizaje. [29]

A su vez, en la documentación se incluye la línea de comandos para poder ejecutar el script principal del proyecto para la detección de objetos:

```
python detect.py --weights yolov5s.pt --source 0                              # webcam
                                            img.jpg                            # image
                                            vid.mp4                            # video
                                            screen                             # screenshot
                                            path/                              # directory
                                            list.txt                           # list of images
                                            list.streams                       # list of streams
                                            'path/*.jpg'                       # glob
                                            'https://youtu.be/LNwODJXcvt4'     # YouTube
                                            'rtsp://example.com/media.mp4'     # RTSP, RTMP, HTTP strea
```

Figura 13 Documentación de cómo ejecutar el script [19]

El código *detect.py* ejecuta la detección de objetos mediante el algoritmo YOLOv5 en fuentes como imágenes, videos, o directorios. El código carga el modelo, procesa las imágenes para la detección de los objetos e incluye la funcionalidad de poder visualizar las

detecciones en tiempo real, que es uno de los objetivos principales del proyecto. Para la configuración de parámetros como el tamaño de la imagen o filtros de clases se utilizará el formato de modelo PyTorch. PyTorch es una plataforma de código abierto utilizada para construir redes neuronales, integrando la biblioteca de aprendizaje automático Torch con una Api de alto nivel en Python. [30]

Las partes fundamentales del código se explican en el capítulo 4, ya que se modifican algunas partes del código para poder obtener los resultados requeridos.

Respecto a la elección de YOLOv5, en el repositorio de GitHub de Glenn-jocher se muestra una Figura que justifica el uso de este algoritmo frente a otros:

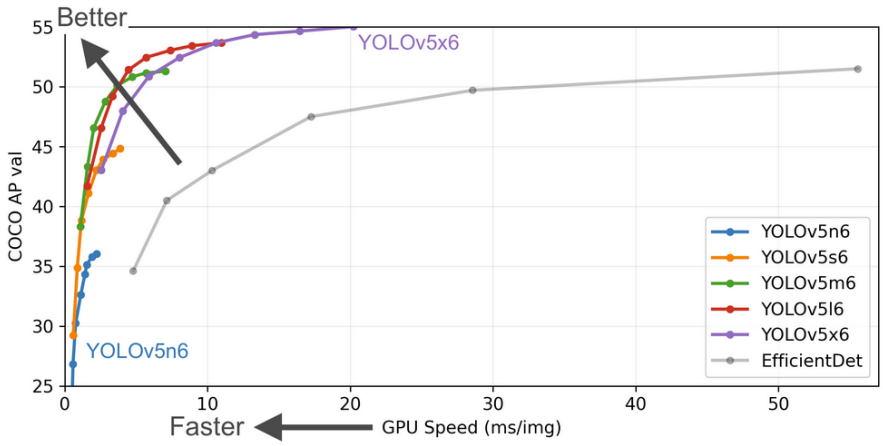

Figura 14 Eficiencia de YOLOv5 [19]

Una de las mejoras en la arquitectura de YOLOv5 fue la integración de la capa Focus, por medio de la cual se reduce el número de capas y parámetros. Adicionalmente se incrementa la velocidad de avance y retroceso sin afectar de manera considerable al mAP.[17]

A continuación, se muestra una Figura en la que se compara el tiempo de entrenamiento entre la versión 4 y la 5:

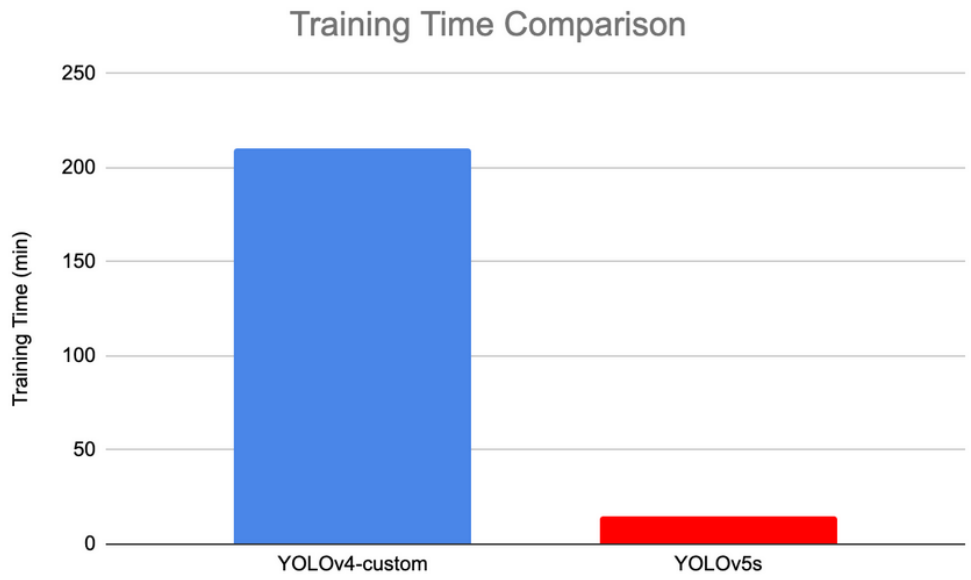

Figura 15 Comparación del tiempo de entrenamiento de YOLOv5[17]

El tiempo de entrenamiento es el tiempo total necesario para entrenar un modelo de aprendizaje automático y la evaluación máxima es el punto de entrenamiento en el que el modelo alcanza su mejor rendimiento. En la Figura 15 se demuestra que el YOLOv4 tarda alrededor de 3.5 horas para alcanzar a su evaluación máxima, mientras que el YOLOV5s tarda alrededor de 14.46 minutos en entrenar.

2.2.2 CÓDIGO ROS.

En cuanto al ámbito de robótica del proyecto, se emplean códigos fundamentados en el Sistema Operativo de Robótica (ROS). ROS fue creado en 2007, por Keenan Wyrobek y Eric Berger, para la simplificación de programación y conexión en robots y otros dispositivos. A partir del 2014, la Fundación de Robótica de Código se convirtió en la administradora de ROS, impulsando su crecimiento exponencial para el desarrollo de aplicaciones robóticas. [31]

ROS se considera un meta-sistema operativo, estando entre un sistema operativo y middleware. ROS ofrece un conjunto de bibliotecas de software de código abierto que ayudan a los usuarios a desarrollar y avanzar en aplicaciones robóticas. [32]

Entre las características que ofrece se encuentran: abstracción de hardware, gestión de conflictos y gestión de procesos. Además, proporciona funcionalidades de altos nivel, asíncronas y síncronas, una base de datos centralizada de datos y un sistema de configuración de robots. [33]

ROS ofrece numerosas ventajas, destacando la capacidad de evitar la repetición de esfuerzos en el desarrollo de un proyecto robótico, ahorrando tiempo y recursos. Esto incluye la reducción del nivel técnico requerido para trabajar en proyectos de robótica, facilitando a empresas iniciarse en el mundo de la robótica e incluso diseñar sistemas complejos de una forma más ágil. Adicionalmente, facilita la colaboración de distintas disciplinas y simplifica el proceso de diseño y programación de robots. [33]

ROS se encarga de ejecutar un gran número de procesos en paralelo para poder intercambiar datos. En el caso de este proyecto, por un lado, se ejecuta el código que gestiona el movimiento del robot con un controlador de la PS4, y de manera paralela el robot va recibiendo los metadatos correspondientes a la inteligencia artificial que se obtienen por medio de su cámara integrada. En el momento en el que el robot se acerca considerablemente a una persona, éste se debe frenar y no dejar al usuario controlarlo.

Este proceso se lleva a cabo de manera continua y en paralelo, manteniendo un acceso eficiente a los recursos del robot.

Hay una serie de conceptos que forman la base de la infraestructura de ROS. Estos conceptos fundamentales permiten que el sistema funcione de manera eficiente y facilitan el desarrollo de las aplicaciones robóticas. Estos conceptos son: los nodos, el maestro, los tópicos, los mensajes y los servicios. En secciones posteriores se abordará tanto la instalación del sistema operativo como el funcionamiento y el uso de estos conceptos.

El código de ROS utilizado es proporcionado por el usuario de GitHub cmauricioae8, el proyecto se titula *ros_cpp_py_basics*.[34]

Para poder trabajar con este proyecto se tuvo que integrar en la carpeta del proyecto de ROS conocida como *catkin_ws* y hacer una serie de ajustes para que funcionase correctamente. Se analizaron los códigos de *Python* (carpeta *scripts*), de *C++* (carpeta *src*), y los *launch* (carpeta *launch*).

Después de analizar el funcionamiento de los códigos y de la simulación del robot, se comenzó a desarrollar *scripts* para controlar el robot de forma autónoma. Tras entender los conceptos de ROS, se empezó a programar un script con el objetivo de mover el robot a partir de las señales obtenidas por el controlador de la PS4. A continuación, se proporciona un esquema de la estructura del proyecto con las carpetas fundamentales que se utilizaron y los códigos que ayudaron a comprender el proceso de simulación de un robot en el simulador Gazebo.

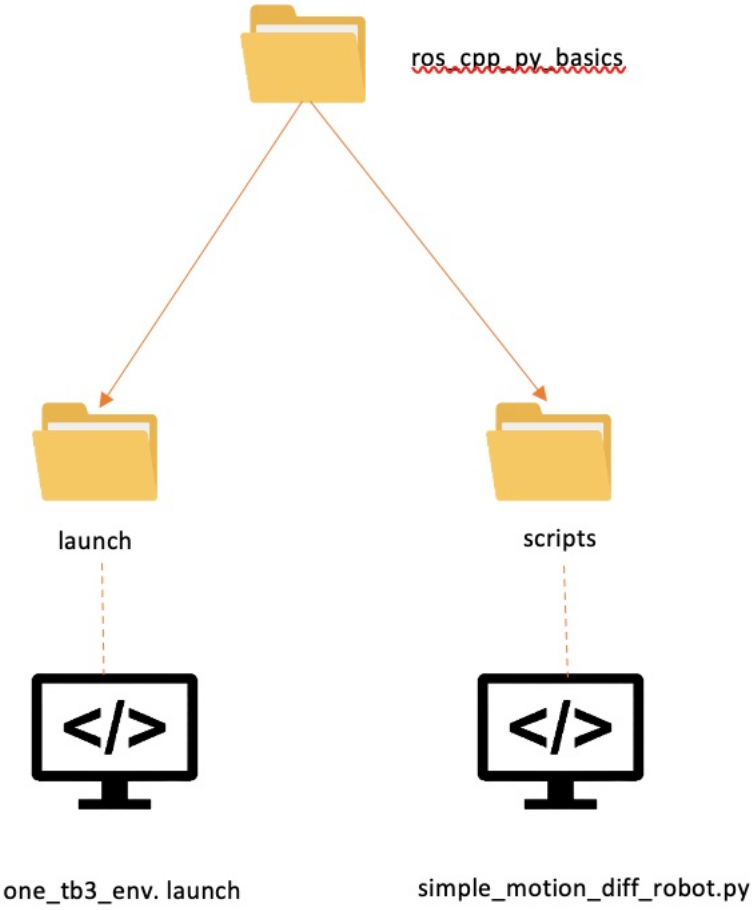

Figura 16 Esquema realizado con los códigos utilizados del proyecto de ROS [35][36]

El uso de estos códigos se detalla más adelante en el apartado de ROS en el Capítulo 4. Los *scripts* no se han utilizado en el proyecto, simplemente han servido como referencia para entender el funcionamiento de ROS. Por otro lado, el *launch* sí que se ha utilizado, pero se ha modificado para adaptarse al proyecto.

2.2.3 CÓDIGO DS4_DRIVER.

El siguiente código, obtenido también de GitHub, corresponde al proyecto *ds4_driver* realizado por Naoki Mizuno.[37] Este proyecto se basa en un controlador diseñado específicamente para interactuar con el control DualShock 4 (DS4), asociado con el utilizado por la consola de juegos de la "*PlayStation 4 (PS4)"* a través de Bluetooth.

Por medio de este proyecto, dentro de ROS se puede controlar el DualShock 4 como un dispositivo de entrada por medio de Bluetooth o USB, para controlar un robot u otros dispositivos. Esto se consigue gracias a la captura de datos generados por el controlador DS4.

Posteriormente, estos datos se publican en lo que se conoce como tópicos de ROS, en este caso específico, en el tópico *'/joy'*. La información que se publica está relacionada con el estado de los botones presionados, así como la orientación y la posición del joystick entre otros datos relevantes. El valor de los botones presionados se consigue a partir de los índices establecidos a cada botón en la Wiki de ROS [37]:

Index	Button name on the controller
0	A
1	B
2	X
3	Y
4	LB
5	RB
6	back
7	start
8	power
9	Button stick left
10	Button stick right

Tabla 1 Número de índice de los /joy.buttons[37]

Index	Axis name on the controller
0	Left/Right Axis stick left
1	Up/Down Axis stick left
2	LT
3	Left/Right Axis stick right
4	Up/Down Axis stick right
5	RT
6	cross key left/right
7	cross key up/down

Tabla 2 Número de índice de los /joy.axes[37]

En el proyecto, se emplearán algunos de los botones mencionados en la Tabla 1 y la Tabla 2. El uso específico de estos botones y de los tópicos se detallará posteriormente en el Capítulo 4.

En resumen, el proyecto de *ds4_driver* actúa como vínculo entre el control DS4 y el entorno ROS, permitiendo al controlador actuar como una interfaz para interactuar con sistemas robóticos.

2.2.4 CÓDIGO UDP.

Para poder establecer la comunicación entre el robot y el ordenador servidor se utiliza el protocolo de comunicación UDP (User Datagram Protocol), el cual permite la transmisión de datos de manera no orientada a conexión y transmisiones sujetas a limitación temporal como es el caso de la reproducción de video. La ventaja de utilizar el protocolo UDP es la velocidad de transmisión de los datos, mientras que la desventaja es la pérdida ocasional de paquetes (unidades de transmisión de datos). [38]

Para implementar la comunicación UDP necesaria, se tomaron como referencia los códigos existentes, adaptándolos para satisfacer las necesidades específicas requeridas. El objetivo es permitir la transferencia de los metadatos generados por la inteligencia artificial en el

ordenador servidor tras recibir el vídeo de la cámara integrada en la cámara del robot. Esta información es fundamental para que el robot comprenda su entorno y pueda tomar decisiones en base a el mismo, mejorando la capacidad de movimiento y navegación, así como la respuesta a su entorno.

Los códigos a utilizar se basarán, por un lado, en el envío de mensajes y, por otro lado, en la recepción de mensajes. A continuación, se muestran los códigos utilizados en cada caso.

Envío de mensajes UDP:

```
import socket

UDP_IP = "127.0.0.1"
UDP_PORT = 5005
MESSAGE = b"Hello, World!"

print("UDP target IP: %s" % UDP_IP)
print("UDP target port: %s" % UDP_PORT)
print("message: %s" % MESSAGE)

sock = socket.socket(socket.AF_INET, # Internet
                     socket.SOCK_DGRAM) # UDP
sock.sendto(MESSAGE, (UDP_IP, UDP_PORT))
```

Figura 17 Ejemplo de código para envío de mensajes en UDP [39]

Recepción de mensajes UDP:

```
import socket

UDP_IP = "127.0.0.1"
UDP_PORT = 5005

sock = socket.socket(socket.AF_INET, # Internet
                     socket.SOCK_DGRAM) # UDP
sock.bind((UDP_IP, UDP_PORT))

while True:
    data, addr = sock.recvfrom(1024) # buffer size is 1024 bytes
    print("received message: %s" % data)
```

Figura 18 Ejemplo de código para recepción de mensajes UDP [39]

Capítulo 3. DEFINICIÓN DEL TRABAJO

3.1 MOTIVACIÓN

Este proyecto se basa en el avance tecnológica de la conducción asistida a través de la inteligencia artificial y la robótica. La necesidad de avanzar en la creación de robots que puedan tomar de decisiones de manera autónoma se realiza con el objetivo de poder crear situaciones más seguras y eficientes en un futuro. Además, se busca promover el uso de la robótica, para que sea más accesible y utilizada en diversos campos, tanto en ámbitos profesionales como en no profesionales.

En cuanto a la toma de decisiones autónoma, una gran motivación es trabajar y avanzar en el desarrollo de la IA en los robots. La capacidad de que un robot pueda analizar y responder a su entorno mientras es controlado por un ser humano contribuye al progreso en la autonomía parcial de los robots, y a su vez, en la autonomía completa.

Para lograr esto, se deberá de llevar a cabo un estudio en profundidad para evaluar por medio de la IA los algoritmos de percepción y la toma de decisiones para que el robot pueda responder de manera correcta a su entorno. Al mismo tiempo, se programará sistemas de control que permita progresar el movimiento del robot junto con lo que le rodea.

Este proyecto tiene la intención de progresar en este apasionante campo de la tecnología aprovechando el soporte de una empresa históricamente puntera en estos ámbitos: Nokia.

3.2 OBJETIVOS DEL PROYECTO

Como se ha mencionado anteriormente, el enfoque del proyecto se basa en la programación de un robot en formato de vehículo, para que pueda desplazarse a medida que va realizando un análisis de su entorno por medio de la IA.

El objetivo principal del proyecto es conseguir la conducción asistida, permitiendo al robot reaccionar de manera autónoma a obstáculos como personas y a cambios en su entorno alcanzando una conducción segura y eficiente.

Este objetivo se alcanza gracias a los siguientes subobjetivos:

1. **Integración de un Sistema de inteligencia artificial*:*** Utilizar e incorporar algoritmos, en este caso, el algoritmo YOLOv5 en Python, para dotar al robot de la capacidad de analizar su entorno durante la conducción asistida. De esta forma, se podrán evitar los obstáculos y garantizando una conducción segura y eficiente.

2. **Exploración en el área de conducción por medio de ROS*:*** Desarrollo un programa de conducción asistida del robot con el sistema operativo ROS. Este programa debe de estar programado para ejecutarlo en el simulador Gazebo. El robot será controlado mediante un mando PS4, denominado en el sistema como 'joystick'. La simulación previa permite aplicar la programación en el robot de manera más eficiente.

3. **Combinar la parte de Inteligencia artificial y la de ROS en el robot físico:** Este último objetivo consiste en conseguir la comunicación efectiva entre el ordenador, que ejecuta la parte de inteligencia artificial (YOLOv5), y con el robot, que gestiona su propio movimiento.

3.3 METODOLOGÍA Y RECURSOS.

Para la realización del proyecto, este se ha dividido en distintas etapas, durante las cuales se llevaron a cabo los procesos necesarios para garantizar alcanzar los objetivos de manera clara y eficiente.

Antes de comenzar con el proyecto, se realizó un estudio de las dos áreas más importantes:

1. El área de la inteligencia artificial junto con la herramienta YOLOv5 a programar para reconocer objetos en imágenes y videos, como funcionan las redes neuronales asociado con la parte de algoritmos.
2. La robótica del proyecto gestionada por el sistema operativo ROS.

En primer lugar, se utilizó el código abierto en el lenguaje Python del algoritmo de YOLOv5 ejecutándolo en "*Visual Studio Code*" para comprobar su funcionamiento. A posteriori, la empresa Nokia suministró una carpeta con varios códigos basados en el mismo algoritmo y para diferentes casos en función de la información que se quiere obtener. En este caso, se aprenderá a trabajar con Docker para la ejecución del código principal, que se suele conocer como *detect.py*. Posteriormente, se analizó el flujo de salida de los dos códigos fundamentales que suministran los metadatos asociados a los objetos detectados por el robot.

En segundo lugar, una vez entendido el sistema operativo de ROS, se comenzó la creación de códigos con el lenguaje Python para gestionar el movimiento de cualquier robot. El código se ajustó a los movimientos del robot demandados por Nokia para conseguir los objetivos del proyecto. El funcionamiento de los códigos se verificó en el simulador de robótica 2D/3D conocido como Gazebo. Una vez verificado el cumplimiento de los requisitos del robot, se trasladó el código al robot físico y se hicieron los cambios necesarios para que el código se ajustase a las condiciones del robot.

En tercer lugar, se trabajó con el robot y con el ordenador servidor. Por un lado, toda la parte de IA se gestionó desde el ordenador servidor y la parte del movimiento del robot y del envío de mensajes a la hora de detectar personas se gestionó desde el mismo robot. Para gestionar

la comunicación entre el ordenador servidor y el robot, se estableció una comunicación UDP. Esta conexión permitió el envío de la información detallada sobre la estructura y características del flujo de vídeo que observaba el robot, conocida como metadatos[40]. Se consiguió que el robot analizase los metadatos en tiempo real y que enviase un mensaje, en el caso de que encontrar a una persona muy cerca de su entorno, al código principal del movimiento del robot, frenando su avance, con el consiguiente impacto en seguridad y eficiencia buscado.

Finalmente, para gestionar la mejora del flujo de imágenes desde el robot al ordenador se añadió un router de 5G. Este aspecto permitió evitar las congestiones de red al ser el único dispositivo utilizando el router. El router se configura sin salida a internet, para focalizar su uso en la conectividad entre el PC servidor y el robot.

