

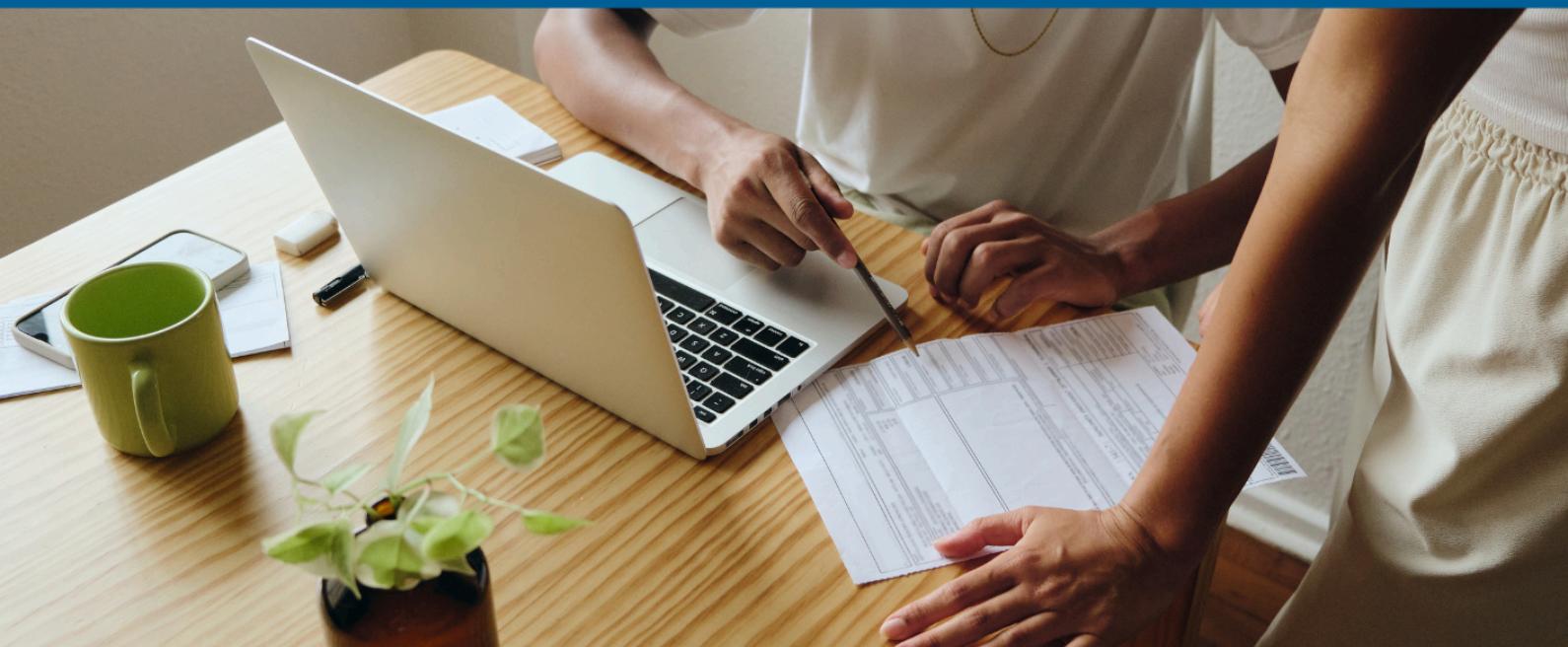
ROBOENTREGA

Diego Baldovi, Jaime Ferrer,
Endika Matute, Jordan David
Phillips Fonta, Hao Xu,
Alejandro Roca

Memoria de proyecto

Proyecto Robótica

equipo 1



Índice

1. Concepción de la idea.....	5
1.2. Descripción de la idea principal.....	5
1.3. Resultados de la vigilancia tecnológica.....	5
2. Descripción del proyecto.....	6
3. Funcionalidades y objetivos.....	6
3.2. Navegación autónoma.....	6
3.3. Reconocimiento de imágenes.....	6
3.4. OpenCV.....	7
1. Detección de cajas.....	7
2. Detección de caras.....	7
3. Detección de personas.....	7
3.5. Interfaz Web.....	8
4. Diseños.....	9
4.2. Página web.....	9
4.3. Base de datos.....	13
4.4. Nodos y topics de ROS2.....	13
4.5. Modelado 3d.....	14
5. Implementación y verificación.....	16
5.2. ROS2.....	16
5.2.1. OfficeWorld.....	17
5.2.2. Slam y ProvideMap.....	17
5.2.3. Navigator2System.....	19
5.2.4. WebRequester.....	20
5.2.5. CaptureImage.....	21
5.2.6. WebVideoServer.....	23
5.2.7. ProjectLauncher.....	24
5.3. Web.....	25
5.3.1. Base de datos.....	26
5.3.2. Conexión Web-ROS2.....	28
5.3.3. Página del trabajador.....	29
5.3.4. Página del administrador.....	30
5.4. Caja física.....	32
6. Gestión del equipo.....	34
7. Reflexiones del equipo.....	35
8. Conclusiones y propuestas de mejora.....	36
9. Programas y servicios utilizados.....	37
10. Historial de contribuciones al documento.....	38

Índice de figuras

Figura 4.2.1 Diagrama de flujo de la web	9
Figura 4.2.2: Diseño de la página principal de la web	10
Figura 4.2.3: Diseño de la página de iniciar sesión de la web	10
Figura 4.2.4: Diseño de la página del administrador de la web	11
Figura 4.2.5: Diseño de la página del trabajador de la web	12
Figura 4.2.6: Diseño de la página de contacto de la web	12
Figura 4.3.1: Diseño de la base de datos SQL	13
Figura 4.4.1: Diagrama de nodos y topics de ROS2	14
Figura 4.5.1: Diseño de las piezas para imprimir para el robot	15
Figura 4.5.2: Diseño de las piezas montadas sobre el robot	15
Figura 5.2.1: Captura de la jerarquía de nodos y carpetas de ros2 hecha en VisualStudio Code	16
Figura 5.2.1.1: Captura del espacio de gazebo final de simulación visto desde arriba	17
Figura 5.2.2.1: Imagen del mapa del office_world	18
Figura 5.2.2.2: Captura de la jerarquía de carpetas y archivos dentro del nodo robo_entrega_provide_map	18
Figura 5.2.3.1: Captura del visualizador rviz2 después de iniciarse mostrando el mapa, la posición inicial del robot y la imagen de la cámara del robot	19
Figura 5.2.4.1: Diagrama para facilitar la comprensión del funcionamiento del nodo	20
Figura 5.2.4.2: Captura de la página del trabajador solicitando una entrega	21
Figura 5.2.4.3: Captura de la página del administrador con mensaje de finalización de entrega	21
Figura 5.2.6.1: Captura de la nueva ventana de navegador donde ver la cámara del robot	24
Figura 5.2.7.1: Captura del terminal y el rviz tras la ejecución del nodo nodes_launcher	25
Figura 5.3.1: Captura de la jerarquía de carpetas del apartado web	26

Figura 5.3.1.1: Diagrama de la base de datos	27
Figura 5.3.2.1: Capturas de la consola del navegador	28
Figura 5.3.2.2: Captura del mapa de la pagina del trabajador	29
Figura 5.3.3.1: Capturas del navegador en la página del trabajador	30
Figura 5.3.4.1: Capturas del navegador en la página del administrador	31
Figura 5.3.4.2: Captura de exito en crear usuario desde admin	32
Figura 5.3.4.3: Captura de edición de datos de usuario desde admin	32
Figura 5.4.1: Frame del video de demostracion del motor físico	33
Figura 6.1: Diagrama de representación de commits y ramas de git	35

1. Concepción de la idea

1.2. Descripción de la idea principal

La idea principal es un robot capaz de navegar por una oficina llevando paquetes y documentos a los trabajadores cuyas entregas de paquetes puedan ser gestionadas desde una interfaz web.

El robot también sería capaz de reconocer a los trabajadores de la oficina y sus portátiles para saber si ya han llegado al trabajo y si es así saber que puede entregarles los paquetes que haya para él, poder ver y reconocer paquetes que se hayan quedado por el suelo para avisar de su estado.

1.3. Resultados de la vigilancia tecnológica

Durante la primera fase de desarrollo de este proyecto realizamos un trabajo de investigación para buscar otros proyectos o productos del mercado que se parecieran a nuestra idea principal, Aquí resumimos aquellos que hemos encontrado:

- Goggo Network

Unos pequeños robots desarrollados por la empresa **Goggo Network**, son utilizados en uno de los supermercados de la empresa **Diá** para entregar la compra a los vecinos del pueblo de **Alcobendas**, cerca de Madrid.

- Dal-e

Un robot desarrollado por **Hyundai** y **Kia** cuyo empleo es recorrer el **Factorial Seongsu de Aegis Asset Management** en **Seúl** repartiendo vasos de café a los trabajadores del edificio aumentando la productividad.

- Robot Apple

Un robot en proceso de creación por la empresa **Apple** que podría ayudarte a realizar labores del hogar como cocinar o poner la lavadora, este pequeño robot sería capaz de traerte lo que necesitarás en ese momento, algún ingrediente, alguna cápsula de jabón, incluso podría traerte las pastillas en un momento dado. Este robot solo es una idea de momento, pero Apple ya está trabajando en traerlo a la vida.

2. Descripción del proyecto

Este proyecto llamado **RoboEntrega** trata de hacer el código de un robot turtlebot3 modelo Burger para que cumpla unos objetivos con unas funcionalidades, al terminar el proyecto el robot sea capaz de navegar una oficina o espacio de trabajo llevando paquetes a sus destinatarios en entregas que son gestionadas desde una interfaz web, capaz de detectar paquetes perdidos por el camino,, etiquetas de paquetes y reconocer a los trabajadores para saber si han entrado a trabajar y si puede entregarles algún paquete a su nombre.

3. Funcionalidades y objetivos

En este proyecto se nos pide cómo mínimo que el robot sea capaz de navegar de forma autónoma el espacio en el que se encuentra, detectar algún tipo de objeto o persona y una página web como interfaz de interacción con el robot.

3.2. Navegación autónoma

El robot es capaz de navegar el espacio de una oficina repleta de cubículos de trabajo esquivando los objetos que pueda haber por en medio y haciendo una ruta por la oficina buscando paquetes perdidos por los pasillos o manchas en el suelo.

3.3. Reconocimiento de imágenes

Además de detectar cajas con la cámara, el robot podrá detectar también personas , portátiles y etiquetas de los paquetes, sin embargo, con estos también los reconocerá.