La metodología se puede resumir en los siguientes esquemas:

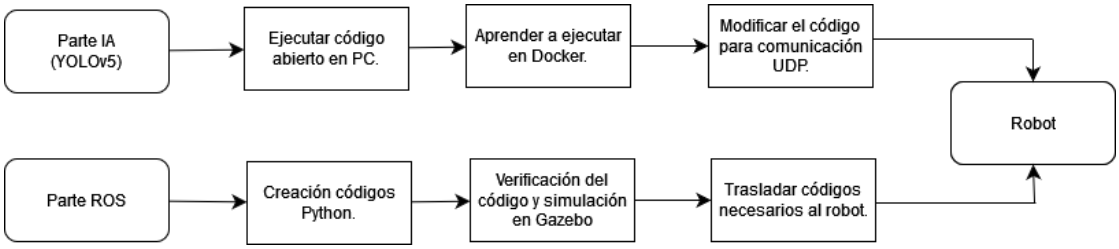

Figura 19 Esquema I de la metodología del proyecto

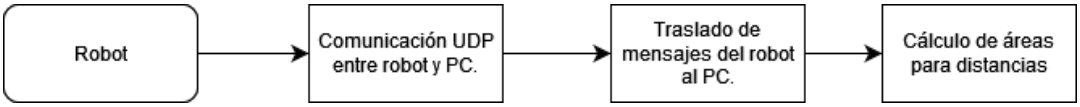

Figura 20 Esquema II de la metodología del proyecto

Los recursos utilizados a lo largo del proyecto son los siguientes:

- Artículos de Internet.
- Códigos ejemplo extraídos de GitHub.
- Documentos proporcionados por Nokia.
- Códigos proporcionados por Nokia.
- Material de hardware (el robot y todas las partes integradas) proporcionado por Nokia.
- Sistema Operativos ROS (Robot Operating System).
- Lenguajes de programación: Python.
- Entorno gráfico Gazebo.
- Controlador de la PS4 para movilizar el robot.

3.4 PLAN DE TRABAJO.

Para cumplir con los objetivos y requisitos del proyecto, así como la metodología propuesta, se ha seguido la siguiente planificación:

PLANIFICACIÓN TRABAJO FIN DE GRADO											
TAREA	2023					2024					
	Junio-Agosto	Septiembre	Octubre	Noviembre	Diciembre	Enero	Febrero	Marzo	Abril	Mayo	Junio
Informarse sobre proyecto											
Investigación en IA y ROS											
Desarollo de códigos en ROS y simulación en Gazebo.											
Finalizar el código del movimiento completo del robot.											
Aprender a utilizar Docker con IA.											
Trasladar código ROS al robot.											
Conseguir comunicación UDP entre PC y robot.											
Gestionar el movimiento del robot y resultados de IA por UDP.											
Mejorar detección de objetos por cálculo de áreas.											
Trabajar en memoria final											

Figura 21 Planificación temporal del proyecto

Al llevar una planificación organizada con distintas tareas interrelacionadas, cada una de ellas solía tomar entre uno o dos meses. El poder llevar una buena organización ha contribuido favorablemente a conseguir los objetivos establecidos y antes de lo esperado.

Capítulo 4. SISTEMA/MODELO DESARROLLADO.

En esta sección del Trabajo de Fin de Grado, se abordará en detalle todos los tipos de software y hardware empleados a lo largo del desarrollo. En primer lugar, se expondrán los componentes del robot y el funcionamiento de cada uno de ellos. En segundo lugar, se describirá el proceso de descarga de los distintos componentes de software necesarios para llevar a cabo el proyecto.

4.1 ESTRUCTURA DEL ROBOT.

El robot del proyecto fue comprado por la empresa Nokia en Internet. A día de hoy, no se encuentran documentos sobre el origen del robot ni su manual de funcionamiento, por lo que se procede a explicar las partes que componen el robot.

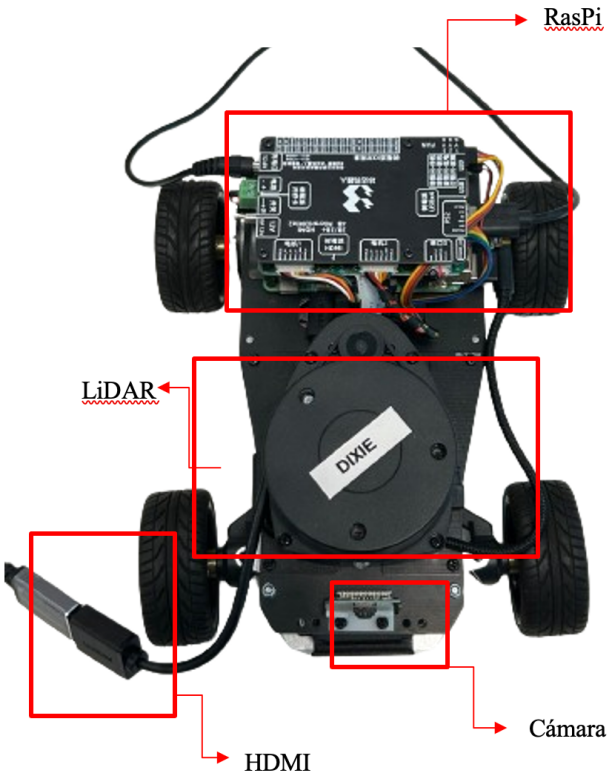

Figura 22 Componentes del robot

El robot se encuentra equipado con una Raspberry Pi (también denominada Raspi), motores, una cámara Raspberry Pi y un sensor LiDAR. Para este proyecto, el sensor LiDAR no se va a utilizar.

La Raspberry Pi es un ordenador de tamaño pequeño que actúa como el cerebro del robot. Se basa en una placa que soporta distintos componentes de un ordenador como un procesador ARM de hasta 1500 MHz, un chip gráfico y una memoria RAM de hasta 8 GB.Entre sus características destacan[41]:

- Tiene una serie de puertos y entradas que permiten conectar dispositivos periféricos. En este caso se usará una pantalla, teclados, ratón y USB.
- Dispone de un procesador gráfico VideoCoreIV que permite, la reproducción de video en alta definición.
- Permite conexión a la red por medio de Wifi, Bluetooth y Ethernet.
- Incluye una ranura para tarjetas SD que facilita la instalación de sistemas operativos libres por medio de una tarjeta microSD.

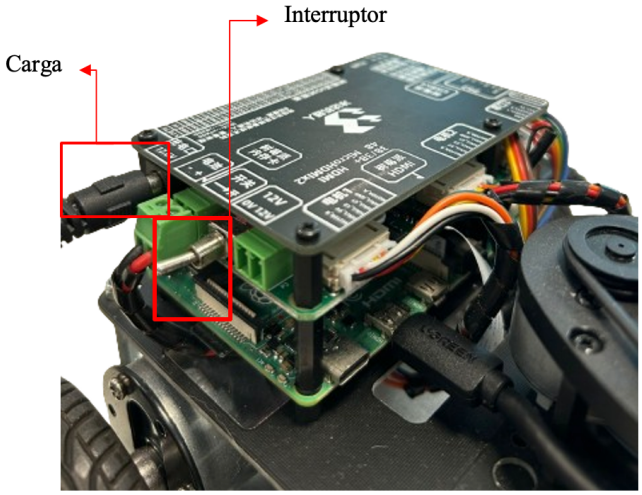

Figura 23 Componentes adicionales en la RasPi

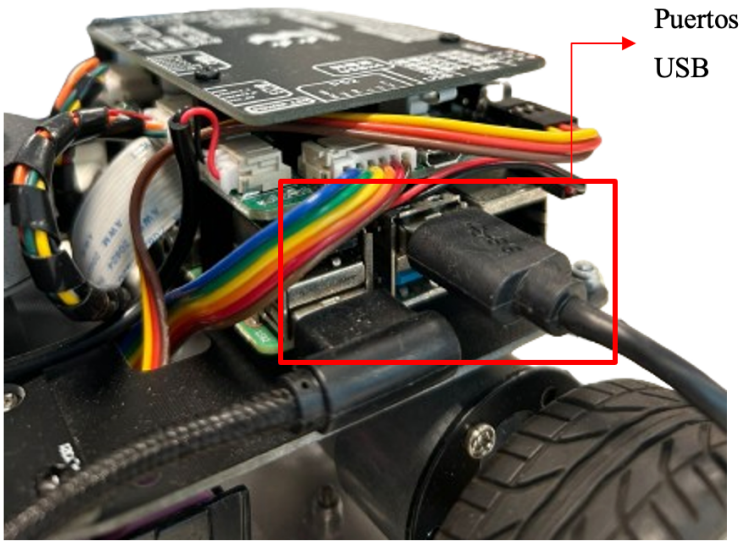

Figura 24 Puertos USB RasPi

En este caso, la RasPi tiene instalado el sistema operativo Ubuntu 16.04, una versión de Linux enfocada al uso de estos dispositivos. La actividad en el robot está gestionada por ROS, de tal forma que cuando la Raspberry Pi se inicia, se carga Ubuntu y se lanza un servicio específico de ROS de manera automática que inicia todos los nodos necesarios para el uso del robot, produciendo un pitido cuando éste se ha levantado correctamente.

Por otro lado, se encuentran los motores del robot, que son los encargados de su movilidad del robot y que se controlan mediante tópicos de ROS. El funcionamiento de la navegación, la velocidad y la dirección de los motores se controla a través del envío de mensajes en ROS, lo cual se detallará en el apartado 4.3, “ROS”, de este capítulo.

La cámara está configurada para lanzar el flujo de datos mediante un pipeline de GStreamer, permitiendo al robot obtener información para ayudar en la navegación y en el análisis del entorno, gracias al procesamiento del vídeo en tiempo real. En este proyecto la cámara se activa con un comando específico usando GStreamer, este proceso se explicará en el apartado 4.2, "Inteligencia artificial: YOLOv5", de este Capítulo.

Por último, el LIDAR (Light Detection and Ranging) es una tecnología de teledetección que utiliza rayos láser para medir distancia y movimientos precisos en tiempo real[42]. El sistema de LIDAR se compone de dos elementos principales: un sensor LiDAR, encargado de detectar y recoger los impulsos del entorno; y un procesador para calcular el tiempo y la distancia de los impulsos, que proporciona un conjunto de datos resultante denominado nube de puntos LiDAR. Por medio de este sistema se permite mapear y analizar el entorno del robot con alta precisión[42].

En este proyecto el LiDAR no se utiliza ya que, para detectar la cercanía de los objetos, se usará la detección de objetos por medio de inteligencia artificial y el cálculo de áreas.

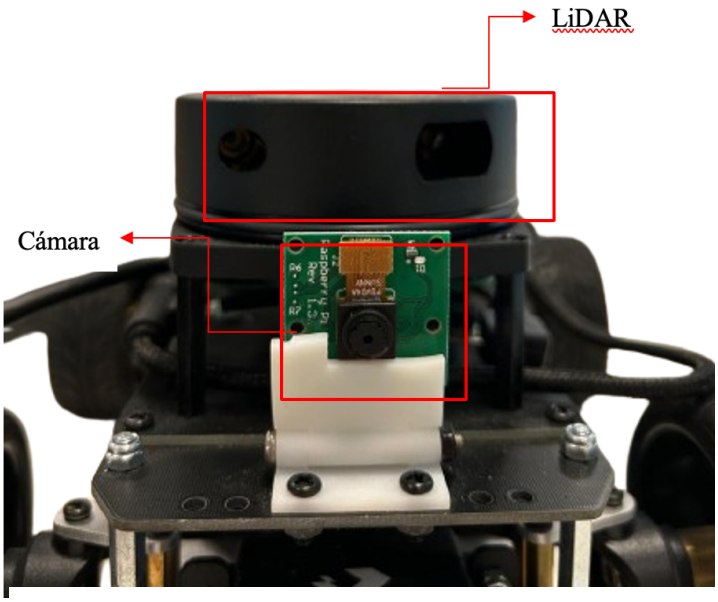

Figura 25 Posición del LiDAR y la cámara en el robot

Finalmente, este conjunto de herramientas y componentes permite que el robot sea operado de manera autónoma o sea controlado remotamente en función de lo que quiera el usuario. Como se mencionó anteriormente, todas las operaciones están centralizadas en el uso de la Raspberry Pi y gestionadas por ROS.

Para finalizar con la parte del hardware, a continuación, se muestra un diagrama de la arquitectura del proyecto, con las comunicaciones establecidas entre el robot y el PC servidor donde se ejecuta la parte de IA.

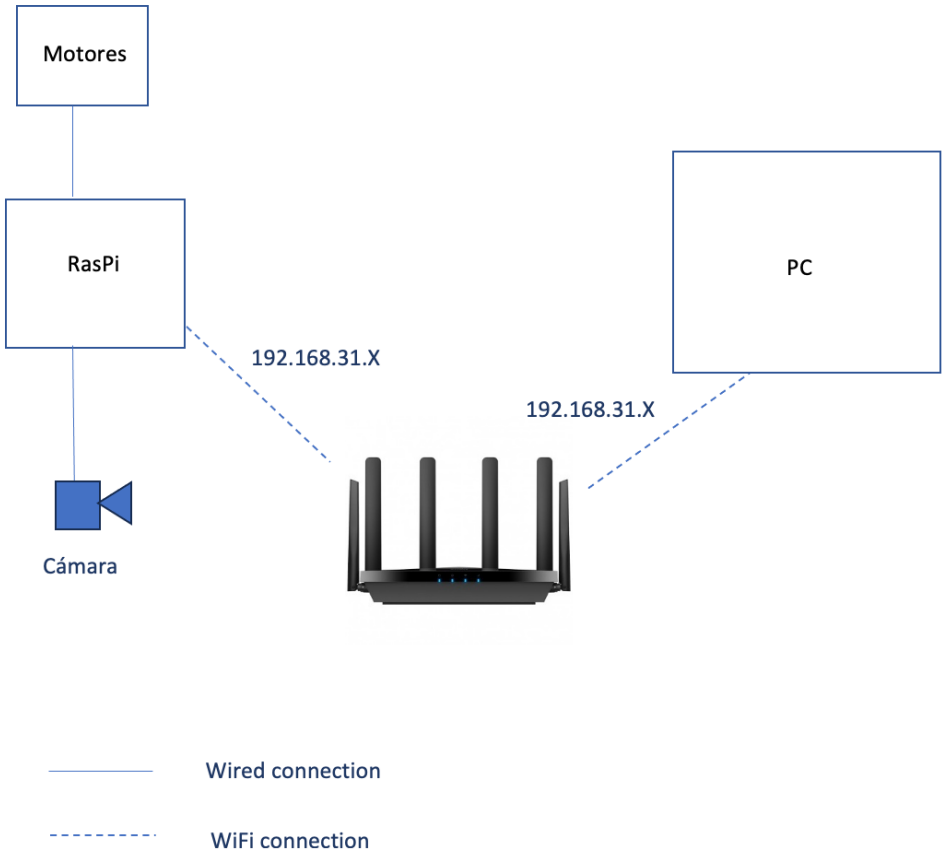

Figura 26 Diagrama de la arquitectura del proyecto [43]

4.2 INTELIGENCIA ARTIFICIAL: YOLOV5.

El algoritmo empleado para gestionar toda la parte de la IA en el robot se basa en la red neuronal YOLOv5, la cual ha sido expuesta en detalle en el Capítulo 2.

Al principio del proyecto, se descarga la carpeta que contiene el código base de YOLOv5 con el objetivo de comprobar y entender su funcionamiento en el ordenador de trabajo con la webcam. Para ello, se siguieron las instrucciones del repositorio de GitHub de Glenn-jocher, tal como se menciona previamente en el Capítulo 2.

En este caso, al estar trabajando con el ordenador perteneciente a la empresa Nokia, no fue necesario descargar una nueva versión de Python, ya que la instalada ya estaba actualizada a la versión 3.8.10.

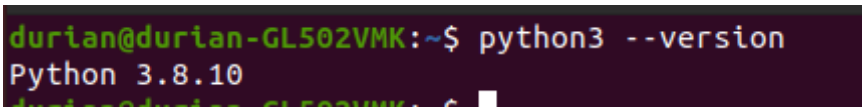

Figura 27 Versión de Python necesaria

Para instalar o actualizar la versión de Python, se deben seguir los siguientes pasos [44]:

1. Antes de instalar Python 3.8, es esencial garantizar que todos los paquetes están actualizados. Al abrir la terminal se ejecutan los primeros comandos:
   sudo apt update

   En el caso de que haya algún paquete desactualizado, se actualiza con el siguiente comando:

   sudo apt upgrade

2. Se añade *deadsnakes* PPA al Sistema. De esta forma, se puede acceder a paquetes adicionales que no están disponibles en los repositorios de Ubuntu. Por medio de *deadsnakes* PPA se puede instalar y manejar con distintas versiones de Python de

una forma más rápida y utilizando herramientas de gestión de paquetes como es el *apt*:
sudo add-apt-repository ppa:deadsnakes/ppa

3. Se hace una actualización de los paquetes para incluir el nuevo repositorio:
   sudo apt update

4. Una vez que el repositorio esté habilitado se instala Python 3.8:
   sudo apt install python3.8

5. Para verificar que la versión es la correcta:
   python3.8 –versión

El resto de las herramientas necesarias para ejecutar el proyecto de YOLOv5 se encuentran en el archivo *requirements.txt.* Como se ha explicado en el apartado, "Códigos externos utilizados", del Capítulo 2 este archivo contiene librerías como: GitPython, Matplotlib, NumPy, OpenCV, Pillow, psutil, PyYAML, request, SciPy, thop, torch, torchvision, tqdm, ultralytics… necesarias para el procesamiento de imágenes, el entrenamiento y la inferencia de modelos de aprendizaje profundo, así como la visualización de resultados, entre otras tareas relacionadas con YOLOv5.

Una vez analizado el código, y todas sus funciones, se procede a ejecutar el archivo principal del proyecto: *detect.py,* que es el encargado de iniciar y llevar a cabo la detección de objetos.

Para ello se ejecuta en la terminal el siguiente comando [19]:

python detect.py --weights yolov5s.pt --source 0

siendo el '*source 0'* lo que está asociado con la webcam del nuestro ordenador. En el caso de querer utilizar una imagen o un video cualquiera, simplemente se tendría que poner la ruta hacia el archivo .jpg o .mp4 deseado en la opción de *source*. En el caso del proyecto no

se utilizará esta parte en el comando a ejecutar, ya que la imagen será obtenida por medio de la cámara del robot y no por el PC servidor, más adelante se mostrará un ejemplo de ello.

4.2.1 INSTALACIÓN DE HERRAMIENTAS NECESARIAS.

Una vez adquirido un mayor conocimiento sobre el algoritmo YOLOv5 y haber comprendido el funcionamiento del código principal *detect.py*, el siguiente paso consiste en trasladar toda la parte de inteligencia artificial a un entorno de Docker para su ejecución. Para realizar esto, Nokia proporciona una carpeta con el nombre *yolo_alga*. Esta carpeta contiene un total de 10 códigos distintos, aunque no todos se han utilizado en el sistema final. Los códigos con los que se ha funcionado a lo largo del proyecto son los siguientes:

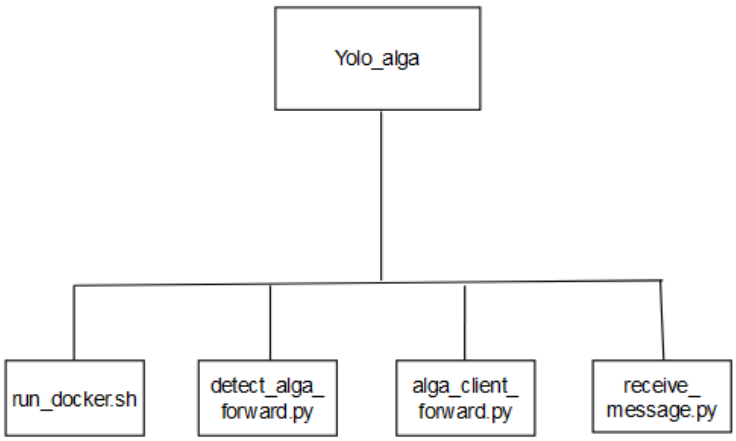

Figura 28 Esquema de los códigos utilizados en la carpeta de YOLO

Estos códigos tienen la siguiente funcionalidad en el sistema:

- *Detect_alga_forward.py*: Este código ejecuta la detección de objetos en fotogramas de video, recibidos por UDP, utilizando el algoritmo YOLOv5. A posteriori, envía los resultados correspondientes de la detección a través de UDP

en dos formatos: un conjunto de metadatos con información sobre las detecciones, y los fotogramas procesados.

- *Alga_client_forward.py*: Se trata del código que se ejecuta en el cliente para la recepción de las detecciones, que muestra los fotogramas de video y los metadatos por pantalla.
- *Receive_message.py*:
  - La primera función de este código se basa en crear un nodo y un tópico de ROS. En este lenguaje de robótica, los nodos se comunican entre sí de manera asíncrona a través de *tópicos* , que son canales de comunicación unidireccional. Un nodo puede publicar mensajes dentro de un tópico y posteriormente leer esos mensajes suscribiéndose al mismo tópico. De esta forma, se facilita la comunicación entre sistemas robóticos. En el caso de este proyecto, el nodo va a escuchar los mensajes UDP en una dirección IP y un puerto específico. El proceso de comunicación a través de UDP se realiza a través de los códigos explicados anteriormente en el capítulo 2, sacados de internet.
  - Los mensajes que se escuchan son los metadatos que proceden de la ejecución de la inteligencia artificial integrada en el PC servidor. A partir de estos mensajes, si se detecta a una persona, se publica el mensaje: "*Person detected, stopping robot*" en el tópico de ROS llamado *'/stop_robot'*, indicando la detección y que se debe de parar el robot en el código que se corresponde con toda la parte del movimiento de este. Si no recibe el mensaje con la palabra "*person*" entonces se envía un mensaje indicando que no se ha identificado a ninguna persona.

4.2.1.1 Instalación de NVIDIA

Antes de empezar a ejecutar los códigos mencionados en el esquema de la Figura 28 de la sección anterior, hay que descargar el NVIDIA Driver.

NVIDIA es una empresa estadounidense que destaca por su contribución a la revolución de la inteligencia artificial (IA) gracias a sus unidades de procesamiento gráfico (GPU). Estas GPUs se consideran como el "cerebro" de las aplicaciones IA, ya que permiten desde la generación de textos hasta la conducción autónoma. NVIDIA es conocida por sus tarjetas gráficas y por el desarrollo de chips y software para entrenar y ejecutar algoritmos de aprendizaje automático. [45]

Para descargar el driver de NVIDIA, se accede a su página web [46]:

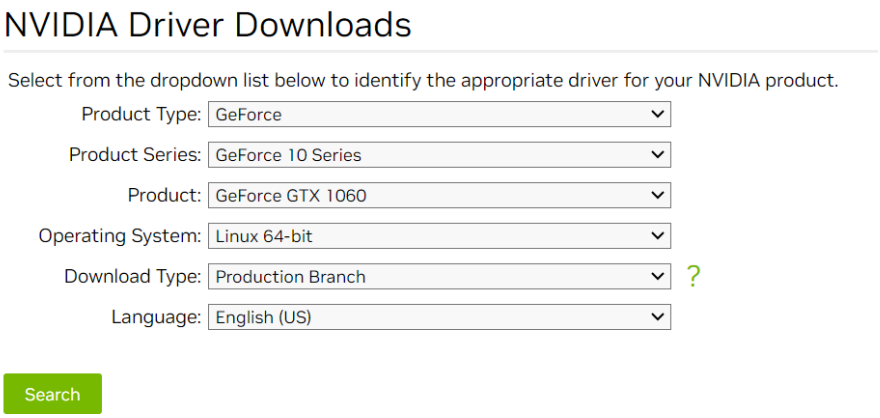

Figura 29 Versión de descarga para tarjetas gráficas NVIDIA [46]

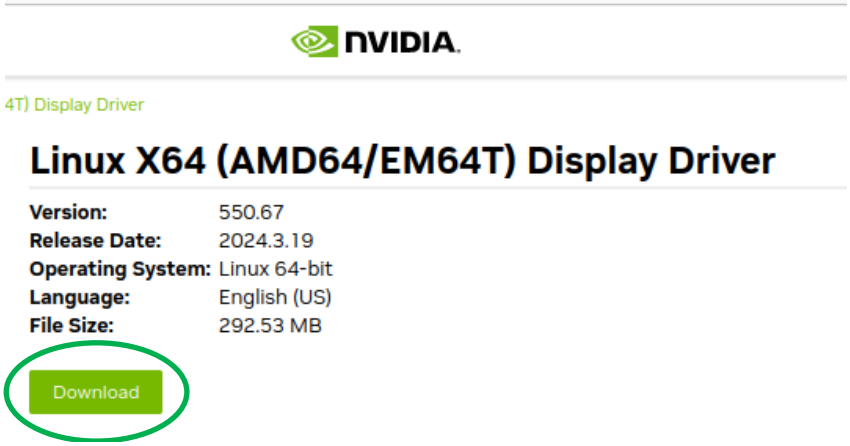

Figura 30 Proceso de descarga de la versión adecuada de NVIDIA [46]

Una vez escogida la versión correspondiente mostrada en la Figura 29, aparece una nueva ventana con el contenido de la Figura 30. A continuación, se presiona el botón de descarga y se abre una terminal para introducir los siguientes comandos desde la carpeta en la que ha sido descargada el driver[47]:

chmod +x NVIDIA-Linux-x86_64-550.54.14.run
sudo ./NVIDIA-Linux-x86_64-550.54.14.run

El primer comando cambia los permisos del archivo de instalación y lo hace ejecutable. Al poner el *'+x'* se añaden permisos de ejecución para el propietario y otros usuarios. En el segundo comando, al poner '*sudo*' asegura al usuario que tiene los permisos necesarios para instalar el controlador y se ejecuta el archivo de instalación con privilegio de superusuario *(root)*.

sudo reboot

Este último comando se utiliza para reiniciar el sistema, así los cambios del controlador se aplican correctamente.

Si todo lo anterior funciona correctamente, se verifica la versión de NVIDIA en el ordenador de la siguiente forma:

```
durian@durian-GL502VMK:~$ nvidia-smi
Mon Apr  1 10:34:14 2024
+---------------------------------------------------------------------------------------+
| NVIDIA-SMI 550.54.14              Driver Version: 550.54.14      CUDA Version: 12.4     |
|-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  NVIDIA GeForce GTX 1060        Off |   00000000:01:00.0  On |                  N/A |
| N/A   40C    P0              25W /   78W |     217MiB /   6144MiB |      0%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+

+---------------------------------------------------------------------------------------+
| Processes:                                                                            |
|  GPU   GI   CI        PID   Type   Process name                            GPU Memory |
|        ID   ID                                                             Usage      |
|=======================================================================================|
|    0   N/A  N/A      1217      G   /usr/lib/xorg/Xorg                            35MiB |
```

Figura 31 Comprobación de la versión de NVIDIA adecuada

4.2.1.2 Instalación de los Dockers.

A continuación, se trata con la parte de instalación de los dos dockers del proyecto.

4.2.1.2.1 Docker de YOLO.

La instalación de YOLO en el docker difiere de los métodos tradicionales. En lugar de la descarga tradicional, se adquiere la imagen del Docker específico que se corresponde con YOLO. Este proceso se hace una única vez con el siguiente comando:

sudo docker pull ultralytics/yolov5

Posteriormente, para ejecutar el proyecto en su totalidad se utiliza el comando *'./run_docker.sh'* , el cual inicia un contenedor utilizando la imagen previamente descargada que contiene todas las librerías necesarias para la correcta ejecución de YOLO.

Dentro del fichero *run_docker.sh* se encuentra lo siguiente:

```
docker run --ipc=host --gpus=all --network=host -it --rm --env="DISPLAY" --
volume="$HOME/.Xauthorityx:/root/.Xauthority:rw" -v "$(pwd)":/usr/src/app/yolo ultralytics/-
yolov5:latest
```

Figura 32 Contenido del fichero run_docker.sh del Docker del YOLO

En concreto, la opcion –rm indica que cada vez que se cierra el contenedor, éste se elimina. Por ello, siempre que se quiera acceder al código de YOLO se debe ejecutar el comando descrito anteriormente: '*sudo ./run_docker.sh'*.

4.2.1.2.2 Docker de polyp.

Una vez que se ha descargado la imagen, se trabaja con un contenedor que pertenece a un entorno de contenedor Docker llamado : polyp.

Para comenzar, se descarga la carpeta spider que se obtiene de una página web proporcionada solo a los usuarios de la empresa Nokia.

Dentro de esta carpeta, los archivos relevantes para el proyecto son los siguientes:

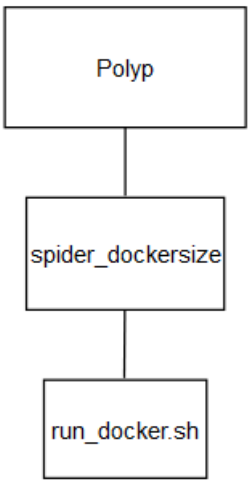

Figura 33 Archivos utilizados dentro de la carpeta polyp

Al ejecutar el fichero *run_docker.sh* se levanta un contenedor y se instala GStreamer, una biblioteca multimedia utilizada para reproducir, crear y manipular multimedia como vídeos o sonidos. Esta biblioteca tiene una compatibilidad con una variedad de formatos, un buen rendimiento y una arquitectura flexible.[49] Para todo lo anterior se ejecutan los siguientes pasos [48]:

1. En la carpeta descargada ejecuta : *sudo run_docker.sh*
2. Dentro del contenedor, actualiza GStreamer ejecutando : *apt-get install libgstreamer1.0-dev libgstreamer-plugins-base1.0-dev libgstreamer-plugins-bad1.0-dev gstreamer1.0-plugins-base gstreamer1.0-plugins-good gstreamer1.0-plugins-bad gstreamer1.0-plugins-ugly gstreamer1.0-libav gstreamer1.0-tools gstreamer1.0-x gstreamer1.0-alsa gstreamer1.0-gl gstreamer1.0-gtk3 gstreamer1.0-qt5 gstreamer1.0-pulseaudio*
3. Ejecuta : *polyp stop*
4. Ejecuta : *polyp start &*

En este caso, la ejecución cambia con respecto al proceso de instalación del Docker de YOLO. No hay que volver a ejecutar el fichero *run_docker.sh* ya que se reutiliza el contenedor. Esto se debe ya que dentro del fichero *run_docker.sh* no se añade la opción '*--rm'*, lo que demuestra que el fichero no se elimina, sino que se reutiliza.

```
#!/bin/bash
#
# The polyp config file (polyp.ini) might be overwriten with the path of the first argument

docker build -t spider .
# Run container with the host network interfaces
echo Running container...
params=('--network host')
if [[ -n $1 ]]
then
  SPIDER_DIR=$(docker inspect spider | jq -r \
    '.[].Config.Env[]|select(match("^SPIDER_DIR"))|.[index("=")+1:]')
  [[ -z $SPIDER_DIR ]] && SPIDER_DIR=/opt/spider
  params+=("-v $1:$SPIDER_DIR/polyp.ini")
fi
echo $params
docker run ${params[@]} -d spider
```

Figura 34 Contenido del fichero run_docker.sh del Docker de polyp

4.2.2 EXPLICACIÓN Y EJECUCIÓN DE LOS CÓDIGOS.

Una vez instalado el contenedor de polyp, se procede a la fase de lanzamiento de los archivos asociados a la parte de la inteligencia artificial siguiendo los pasos de la página web anterior.[48]

En primer lugar, se localiza el contenedor creado en la instalación de polyp:

```
durian@durian-GL502VMK:~$ sudo docker ps --all
[sudo] password for durian:
CONTAINER ID   IMAGE                  COMMAND                  CREATED         STATUS
cac14fef63fe   spider                 "/bin/sh -c /opt/doc…"   4 weeks ago     Exited
```

Figura 35 Contenedor de polyp

Como se puede observar en la Figura 35 si el contenedor no está activo (el estado es *"Exited"),* se ejecutan los siguientes comandos para entrar dentro del contenedor:

```
durian@durian-GL502VMK:~$ sudo docker start cac14fef63fe
cac14fef63fe
durian@durian-GL502VMK:~$ sudo docker exec -it cac14fef63fe /bin/bash
root@durian-GL502VMK:/opt# cd spider
```

Figura 36 Iniciamos el contenedor de polyp

En caso de que sí estuviese activo el contenedor, únicamente habría que ejecutar el segundo paso.

Posteriormente, se accede a la carpeta de spider y se comprueban las reglas dentro del contendor polyp para la aplicación.

```
durian@durian-GL502VMK:~$ sudo docker exec -it cac14fef63fe /bin/bash
root@durian-GL502VMK:/opt# cd spider
root@durian-GL502VMK:/opt/spider# polyp list
[
]
root@durian-GL502VMK:/opt/spider# polyp add "udp://192.168.31.204:5000?p=rtp" "v
ideo360-to-frames360" "zmq+tcp://127.0.0.1:3000?p=picture&t=/demo/video/&zbind=0
"
Success
root@durian-GL502VMK:/opt/spider#
```

Figura 37 Agregación de la configuración en la carpeta spider

Si no existen reglas (el resultado de '*polyp list'* es: '[]'), se añade una nueva configuración con '*polyp add'*, para conectarse a una transmisión de video en la red y poder procesarlo en la aplicación. Al añadir esta regla se le indica al contenedor que debe escuchar el flujo de vídeo RTP que le llega al PC por el puerto 5000 y, mediante GStreamer, transformarlo en frames que se envían por TCP al puerto 3000 del mismo PC bajo el tópico *'demo/video'*.

Una vez establecidas las reglas de polyp, se abre un nuevo terminal desde la carpeta de nuestro proyecto YOLO para ejecutar el código de detección de objetos.

Para ello, se levanta el contenedor de YOLO, y, dentro de él se lanza el script mencionado anteriormente: *detect_alga_forward.py.*

Este script permite añadir las siguientes opciones de pesos en su ejecución:

```
Usage - formats:
    $ python detect_alga.py --weights yolov5s.pt                 # PyTorch
                                yolov5s.torchscript        # TorchScript
                                yolov5s.onnx               # ONNX Runtime or OpenCV DNN with --dnn
                                yolov5s_openvino_model     # OpenVINO
                                yolov5s.engine             # TensorRT
                                yolov5s.mlmodel            # CoreML (macOS-only)
                                yolov5s_saved_model        # TensorFlow SavedModel
                                yolov5s.pb                 # TensorFlow GraphDef
                                yolov5s.tflite             # TensorFlow Lite
                                yolov5s_edgetpu.tflite     # TensorFlow Edge TPU
                                yolov5s_paddle_model       # PaddlePaddle
"""
```

Figura 38 Opciones de pesos en el scrip detect_alga_forward.py

En este caso al ejecutar el fichero se añade el argumento *'--weights yolov5s.pt'*, que indica el uso del archivo de pesos *yolov5s.pt* para la detección. El comentario *"#PyTorch"* hace referencia a que este archivo de pesos es para un modelo entrenado con PyTorch, obtenido previamente en la instalación con el *requirements.txt*.

Para añadir más argumentos en función de lo que se necesite se encuentra la función '*parse_opt()*' , la cual se encarga de definir y analizar los argumentos a utilizar para construir el modelo de detección de objetos. Para la gestión de los argumentos se utiliza el módulo '*argparse*'.

```
def parse_opt():
    parser = argparse.ArgumentParser()
    parser.add_argument('--weights', nargs='+', type=str, default=ROOT / 'yolov5s.pt', help='model path or triton URL')
    parser.add_argument('--data', type=str, default=ROOT / 'data/coco128.yaml', help='(optional) dataset.yaml path')
    parser.add_argument('--imgsz', '--img', '--img-size', nargs='+', type=int, default=[640], help='inference size h,w')
    parser.add_argument('--conf-thres', type=float, default=0.50, help='confidence threshold')
    parser.add_argument('--iou-thres', type=float, default=0.50, help='NMS IoU threshold')
    parser.add_argument('--max-det', type=int, default=1000, help='maximum detections per image')
    parser.add_argument('--device', default='', help='cuda device, i.e. 0 or 0,1,2,3 or cpu')
    parser.add_argument('--classes', nargs='+', type=int, help='filter by class: --classes 0, or --classes 0 2 3')
    parser.add_argument('--agnostic-nms', action='store_true', help='class-agnostic NMS')
    parser.add_argument('--augment', action='store_true', help='augmented inference')
    parser.add_argument('--half', action='store_true', help='use FP16 half-precision inference')
    parser.add_argument('--dnn', action='store_true', help='use OpenCV DNN for ONNX inference')
    parser.add_argument('--input-url', type=str, default='tcp://*:3000', help='Input ZMQ binding url')
    parser.add_argument('--input-topic', type=str, default='/demo/video/', help='Input topic')
    parser.add_argument('--output-url', type=str, default='tcp://*:9000', help='Output ZMQ binding url (for detection)')
    parser.add_argument('--output-topic', type=str, default='/demo/semantics/',
                        help='Output topic (for detection)')
    parser.add_argument('--json-url', type=str, default='tcp://*:9999', help='Output ZMQ binding url for KPIs')
    parser.add_argument('--json-topic', type=str, default='/demo/metadata/', help='Output topic for KPIs')
    parser.add_argument('--do-segment', action='store_true', help='Do segmentation? If not, detection')
    parser.add_argument('--just-one-class', type=str, default='None', help='Select the name of the class you wanto to detect')
    parser.add_argument('--plot-boxes', action='store_true', help='Draw classes boxes/mask in the output images')

    opt = parser.parse_args()
    opt.imgsz *= 2 if len(opt.imgsz) == 1 else 1  # expand
    print_args(vars(opt))
    return opt
```

Figura 39 Opciones de argumentos en el script detect_alga_forward.py

En el proyecto, se van a utilizar los siguientes argumentos:

- *'--weights'*, indica el archivo de pesos que se debe de cargar para la detección de objetos. El uso de los pesos preentrenados es lo que permite que el modelo sea capaz de realizar detecciones con alta precisión, ya que estos se encargan de encapsular el conocimiento del entrenamiento en grandes conjuntos de datos. Además, son muy útiles para transferir un conocimiento general sobre objetos en contexto a conjuntos de datos personalizados.[50]
- *'--classes'*, este argumento filtra los resultados según los objetos que se quieran detectar. En este caso, se quieren filtrar los resultados para la detección de personas y que se reaccione ante ésta a una determinada distancia.

- *'--plot-boxes'*, indica que se quiere dibujar las cajas delimitadoras de los objetos que se detectan en la imagen de salida.

A continuación, se ejecuta el comando con los argumentos especificados anteriormente.

```
root@durian-GL502VMK:/usr/src/app/yolo# python detect_alga_forward.py --weights yolov5s-seg.pt --plot-
boxes
yolo/detect_alga_forward: weights=['yolov5s-seg.pt'], data=../data/coco128.yaml, imgsz=[640, 640], con
f_thres=0.5, iou_thres=0.5, max_det=1000, device=, classes=None, agnostic_nms=False, augment=False, ha
lf=False, dnn=False, input_url=tcp://*:3000, input_topic=/demo/video/, output_url=tcp://*:9000, output
_topic=/demo/semantics/, json_url=tcp://*:9999, json_topic=/demo/metadata/, do_segment=False, just_one
_class=None, plot_boxes=True
requirements: /opt/conda/lib/python3.10/site-packages/requirements.txt not found, check failed.
YOLOv5 🚀 2024-1-18 Python-3.10.9 torch-2.0.0 CUDA:0 (NVIDIA GeForce GTX 1060, 6063MiB)

Fusing layers...
YOLOv5s-seg summary: 224 layers, 7611485 parameters, 0 gradients, 26.4 GFLOPs
Starting server...
Image received
Class Index: 0.0, Class Name: person, Color: [187, 123, 0]
Class Index: 56.0, Class Name: chair, Color: [199, 9, 3]
Class Index: 56.0, Class Name: chair, Color: [199, 9, 3]
Box: [2, 19, 210, 372], Color: [187, 123, 0], Text: person (0.727)
Box: [584, 224, 640, 364], Color: [199, 9, 3], Text: chair (0.720)
Box: [407, 238, 587, 430], Color: [199, 9, 3], Text: chair (0.586)
```

Figura 40 Comando para ejecutar por terminal el script detect_alga_forward.py

Como se puede observar en la Figura 40, si todo va como debe, aparecerá el mensaje *"Starting server…."*

Por otro lado, se abre un nuevo terminal dentro de la misma carpeta, y se ejecuta el script *alga_client_forward.py*. Este código se va a focalizar en recibir y mostrar las imágenes así como el resultado de la detección (metadatos).

Además, se establece una conexión UDP mediante sockets con el robot para poder informarle sobre la detección de obstáculos. Para ello, se indica en el código la IP y el puerto del robot que va a escuchar los mensajes UDP y se abre el socket con él.

```
UDP_IP = "192.168.31.162"  # IP address of the robot 192.168.250.52
UDP_PORT = 7000  # Port number for UDP communication

# UDP socket
sock_udp = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
```

Figura 41 Implementación de comunicación UDP con el robot

A continuación, se ejecuta un bucle contínuo que comprueba si se recibe información de detección desde el script *detect_alga_forward.py*.

```
while True:

    instances = metadata.recv().data
    if instances:
        print("Received metadata:", instances)
        metadata_str = str(instances)

        sock_udp.sendto(bytes(metadata_str, "utf-8"), (UDP_IP, UDP_PORT))
        #socket.sendto(str.encode(txt), (UDP_IP, UDP_PORT))

    #video frame from the server
    frame = input.recv().data

    # Display video frame
    cv2.imshow("RECEIVING VIDEO", frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
```

Figura 42 Bucle de detección de información en el script alga_client_forward.py

En primer lugar, se comprueba si se reciben metadatos (‘*instances’*) con el resultado de la detección. En caso afirmativo, se convierten en una cadena de “*strings*” y se envían en formato de bytes por medio del socket UDP creado. Después, se obtiene la imagen con las detecciones del servidor y se visualiza con ‘*cv2.imshow(“RECEIVING VIDEO”, frame)*’.

La configuración de las direcciones y parámetros para la recepción del metadatos se realiza en la función principal, '*main*':

```
if __name__ == '__main__':
    inaddr = 'tcp://192.168.31.204:9000' #IP address of computer
    metaaddr = 'tcp://192.168.31.204:9999'
    slice = '/demo/'
    device = 'client2'
    run(inaddr, metaaddr, slice, device)
```

Figura 43 Configuración de las direcciones del PC servidor

La variable *'inaddr'* establece la dirección IP y el puerto para recibir las imágenes desde el servidor y la variable *'metaaddr'* establece la dirección IP y el puerto para recibir los metadatos desde el servidor.

La ejecución del fichero *alga_client_forward.py* se realiza por terminal con el siguiente comando:

```
durian@durian-GL502VMK:~/yolo_alga$ python alga_client_forward.py
Waiting for images...
Received metadata: [{'inferences': 1, 'instances': 3, 'number_classes': 2, 'classes': ['person', '
r', 'chair'], 'boxes': [[2, 19, 210, 372], [584, 224, 640, 364], [407, 238, 587, 430]], 'pre_proces
s': 0.4162788391113281, 'inference_ms': 45.71843147277832, 'nms_ms': 169.99101638793945, 'post_proc
 ms': 88.83309364318848}]
```

Figura 44 Comando para ejecutar por terminal el script alga_client_forward.py

Tras tener funcionando el servidor que realiza la detección y el código de visualización, se ejecuta el GStreamer en el robot, para comenzar a enviar el vídeo al servidor con el siguiente comando:

```
bingda@robot:~$ gst-launch-1.0 -v v4l2src device=/dev/video0 ! 'image/jpeg,width
=640,height=480,framerate=30/1' ! jpegdec ! x264enc tune=zerolatency bitrate=409
6 speed-preset=superfast ! rtph264pay ! udpsink host=192.168.250.81 port=5000
Setting pipeline to PAUSED ...
Pipeline is live and does not need PREROLL ...
Setting pipeline to PLAYING ...
New clock: GstSystemClock
/GstPipeline:pipeline0/GstV4l2Src:v4l2src0.GstPad:src: caps = image/jpeg, width=
(int)640, height=(int)480, framerate=(fraction)30/1, colorimetry=(string)1:4:7:1
, interlace-mode=(string)progressive
/GstPipeline:pipeline0/GstCapsFilter:capsfilter0.GstPad:src: caps = image/jpeg,
width=(int)640, height=(int)480, framerate=(fraction)30/1, colorimetry=(string)1
:4:7:1, interlace-mode=(string)progressive
```

Figura 45 Comando para ejecutar el GStreamer

Este comando inicia una operación por medio de GStreamer, en la que se captura y transmite en tiempo real el vídeo obtenido por la cámara del robot. El flujo se envía hacia una dirección IP y puerto determinado, empleando el protocolo RTP.