En el caso de la etiqueta, el robot leerá el código de barras presente en las cajas o identificaciones del personal de la oficina y añadirá automáticamente ese paquete a la base de datos para poder mantener registro de la entrega de ese paquete en el caso de haberlo encontrado durante su patrulla por los pasillos o en caso de ser una identificación informará al servidor web de que ese trabajador ha llegado a la oficina y se encuentra presente en su puesto.

Al detectar una persona, el robot también podrá escanear la cara del sujeto y reconocerle teniendo la misma funcionalidad que al detectar el código de barras de su identificación en caso de que éste no la tenga o no sea visible.

3.4. OpenCV

En este proyecto se utiliza OpenCV como herramienta principal para realizar el procesamiento de imágenes capturadas en tiempo real desde una cámara. El objetivo principal del procesado es detectar elementos específicos dentro del entorno, como cajas (paquetes), rostros humanos y cuerpos completos, a través de diferentes técnicas de visión por computadora.

1. Detección de cajas

Se realiza mediante segmentación por color en el espacio HSV (Tono, Saturación, Valor). Este método convierte la imagen de BGR a HSV para facilitar la identificación de colores característicos (como marrones o zonas oscuras). Se aplican umbrales de color (`cv2.inRange`) para crear una máscara binaria, y luego se filtran los objetos por área y relación de aspecto. Esto permite descartar manchas pequeñas o formas que no coinciden con la geometría típica de una caja.

2. Detección de caras

Se aplica un clasificador Haar Cascade (`haarcascade_frontalface_default.xml`) sobre la imagen en escala de grises. Esta técnica es rápida y efectiva para identificar rostros humanos en diferentes escalas y posiciones. El clasificador analiza regiones de la imagen buscando patrones faciales conocidos, y en caso de detección, se dibuja un recuadro y se etiqueta como "Cara".

3. Detección de personas

De forma similar, se usa otro clasificador Haar (`haarcascade_fullbody.xml`) que está entrenado para reconocer la forma completa del cuerpo humano. Este permite detectar personas aunque no estén mirando directamente a la cámara, siempre que la figura sea reconocible en términos de proporción y postura.

4. Visualización y publicación

Cada imagen procesada se muestra visualmente en una ventana única usando `cv2.imshow()`, y también se convierte a un mensaje de ROS 2 (`sensor_msgs/msg/Image`) usando `CvBridge`. Estas imágenes son publicadas en un tópico ROS (`/deteccion/detecta_caja`), lo que permite que otros nodos puedan suscribirse a la imagen procesada y usar esa información para tareas de navegación, decisión o registro.

Este procesamiento en tiempo real permite que el sistema identifique eventos importantes en el entorno (personas, paquetes, manchas) de manera autónoma, eficiente y reutilizable dentro del ecosistema de ROS 2.

3.5. Interfaz Web

Las funcionalidades básicas del robot podrán ser monitorizadas y gestionadas desde la página web, ésta cuenta con las páginas de: índice, inicio de sesión, gestor de trabajador y gestor de administrador.

La página de índice contiene mínima información sobre la startup ficticia creada para este proyecto.

La página de inicio de sesión contiene un formulario donde los trabajadores ingresan sus datos y pueden acceder a sus respectivas páginas, los trabajadores a la página del trabajador y el administrador a la página del admin.

La página del trabajador contiene un mapa de su oficina con un indicador actualizado en tiempo real de la ubicación del robot, también tiene un cuadro con la imagen actual del robot, es decir, qué es lo que ve el robot, y por último un pequeño cuadro donde el trabajador puede decirle al robot que tiene un paquete que quiere entregarle a otra persona y el robot debe ir a su puesto a recoger el paquete y hacer la entrega.

La página del administrador contiene una serie de tablas que contienen toda la información sobre los trabajadores, los puestos de trabajo y los robots contratados por la oficina. Además, hay un cuadro donde se reciben mensajes del estado del servidor web y las actualizaciones sobre las actividades del robot como “He encontrado una caja” o “Estoy realizando una entrega de Pedro para Manuel”.

4. Diseños

4.2. Página web

El diagrama del flujo de la web, hecha en la web [drawio](#).

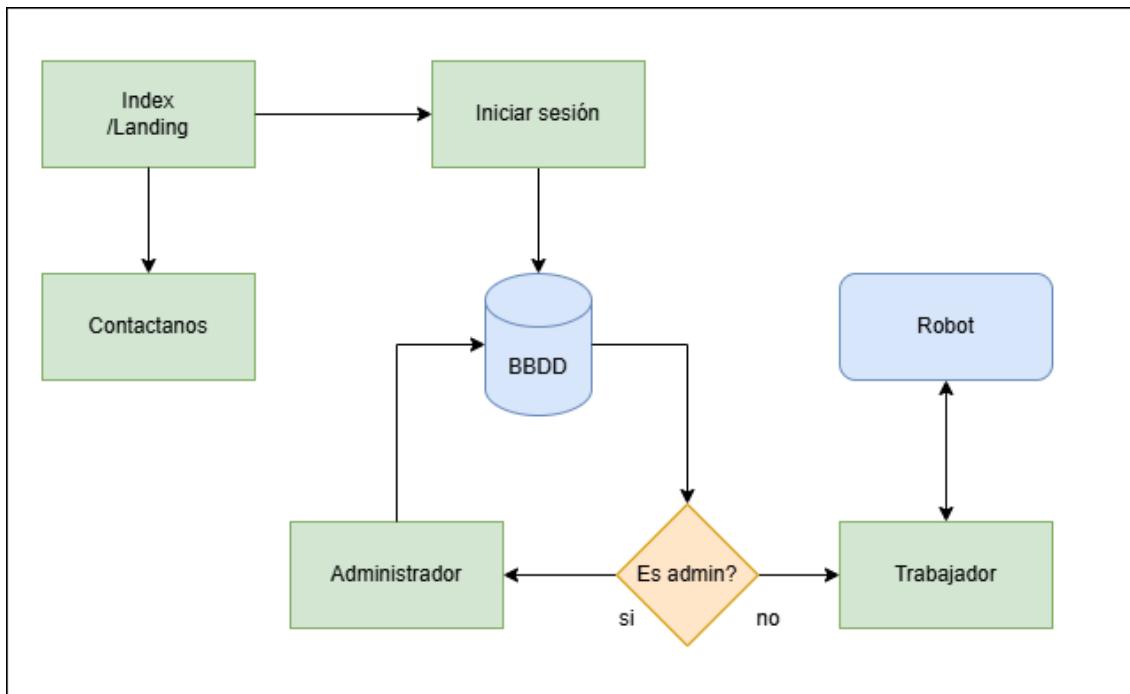


Figura 4.2.1: Diagrama de flujo de la web

A continuación, los diseños de las páginas hechas en [Axure](#).

La página de inicio / Índex / Landing.

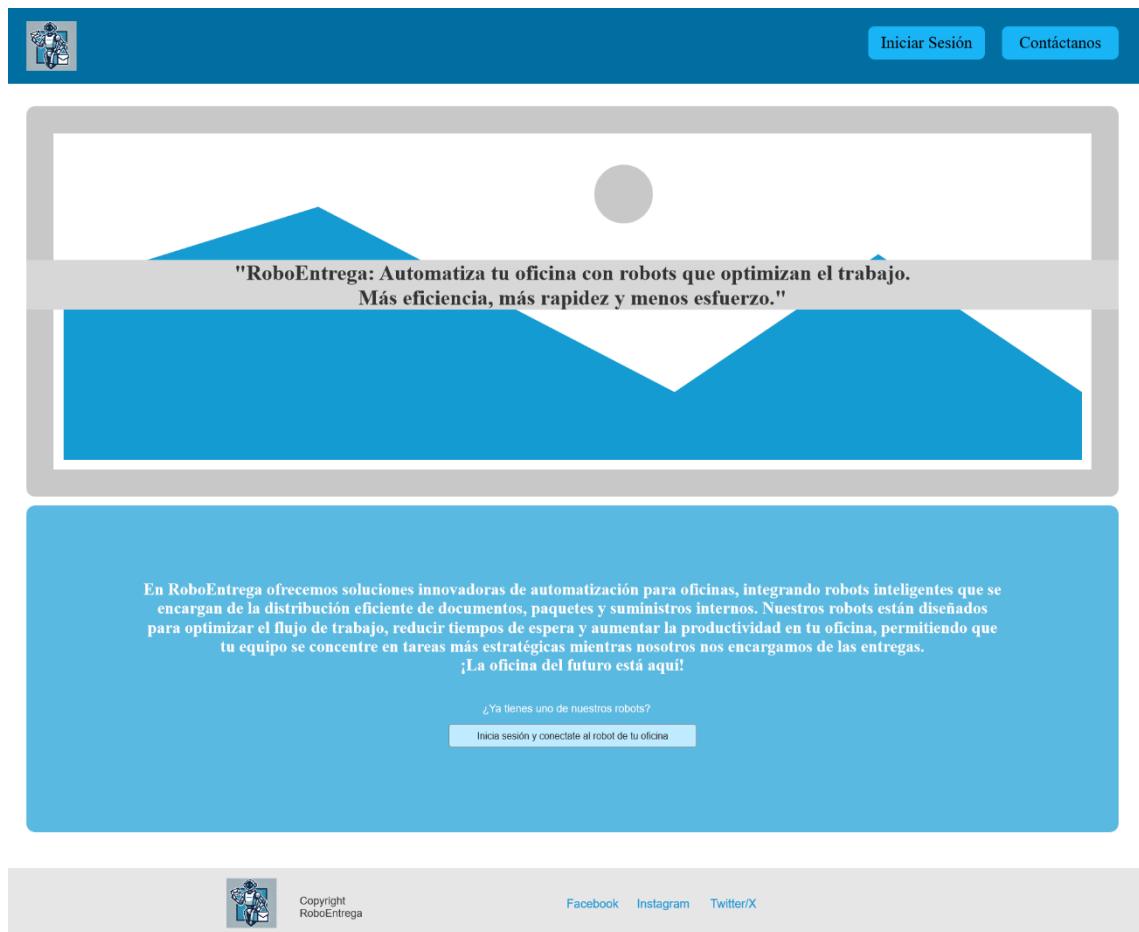


Figura 4.2.2: Diseño de la página principal de la web

La página de inicio de sesión.