Finalmente, debería de aparecer una ventana con el título de *"RECEIVING VIDEO"* de la siguiente forma:

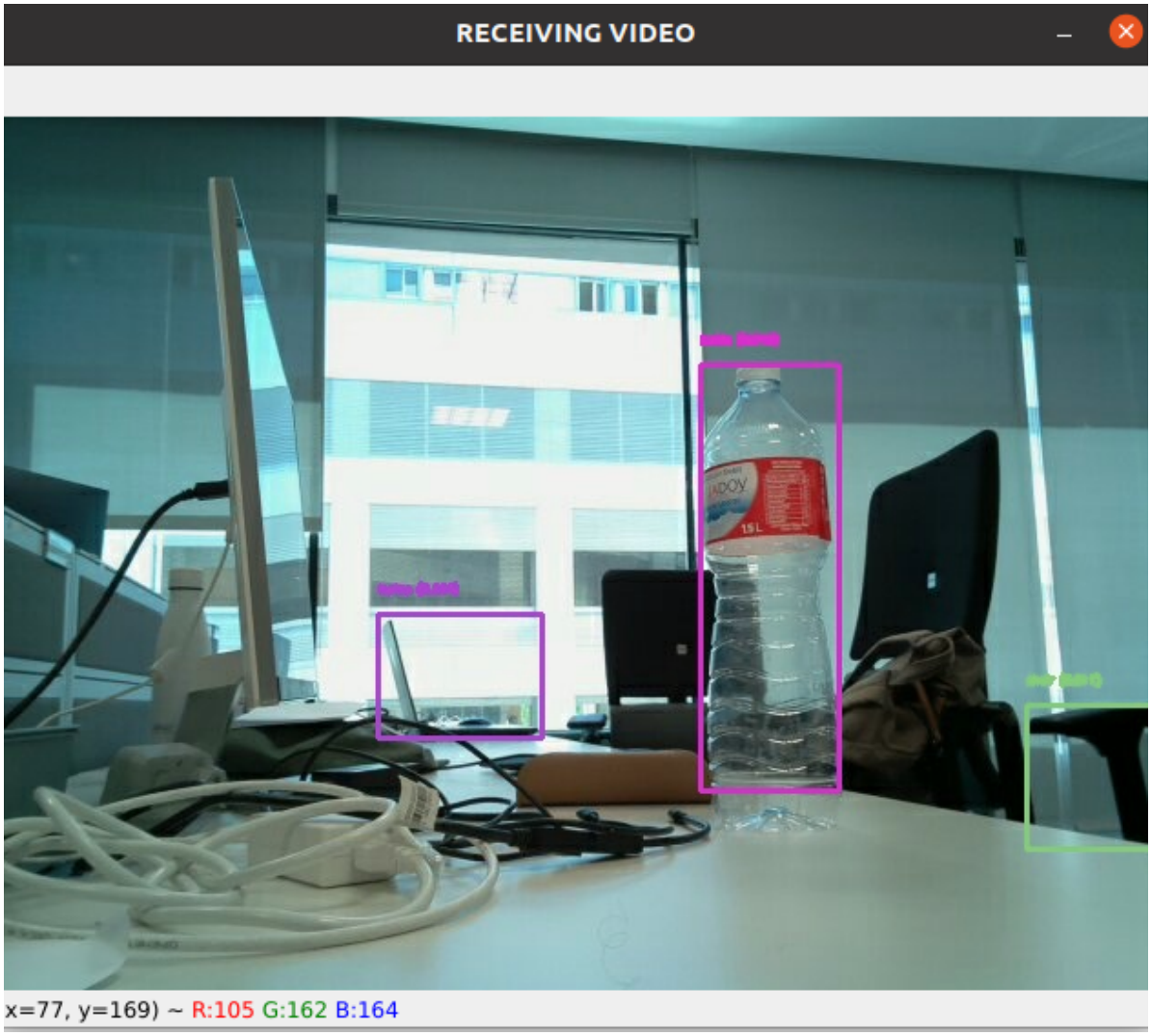

Figura 46 Secuencia de vídeo capturada por la cámara del robot

A su vez, se irán imprimiendo los resultados en el *detect_alga_forward.py* y en el *alga_client_forward.py*.

En primer lugar se analiza la salida del *detect_alga_forward.py*:

```
Image received
Class Index: 39.0, Class Name: bottle, Color: [216, 49, 204]
Class Index: 63.0, Class Name: laptop, Color: [170, 68, 207]
Class Index: 56.0, Class Name: chair, Color: [134, 204, 124]
Box: [388, 135, 464, 370], Color: [216, 49, 204], Text: bottle (0.914)
Box: [211, 273, 301, 342], Color: [170, 68, 207], Text: laptop (0.786)
Box: [567, 323, 640, 403], Color: [134, 204, 124], Text: chair (0.622)
Image received
Class Index: 39.0, Class Name: bottle, Color: [216, 49, 204]
Class Index: 63.0, Class Name: laptop, Color: [170, 68, 207]
Class Index: 56.0, Class Name: chair, Color: [134, 204, 124]
Box: [388, 136, 464, 371], Color: [216, 49, 204], Text: bottle (0.915)
Box: [210, 273, 301, 341], Color: [170, 68, 207], Text: laptop (0.793)
Box: [567, 323, 640, 402], Color: [134, 204, 124], Text: chair (0.614)
```

Figura 47 Objetos detectados a partir de la imagen

- Class Index: indica el índice de la clase detectada.
- Class Name: indica el nombre de la clase detectada.
- Color: señala el color assignado a la caja delimitadora de la clase detectada.
- Box: muestra las coordenadas de la caja delimitadora del objeto detectado. El formato es el siguiente: [x_min,y_min,x_max,y_max] [17]
- Text: contiene el nombre de la etiqueta del objeto detectado y entre paréntesis se encuentra lo que se conoce como "*confidence score*". Este número sirve como estimación para indicar el grado de confianza de que el objeto pertenezca a esa clase[51].

A continuación, se analiza la salida del *alga_client_forward.py*:

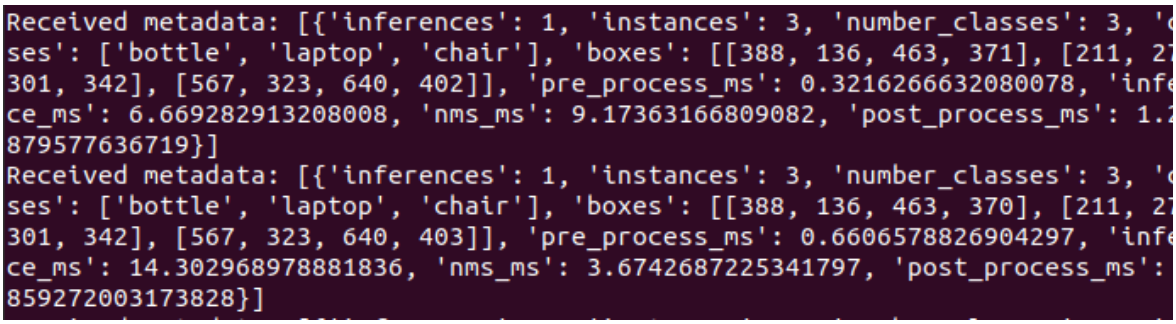

Figura 48 Información de metadatos generados a partir de la imagen

Por cada entrada de metadatos se recibe una lista que contiene un diccionario con los siguientes elementos:

- Inferences: Muestra un valor entero indiciando el número de inferencias. En este caso siempre va a haber una inferencia por entrada.
- Instances: Muestra un valor entero indicando el número de instancias detectadas. En este caso se han detectado 3 instancias en la imagen.
- Number_classes: Indicia el número de clases detectadas en la imagen.
- Classes: Expone los nombres de los tipos de objetos detectados indicados por la etiqueta o por el número de índice.
- Boxes: Igual que en el caso anterior, indica las coordenadas de la misma forma.
- Pre_process_ms: Representa el tiempo en milisegundos que tarda el pre-procesamiento de la imagen.
- Inference_ms: Es el tiempo en milisegundos en el que estima el paso de la inferencia o de la predicción real.
- Nms_ms: Esta variable estima en tiempo en milisegundos para evitar los no máximos, técnica utilizada para eliminar la redundancia de cajas delimitadoras.
- Post_process_ms: Es el tiempo en milisegundos necesario para el post-procesamiento de las imágenes.

Este resultado se representa en esta sección del código *detect_alga_forward.py*:

```
results = [
    {
        "inferences": 1,
        "instances": n_instances,
        "number_classes": n_classes,
        "classes": list(det_classes),
        "boxes": list(boxes),
        "pre_process_ms": t[0],
        "inference_ms": t[1],
        "nms_ms": t[2],
        "post_process_ms": t[3]
    }
]
response = metadata.response(request)
response.data = results
metadata.send(response)
```

Figura 49 Porción de código asociado a la salida por terminal

En conclusión, todo lo detallado anteriormente se sintetiza mediante el diagrama que se presenta a continuación:

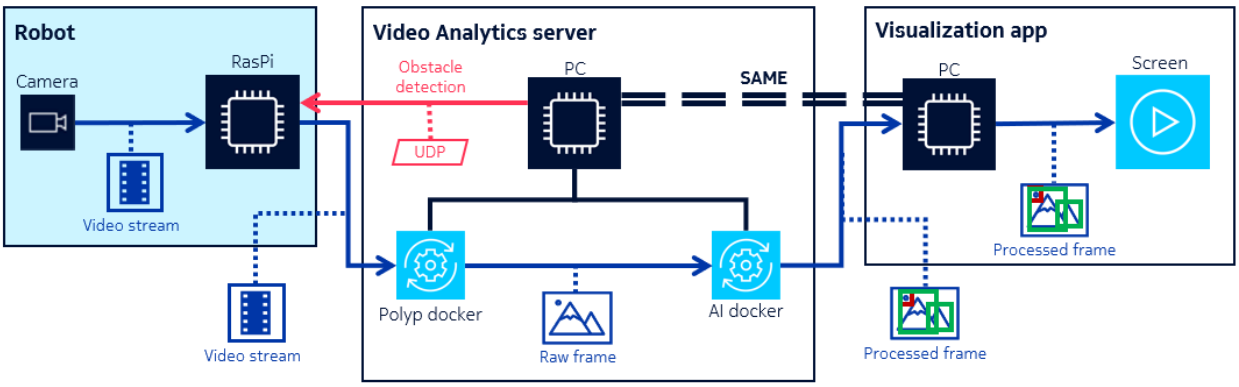

Figura 50 Diagrama del proceso de detección de objetos

La Figura 50 muestra a la izquierda el robot equipado con una cámara y una Raspberry Pi. Este robot es el que se mueve y transmite el flujo de video. En el centro, se encuentra el servidor de análisis de vídeo que obtiene el flujo video lo obtiene el ordenador servidor a partir de la cámara del robot por medio de una conexión en tiempo real, RTP.

El funcionamiento del servidor consta de los dos dockers descritos anteriormente. Por un lado, está el docker Polyp el cual obtiene el flujo de video y manda los frames sin procesar al docker de IA. Por medio del contenedor de YOLO dentro del docker de IA, se procesan los frames para realizar la detección de objetos.

Finalmente, la imagen procesada se muestra a través de una aplicación de visualización en el propio ordenador servidor, que se encarga también de informar al robot sobre la detección de los objetos en su entorno a través de una conexión UDP.

4.3 ROS.

ROS es un meta sistema operativo de código abierto enfocado al diseño de robots. Ofrece una serie de servicios típicos utilizados en sistemas operativos, entre estos, la abstracción de hardware, control de dispositivos de bajo nivel, funcionalidades comunes, mensajes entre procesos y gestión de paquetes. A su vez, facilita herramientas y bibliotecas para desarrollar código en todo tipo de computadoras. [1]

Se compone de procesos paralelos denominados "*nodes*", que se pueden ejecutar sobre diferentes máquinas en un mismo sistema de ROS. La comunicación entre nodos se lleva a cabo por medio del publicador y suscriptor, y a los canales de comunicación se les conoce como *"topics"*. En resumen, un nodo tiene la capacidad de publicar información sobre uno o más tópicos, y a su vez, se subscribe a diferentes tópicos para obtener la información compartida por otros nodos. Todo esto se tiene que llevar a cabo ejecutando el nodo principal conocido como el ROS Máster, el cual actúa como el directorio central, proporcionando el registro y localización de nodos para facilitar la comunicación entre ellos. [52]

4.3.1 PROCESO DE INSTALACIÓN

Hay varias versiones de ROS, este proyecto trabaja con la versión Noetic Ninjemys:

Distro	Release date	Poster	*Tuturtle*, turtle in tutorial	EOL date
ROS Noetic Ninjemys (**Recommended**)	May 23rd, 2020			May, 2025 (Focal EOL)
ROS Melodic Morenia	May 23rd, 2018			May, 2023 (Bionic EOL)
ROS Lunar Loggerhead	May 23rd, 2017			May, 2019
ROS Kinetic Kame	May 23rd, 2016			April, 2021 (Xenial EOL)

Figura 51 Diversas versiones de ROS [53]

Para poder instalar ROS es fundamental habilitar en Ubuntu los repositorios *"restricted"*, *"universe"* y *"multiverse"* [54]. De esta forma, aseguramos el acceso a todas las dependencias necesarias, incluyendo software específico y actualizaciones para el funcionamiento completo del sistema.

El proceso de instalación se basa en los siguientes pasos [54]:

1. Configuración del sources.list

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

   Por medio de este comando se configura el ordenador para que sea capaz de aceptar el software del repositorio de ROS. Se crea un nuevo archivo dentro del directorio

'*/etc/apt/sources.list.d/'* que contiene la URL del repositorio de ROS, permitiendo la descarga y la instalación de los paquetes de ROS.

2. Configuración de claves.
   sudo apt install curl # if you haven't already installed curl
   curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo apt-key add –
   La función de este comando es instalar *curl*, una herramienta de línea de comando utilizada para transferir datos con URLs. [55]. A su vez, el comando descarga y añade una clave GPG para poder añadir repositorios de ROS en tus claves de confianza. De esta forma, se asegura la autenticidad de los paquetes de ROS que se instalan.

3. Instalación.
   sudo apt update
   Primero se ejecuta el comando anterior cuya función es actualizar la lista de paquetes disponibles en los repositorios del sistema.[56]. Este paso es esencial antes de instalar o actualizar cualquier tipo de software de últimas versiones.

   A continuación, se hace la instalación de ROS *Noetic*:

   sudo apt install ros-noetic-desktop-full

   Con este comando, se instala ROS con todas sus características, de forma que se podrá trabajar en el área de robótica avanzada con procesamientos de imágenes, en entornos virtuales o en tiempo real.

4. Configuración del entorno.
   Para empezar la configuración del entorno primero, se ejecuta el siguiente comando por terminal:

   source /opt/ros/noetic/setup.bash

De esta forma se cargan las variables de entorno y funciones necesarias, como bibliotecas o paquetes, para usar ROS. Cualquier cambio adicional se realiza en el script ejecutado *setup.bash*.

5. Instalación de herramientas y dependencias necesarias.
   Las herramientas que se muestran en el comando se encargan de descargar, construir e instalar paquetes de ROS.

   sudo apt install python3-rosdep python3-rosinstall python3-rosinstall-generator python3-wstool build-essential

   - python3-rosdep: asegura tener las bibliotecas y herramientas necesarias para ejecutar paquetes de ROS[57].
   - python3-rosinstall: gestiona varios códigos fuente entre varios paquetes de ROS. [58]
   - python3-rosinstall-generator: genera archivos de rosinstall para un conjunto de paquetes de ROS.
   - python3-wstool: facilita el uso de los proyectos administrando múltiples repositorios de código fuente. [59]
   - build-essential: compila software desde el código fuente por medio de una variedad de herramientas de dessarollo, como por ejemplo el compilador gcc.[60]

6. Instalación de dependencias para ejecutar paquetes.
   Para empezar, se ejecuta el comando:

   sudo apt install python3-rosdep [54]

   Este commando instala *rosdep*, el cual se encarga de manejar las dependencias de los paquetes de ROS que se quieren ejecutar. [57]

   A posteriori, se ejecutan las siguientes líneas en terminal para inicializar y actualizar la base de datos local de *rosdep*.

Una vez que se ha realizado la instalación y se ha configurado ROS de manera correcta se crea el primer espacio de trabajo ROS. Para ello, se crea el espacio de trabajo de *catkin_ws* con los siguientes comandos [61]:

mkdir -p ~/catkin_ws/src

cd ~/catkin_ws/

catkin_make

A continuación, se obtiene el siguiente resultado:

Figura 52 Contenido dentro de la carpeta catkin_ws

Como se puede observar en la Figura 52, ahora se dispone de las carpetas *build* y *devel*. Por un lado la carpeta *build* contiene todos los archivos intermedios, entre ellos se llama a *cmake* y make para configurar y construir sus paquetes. [62]

Por otro lado, la carpeta devel contiene la parte de los ejecutables, bibliotecas y scripts.[62] Entre los scripts:

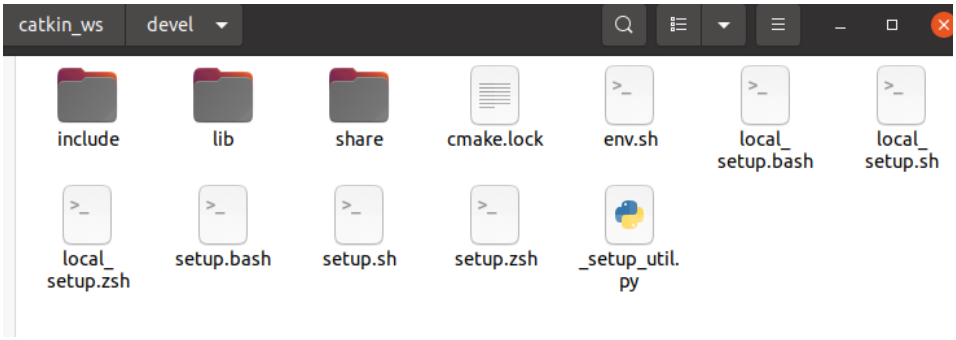

Figura 53 Contenido dentro de la carpeta devel

El script en el que se crea la atención es el *setup.bash*, este se ejecuta para poder configurar y preparar el entorno actual. De esta forma, se facilita el uso de las herramientas y comandos de ROS para encontrar y acceder a los paquetes correctamente, y poder ejecutar los programas de ROS de manera eficaz.

A continuación, para empezar a programar proyectos en ROS se crea un primer paquete con los siguientes comandos por terminal [63]:

cd ~/catkin_ws/

ls src

beginner_tutorials/ CMakeLists.txt@

Por cada cambio que se realiza en la carpeta de *catkin_ws*, se tiene que ejecutar posteriormente el comado: *'catkin_make'*.

Este comando tiene tres propósitos fundamentales [64]:

1. Compilación : compila el código fuente de los paquetes, y a su vez, genera ejecutables y bibliotecas necesarias para el funcionamiento.
2. Dependencias: Compilación de las dependencias en los diferentes paquetes.
3. Entorno de Ejecución: se tiene un entorno en el que se facilita la ejecución, el desarrollo y las pruebas de los programas en ROS.
4. Creación archivos: Crea todo tipo de archivos necesarios para la compilación de los nodos de ROS.

Al hacer el '*catkin_make'* en la carpeta *catkin_ws,* si todo se ha configurado correctamente, debería de salir el siguiente resultado:

```
durian@durian-GL502VMK:~$ cd catkin_ws
durian@durian-GL502VMK:~/catkin_ws$ catkin_make
Base path: /home/durian/catkin_ws
Source space: /home/durian/catkin_ws/src
Build space: /home/durian/catkin_ws/build
Devel space: /home/durian/catkin_ws/devel
Install space: /home/durian/catkin_ws/install
####
#### Running command: "make cmake_check_build_system" in "/home/durian/catkin_ws/build"
####
####
#### Running command: "make -j8 -l8" in "/home/durian/catkin_ws/build"
####
[  0%] Built target sensor_msgs_generate_messages_cpp
[  0%] Built target std_msgs_generate_messages_py
[  0%] Built target std_msgs_generate_messages_cpp
[  0%] Built target sensor_msgs_generate_messages_py
[  0%] Built target std_msgs_generate_messages_eus
[  0%] Built target sensor_msgs_generate_messages_eus
[  0%] Built target std_msgs_generate_messages_lisp
[  0%] Built target sensor_msgs_generate_messages_lisp
[  0%] Built target std_msgs_generate_messages_nodejs
[  0%] Built target sensor_msgs_generate_messages_nodejs
[  0%] Built target _ds4_driver_generate_messages_check_deps_Feedback
[  0%] Built target _ds4_driver_generate_messages_check_deps_Trackpad
[  0%] Built target _ds4_driver_generate_messages_check_deps_Report
[  0%] Built target _ds4_driver_generate_messages_check_deps_Status
[ 18%] Built target ds4_driver_generate_messages_cpp
[ 40%] Built target ds4_driver_generate_messages_py
[ 59%] Built target ds4_driver_generate_messages_lisp
[100%] Built target ds4_driver_generate_messages_nodejs
[100%] Built target ds4_driver_generate_messages_eus
[100%] Built target ds4_driver_generate_messages
```

Figura 54 Resultado del comando catkin_make

4.3.2 CÓDIGOS ROS.

4.3.2.1 Códigos ROS con Gazebo.

Para comenzar a tratar con ROS se probaron los códigos dados en C++ y en Python. En este caso, se tomaron como ejemplo los códigos obtenidos del GitHub *cmauricioae8* con el nombre *ros_cpp_py_basics,* mostrado en el apartado, "Códigos externos utilizados", del Capítulo 2. En un principio, se ejecutaron los códigos con su respectivo *launch* siguiendo los pasos puestos por el autor en cada fichero dentro de la carpeta *src*. Se compilaron los códigos *simple_motion_diff_robot.py* y *diff_robot_controller.py* para poder observar cómo funcionaba y poder tomarlo como referencia para el robot, pero principalmente se analizó el código *simple_motion_diff_robot.py.*

El objetivo del primer código es publicar las velocidades en el tópico '/cmd_vel' para hacer que el robot se mueva en círculos y, a su vez, hacer una subscripción a la odometría desde el tópico '*/odom'* para que el robot se localice a sí mismo gracias a su propio sistema de odometría[65].