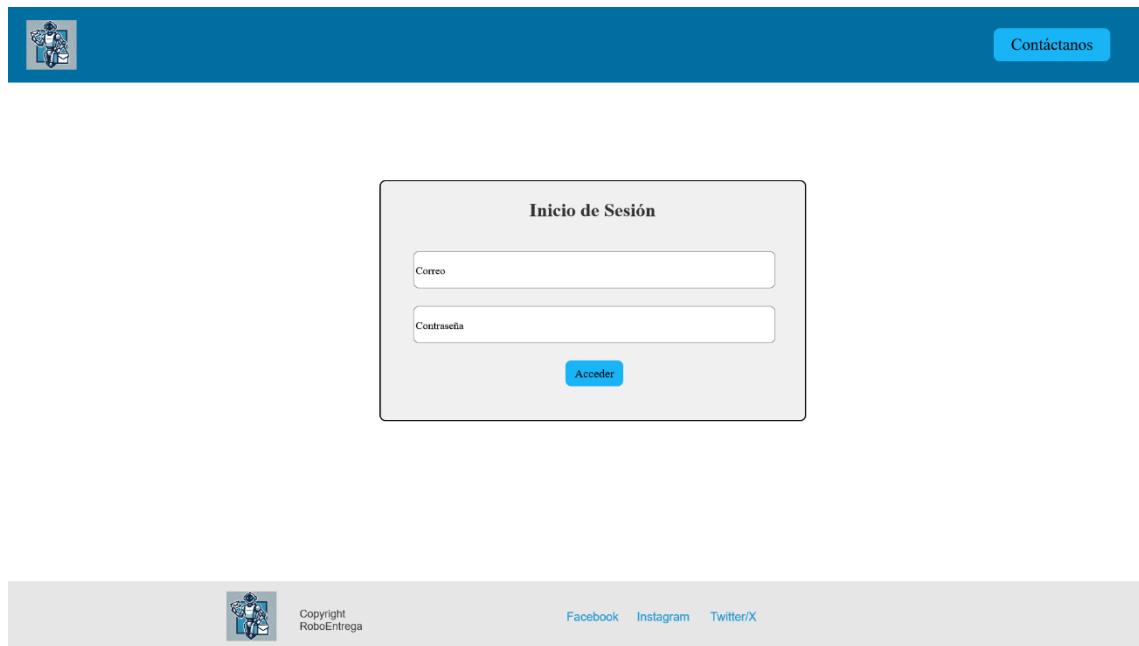


Figura 4.2.3: Diseño de la página de iniciar sesión de la web

La página del Administrador. Donde las tablas representan tablas que se llenarán con los datos correspondientes de la base de datos.

The screenshot shows a dark blue header bar with a logo on the left, 'Cerrar Sesión' (Logout) and 'Contáctanos' (Contact us) buttons on the right. Below the header, there are three tables:

- Mis robots**: A table with columns 'id_robot', 'Batería', and 'Acciones'. It has two empty rows.
- Trabajadores**: A table with columns 'dni', 'Correo', and 'Column 3'. It has two empty rows.
- Mesas**: A table with columns 'id_mesa', 'lat', and 'lng'. It has two empty rows.

Below these tables is a large gray box labeled 'Mensajes del robot' (Messages from the robot), which is currently empty.

At the bottom of the page is a footer bar with the RoboEntrega logo, 'Copyright RoboEntrega', and social media links for Facebook, Instagram, and Twitter/X.

Figura 4.2.4: Diseño de la página del administrador de la web

La página del trabajador.

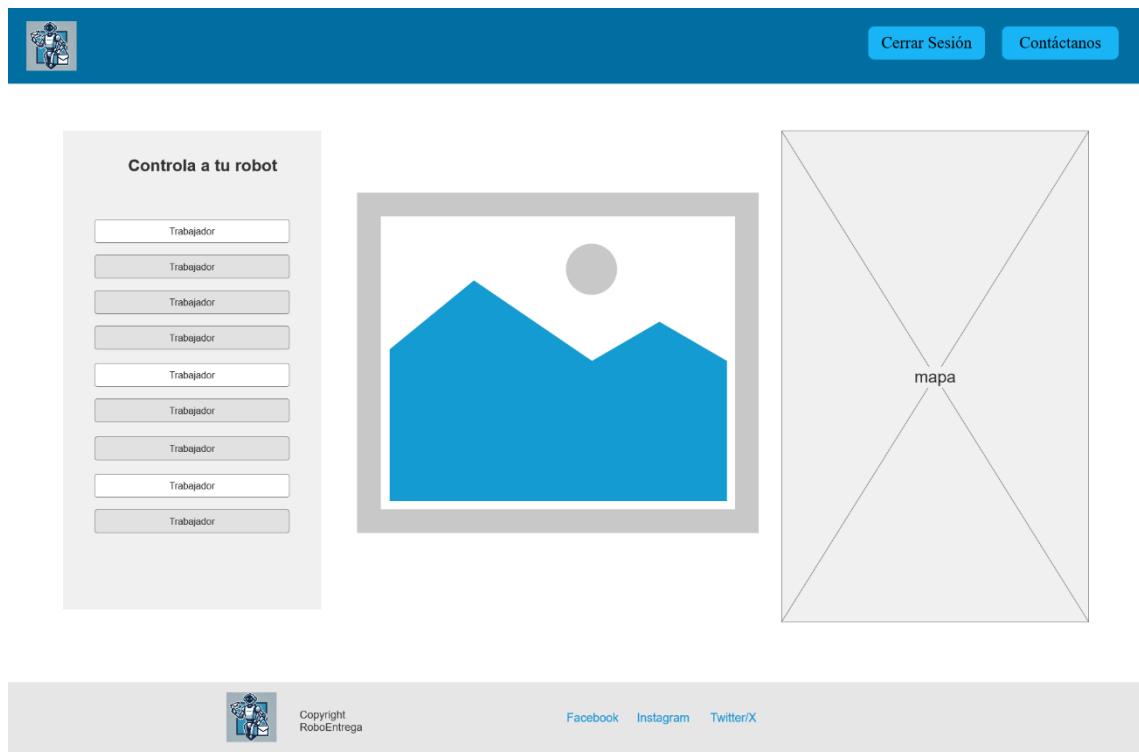


Figura 4.2.5: Diseño de la página del trabajador de la web

La página de contacto.

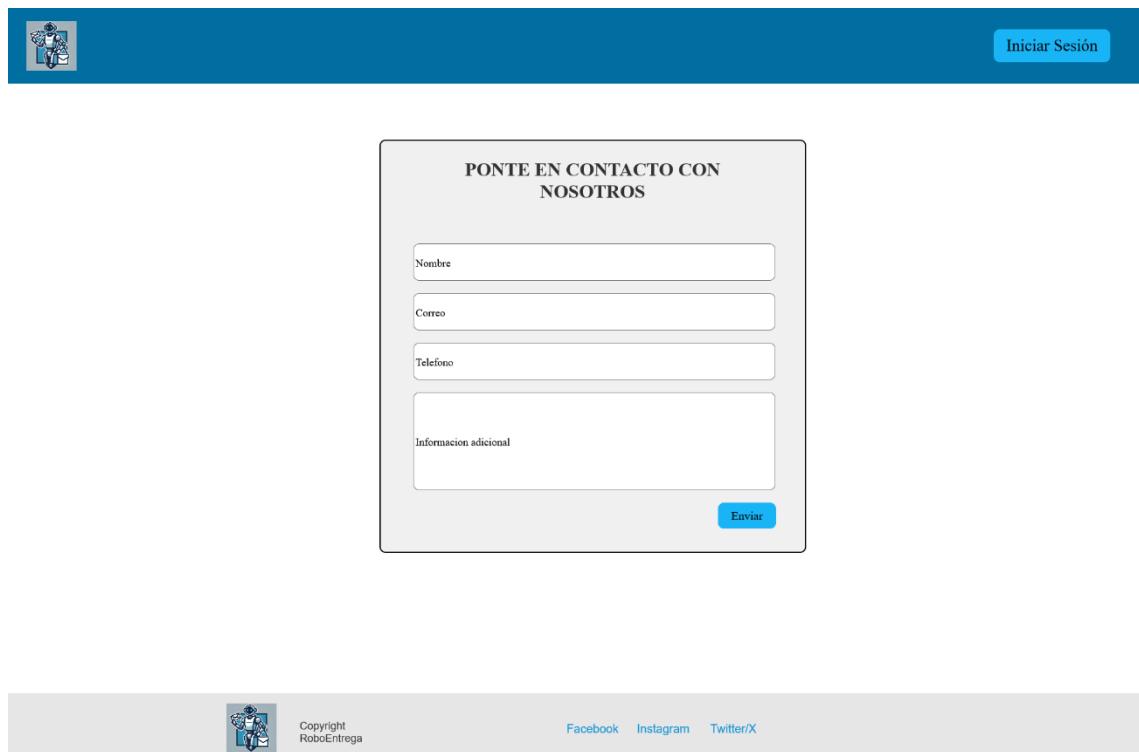


Figura 4.2.6: Diseño de la página de contacto de la web

4.3. Base de datos

Nuestra base de datos contiene la información del robot que ronda la oficina, los trabajadores y la mesa donde trabaja el empleado, relacionado entre el trabajador y el robot se encuentra la tabla de entregas, estas representan las entregas que puede realizar el robot, principalmente aquellas solicitadas por el empleado desde la página web.

La tabla de mesas contiene en todo momento las coordenadas de rviz2 de las ubicaciones de cada mesa en el mapa escaneado del robot para que así el robot sepa a qué mesa/cubículo ir para entregar un paquete a alguien.

Este diseño fue creado usando [MySQL Workbench](#).

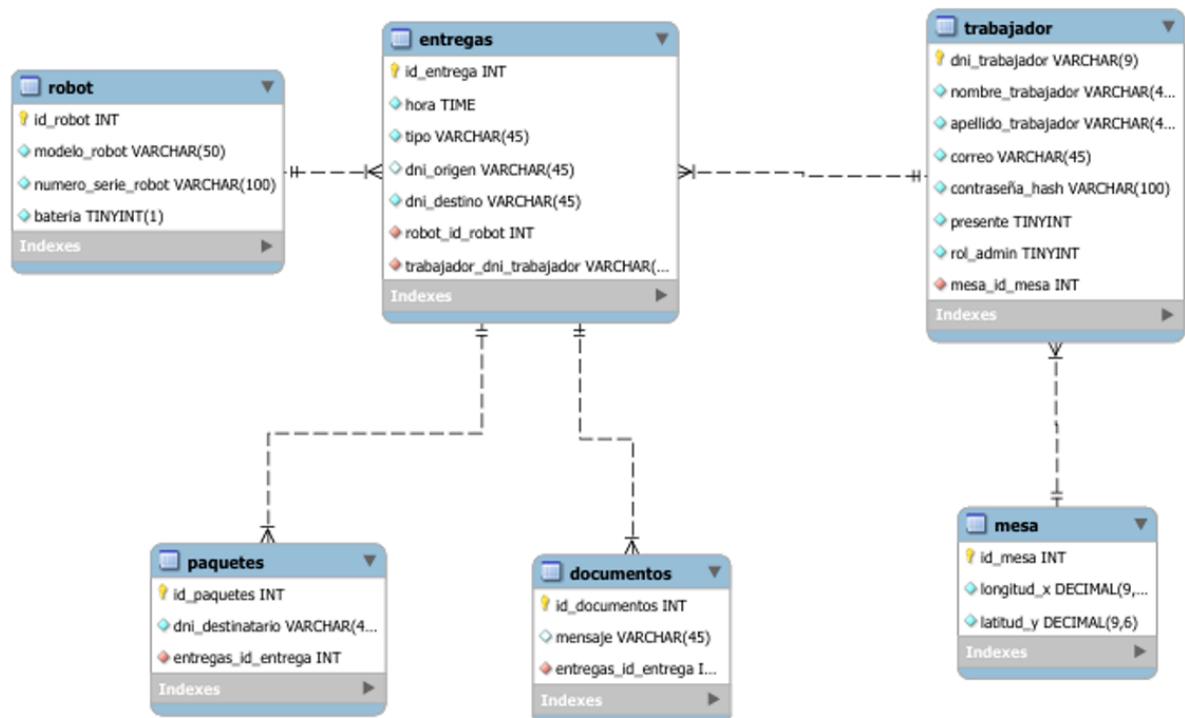


Figura 4.3.1: Diseño de la base de datos SQL

4.4. Nodos y topics de ROS2

Nuestros elementos tanto de la web a los nodos de ROS2 como entre los nodos se comunican mediante topics, este diagrama contiene nuestros nodos creados y cómo se comunican con otros elementos mediante topics.

Este diagrama fue creado usando la web [drawio](#).

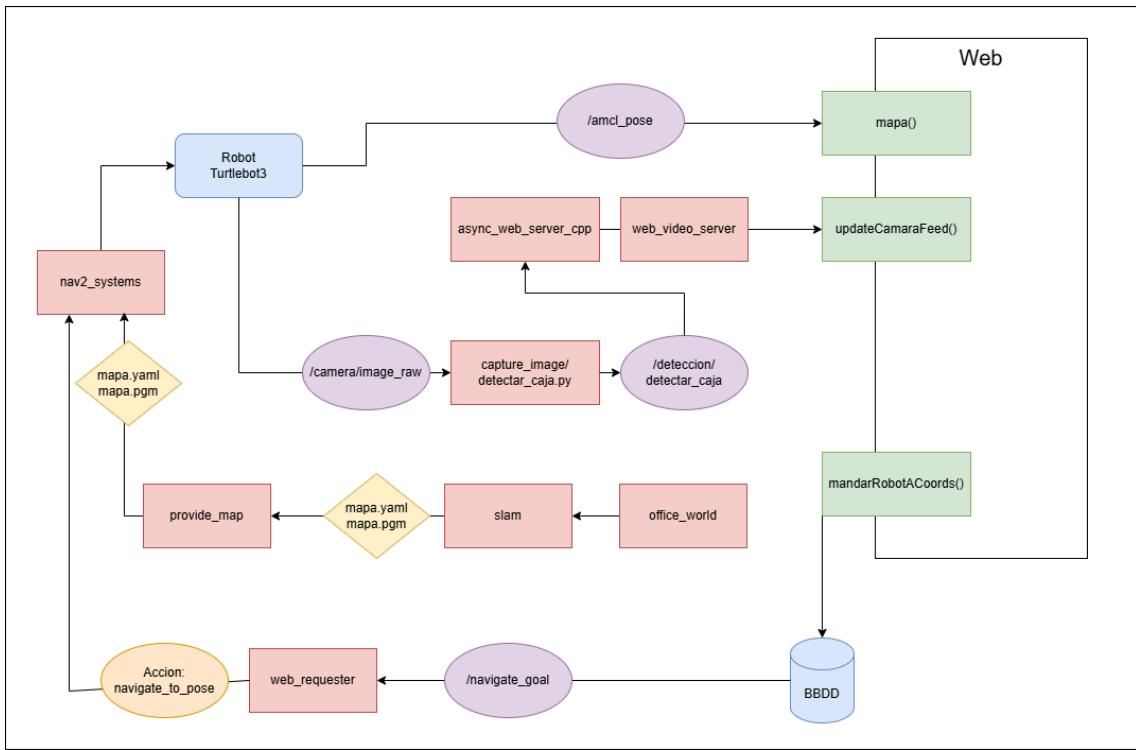
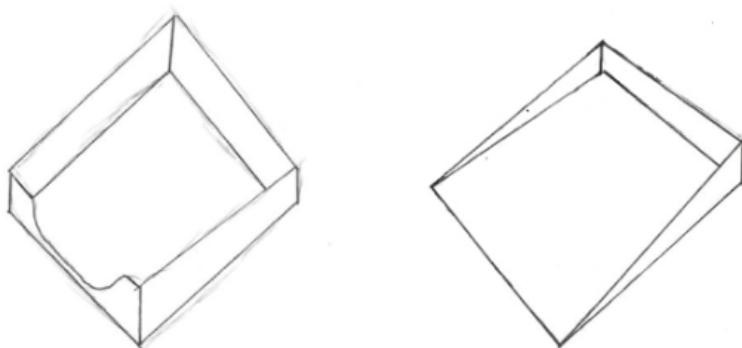


Figura 4.4.1: Diagrama de nodos y topics de ROS2

4.5. Modelado 3d

Como extra en este proyecto hemos diseñado e impreso una pequeña caja unida al cuerpo del robot para llevar los paquetes y una pequeña pala para levantar el paquete y entregarlo a la persona.

El diseño fue hecho a mano con lápiz y papel y el modelado 3d fue hecho con [Blender](#) y finalmente se imprimió con la impresora 3d [Ender 3 V3 KE](#).



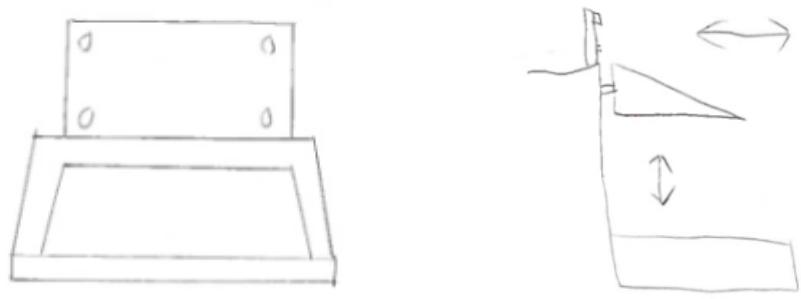


Figura 4.5.1: Diseño de las piezas para imprimir para el robot

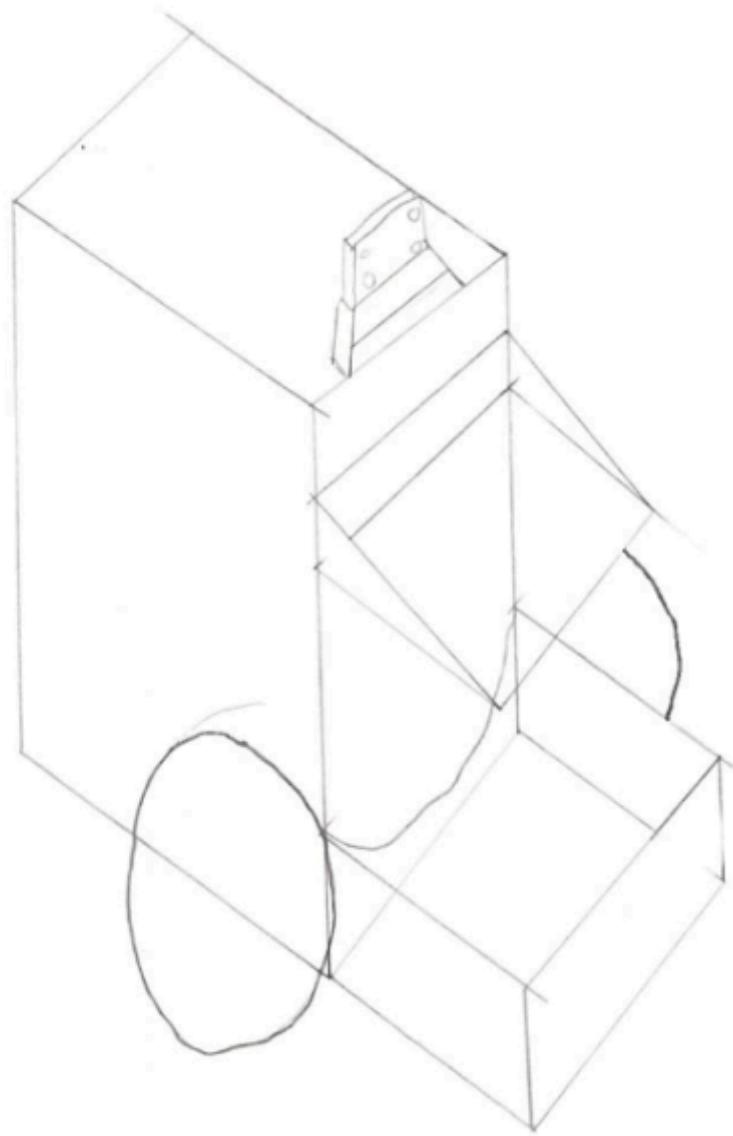


Figura 4.5.2: Diseño de las piezas montadas sobre el robot

5. Implementación y verificación

5.2. ROS2

En esta sección se detallará cómo funciona cada nodo que hemos creado de ROS2 y cómo verificamos que funciona.

En esta imagen se muestra nuestra jerarquía de carpetas de ROS2 conteniendo todos nuestros nodos.

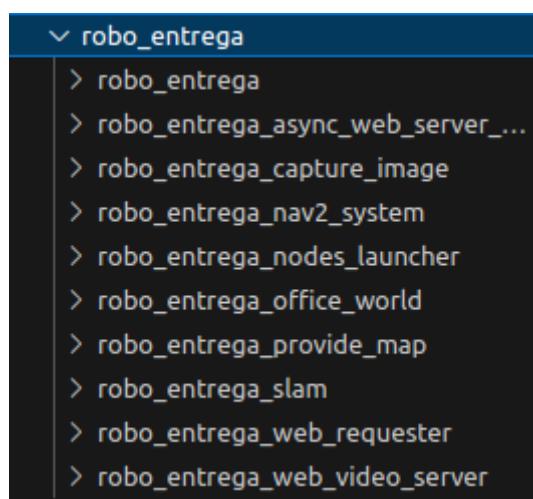


Figura 5.2.1: Captura de la jerarquía de nodos y carpetas de ros2 hecha en VisualStudio Code

Como se puede ver, cada uno de nuestros nodos cuelga de un metapquete llamado **robo_entrega**.