Por otro lado, el segundo código se focaliza en enviar al tópico '*/cmd_vel'* las velocidades calculadas para seguir una cierta trayectoria y publicar los errores de seguimiento publicarlos en el tópico '*/tracking_errors'*. Posteriormente, procesa la posición y orientación del robot obtenidas del tópico '*/odom'*. Una utilidad adicional que se implementa en este código es la interacción con el usuario por medio de la tecla '*ESC'* o '*q'* para finalizar el nodo.

4.3.2.2 Códigos ROS modificados con Gazebo.

En primer lugar, se muestra un esquema de los códigos que se van a ejecutar dentro de la carpeta *catkin_ws*. Para la simulación se trabajó en la misma carpeta del proyecto descargado de GitHub y la simulación del robot se lanzó con uno de los *launch* establecidos.

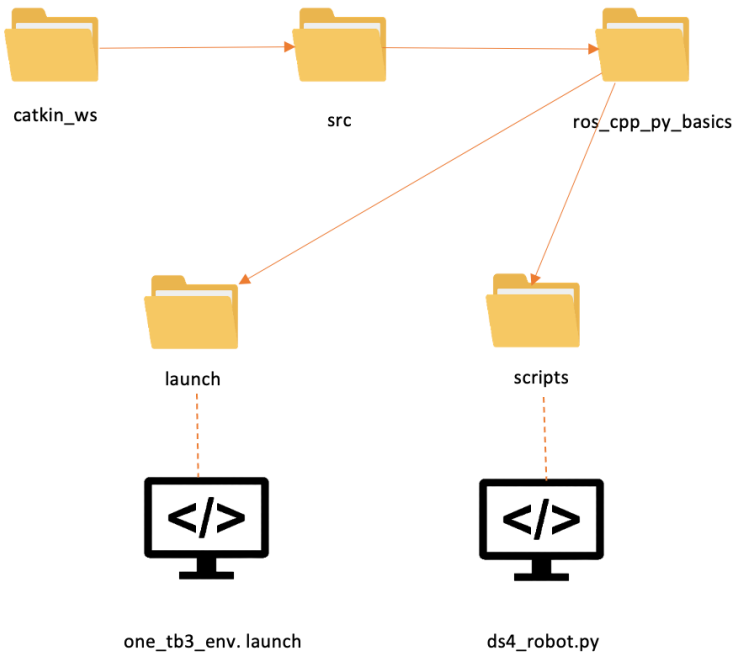

Figura 55 Esquema de códigos utilizados para la simulación [35][36]

Inicialmente, se fue experimentado con los códigos en C++ del proyecto, pero al probar también los códigos de Python y entender su funcionamiento mejor, la programación del robot se focalizó en el uso de Python.

Los requisitos a seguir para el movimiento del robot se resumen en los siguientes:

1. Para arrancar el robot y poder controlarlo el robot se necesita presionar el botón R1. En caso de que no se presione, el robot se mantendrá parado independientemente de lo que se marque en el mando.

2. Por medio de los dos joysticks analógicos se controla la dirección del robot. Para mover el robot en el eje x de manera lineal se utiliza el joystick de la izquierda, y para desplazar el robot en el eje angular se utiliza el joystick derecho.
3. Aumentar la velocidad del robot con el botón triángulo.
4. Decrementar la velocidad del robot con el botón X.

La funcionalidad de todos los requisitos se concentra en esta sección del código:

```
if current_state_button_start == 1:
    self.rl_pressed = True
else:
    self.rl_pressed = False

if self.rl_pressed:
    vel_msg.linear.x = joy_msg.axes[linear_axis_left]
    vel_msg.linear.y = joy_msg.axes[linear_axis_right]
    e_orientation = 0.0 - joy_msg.axes[angular_axis_right]
    vel_msg.angular.z = self.ctrld * e_orientation

    current_state_button = joy_msg.buttons[max_vel_buttons_tr]
    current_state_button2 = joy_msg.buttons[decrease_vel_buttons_x]


    if current_state_button == 1 and self.prev_max_vel_button_state == 0:
        self.speed_scale += 2
    elif current_state_button2 == 1 and self.prev_max_vel_button_state2 == 0:
        self.speed_scale -= 0.5
    self.prev_max_vel_button_state = current_state_button
    self.prev_max_vel_button_state2 = current_state_button2

    # Range of velocity
    self.speed_scale = max(self.vel_low, min(self.speed_scale, self.vel_max))

    vel_msg.linear.x *= self.speed_scale
    vel_msg.linear.y *= self.speed_scale
    vel_msg.angular.z *= self.speed_scale

    print(self.speed_scale)

    self.velocity_pub.publish(vel_msg)

else:
    self.velocity_pub.publish(Twist())
```

Figura 55 Funcionalidad del código del robot

Para que esta implementación funcione, primero se inicia un nodo ROS con el nombre *ps4_controller_teleop* ,y, a continuación, se crea la variable '*velocity_pub'* para publicar los mensajes de tipo '*Twist'* en el tópico '*\cmd_vel'* para controlar el movimiento del robot. Los mensajes tipo '*Twist'* siguen una estructura de datos que representa la velocidad lineal y angular de un robot en ROS. Se emplea '*Twist.linear'* para representar la velocidad lineal y '*Twist.Angular' para* la velocidad angular en las tres direcciones (x,y,z).[66]

Posteriormente se crea una variable en la que se subscriben los mensajes que se reciben del tipo *'Joy'* por medio del controlador de la PS4 en el tópico '*/joy'*. El código correspondiente a los nodos y los tópicos se muestra en el Capítulo 5, al mostrar la interacción por medio de la herramienta de ROS *rqt_graph*.

Todo el código mostrado en la Figura 55 se implementa dentro de la función *joy_callback*, que se activa cada vez que recibe un mensaje del joystick. Por medio del código se permite ajustar los movimientos y la velocidad del robot, para ello se publica un mensaje '*Twist'* en el tópico *'/cmd_vel'*.

4.3.2.3 Códigos ROS del robot físico.

En este caso la estructura de carpetas es diferente a la utilizada en la simulación. La carpeta de *catkin_ws* del robot contiene varias carpetas y códigos que no se utilizan en este proyecto, en concreto, por ello, en el siguiente esquema se destacan las carpetas y los códigos utilizados:

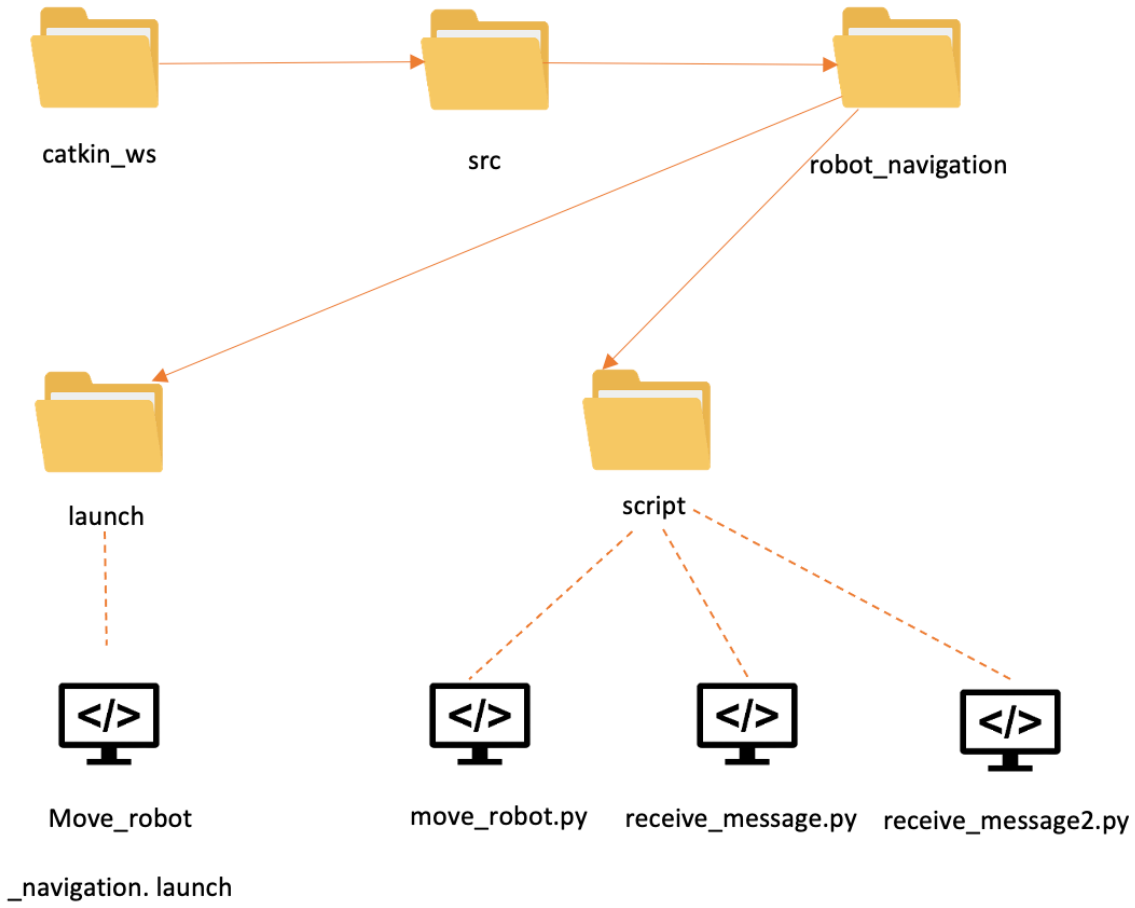

Figura 56 Códigos utilizados para el proyecto con el robot físico [35][36]

El código *move_robot.py* contiene la misma lógica que el código *ds4_robot.py,* explicado en el apartado anterior. Las únicas variaciones del código son las siguientes:

- El cambio del valor de la velocidad angular del robot físico con respecto al robot de la simulación, debido a el eje rotacional.
- Se introduce una nueva función que gestionará los mensajes que se reciben desde los códigos *receive_message.py* o *receive_message2.py* para frenar el robot en caso de detectar a una persona.

Los resultados de ambas variaciones y su explicación se muestran en detalle en el apartado, "Resultados en el robot" del Capítulo 5.

Antes de analizar los resultados, se va a explicar el funcionamiento de los códigos receive_message.py que se asocia con la Parte I y receive_message2.py que se asocia con la Parte II.

4.3.2.3.1 Parte I.

A la hora de integrar el código con el robot físico ha sido relativamente sencillo ya que el código de la simulación plasmó bien lo que había que integrar. Una vez trasladados al robot los códigos asociados con su movimiento, se procedió a establecer la comunicación entre el PC servidor y el robot. Esto se consigue por medio de un socket UDP que se realizó creando el código *receive_message.py*:

```
#Creating a new and different node
rospy.init_node('udp_ros')

#Creation of a ros topic to publish
stoprobot_pub = rospy.Publisher('/stop_robot', String, queue_size=10)

#UDP socket
UDP_IP = "192.168.31.162" #ip address robot
UDP_PORT = 7000
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((UDP_IP, UDP_PORT))
```

Figura 57 Integración comunicación UDP en el robot

En primer lugar, se crea el nuevo nodo ROS y el nuevo tópico *'/stop_robot'* en el que se van a publicar los mensajes de tipo *String*. A continuación, entramos en un bucle *'while'* que va a analizar los datos obtenidos por el Socket.

```
while not rospy.is_shutdown():
    data, addr = sock.recvfrom(1024)
    ai_data = data.decode('utf-8')
    #print("Received message:", ai_data)
```

Figura 58 Bucle while para analizar los datos

El script actúa como como punto de entrada para señales externas inicializando un nodo ROS '*robot_controler*'. Analiza el mensaje recibido por medio de UDP y si contiene la palabra "*person*", se publica un mensaje al tópico '*/stop_robot*' que dice: "*Person detected, stop robot*". En caso contrario se publicará "*Person not detected, safe*". Esta funcionalidad del código se detallará posteriormente en el apartado, "Resultados en el robot", del Capítulo 5.

4.3.2.3.2 Parte II.

La primera versión del código se focaliza en detectar a una persona y detener al robot, posteriormente se pasó a mejorar la condición de parada de tal forma que el robot se detuviese a una distancia muy cercana al humano. Este código se implementa en el script *receive_message2.py*.

La comprobación de distancia se puede desarrollar de varias formas, en este caso se optó por realizarlo mediante cálculo de áreas, tomando como referencia el código de una compañera, que hizo un trabajo similar con un robot y software completamente distintos. En el caso de referencia, el robot tiene forma de perro y dispone de dos cámaras, por lo que el reparto de las imágenes y por tanto de las áreas era distinto este caso.

Se tomó como referencia el cálculo del área de la caja que contiene a la persona detectada dentro de un frame para poder compararla con la totalidad de la imagen. En caso de que el área ocupe más de un 60% del total de la imagen se identifica como muy cercano, impidiendo el movimiento. La configuración del socket UDP y el bucle principal para obtener los metadatos coincide con el código de la parte I.

Para realizar el cálculo de área hay que tener en cuenta el ancho y el alto del frame definido anteriormente en el pipeline GStreamer de la siguiente forma:

```
bingda@robot:~$ gst-launch-1.0 -v v4l2src device=/dev/video0 ! 'image/jpeg,width
=640,height=480,framerate=30/1' ! jpegdec ! x264enc tune=zerolatency bitrate=409
6 speed-preset=superfast ! rtph264pay ! udpsink host=192.168.250.81 port=5000
```

Figura 59 Altura y Anchura establecidas en el GStreamer

Según la Figura 59, se tiene una anchura de 640 píxeles y una altura de 480 píxeles. Teniendo esto en cuenta se calcula el área total del frame de la siguiente forma:

```
# Frame dimensions
WIDTH = 640.0
HEIGHT = 480.0
AREA = WIDTH * HEIGHT
print(WIDTH, HEIGHT, AREA)
```

Figura 60 Implementación de las dimensiones en el código

A continuación, se entra en el bucle '*while*' establecido anteriormente (parte I) y se añade lo siguiente para el cálculo del área de la caja correspondiente a la persona detectada:

```
try:
    # Decode the data and convert it into a listt
    ai_data_list = eval(ai_data)


    #Process of the first dictionary
    ai_data_dict = ai_data_list[0]


    if "person" in ai_data_dict['classes']:
        person_index = ai_data_dict['classes'].index("person")
        frame_box = ai_data_dict['boxes'][person_index]
        frame_area = (frame_box[2] - frame_box[0]) * (frame_box[3] - frame_box[1])
        print("F.AREA:", frame_area, " - AREA:", AREA)
        obj_area = frame_area/AREA

        print("OBJ. AREA:", frame_area/AREA)
```

Figura 61 Cálculo de área según las dimensiones

La cadena de datos recibida por UDP se transforma en una lista de Python y se extrae el primer elemento de la lista, que es el diccionario que contiene la información. A continuación, si se ha detectado una persona, se busca la posición en la que se encuentra la persona en la detección, se guarda y se obtiene la caja delimitadora con ella.

Posteriormente, se calcula el área de la caja delimitadora multiplicando su anchura por su altura[18]:

- Cálculo de la anchura de la caja delimitadora: frame_box[2] – frame_box[0]
- Cálculo de la altura de la caja delimitadora: frame_box[3] – frame_box[1]

Finalmente, se divide el área de la caja delimitadora de la persona entre el área total del frame para obtener el área relativa del objeto.

Los resultados asociados al código se presentan en el apartado, “Resultados en el robot”, del Capítulo 5.

Capítulo 5. ANÁLISIS DE RESULTADOS

5.1 RESULTADOS EN GAZEBO

Una vez realizadas las pruebas con el código obtenido de GitHub, *ros_cpp_py_basics*, se pone a prueba el código modificado con los requisitos. Dentro del código *launch*, se modifica el fichero *world* que es el mundo de simulación en el que se encuentra el robot para añadir unos bloques por si acaso añadía la detección de obstáculos en la simulación.

Dentro del *launch* se añaden todos los códigos necesarios para realizar una ejecución en consola con un solo comando.

```
<arg name="world_file" default="$(find ros_cpp_py_basics)/worlds/my_world.world"/>
<node name="gazebo" pkg="gazebo_ros" type="gazebo" args="$(arg world_file)"/>

<!-- Send the robot XML to param server -->
<param name="robot_description" command="$(find xacro)/xacro --inorder $(find turtlebot3_description)/urdf/turtlebot3_$(arg model).urdf.xacro" />

<!-- Push robot_description to factory and spawn robot in Gazebo -->
<node name="spawn_urdf" pkg="gazebo_ros" type="spawn_model" args="-urdf -model $(arg robot_name) $(arg robot_posture) -param robot_description" />

<!-- Run the joy_node -->
<node pkg="joy" type="joy_node" name="joy_node" />

<node pkg="ros_cpp_py_basics" type="ds4_robot.py" name="ds4_robot" output="screen"/>

<!-- <node pkg="ros_cpp_py_basics" name="ds4_chat_test.py" type="ds4_chat_test.py" output="screen"/>-->
```

Figura 62 Implementación de los códigos en el launch

En la imagen anterior se observan dentro de los recuadros los códigos del proyecto que se quieren integrar para que se ejecuten todos al mismo tiempo a la hora de ejecutar el siguiente comando por terminal:

```
durian@durian-GL502VMK:~$ roslaunch ros_cpp_py_basics one_tb3_env.launch
```

Figura 63 Comando para ejecutar el fichero launch

Antes de ejecutar el comando anterior es necesario cargar las configuraciones del entorno de trabajo de *catkin_ws* de ROS *Noetic*, de esta forma se asegura que las variables necesarias están configuradas de manera correcta.

```
durian@durian-GL502VMK:~$ source ~/catkin_ws/devel/setup.bash
durian@durian-GL502VMK:~$ env | grep ROS
ROS_VERSION=1
ROS_PYTHON_VERSION=3
ROS_PACKAGE_PATH=/home/durian/catkin_ws/src:/opt/ros/noetic/share
ROSLISP_PACKAGE_DIRECTORIES=/home/durian/catkin_ws/devel/share/common-lisp
ROS_ETC_DIR=/opt/ros/noetic/etc/ros
ROS_MASTER_URI=http://localhost:11311
ROS_ROOT=/opt/ros/noetic/share/ros
ROS_DISTRO=noetic
```

Figura 64 Configuraciones del entorno del trabajo de catkin_ws

Al ejecutar el fichero con todos los códigos necesarios, debe de iniciarse la ventana del simulador Gazebo y mostrar al robot dentro del mundo tridimensional creado de la siguiente forma:

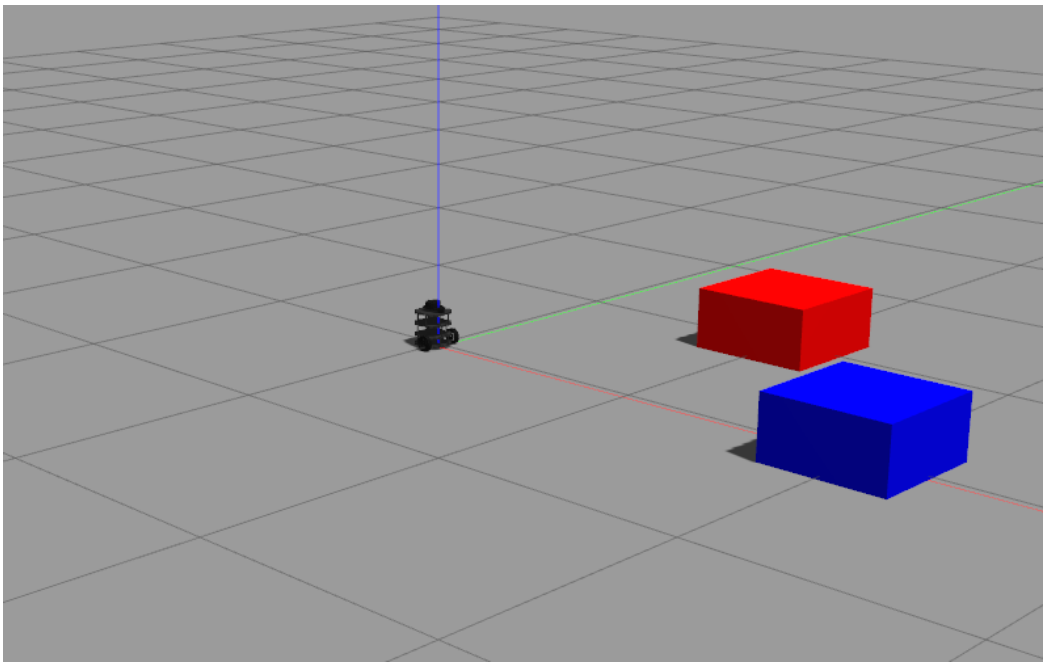

Figura 65 El robot en el simulador Gazebo

Si se mueve el robot, en la terminal se irá imprimiendo constantemente la velocidad que lleva el robot a medida que éste vaya avanzando:

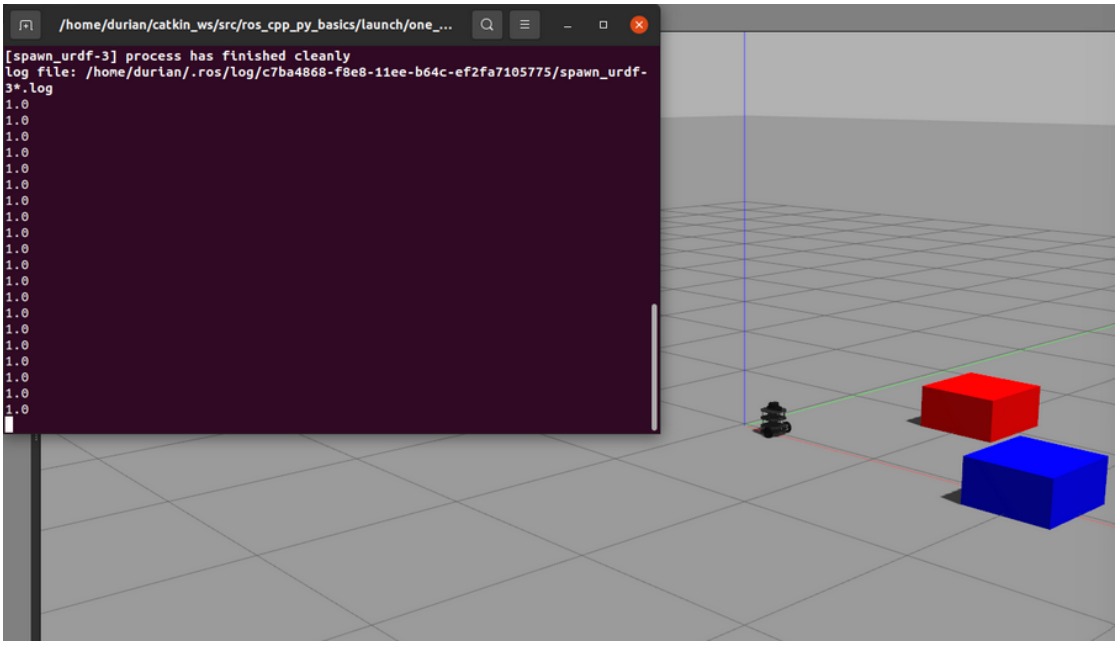

Figura 66 Navegación del robot en Gazebo

La razón por la que se imprime la velocidad que lleva el robot es para comprobar que los valores concuerdan con la lógica del código:

```
current_state_button = joy_msg.buttons[max_vel_buttons_tr]
current_state_button2 = joy_msg.buttons[decrease_vel_buttons_x]


if current_state_button == 1 and self.prev_max_vel_button_state == 0:
    self.speed_scale += 2
elif current_state_button2 == 1 and self.prev_max_vel_button_state2 == 0:
    self.speed_scale -= 0.5
self.prev_max_vel_button_state = current_state_button
self.prev_max_vel_button_state2 = current_state_button2
```

Figura 67 Porción de código que gestiona los cambios de velocidad

Si se pulsa el botón del triángulo de la consola, la velocidad aumenta dos unidades:

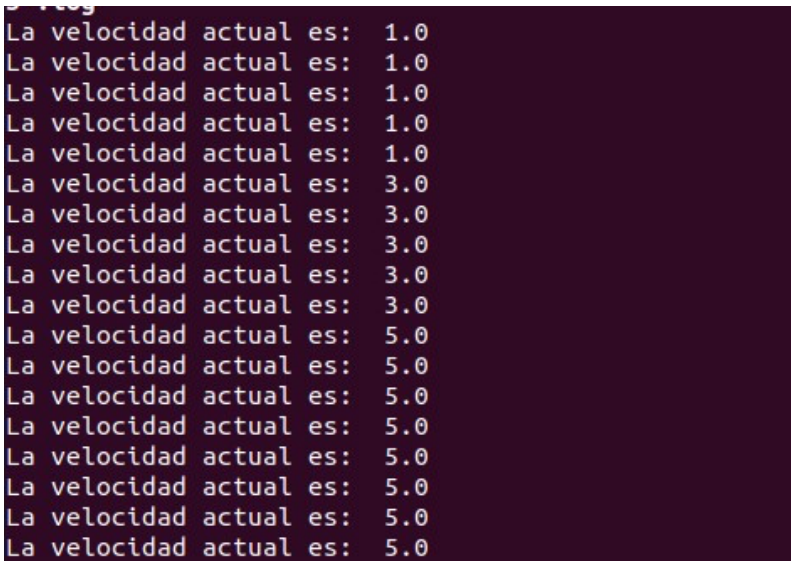

Figura 68 Demostración de aumento de la velocidad del robot

Si se pulse el botón X de la consola, la velocidad se reduce con un valor de 0.5:

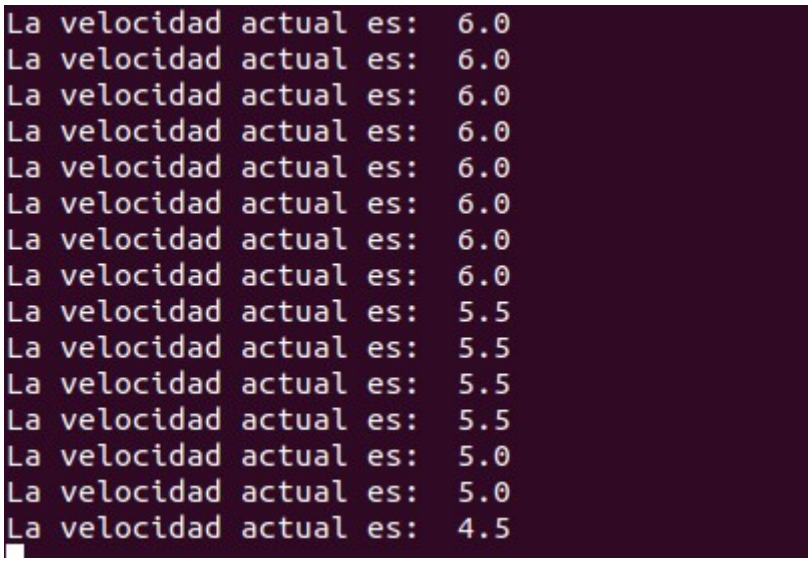

Figura 69 Demostración de la disminución de la velocidad del robot

Los valores de la velocidad siempre se mantendrán dentro de los rangos de velocidad establecidos por el código, que son los mismos tanto en el código de la simulación como en el robot.

```
self.vel_max = 10
self.vel_low = 0.5
```

Figura 70 Valores límite de la velocidad

Se decidió establecer un rango máximo de velocidad de 10 y un rango mínimo de 0.5 para que el robot no se detuviera. El robot solo estará parado si no se pulsa el botón R1. Para mantener la velocidad del robot dentro de los límites, se define el rango de velocidad de la siguiente forma:

```
self.speed_scale = max(self.vel_low, min(self.speed_scale, self.vel_max))
```

Figura 71 Rango de la velocidad permitida

5.2 RESULTADOS EN EL ROBOT

Una vez comprobado el funcionamiento del código en el entorno de simulación Gazebo, se transfirió al robot físico. En Gazebo, el código estaba diseñado para controlar un robot basado en las coordenadas y los ejes definidos en ese entorno virtual. Debido a esto, al pasar al robot físico fue necesario revisar y ajustar el código a las coordenadas y los ejes definidos en el hardware del robot. Los pasos a seguir para la implementación:

1. Verificar la orientación de los ejes: en la simulación de Gazebo, los ejes X,Y,Z estan orientados de manera diferente en comparación con el robot. Mientras que en Gazebo utilizaba todos los ejes para mover el robot, en este caso simplemente se

necesitan los ejes X y Z. En el eje X el robot tiene el mismo funcionamiento que en Gazebo, mientras que en el Z había que cambiar la orientación multiplicando el valor por un ángulo de 90 grados para que el robot puediese girar según su eje de rotación.

2. Ajustar la velocidad: se tiene que comprobar que la velocidad establecida por defecto y los límites establecidos concuerdan con el robot. Hay que tener más precaución con la velocidad que se establece al robot, ya que este puede ser más sensible a ello.
3. Considerar las diferencias físicas del entorno: en la simulación de Gazebo no se tienen en cuenta factores como la fricción o la resistencia del aire, que pueden influir en el comportamiento del robot. Además, el robot está en un entorno con muchos más obstáculos, los cuales hay que tener en cuenta para poder evitarlos. En este caso, no influyeron factores externos al robot, pero se tuvo que modificar el código para la detección de objetos en el entorno físico.

Para conseguir la detección de obstáculos, se realizaron dos enfoques al proyecto. En la primera parte del estudio, como se ha mostrado en el apartado, "Códigos ROS" , del Capítulo 4, el enfoque principal del código se basa en la detención automática del robot ante la presencia de una persona. Para poder ejecutar todo en una línea de comandos y obtener los resultados deseados, en el fichero launch se añaden todos los códigos a ejecutar de la siguiente forma:

```
<launch>

    <!--<node name="robot_pose_publisher" pkg="robot_pose_publisher" type="robot_pose_publisher" />-->
    <node name="multi_mark" pkg="robot_navigation" type="move_robot.py" output="screen" />

    <!--Launch of the Receive message code -->
    <node name="receiver" output="screen" pkg="robot_navigation" type="receive_message.py" />
    <!--<node name="receiver" output="screen" pkg="robot_navigation" type="receive_message2.py"/>-->

    <!-- node pkg="robot_navigation" type="receive_message.py" output="screen" -->


     <!-- Run the joy_node -->
    <node pkg="joy" type="joy_node" name="joy_node" />

</launch>
```

Figura 72 Fichero launch del robot físico

Como se puede observar en la Figura 72, se señala en rojo las líneas que corresponden, por un lado, al control del movimiento del robot y, por otro, al control de los datos de inteligencia artificial obtenidos.

Al tener todos los códigos necesarios puestos en el fichero de la Figura 72, para ejecutar todo el proyecto simplemente tenemos que lanzar el siguiente comando en la terminal del robot:

```
bingda@robot:~$ roslaunch robot_navigation move_robot_navigation.launch
```

Figura 73 Línea de comando para ejecutar el launch del robot

5.2.1 RESULTADOS CON EL CÓDIGO UDP

En este caso si el código principal recibe el mensaje publicado en el tópico '*/stop_robot*' a través del código *receive_message.py*, siendo el mensaje: '*stoprobot_pub.publish("Stop robot")*', el robot se deberá de parar y se deberá de imprimir por terminal el mensaje "*Person detected, stop robot*". En el caso de que no reciba ese mensaje, el robot podrá seguir circulando y se imprimirá por terminal el mensaje "*Person not detected, safe*".

```
if "'person'" in ai_data:
    stoprobot_pub.publish("Stop robot")
    print("Person detected, stop robot")

else:
    stoprobot_pub.publish("Person not detected")
    print("Person not detected, safe")
```

Figura 74 Generación y publicación de mensajes según la detección de una persona

En el caso de que el código principal del robot reciba el mensaje "*Stop robot*" publicado por el tópico '*/stop_robot'*, frenará el movimiento del robot y pondrá la variable booleana de '*person_detected'* a *True*. En el caso contrario la variable booleana de '*person_detected*' estará a *False*.

```
def stop_robot_callback(self, stop_msg):
    #We obtain the message that comes from the ros topic /stop_robot
    print("Received stop message:", stop_msg.data)
    if stop_msg.data == "Stop robot":
    #if stop_msg.data == "Person detected is too close, stopping robot":
        self.person_detected=True

        self.velocity_pub.publish(Twist())
    else:
        self.person_detected=False
```

Figura 75 Función que analiza los mensajes publicados

Si la variable booleana de '*person_detected'* está a *True*, dentro de la función de '*joy_callback'* el movimiento del robot se detendrá y se impedirá que el robot pueda ser controlado a través del mando remoto. Esta funcionalidad se conseguirá con el siguiente condicional:

```
if self.person_detected:
    self.velocity_pub.publish(Twist())
```

Figura 76 Control de velocidad basado en la detección de personas

Al ejecutar ambos códigos, se obtienen los siguientes resultados en terminal:

```
Person not detected, safe
('Received stop message:', 'Person not detected')
Person detected, stop robot
('Received stop message:', 'Stop robot')
```

Figura 77 Resultado de los mensajes publicados en terminal

En el primer caso, no se detecta a ninguna persona en el frame por lo que el robot sigue con su trayectoria controlada por el usuario. Aparece el mensaje "*Person not detected, safe*" y el mensaje que obtiene el código principal a través del tópico '*/stop_robot'*, que indicia la situación en la que se encuentra el robot.

En el segundo caso, una persona se pone delante de la cámara del robot y como se puede observar en la Figura 77, aparece el mensaje "*Person detected, stop robot*" y el mensaje publicado "*Stop robot*", indicando al robot que debe de pararse.

5.2.2 RESULTADOS CON EL CÓDIGO UDP Y EL CÁLCULO DE ÁREAS

En el caso de la segunda parte, se programa un código más avanzado para la detección de las personas basado en el cálculo de áreas. El resultado que debe aparecer en la terminal se encuentra definido en la lógica definida en el fichero *receive_message2.py*:

```
            if obj_area > 0.6:
                print("Person detected is too close, stopping robot.")
                stoprobot_pub.publish("Person detected is too close, stopping robot")

            else:
                print("Person detected, still safe")
                stoprobot_pub.publish("Person detected, still safe")

        else:
            print("Person not detected, safe.")
            stoprobot_pub.publish("Person not detected, safe")

    except Exception as e:
        print("Error processing AI data:", e)
        stoprobot_pub.publish("Error processing AI data")
```

Figura 78 Gestión de personas basada en su proximidad

En este código tenemos 3 escenarios. En primer lugar, no se detecta a la persona por lo que el robot esta seguro. En segundo lugar, se detecta una persona pero a una distancia lejos del robot que asegura que se encuentra en un estado seguro. En tercer lugar, la persona abarca un 60% o más de la visión del robot, indicando que está demasiado cerca y, por tanto, provoque que el robot deba detenerse.

```
root@durian-GL502VMK:/usr/src/app/yolo# python detect_alga_forward.py --weights
 yolov5s-seg.pt --plot-boxes --classes 0
yolo/detect_alga_forward: weights=['yolov5s-seg.pt'], data=../data/coco128.yaml
, imgsz=[640, 640], conf_thres=0.5, iou_thres=0.5, max_det=1000, device=, class
es=[0], agnostic_nms=False, augment=False, half=False, dnn=False, input_url=tcp
://*:3000, input_topic=/demo/video/, output_url=tcp://*:9000, output_topic=/dem
o/semantics/, json_url=tcp://*:9999, json_topic=/demo/metadata/, do_segment=Fal
se, just_one_class=None, plot_boxes=True
requirements: /opt/conda/lib/python3.10/site-packages/requirements.txt not foun
d, check failed.
YOLOv5 🚀 2024-1-18 Python-3.10.9 torch-2.0.0 CUDA:0 (NVIDIA GeForce GTX 1060,
6063MiB)

Fusing layers...
YOLOv5s-seg summary: 224 layers, 7611485 parameters, 0 gradients, 26.4 GFLOPs
Starting server...
```

Figura 79 Línea de comando para la detección de personas

En la Figura 79 se muestra la línea de comando a ejecutar por consola para que el sistema se focalice únicamente en la detección de personas. Esto se consigue añadiendo al comando el argumento '*--classes 0*'. Cada clase lleva asociada un número entero que hace referencia al objeto detectado; en este caso al detectar una persona el número entero asociado a la clase de persona es 0.

Los resultados son los siguientes:

```
Person not detected, safe.
Person not detected, safe.
Person not detected, safe.
Person detected, still safe
Person detected, still safe
Person detected, still safe
Person detected, still safe
Person detected, still safe
```

Figura 80 Registro de detección de personas en tiempo real

Al principio, se inicia la ejecución del código y se observa cómo el robot no detecta ninguna persona. A continuación, una persona se sitúa delante de la cámara, pero alejada del robot, y se observa que el robot detecta a la persona a una distancia segura. Este resultado equivale al siguiente resultado visto por la cámara del robot y analizado por la inteligencia artificial en el servidor:

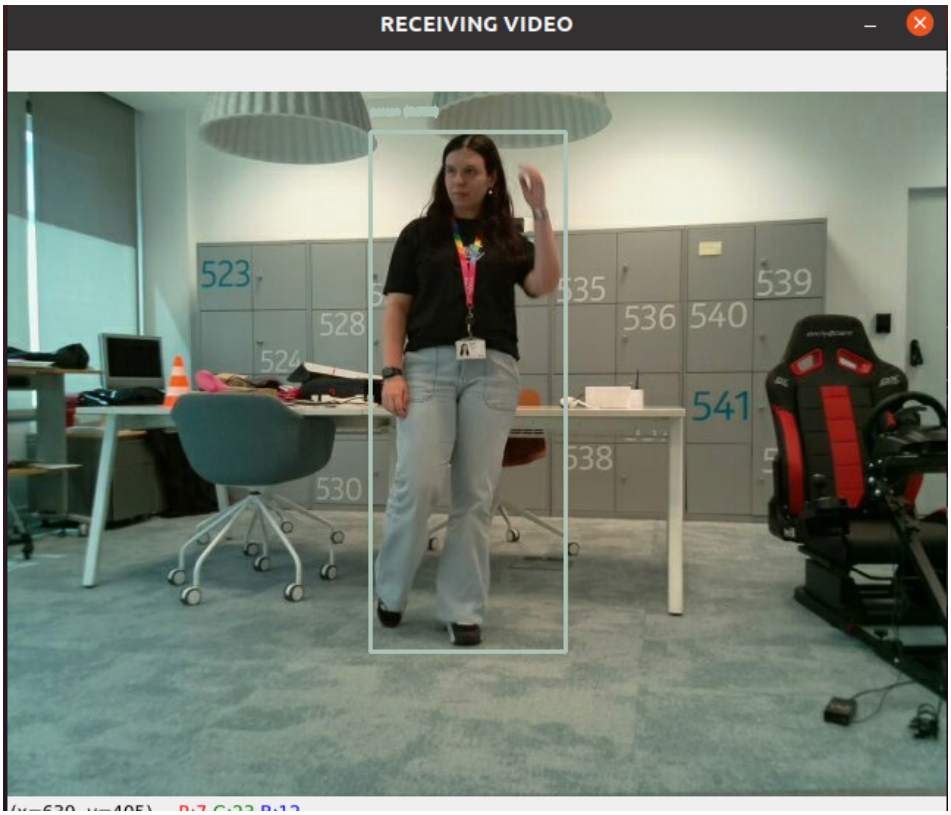

Figura 81 Detección de persona en tiempo real

En la terminal del ordenador servidor se observan los siguientes resultados que nos confirman que se observa una persona.

En el *detect_alga_forward.py*:

```
Image received
Image received
Class Index: 0.0, Class Name: person, Color: [189, 252, 232]
Box: [378, 20, 638, 474], Color: [189, 252, 232], Text: person (0.872)
Image received
Class Index: 0.0, Class Name: person, Color: [189, 252, 232]
Box: [322, 16, 637, 472], Color: [189, 252, 232], Text: person (0.861)
```

Figura 82 Resultados de la detección de personas en tiempo real

Se observa a partir de la Figura 82 que el índice de la clase es 0.0 lo que corresponde con la clase que detecta personas. También se indica una puntuación de confianza alta de que el objeto detectado es una persona.

En el *alga_client_forward.py*:

```
Received metadata: [{'inferences': 1, 'instances': 0, 'number_classes': 0, 'classes'
: [], 'boxes': [], 'pre_process_ms': 0.6930828094482422, 'inference_ms': 21.37374877
9296875, 'nms_ms': 1.829385757446289, 'post_process_ms': 0.6129741668701172}]
Received metadata: [{'inferences': 1, 'instances': 0, 'number_classes': 0, 'classes'
: [], 'boxes': [], 'pre_process_ms': 0.7243156433105469, 'inference_ms': 19.57464218
1396484, 'nms_ms': 2.0945072174072266, 'post_process_ms': 0.7467269897460938}]
Received metadata: [{'inferences': 1, 'instances': 0, 'number_classes': 0, 'classes'
: [], 'boxes': [], 'pre_process_ms': 0.6821155548095703, 'inference_ms': 19.94919776
916504, 'nms_ms': 0.91552734375, 'post_process_ms': 0.6270408630371094}]
Received metadata: [{'inferences': 1, 'instances': 1, 'number_classes': 1, 'classes'
: ['person'], 'boxes': [[568, 212, 640, 461]], 'pre_process_ms': 0.6406307220458984,
 'inference_ms': 20.362377166748047, 'nms_ms': 3.847360610961914, 'post_process_ms':
 2.2907257080078125}]
Received metadata: [{'inferences': 1, 'instances': 1, 'number_classes': 1, 'classes'
: ['person'], 'boxes': [[556, 247, 640, 479]], 'pre_process_ms': 0.6401538848876953,
 'inference_ms': 21.24643325805664, 'nms_ms': 2.5403499603271484, 'post_process_ms':
 1.7027854919433594}]
Received metadata: [{'inferences': 1, 'instances': 1, 'number_classes': 1, 'classes'
: ['person'], 'boxes': [[499, 219, 630, 432]], 'pre_process_ms': 0.6380081176757812,
 'inference_ms': 22.4456787109375, 'nms_ms': 2.2351741790771484, 'post_process_ms':
1.7163753509521484}]
```

Figura 83 Análisis de los metadatos en la detección de personas

En la Figura 83 se muestra, que cuando no hay una persona en la imagen recibida, las variables: *'instances', 'number_classes', 'classes'* y *'boxes'* se encuentran vacías. La razón

por la que no detecta más objetos es por la condición establecida en la Figura 80, en la que se limita el sistema a detectar únicamente personas.

Estos resultados del *detect_alga_forward.py* y del *alga_client_forward.py* coinciden con la primera parte del código UDP. En este caso, lo único que varía son los mensajes por terminal, en cuanto a la detección de personas, se detectan igual que el primer caso.