Cada uno de estos nodos se da por verificado, a menos que se indique lo contrario en este documento, cuando a la hora de ejecutarlo se obtienen los resultados esperados. En cualquier caso, se adjuntan capturas de éstos resultados aunque no se indiquen explícitamente.

5.2.1. OfficeWorld

Este nodo llamado **robo_entrega_office_world** contiene un mundo de gazebo para la simulación y testeo de las funcionalidades del robot. Este mundo está basado en el creado por [Melih Erdogan](#), se puede encontrar la versión original de nuestro mundo en [su repositorio de datasets de mundos gazebo](#).

Para nuestro proyecto redujimos la carga computacional del mundo eliminando puertas, mesas, objetos muy detallados y habitaciones que no necesitábamos tener. Además, añadimos modelos de cajas/paquetes para testear la detección de objetos y modelos de personas para testear la detección de personas.

Así quedó después de nuestros cambios:

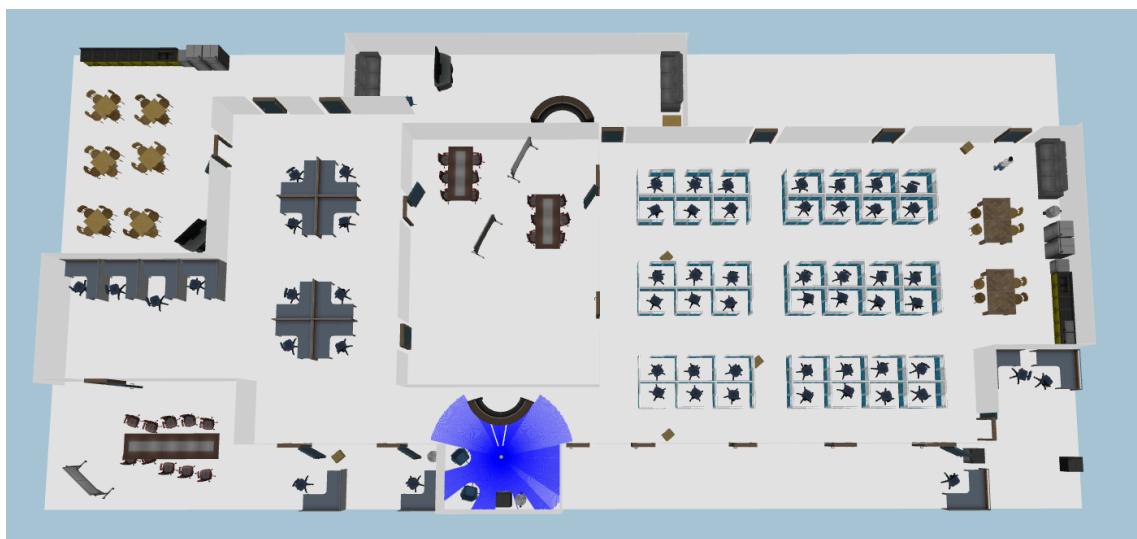


Figura 5.2.1.1: Captura del espacio de gazebo final de simulación visto desde arriba

5.2.2. Slam y ProvideMap

Estos dos nodos llamados **robo_entrega_slam** y **robo_entrega_provide_map** tienen como objetivo proveer el mapa al robot para que sepa por dónde navegar.

Robo_entrega_slam es un paquete que ejecutamos cuando está en un espacio nuevo, por ejemplo, la primera vez que abrimos gazebo o cuando debíamos escanear el espacio real de la clase. Este nodo escanea los alrededores del robot conforme se mueve para generar un mapa en escala de grises con 3 tonos, **blanco** para zonas por donde el robot puede ir, **negro** para paredes y objetos sólidos que el robot no puede atravesar y **gris** para aquellas zonas que el robot no puede escanear, ya sea porque está detrás de una pared o porque no ha ido a ese lugar. Un mapa de este tipo contiene

un archivo **.yaml** con la información de la imagen y una imagen **.pgm**. Este es el mapa pgm de nuestro mundo de gazebo **office_world**:

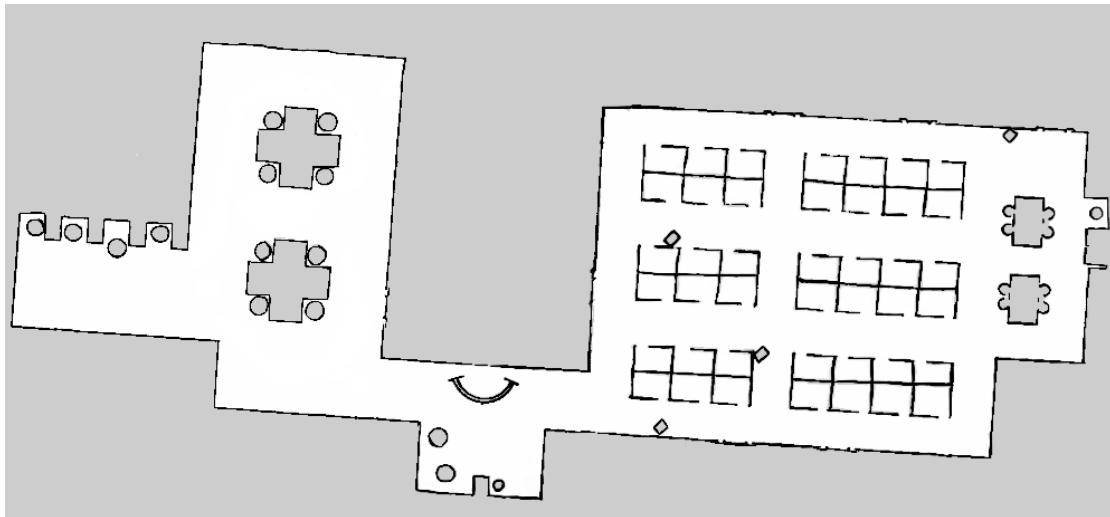


Figura 5.2.2.1: Imagen del mapa del *office_world*

Cuando este mapa se ha terminado de escanear, tanto el **.pgm** como el **.yaml**, se guardan ambos en la carpeta **map** dentro del nodo **robo_entrega_provide_map**, y cuando el nodo **robo_entrega_nav2_system** se ejecuta realiza una llamada a este nodo para obtener el mapa.

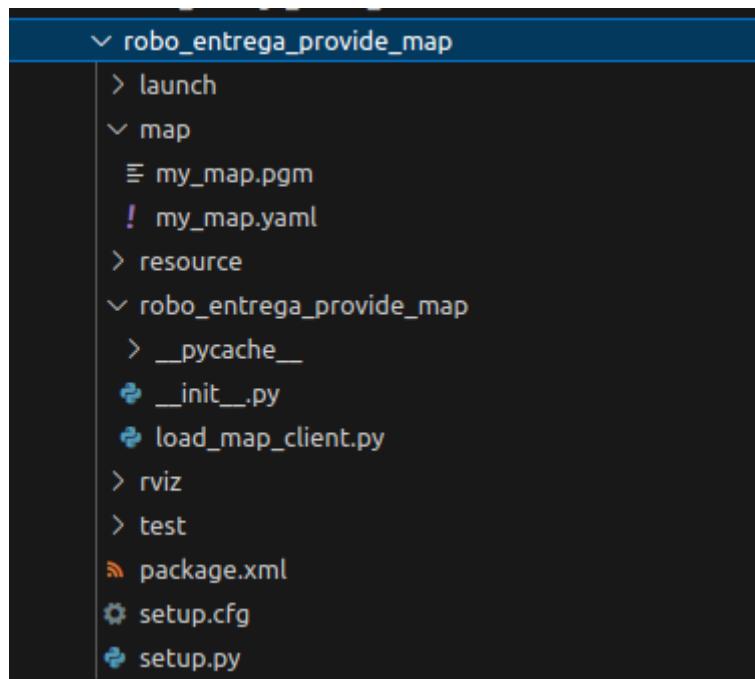


Figura 5.2.2.2: Captura de la jerarquía de carpetas y archivos dentro del nodo *robo_entrega_provide_map*

5.2.3. Navigator2System

El nodo **robo_entrega_nav2_system** se encarga de inicializar todos los servicios de movimiento y navegación del robot y, una vez inicializados, lanza el visualizado rviz2 para poder monitorizar al robot.

Más concretamente lanza los siguientes servicios: **nav2_map_server**, **nav2_amcl**, **nav2_planner**, **nav2_controller**, **nav2_bt_navigator**, **nav2_recoveries** y **nav2_lifecycle_manager**. Despues de lanzarlos, el nodo lanza también el **rviz2** con un archivo de configuración ya preparado y ejecuta nuestros archivos **load_map_client.py** del nodo **provide_map** el cual le da el mapa al rviz2 para poder visualizarlo y el archivo **initial_pose_pub.py** que le indica al robot en qué punto del mapa se encuentra como punto inicial.

La verificación de funcionalidad de este nodo se comprueba en 2 fases, la primera es que el rviz2 se inicia como se ha explicado y se muestra el visualizado como se observa en la siguiente imagen. La segunda fase se confirma cuando, con todo el proyecto ejecutado, tanto desde terminal como desde otro de los nodos se le pide al robot un movimiento a un punto concreto y lo hace.

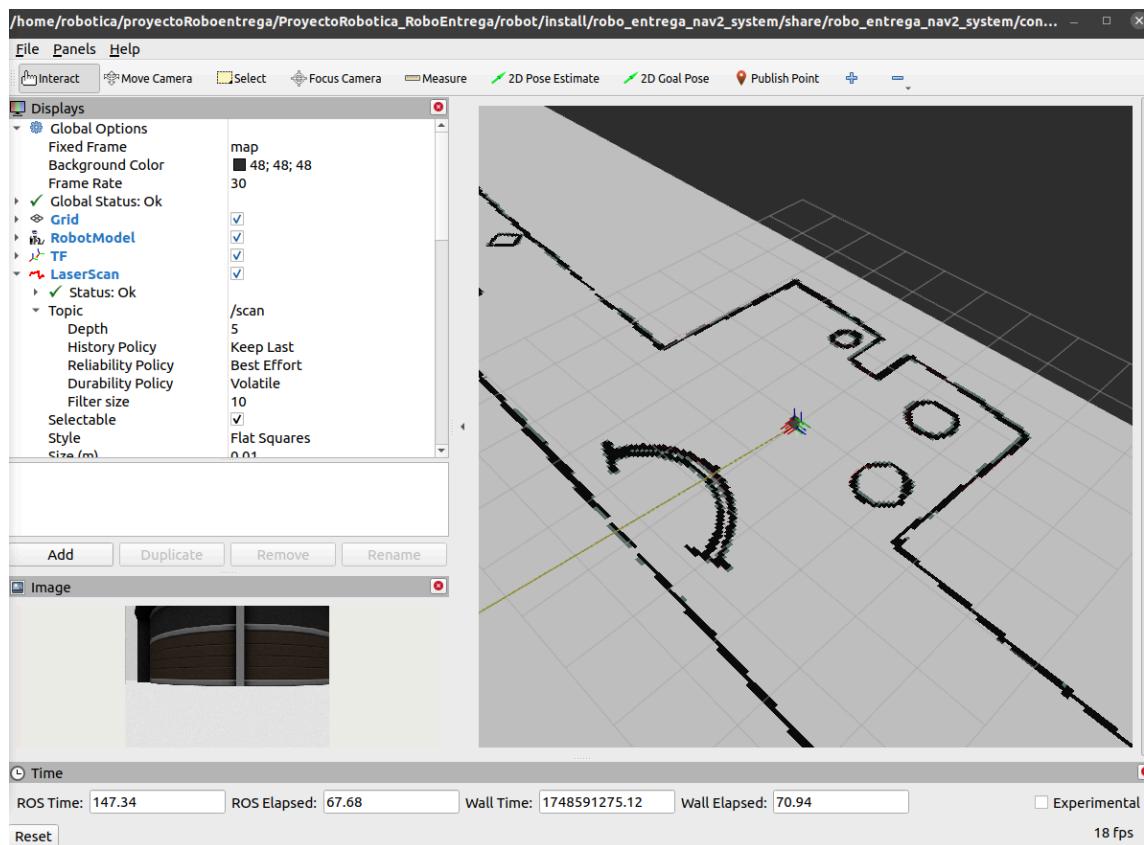


Figura 5.2.3.1: Captura del visualizador rviz2 después de iniciarse mostrando el mapa, la posición inicial del robot y la imagen de la cámara del robot

5.2.4. WebRequester

Este nodo **robo_entrega_web_requester** se conecta a la base de datos del servidor web y cada 10 segundos llama a la api de la base de datos para comprobar de la tabla de entregas, aquellas que están sin realizar, de haber alguna o más de una, el nodo comprobará qué entrega es la que lleva menos tiempo o, lo que es lo mismo, cual es más cercana o tiene un menor recorrido. Una vez ha decidido qué entrega hacer, crea una acción para decirle a donde debe ir, de ser una entrega donde debe ir a algún puesto a recoger el paquete para luego ir a entregarlo a otro, primero hace la primera acción y cuando recibe la confirmación de que lo ha hecho, hace la segunda acción de ir a la segunda ubicación. Cuando ha recibido la confirmación de que ha llegado al segundo puesto y, por tanto, ha terminado la entrega, vuelve a llamar a la api de la base de datos para modificar la entrada de la tabla e indicar que esa entrega ya se ha realizado.

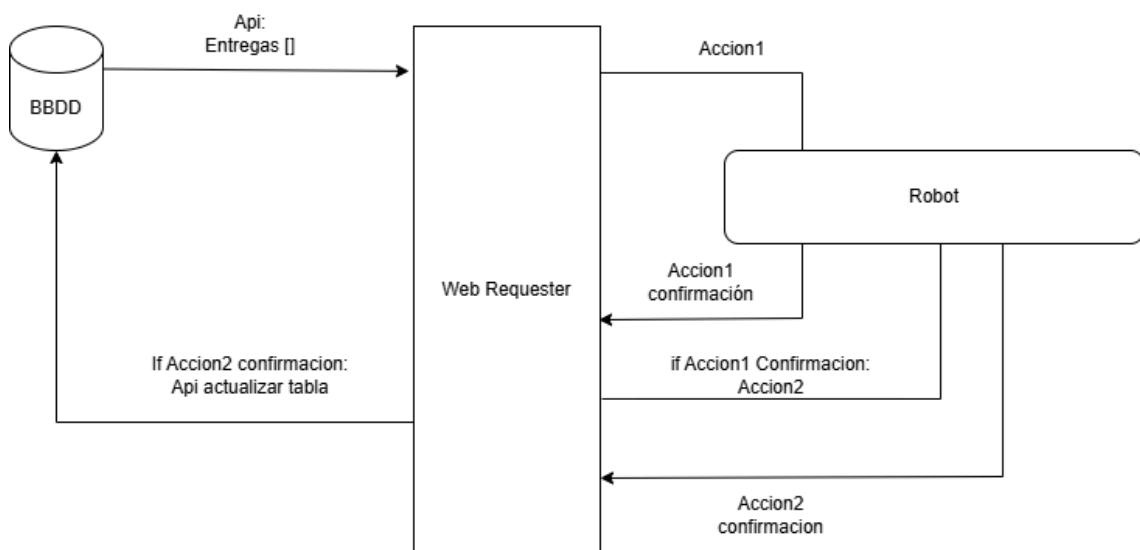


Figura 5.2.4.1: Diagrama para facilitar la comprensión del funcionamiento del nodo

Aquí se muestran capturas demostrando el funcionamiento del nodo; Primero se realiza una petición de un usuario a otro desde la página de trabajador (Figura 5.2.4.2) y el robot navega el espacio para realizar la entrega, una vez terminada manda un mensaje al administrador para indicar que ha terminado (Figura 5.2.4.3).

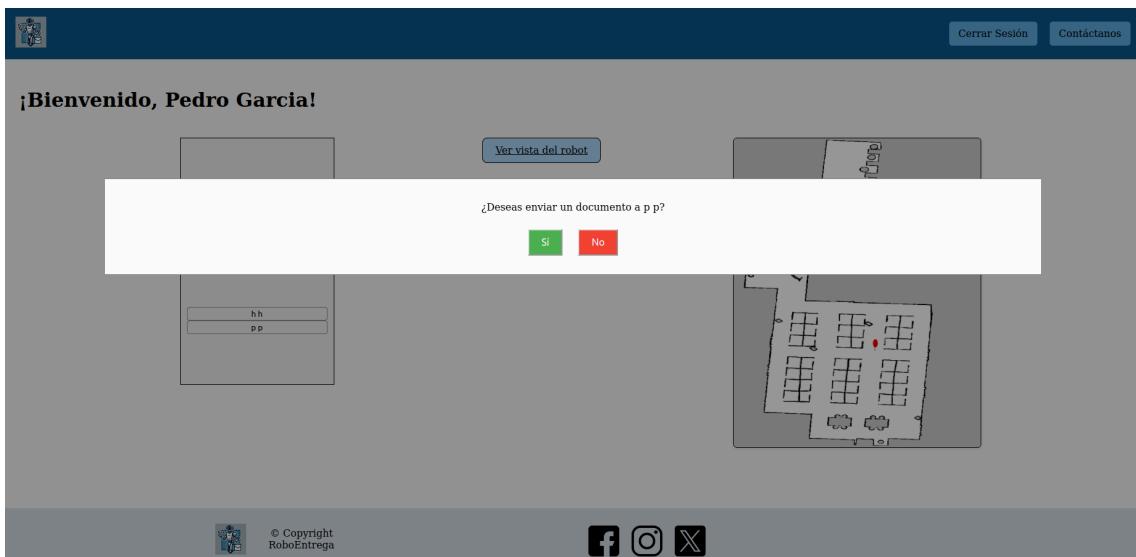


Figura 5.2.4.2: Captura de la página del trabajador solicitando una entrega

Actividad



Figura 5.2.4.3: Captura de la página del administrador con mensaje de finalización de entrega

5.2.5. CaptureImage

Este nodo, **robo_entrega_capture_image**, recibe la imagen que envía el robot de su cámara a través del topic **/camera_raw** y posteriormente detecta una serie de objetos distintos con diferentes métodos y realiza acciones en consecuencia:

La detección de estos objetos se ha llevado a cabo mediante el entrenamiento de un modelo de detección de objetos basado en YOLOv3, integrado en un entorno ROS2. El objetivo de esto es permitir que el robot pueda detectar objetos

específicos en su entorno, mediante la cámara instalada en la parte frontal, como pueden ser en nuestro caso, cajas, personas o portátiles que hay por la oficina, utilizando visión artificial en tiempo real.

Para llevar a cabo dicho entrenamiento se han seguido una serie de pasos:

1.- Preparación del Dataset

Este primer paso consistió en la recopilación de imágenes representativas de los objetos a detectar. Estas imágenes fueron organizadas en dos conjuntos, train y test. A cada imagen se le asoció un archivo .txt con las anotaciones en el formato YOLO: <clase><x_center> <y_center> <width> <height>, con valores normalizados entre 0 y 1.

Se estableció un conjunto de clases con sus identificadores correspondientes, descritos en el archivo new_names.names. Cada línea del archivo contiene el nombre de una clase, en el orden de su identificador numérico.

Antes de continuar, se verificó que cada imagen contara con su anotación correspondiente.

En nuestro caso se utilizaron unas 500 imágenes para el aprendizaje de las cajas, alrededor de otras 500 para el aprendizaje de los portátiles, y poco más de 100 imágenes para el aprendizaje de personas.

2.- Conversión de Anotaciones a Formato CSV

Para facilitar la generación de los archivos TFRecord, se convirtió el conjunto de anotaciones en formato YOLO a formato CSV mediante el script Python (txt_a_csv.py). Este script también transformaba las coordenadas normalizadas a coordenadas absolutas (xmin, ymin, xmax, ymax), tomando como referencia un tamaño fijo de imagen (640x640 píxeles).

Se generaron así dos archivos: train_labels.csv y test_labels.csv, correspondientes a los conjuntos de entrenamiento y validación, respectivamente.

3.- Generación de TFRecord

Con los archivos CSV preparados, se procedió a generar los archivos TFRecord, formato nativo de TensorFlow para la lectura eficiente de datos. Esta operación se llevó a cabo con el script generate_tfrecord_n.py, que recorría cada fila del CSV, leía la imagen asociada y empaquetaba los datos de forma binaria.

Es importante destacar que este paso requiere que las rutas de las imágenes sean correctas. Se verificó cuidadosamente que las rutas del script apuntaran al directorio adecuado, ya que errores frecuentes en este paso suelen estar relacionados con imágenes faltantes o mal ubicadas.

4.- Entrenamiento del Modelo YOLOv3

Una vez preparados los TFRecord, se procedió al entrenamiento del modelo YOLOv3. Para ello, se utilizó el script train.py, configurado para aceptar como parámetros el archivo de clases, el número de clases y las rutas a los datos de entrenamiento y validación.

El modelo se definía en el archivo models.py, basado en la implementación de YOLOv3 para TensorFlow 2. Se utilizó un esquema de callbacks para guardar los pesos del modelo a lo largo del entrenamiento.