Los resultados del código realmente varían cuando una persona se acerca a una distancia peligrosa al robot. En consecuencia, cambian los mensajes que se muestran en la terminal y los mensajes de tipo *'String'* enviados a través del tópico *'/stop_robot':*

```
Person detected, still safe
Person detected, still safe
Person detected, still safe
Person detected is too close, stopping robot.
Person detected is too close, stopping robot.
Person detected is too close, stopping robot.
Person detected is too close, stopping robot.
Person detected is too close, stopping robot.
```

Figura 84 Registro de detección de personas a una distancia muy cercana al robot

Este resultado en la terminal indica que la persona se ha acercado demasiado al robot, ocupando un 60% o más de su visión. En la imagen recibida de la cámara del robot debería de aparece un caso similar al que se adjunta en la Figura a continuación:

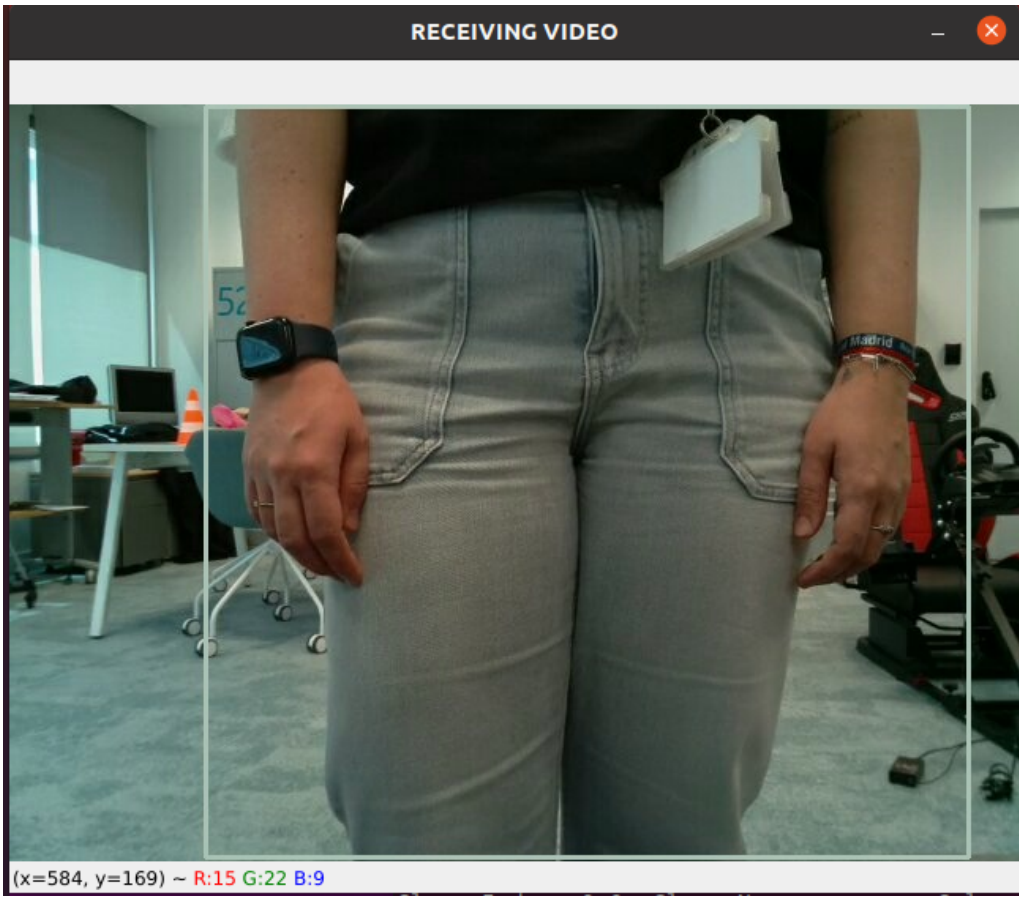

Figura 85 Detección de proximidad de una persona

En el momento en el que la persona de aleja del robot, se vuelve a la situación anterior en la que se detecta a la persona, pero a una distancia adecuada.

```
Person detected is too close, stopping robot.
Person detected is too close, stopping robot.
Person detected, still safe
Person detected, still safe
```

Figura 86 Detección de persona alejándose del robot en tiempo real

Para demostrar que el robot se frena en caso de que la persona se acerque a una distancia considerada peligrosa, utilizamos el comando '*rostopic echo /cmd_vel',* comando que se encarga de imprimir lo que se publica en los tópicos; en este caso, los valores de la velocidad publicados en *'/cmd_vel'*. Este comando es una herramienta que proporciona ROS para mostrar los datos en tiempo real que se van publicando en el tópico '*/cmd_vel'*. En la terminal se mostrarán los comandos de velocidad lineal y angular del robot.

Para verificar que los resultados coinciden con el código, se muestra en la Figura 87 el cambio cuando una persona se acerca más de un 60% del marco de la imagen.

```
('Received stop message:', 'Person detected, still safe')
('La velocidad actual es: ', 1.0)
('F.AREA:', 180306, ' - AREA:', 307200.0)
('OBJ. AREA:', 0.58693359375)
Person detected, still safe
('Received stop message:', 'Person detected, still safe')
('F.AREA:', 184632, ' - AREA:', 307200.0)
('OBJ. AREA:', 0.601015625)
Person detected is too close, stopping robot.
('Received stop message:', 'Person detected is too close, stopping robot')
```

Figura 87 Demostración de la respuesta de seguridad del robot

El resultado de la salida en la terminal al ejecutar '*rostopic echo /cmd_vel'* es la siguiente:

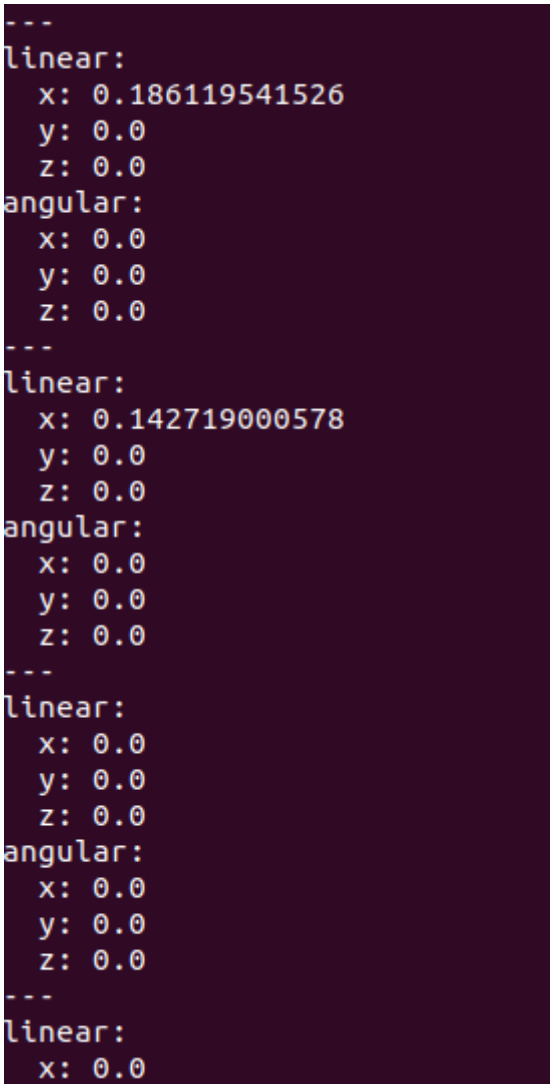

Figura 88 Demostración de los valores de velocidad del robot

En este caso el robot se está moviendo de manera lineal hacia adelante (al ser el valor de x positivo) a un ritmo de velocidad bastante lenta. En el momento en el que el área de la persona pasa a ser mayor que 0.6, como se puede ver en la Figura 88, se observa que el robot se detiene, ya que se demuestra que la velocidad lineal y angular se mantiene a 0.0.

Finalmente, comprobamos que el robot percibe los objetos que le rodean por medio de la inteligenica artificial, calcula correctamente las áreas de los frames y se detiene a una distancia que se considera como peligrosa mediante el código ROS.

5.2.3 RESULTADOS EN RQT_GRAPH.

Las herramientas rqt son un conjunto de plugins de interfaz gráfica de usuario (GUI) que ofrecen diversas funcionalidades en el desarrolllo y análisis de ROS.[67] Estas herramientas se basan en el marco Qt, que contiene un conjunto de bibliotecas y herramientas para gestionar aplicaciones con GUI y software. Qt se emplea para crear interfaces de usuario intuitivas que se usan cada vez más en dispositivos móviles.[68] En este proyecto, permite a los usuarios interactuar con robots y visualizar datos de manera intuitiva.

Rqt_graph es una herramienta de rqt que permite visualizar los nodos y tópicos del sistema ROS. Muestra un esquema con las conexiones de los nodos a través de los tópicos en los que se publican y se subscriben, y los tipos de mensajes que se intercambian. Por medio de esta herramienta, se obtiene una visión general de la arquitectura y de la comunicación del proyecto

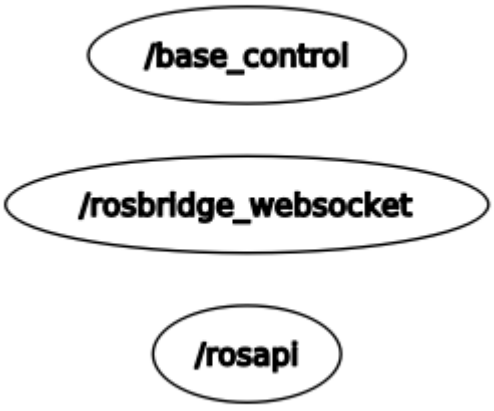

Figura 89 Nodos iniciales al arrancar el robot

En este primer caso, se lanza el *rqt_graph* sin ejecutar ningún código correspondiente al robot.

En primer lugar, el nodo *'/base_control'* se encarga de gestionar el movimiento del robot. Controla los motores por medio de los comandos recibidos por el tópico *'/cmd_vel'*. En segundo lugar, el nodo *'/rosbridge_websocket'* se levanta al iniciar un nodo de ROS, junto con el nodo *'/rosapi'*. El nodo *'/rosbridge_websocket'* gestiona la comunicación basada en la transmisión de mensajes entre el servidor y los clientes por medio de WebSockets. Por último, el nodo *'/rosapi'* maneja las funciones básicas del núcleo de ROS por medio de una API que interactúa con el sistema ROS. De esta forma, se facilita la comunicación, la integración de servicios y las aplicaciones externas.

Si ejecutamos el código principal del robot nos sale el siguiente gráfico:

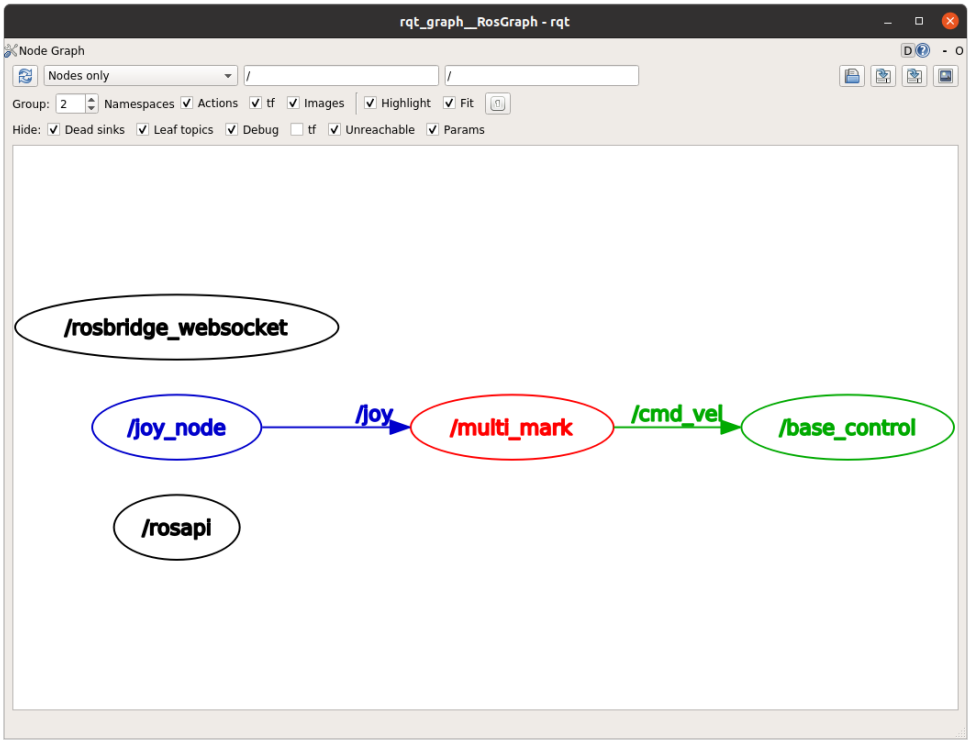

Figura 90 Relación entre nodos al ejecutar el código de movimiento

En la Figura 90 se ejecuta el código principal del robot *move_robot.py*. Al principio del código se establecen los tópicos que aparecen en el esquema de la Figura 90.

```
rospy.init_node('ps4_controller_teleop')
self.velocity_pub = rospy.Publisher('/cmd_vel', Twist, queue_size=10)
self.joy_sub = rospy.Subscriber('/joy', Joy, self.joy_callback)
self.rate = rospy.Rate(10)  # frecuencia ciclo principal a 10 Hz
```

Figura 91 Tópicos establecidos en el código move_robot.py

Se empieza con el nodo *'/joy_node'* que procesa las entradas del *joystick* y publica los mensajes en el tópico *'/joy'*. Posteriormente, el nodo *'/multi_mark'* subscribe los datos publicados anteriormente en el tópico *'/joy'*.

A continuación se crea un *'Publisher'* que publica mensajes de tipo *'Twist'* en el tópico *'/cmd_vel'*. El nodo *'/multi_mark'* publica los comandos de velocidad en el tópico *'/cmd_vel'* y los lleva al nodo *'/base_control'* para poder gestionar el movimiento del robot.

Si ejecutamos los dos códigos:

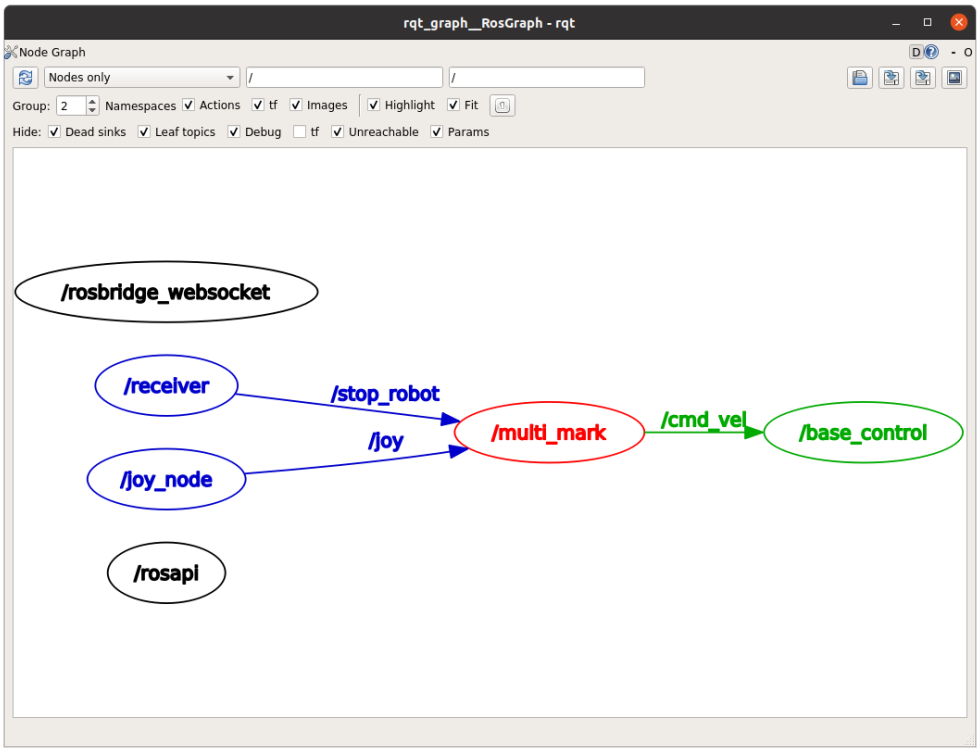

Figura 92 Relación entre nodos al ejecutar todos los códigos

En este caso se ejecutan los códigos *move_robot.py* y *receive_message2.py* lo que modifica el esquema. Se tiene un nuevo nodo correspondiente al código de *receive_message2.py* en el que se publican mensajes de tipo *'String'* en el tópico '/stop_robot'.

El nodo *'multi_mark'* subscribe los datos del joystick por medio del tópico *'/joy'* y por otro lado los datos que se reciben de la cámara del robot desde el tópico *'/stop_robot'*.

En el caso de que solo se ejecute cualquier de los códigos correspondientes al nodo *'/receiver'*, en concreto, la parte de inteligencia artificial del proyecto:

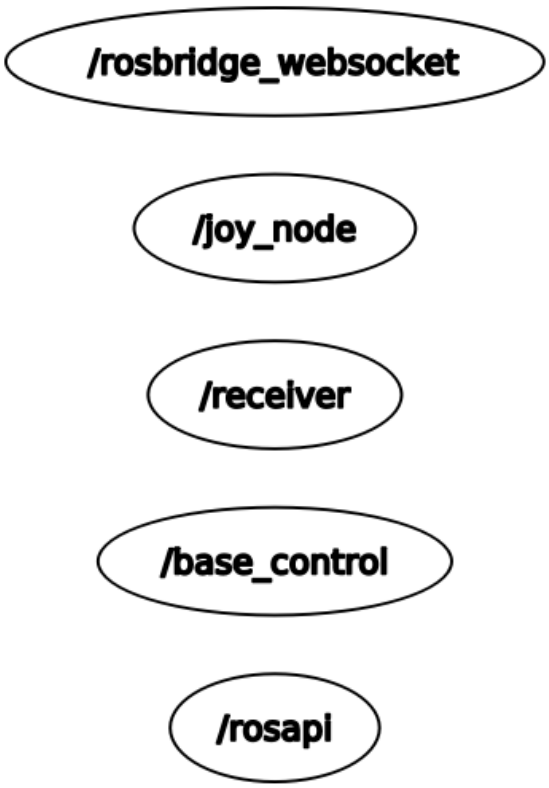

Figura 93 Esquema de nodos ejecutando la parte de IA

Como se puede observar, no hay ninguna interacción entre nodos, a diferencia de la Figura 92. Podemos concluir, que las funcionalidades relacionadas con la inteligencia artificial se operan y proporcionan resultados de manera independiente al resto de los códigos.

Capítulo 6. CONCLUSIONES Y TRABAJOS FUTUROS.

6.1.1 CONCLUSIONES

A lo largo del proyecto se han cumplido todos los objetivos propuestos.

1. Adquisición y aprendizaje de los conceptos relevantes y necesarios para poder manejar los principios fundamentales de la robótica con ROS. Por medio de la ROS Wiki se facilitó el proceso de aprendizaje desde un principio.
2. Estudio y familiarización con los conceptos de inteligencia artificial como las redes neuronales, que son aquellas utilizadas por los algoritmos de código abierto de la IA para la detección de objetos en tiempo real. Este es el caso de YOLOv5, el algoritmo utilizado en este proyecto.
3. Implementación y configuración del comportamiento del robot en el simulador Gazebo. De esta forma, se facilitó posteriormente el uso del código transferido al robot físico. Se redujeron los riesgos y errores que se podrían haber cometido a la hora de trabajar con el robot físico. Además, se pudo comprobar con antelación un código robusto, que permitía todas las funcionalidades del movimiento del robot en la simulación.
4. Manejo del robot con un mando, conectado por medio de bluetooth, y ROS, tanto en la simulación como el robot físico.
5. Transferencia de los códigos al hardware del robot. Tras haber validado el funcionamiento de los códigos tanto de la inteligencia artificial como de ROS en el PC servidor, se transladan al robot físico y se lanzan las pruebas en tiempo real para verificar la estabilidad y el correcto funcionamiento del modelo.
6. Implementación de la comunicación entre el robot y servidor por medio del protocolo de comunicación sin conexión UDP y los tópicos de ROS. De esta forma se pudo conectar la parte de inteligencia artificial que recibe las imágenes que provienen de

la cámara del robot lanzada en el PC servidor y la parte del movimiento del robot por medio de ROS.

7. Establecer conexión WiFi por medio de un router independiente de 5G, para mejorar el resultado del flujo de datos que llegan al PC servidor y viceversa.

De los objetivos mencionados anteriormente, se destacan las siguientes conclusiones:

- Haber realizado la simulación de Gazebo antes de trabajar con el robot físico fue de gran ayuda ya que permitió que el proceso fuese más cómodo y sencillo a la hora de trasladar el código al hardware del robot. A su vez, redujo la aparición de futuroa errores así como de riesgos que podrían haber afectado al rendimiento del robot.
- El uso de la IA, y en concreto, del algoritmo YOLOv5, para la detección de objetos en tiempo real, demuestra una gran robustez para sistemas que precisan de la detección de objetos con una baja latencia. Esto representa un gran avance para el desarrollo y la confianza de la conducción autónoma en un futuro.
- El uso de ROS no solo contribuyó a la programación de la navegación del robot, sino también facilitó el proceso de comunicación entre el robot y el PC servidor. Gracias a la comunicación sin conexión UDP y al uso de los tópicos de ROS, se pudo conseguir de manera más sencilla los resultados deseados.
- La combinación del algoritmo YOLOv5 y el sistema operativo ROS ha logrado alcanzar el objetivo principal del proyecto: crear un sistema de conducción autónoma eficiente y robusto.

6.1.2 TRABAJOS FUTUROS

A pesar de haber cumplido los objetivos principales del proyecto, este se podría haber extendido en dos situaciones específicas. Estas dos situaciones reforzarían la meta del proyecto, basada en conseguir una conducción autónoma de nuestro sistema. Estas metas se basan en:

- Uso del LiDAR: Para reforzar la detección de objetos, en este caso de personas, que se acercan al robot. De esta forma por medio del uso de un láser se podría determinar la posición y distancia de objetos.[69] Esta funcionalidad se añadiría como algo adicional a la detección de objetos por medio de áreas, realizado en este proyecto. De esta forma se refuerza el valor de confianza de cercanía de los objetos al robot, para que pueda frenar en caso de distancias peligrosas.
- Realidad virtual (RV): En el Capítulo 2, se mencionaba brevemente la realidad virtual ya que la meta de este proyecto es compaginarlo con el uso del metaverso. La realidad virtual y el uso del Oculus Rift se está integrando en una gran cantidad de proyectos de Nokia, en este incluido.
- Integrando la RV junto con la robótica permitiría a el usuario controlar el robot en lugares remotos, lugares a los que el usuario no puede acceder directamente.[70] Por ejemplo, Si queremos reforzar la seguridad a la hora de detectar personas, se podría simular el robot como si fuese un coche en una carretera. La conexión a la RV se haría por medio de USBC a Unity, una plataforma que permite a los desarrolladores de VR crear aplicaciones sin necesidad de otras herramientas externas adicionales.

El diagrama del proyecto final sería de la siguiente forma:

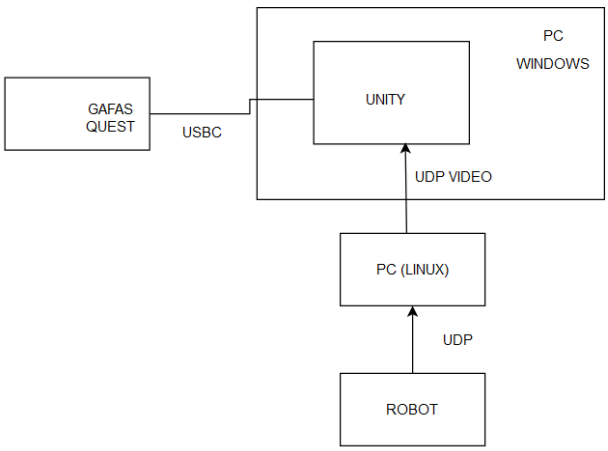

Figura 94 Diagrama del proyecto en un futuro

Capítulo 7. BIBLIOGRAFÍA

[1] J. E. Riva, «Introducción a ROS,» ROS wiki, 18 08 2021. [En línea]. Available: http://wiki.ros.org/es/ROS/Introduccion. [Último acceso: 21 10 2023].

[2] J. Report, «Historia de la robótica,» La Vanguardia, 10 06 2022. [En línea]. Available: https://www.lavanguardia.com/vida/junior-report/20220610/8330874/historia-robotica.html. [Último acceso: 22 03 2024].

[3] A. Manchali, «Did Leonardo da Vinci build a Robot?,» Medium, 4 08 2020. [En línea]. Available: https://medium.com/@aashuthoshm/did-leonardo-da-vinci-build-a-robot-29ecd899fcf1. [Último acceso: 22 03 2024].

[4] P. Gómez, «Origen e historia de la robótica,» Hipernexo, 2020. [En línea]. Available: https://www.hipernexo.com/robotica/historia-de-la-robotica/. [Último acceso: 22 03 2024].

[5] E. Medina, «El Telar de Jacquard, máquina precursora de los ordenadores modernos, es subastado por 43.750 dólares,» Muy Computer, 18 12 2019. [En línea]. Available: https://www.muycomputer.com/2019/12/18/telar-de-jacquard-maquina-subastado-43750-dolares/. [Último acceso: 22 03 2024].

[6] Gamco, «Prueba de Turing: Concepto y definición,» Gamco, 2021. [En línea]. Available: https://gamco.es/glosario/prueba-de-turing/. [Último acceso: 22 03 2024].

[7] E. d. Comunicación, «Las leyes de la robótica: más allá de Isaac Asimov,» Telefónica, 28 12 2023. [En línea]. Available: https://www.telefonica.com/es/sala-comunicacion/blog/leyes-robotica/. [Último acceso: 22 03 2024].

[8] Wikipedia, «George Devol,» Wikipedia, 25 12 2023. [En línea]. Available: https://es.wikipedia.org/wiki/George_Devol#:~:text=Su%20Primer%20Robot%20Industrial%3A%20Unimate,-

Sin%20embargo%2C%20fue&text=Fue%20en%201954%20cuando%20Devol,fue%20el%20primer%20robot%20programable. [Último acceso: 22 03 2024].

[9] EcuRed, «George Devol,» EcuRed, 2021. [En línea]. Available: https://www.ecured.cu/George_Devol. [Último acceso: 22 03 2024].

[10] R. D. López, «El futuro de la conducción autónoma,» Knowmadmood, 21 01 2020. [En línea]. Available: https://www.knowmadmood.com/blog/el-futuro-de-la-conduccin-autnoma. [Último acceso: 08 06 2024].

[11] N. Pastor, «La Inteligencia Artificial llega a la gran industria: así serán las fábricas del futuro,» La Vanguardia, 28 03 2019. [En línea]. Available: https://www.lavanguardia.com/tecnologia/20190328/461296182151/inteligencia-artificial-industria-fabricas-futuro-brl.html. [Último acceso: 23 03 2024].

[12] Iberdrola, «Historia de la Inteligencia Artificial,» Iberdrola, [En línea]. Available: https://www.iberdrola.com/innovacion/historia-inteligencia-artificial. [Último acceso: 23 03 2024].

[13] E. Sandu, «Historia de la Realidad Virtual,» Metaverso Pro, [En línea]. Available: https://metaverso.pro/blog/historia-de-la-realidad-virtual/. [Último acceso: 23 03 2024].

[14] Oculus VR, LLC, "Advertencias de salud y seguridad para Oculus Rift y Touch," documento 310-30132-01, Menlo Park, CA, USA, 2020. [En línea]. Available: https://www.oculus.com/warnings. [Último acceso: 30 03 2024].

[15] Nokia, «The metaverse,» Nokia, 2024. [En línea]. Available: https://www.nokia.com/metaverse/. [Último acceso: 23 03 2024].

[16] Wikipedia, «Algoritmo You Only Look Once (YOLO),» Wikipedia, 1 05 2024. [En línea]. Available: https://es.wikipedia.org/wiki/Algoritmo_You_Only_Look_Once_(YOLO). [Último acceso: 20 03 2024].

[17] Z. Keita, «Explicación de la detección de objetos YOLO,» Datacamp, 01 2024. [En línea]. Available: https://www.datacamp.com/es/blog/yolo-object-detection-explained?utm_source=google&utm_medium=paid_search&utm_campaignid=20616617505&utm_adgroupid=154290359997&utm_device=c&utm_keyword=&utm_matchtype=&utm_network=g&utm_adpostion=&utm_creative=691823521640. [Último acceso: 24 03 2024].

[18] M. Menegaz, «Entendiendo YOLO,» Hackernoon, 16 03 2018. [En línea]. Available: https://hackernoon.com/es/comprension-yolo-f5a74bbc7967. [Último acceso: 24 03 2024].

[19] G. Jocher y P. Derrenger, «ultralytics/yolov5: YOLOv5 in PyTorch,» GitHub, 2020. [En línea]. Available: https://github.com/ultralytics/yolov5?tab=readme-ov-file. [Último acceso: 09 2023].

[20] TEAM CSIRT, «Vulnerabilidad en librería GitPython permite acceso a usuarios no identificados,» Telconet, 30 08 2023. [En línea]. Available: https://csirt.telconet.net/comunicacion/noticias-seguridad/vulnerabilidad-en-libreria-gitpython-permite-acceso-a-usuarios-no-identificados/#:~:text=GitPython%20es%20una%20librería%20de,y%20luego%20el%20entorno%20PATH. [Último acceso: 31 05 2024].

[21] Wikipedia, «Matplotlib,» Wikipedia, 13 02 2024. [En línea]. Available: https://es.wikipedia.org/wiki/Matplotlib. [Último acceso: 31 05 2024].

[22] Wikipedia, «NumPy,» Wikipedia, 17 03 2024. [En línea]. Available: https://es.wikipedia.org/wiki/NumPy. [Último acceso: 31 05 2024].

[23] Geeksforgeeks, «Tutorial de OpenCV en Python,» geeksforgeeks, 14 03 2024. [En línea]. Available: https://www.geeksforgeeks.org/opencv-python-tutorial/. [Último acceso: 31 05 2024].

[24] Wikipedia, «Pillow (biblióteca de código abierto),» Wikipedia, 16 11 2022. [En línea]. Available: https://es.wikipedia.org/wiki/Pillow_(biblioteca_de_código_abierto). [Último acceso: 31 05 2024].

[25] B. Joex y G. Rodola, «psutil 5.9.8,» PyPI, 19 01 2024. [En línea]. Available: https://pypi.org/project/psutil/. [Último acceso: 31 05 2024].

[26] T. Khan y M. Goodwin, «¿Qué es YAML?,» IBM, 11 12 2023. [En línea]. Available: https://www.ibm.com/es-es/topics/yaml. [Último acceso: 31 05 2024].

[27] Wikipedia, «SciPy,» Wikipedia, 24 01 2024. [En línea]. Available: https://es.wikipedia.org/wiki/SciPy. [Último acceso: 31 05 2024].

[28] R. Shah, «Overview of TQDM: Python Progress Bar Library (Updated 2024),» Analytics Vidhya, 24 04 2024. [En línea]. Available: https://www.analyticsvidhya.com/blog/2021/05/how-to-use-progress-bars-in-python/. [Último acceso: 31 05 2024].

[29] Ultralytics, «Revolucionando el mundo de la IA de visión,» Ultralytics, 2024. [En línea]. Available: https://www.ultralytics.com/es#:~:text=Ultralytics%20HUB%20es%20una%20plataforma,un%20marco%20de%20aprendizaje%20profundo.. [Último acceso: 31 05 2024].

[30] IBM, «¿Qué es PyTorch?,» IBM, [En línea]. Available: https://www.ibm.com/es-es/topics/pytorch. [Último acceso: 31 05 2024].

[31] By Automation Review, «ROS: una palanca estratégica para el futuro de la robótica,» AER Automation, 3 03 2022. [En línea]. Available: https://www.aer-automation.com/automation_review/ros-una-palanca-estrategica-para-el-futuro-de-la-robotica/. [Último acceso: 25 03 2024].

[32] J. E. Riva, «Comenzando con ROS: Introducción,» ROS Wiki, 18 08 2021. [En línea]. Available: http://wiki.ros.org/es/ROS/Introduccion. [Último acceso: 10 2023].

[33] V. Mazzari, «ROS - Robot Operating System,» Génération Robots, 26 03 2016. [En línea]. Available: https://www.generationrobots.com/blog/en/ros-robot-operating-system-2/. [Último acceso: 25 03 2024].

[34] C. Arteaga-Escamilla, «ROS tutorials with C++ and Python for beginners,» GitHub, 2021. [En línea]. Available: https://github.com/cmauricioae8/ros_cpp_py_basics. [Último acceso: 10 2023].

[35] PNGWing, «Exclusive png design images,» PNGWing, [En línea]. Available: https://www.pngwing.com/en/free-png-xprdn. [Último acceso: 24 05 2024].

[36]DinosoftLabs, «Folder free icon,» Flaticon, [En línea]. Available: https://www.flaticon.com/free-icon/folder_3516096. [Último acceso: 24 05 2024].

[37] M. Pecka, «joy,» ROS Wiki, 19 08 2022. [En línea]. Available: https://wiki.ros.org/joy. [Último acceso: 10 2023].

[38] Cloudflare, «¿Qué es el UDP?,» Cloudflare, [En línea]. Available: https://www.cloudflare.com/es-es/learning/ddos/glossary/user-datagram-protocol-udp/. [Último acceso: 25 03 2024].

[39] J. Virga, «UDP Communication,» Python.org, 19 05 2020. [En línea]. Available: https://wiki.python.org/moin/UdpCommunication. [Último acceso: 13 03 2024].

[40] Entel Comunidad Empresas, «¿Qué son los metadatos y para qué sirven?,» Entel Comunidad Empresas, [En línea]. Available: https://ce.entel.cl/articulos/que-son-los-metadatos-y-para-que-sirven/#:~:text=En%20palabras%20más%20simples%2C%20los,el%20contenido%20del%20propio%20archivo. [Último acceso: 10 06 2024].

[41] L. Calvo, «¿Qué es una Raspberry PI y para qué sirve?,» GoDaddy, 10 03 2022. [En línea]. Available: https://www.godaddy.com/resources/es/tecnologia/que-es-raspberry-pi. [Último acceso: 19 03 2024].

[42] «¿Qué es LiDAR?,» IBM, [En línea]. Available: https://www.ibm.com/es-es/topics/lidar. [Último acceso: 19 03 2024].

[43] «AX3000 Wi-Fi 6 5G CPE Mesh Router,» Aryan, [En línea]. Available: https://www.aryan.es/1repetidores-wifi-ac/10889-cudy-p5-router-inalambrico-gigabit-ethernet-doble-banda-24-ghz-5-ghz-5g-negro-6971690792282.html. [Último acceso: 25 05 2024].

[44] J. James, «How to Install Python 3.8 on Ubuntu 22.04 or 20.04,» LinuxCapable, 1 11 2023. [En línea]. Available: https://linuxcapable.com/install-python-3-8-on-ubuntu-linux/. [Último acceso: 19 03 2024].

[45] G. Godoy, «Cómo NVIDIA se ha convertido en la líder de la inteligencia artificial con sus GPU,» Cointelegraph, 03 06 2023. [En línea]. Available: https://es.cointelegraph.com/news/how-nvidia-has-become-the-leader-in-artificial-intelligence-with-its-gpus. [Último acceso: 5 06 2024].

[46] «NVIDIA Driver Downloads,» NVIDIA, [En línea]. Available: https://www.nvidia.com/download/index.aspx. [Último acceso: 19 02 2024].

[47] Amir, «How to Install GPU Driver on Ubuntu,» Amir Pourmand, 02 03 2024. [En línea]. Available: https://amirpourmand.ir/posts/2024/install-gpu-driver-ubuntu/. [Último acceso: 21 01 2024].

[48] N. O. d. Frutos, «Perkins Eyes,» Nokia, 2023. [En línea]. Available: http://nokia.sharepoint.com/sites/blspain/blswiki/PerkinsEyes.aspx. [Último acceso: 12 04 2024].

[49] « GSTREAMER,» Genie Doc, [En línea]. Available: https://geniedoc.blogspot.com/p/gstreamer.html. [Último acceso: 09 06 2024].

[50] D. Landup, «Object Detection Inference in Python with YOLOv5 and PyTorch,» Stack Abuse, 16 11 2023. [En línea]. Available: https://stackabuse.com/object-detection-inference-in-python-with-yolov5-and-pytorch/. [Último acceso: 03 06 2024].

[51] G. TONYE, «Machine Learning Confidence Scores — All You Need to Know as a Conversation Designer,» Medium, 25 08 2021. [En línea]. Available: https://medium.com/voice-tech-global/machine-learning-confidence-scores-all-you-need-to-know-as-a-conversation-designer-8babd39caae7. [Último acceso: 09 06 2024].

[52] «ROBOT OPERATING SYSTEM (ROS),» Tknika, 28 10 2022. [En línea]. Available: https://tknika.eus/cont/robot-operating-system-ros-3/. [Último acceso: 02 04 2024].

[53] L. Joseph y J. Cacace, «Mastering ROS for Robotics Programming, Third edition - Third Edition,» Packt Publishing, 2021. [En línea]. Available: https://www.packtpub.com/product/mastering-ros-for-robotics-programming-third-edition-third-edition/9781801071024. [Último acceso: 10 06 2024].

[54] I. Saito, «Ubuntu install of ROS Noetic,» ROS Wiki, 04 03 2023. [En línea]. Available: https://wiki.ros.org/noetic/Installation/Ubuntu. [Último acceso: 10 2023].

[55] Equipo de Contenidos de GoDaddy, «¿Qué es un comando Curl y cómo usarlo?,» GoDADDY, 30 08 2023. [En línea]. Available: https://www.godaddy.com/resources/latam/stories/que-es-comando-curl-como-usarlo. [Último acceso: 30 05 2024].

[56] A. Casero, «apt-get update en Linux para actualizar repositorios,» KeepCoding Tech School, 15 03 2024. [En línea]. Available: https://keepcoding.io/blog/apt-get-update-en-linux/. [Último acceso: 05 04 2023].

[57] Open Robotics, «Managing Dependencies with rosdep,» ROS 2 Documentation: Humble, 10 04 2022. [En línea]. Available: https://docs.ros.org/en/humble/Tutorials/Intermediate/Rosdep.html. [Último acceso: 2024].

[58] W. Garage, «rosinstall: source-based install tool,» 2011. [En línea]. Available: https://docs.ros.org/en/independent/api/rosinstall/html/#:~:text=That%20way%20you%20can%20more,rosinstall%20files. [Último acceso: 05 04 2024].

[59] D. Thomas, «wstool,» 11 06 2020. [En línea]. Available: https://wiki.ros.org/wstool#:~:text=wstool%20provides%20commands%20to%20manage,rosws%20tool%20for%20catkin%20workspaces. [Último acceso: 05 04 2024].

[60] 162, «Compiling Software,» Community Help Wiki, Ubuntu, 19 08 2014. [En línea]. Available: https://help.ubuntu.com/community/CompilingSoftware. [Último acceso: 05 04 2024].

[61] C. Lalancette, «Installing and Configuring Your ROS Environment,» ROS Wiki, 30 01 2024. [En línea]. Available: http://wiki.ros.org/ROS/Tutorials/InstallingandConfiguringROSEnvironment. [Último acceso: 05 04 2024].

[62] J. E. Riva, «Construyendo un Paquete ROS,» ROS Wiki, 13 08 2021. [En línea]. Available: https://wiki.ros.org/es/ROS/Tutoriales/ConstruyendoPaquetes#:~:text=La%20carpeta%20build%20es%20la,antes%20de%20instalar%20sus%20paquetes. [Último acceso: 10 2023].

[63] P. Alcantara, «Building a ROS package,» ROS Wiki, 18 04 2020. [En línea]. Available: http://wiki.ros.org/ROS/Tutorials/BuildingPackages. [Último acceso: 10 2023].

[64] Marguedas, «catkin_make,» ROS Wiki, 22 05 2017. [En línea]. Available: http://wiki.ros.org/catkin/commands/catkin_make. [Último acceso: 10 2023].

[65] «ROS Components,» AIRLab, [En línea]. Available: https://airlab.deib.polimi.it/life-in-airlab/service-pages/resources/ros-howto/ros-components/#:~:text=odom%20is%20the%20frame%20where,to%20its%20own%20odometry%20system. [Último acceso: 05 04 2024].

[66] «Twist Message,» 02 03 2022. [En línea]. Available: http://docs.ros.org/en/noetic/api/geometry_msgs/html/msg/Twist.html. [Último acceso: 10 04 2024].

[67] «¿Cómo se analizan los mensajes ROS con las herramientas rqt?,» [En línea]. Available: https://www.linkedin.com/advice/1/how-do-you-analyze-ros-messages-rqt-tools-skills-ros?lang=es&originalSubdomain=es. [Último acceso: 22 05 2024].

[68] «QtCreatorWhitepaper Spanish,» 17 04 2015. [En línea]. Available: https://wiki.qt.io/QtCreatorWhitepaper_Spanish. [Último acceso: 22 05 2024].

[69] W. B. Services, «Desde la detección de satélites hasta yacimientos arqueológicos: cómo funciona la tecnología lidar y por qué es revolucionaria en la seguridad del coche,» 21 03 2023. [En línea]. Available: https://innovacionvolvo.xataka.com/deteccion-satelites-yacimientos-arqueologicos-como-funciona-tecnologia-lidar-que-revolucionaria-seguridad-coche/. [Último acceso: 31 05 2024].

[70] E. Sandu, «Realidad aumentada y virtual en robótica,» Metaverso Pro, [En línea]. Available: https://metaverso.pro/blog/realidad-aumentada-y-virtual-en-robotica/. [Último acceso: 31 05 2024].

[71] N. XR-Lab, «Carpeta spider» Nokia, 2020. [En línea]. Available: https://gitlabe2.ext.net.nokia.com/drs/spider/-/tree/dockerize. [Último acceso: 17 01 2023].

[72] U. N. D. Programme, «Sustainable Development Goals,» [En línea]. Available: https://www.undp.org/es/sustainable-development-goals. [Último acceso: 12 06 2024].

[73] Noticias ONU, 2015 09 2015. [En línea]. Available: https://www.un.org/sustainabledevelopment/es/2015/09/la-asamblea-general-adopta-la-agenda-2030-para-el-desarrollo-sostenible/. [Último acceso: 12 06 2024].

ANEXO I: ALINEACIÓN DEL PROYECTO CON LOS ODS

Los objetivos de Desarrollo Sostenible (ODS) son una lista de 17 objetivos globales establecidos por las Naciones Unidas en 2015 con el objetivo de erradicar la pobreza y desigualdad, proteger el medio ambiente y garantizar que la población se beneficie de una situación de paz y prosperidad. [72]

Figura 95 Objetivos de Desarrollo Sostenible [73]

Este proyecto se alinea con el Objetivo 9: Industria, Innovación e Infraestructura, al promover la innovación tecnológica en la conducción autónoma y asistida. Además, incluye la integración de inteligencia artificial como una innovación significativa, ya que, en este caso, actúa como una herramienta de análisis muy útil. Esta exploración va a permitir una

gran eficiencia en varios campos, entre ellos el de la conducción, aportando un avance considerable para mejor la movilidad.

Asimismo, se alinea con el Objetivo 11: Ciudades y Comunidades Sostenibles. Este proyecto busca promover la conducción autónoma, avanzando tecnológicamente en una mayor seguridad y eficiencia en la conducción. Por ello, uno de los objetivos en esta área, es crear soluciones de transporte más seguras y eficientes, fomentando el desarrollo de comunidades sostenibles al buscar una mejora en la movilidad y en la calidad de vida del ser humano.

Finalmente, se busca que en un futuro este proyecto pueda ayudar a mejorar el acceso a los servicios de salud, alineándose con el Objetivo 3: Salud y Bienestar. De esta forma, se contribuirá a las comunidades que no tienen facilidad de acceso a instalaciones médicas.