El entrenamiento se llevó a cabo con éxito, generando un archivo de pesos (yolov3_train_20.tf) que contenía la información necesaria para realizar inferencias posteriormente.

5.- Evaluación y Pruebas del Modelo

Una vez finalizado el entrenamiento, se utilizó el script predict.py para realizar pruebas sobre imágenes individuales. Se pasaron como argumentos el archivo de clases, el número de clases, el archivo de pesos entrenado, la imagen de entrada y la imagen de salida deseada.

El modelo fue capaz de detectar con éxito los objetos entrenados, mostrando en pantalla las etiquetas y las probabilidades de detección. El video de verificación de la funcionalidad se encuentra en el directorio **documentacion** del repositorio git:

[Video de verificación de captura de detección de objetos](#)

La detección de etiquetas se llevó a cabo usando **pymzbar**

5.2.6. WebVideoServer

El nodo **robo_entrega_web_video_server** y el nodo **robo_entrega_async_web_server_cpp** trabajan juntos en transformar la imagen dada por un topic, por ejemplo **/camera_raw** o **/deteccion/detectarcaja** (un topic creado por nosotros para transmitir las imágenes donde ya se haya hecho la detección) para poder mostrarse en la página web de manera que parezca un video cuando en realidad son muchos frames o imágenes seguidas.

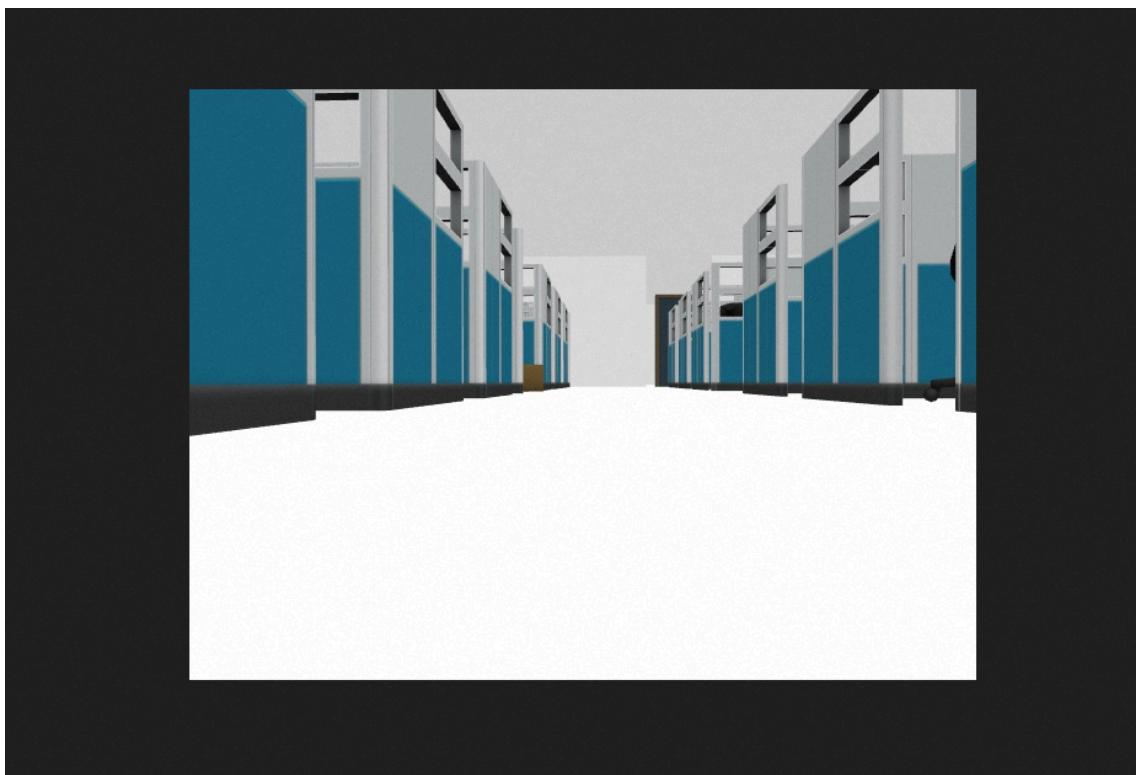


Figura 5.2.6.1: Captura de la nueva ventana de navegador donde ver la cámara del robot

5.2.7. ProjectLauncher

Este nodo **robo_entrega_nodes_launcher** tiene como único objetivo lanzar todos los demás nodos de ROS2 necesarios para ejecutar el proyecto sin tener que abrir tantos terminales. Utiliza acciones del launcher de ROS2 para controlar el orden de ejecución de los nodos lanzando los nodos usando **ProcessEventHandler** junto a **OnProcessStart()** para esperar a cuando un nodo ha terminado de inicializarse para empezar a lanzar el siguiente en este orden:

Nav2_system -> capture_image -> waypoint_follower

El resto de nodos que no se lanzan mediante este nodo se debe a que o no son necesarios para su funcionamiento, como el **office_world** o requieres primero el lanzamiento del servidor web el cual no se lanzó desde el entorno de ROS2 como **web_video_server** o **web_requester**.

La verificación del funcionamiento de este nodo se confirma cuando al ejecutarse todos los demás nodos funcionan como deberían, no hay ningún indicador visual ni textual de que haya funcionado incorrectamente.

En la siguiente imagen se muestra una captura de pantalla justo después de ejecutar el launch de este nodo y como se puede observar, se abre el rviz2 que es propio del nodo **nav2_system**, se ve la cámara del nodo **capture_imagen** y comienza a moverse siguiendo la ruta del **waypoint_follower**:

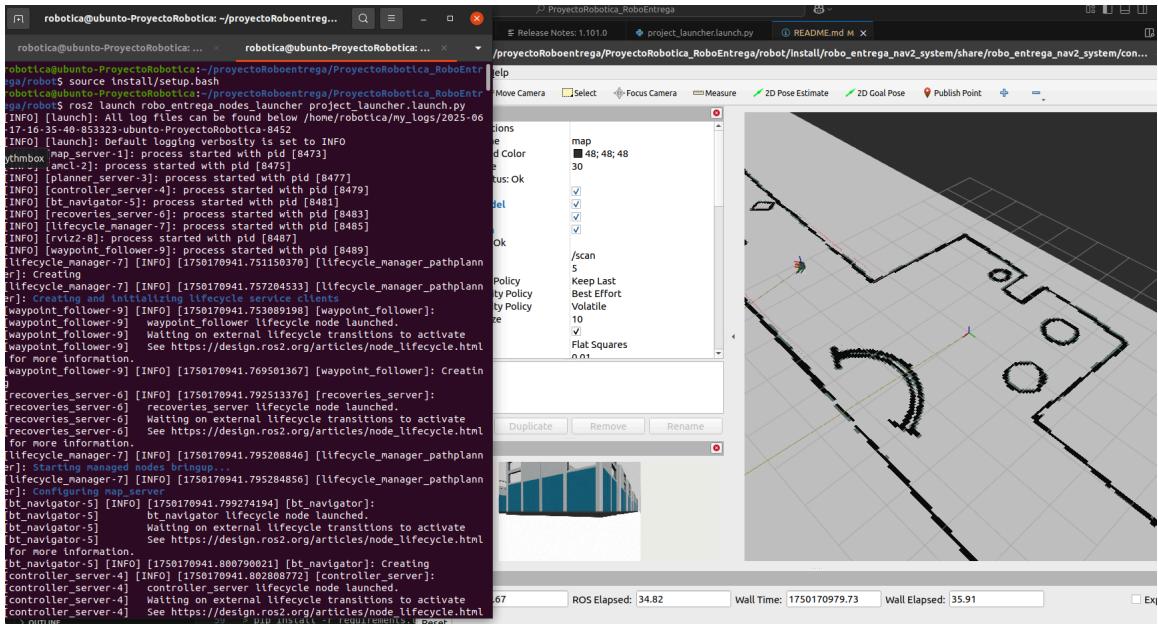


Figura 5.2.7.1: Captura del terminal y el rviz tras la ejecución del nodo **nodes_launcher**

5.3. Web

Nuestra web utiliza el framework backend es **Hask**. Nuestra web contiene la siguiente jerarquía de carpetas y archivos:

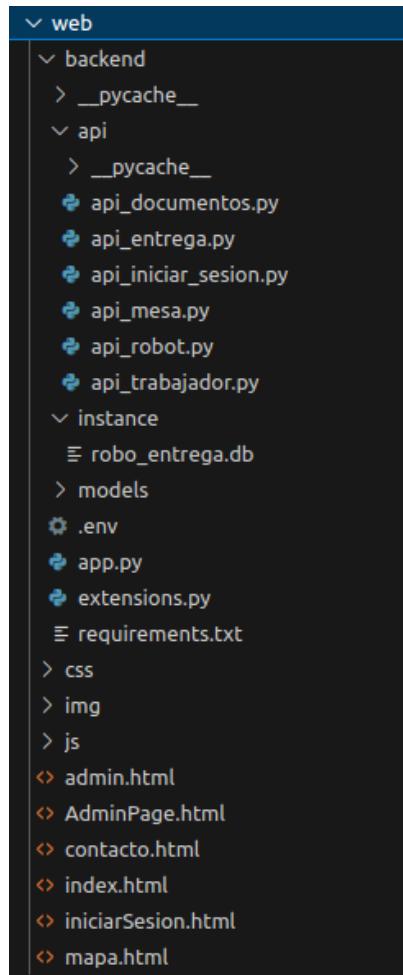


Figura 5.3.1: Captura de la jerarquía de carpetas del apartado web

Como se observa, la base de datos se encuentra dentro de **/backend/instances** justo dentro de la carpeta de **Backend** hay un documento llamado requirements.txt, este documento contiene las versiones de los programas y dependencias de flask para poder ejecutar el servidor backend con la base de datos.

5.3.1. Base de datos

Nuestra base de datos fue diseñada en [MySQL Workshop](#) y creada mediante código.

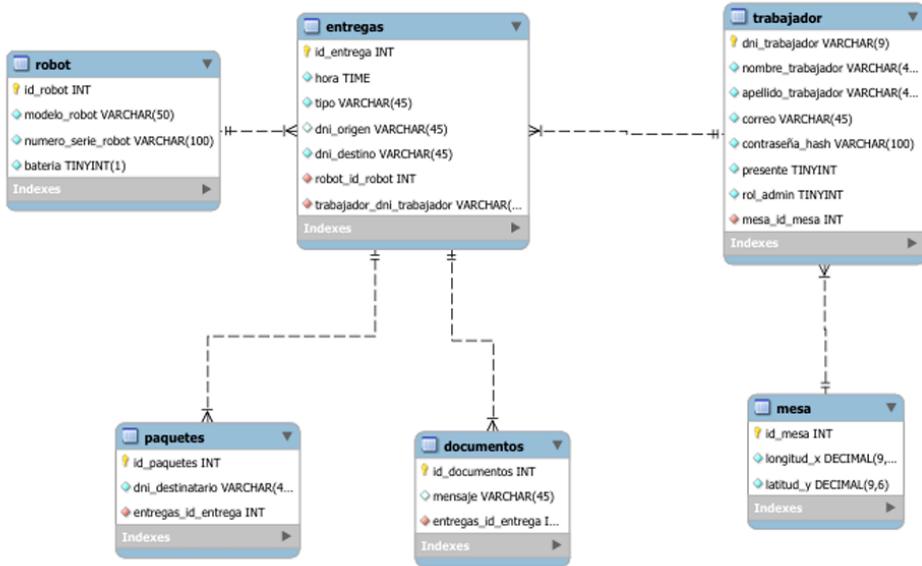


Figura 5.3.1.1: Diagrama de la base de datos

Esta BD contiene una tabla llamada mesa que en una situación real siempre estará llena con las coordenadas de rviz2 de los puestos de trabajo de la oficina para que el robot pueda encontrarlos. La mesa está relacionada con una entrada de la tabla trabajador, aunque una mesa puede existir sin trabajador (ya que el puesto podría estar vacío), un trabajador puede existir sin puesto, es decir, un trabajador debe crearse asignado a un puesto ya creado.

El trabajador cuenta con un campo llamado **presente**, este campo sirve para saber si el trabajador ha llegado ya a la oficina o no, en la lógica de nuestro robot esto se traduce a si puede hacerle la entrega de un paquete o no, o si un trabajador puede siquiera solicitar hacer una entrega a esta persona.

Un trabajador, como se puede ver en el diagrama, también está relacionado con la tabla de entregas, esto es porque cada entrega, sea del tipo que sea, necesita saber dos personas, el emisor y el receptor del paquete, ya que puede ser que llegue un paquete por recepción, en cuyo caso solo se sabe el receptor, o puede ser un trabajador de dentro de la oficina el que quiera entregarle algo u otro, en cuyo caso sabremos emisor y receptor. Saber ambos trabajadores nos sirve para cuando el robot en el nodo **web_requester** pueda leer todas las entregas que no se han realizado y sepa el recorrido completo de todas ellas y poder decidir cuál es el más corto.

La verificación de esta funcionalidad se confirma mediante una pequeña “consola” en la página del administrador que indica si ha habido algún fallo y mediante el resto de las funcionalidades que requieren de base de datos como **web_requester**, **página de trabajador**, **página de administrador** o **página de inicio de sesión**.

5.3.2. Conexión Web-ROS2

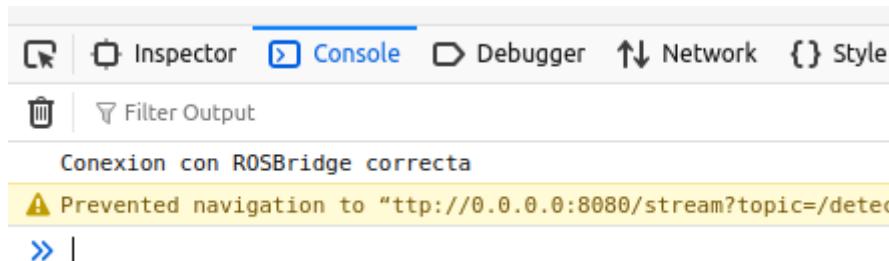
La conexión entre ROS2 y el servidor web es en ambas direcciones usando 2 métodos distintos.

Entre ROS2 y web se conectan topics mediante **RosBridge**, esto nos permite tanto leer como escribir topics desde la web para que el robot con ROS2 pueda leerlos, esta funcionalidad la usan elementos de nuestra web como el mapa en la página del trabajador.

Además de **Rosbridge** también usamos llamadas a la api desde ROS2, usamos la librería de Python **requests** para mandar solicitudes http al servidor y esperar su respuesta. Lo usamos de la siguiente manera:

En el nodo **web_requester** pedimos cada 10 segundos que se haga una petición GET a la api del servidor que le devuelve al nodo una lista de todas las entregas sin realizar para que el robot decida cual hacer, una vez ha hecho una, vuelve a hacer una solicitud POST donde modifica el estado de esa entrega como que ya está hecha.

Se verifica su funcionamiento por los dos siguientes imágenes donde se muestra el mensaje en consola donde se confirma que la conexión ROSBridge es correcta y en la segunda captura donde se puede comprobar que se muestra en el mapa la posición del robot, esto no sería posible si la conexión no funcionara;



The screenshot shows the developer tools console tab in a browser. The tabs at the top are Inspector, Console (which is selected), Debugger, Network, and Style. Below the tabs are two buttons: a trash can icon and a filter icon labeled 'Filter Output'. The main area of the console displays the following text:

```
Conexion con ROSBridge correcta
⚠ Prevented navigation to "http://0.0.0.0:8080/stream?topic=/deteccion"
» |
```

Figura 5.3.2.1: Capturas de la consola del navegador

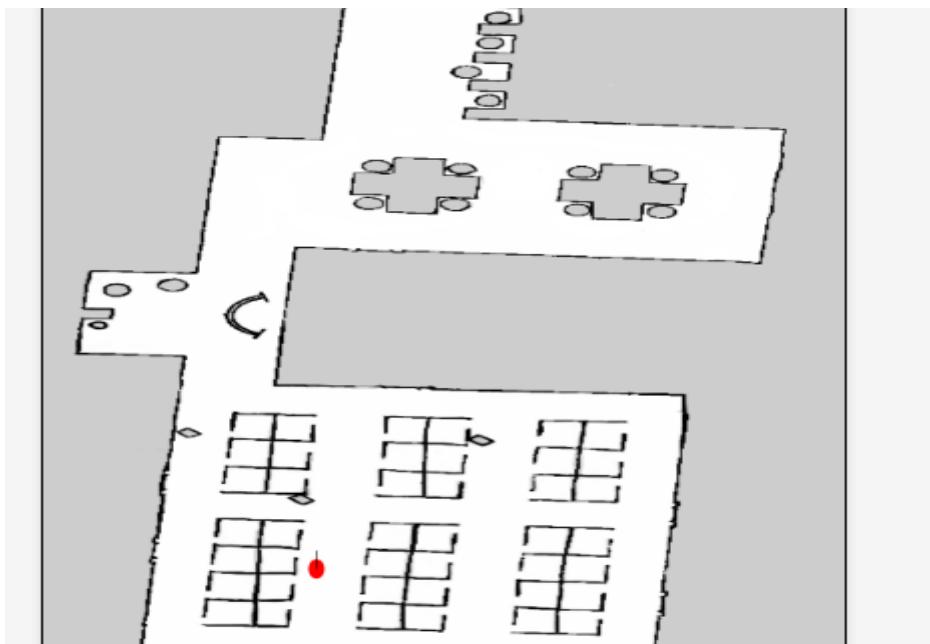


Figura 5.3.2.2: Captura del mapa de la página del trabajador

5.3.3. Página del trabajador

La página del trabajador tiene tres funcionalidades principales:

Primero, tiene una lista de todos los trabajadores de la oficina como botones, algunos están activos y otros no, aquellos activos significan que el trabajador se encuentra en la oficina y al poder clicar el botón se le manda una solicitud a la base de datos para añadir una entrega siendo el emisor tú mismo, es decir el trabajador de la sesión iniciada, y el receptor el trabajador al que has clicado en la lista de botones. Así la próxima vez que el robot lea las entregas sin hacer de la base de datos podrá leer la entrega que has creado vendrá a tu puesto para recoger el paquete de la entrega.

Segundo, la página cuenta con un botón que dice “Ver vista del robot”, al no poder mostrar la imagen de la detección en la propia página, con este botón redirigimos al usuario a una ventana nueva donde únicamente se ve la cámara del robot con la detección.

Por último, la página cuenta con un pequeño mapa que muestra en tiempo real la posición del robot en la oficina para saber en todo momento por donde está.

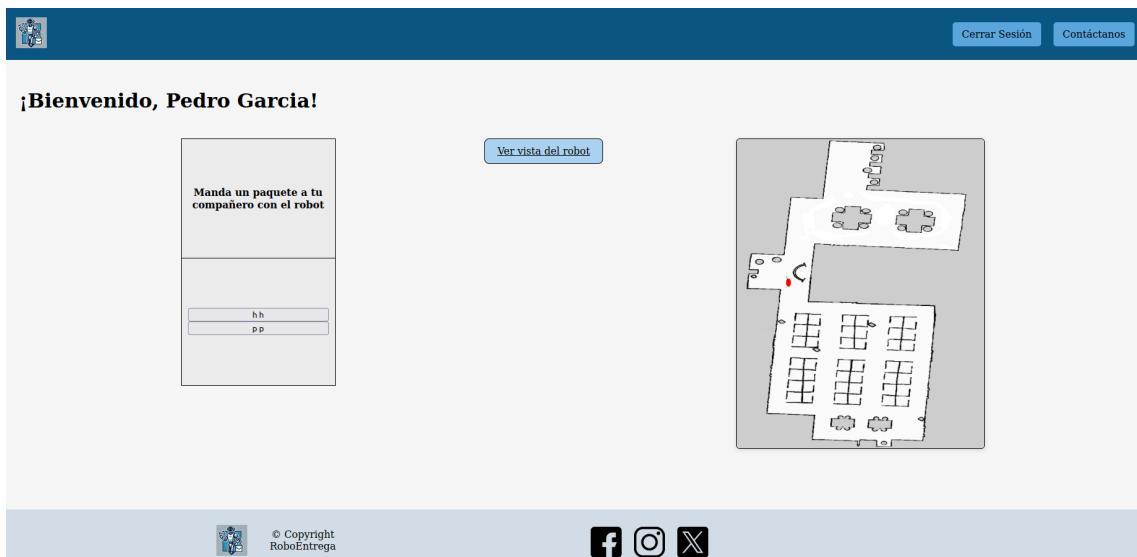


Figura 5.3.3.1: Capturas del navegador en la página del trabajador

5.3.4. Página del administrador

A la página del administrador se accede de la misma forma en la que se accede a la página del trabajador, mediante la página de inicio de sesión.

El propósito de esta página es que el administrador pueda gestionar la base de datos, contiene tablas para poder añadir, editar y eliminar entradas en las tablas de robot, trabajador y mesa. Además, cuenta con un cuadro inspirado en un terminal de consola cmd de un ordenador donde se muestran los mensajes de la base de datos en respuesta a las acciones realizadas en esta página con alguna de las funciones de las tablas ya comentadas y también muestra mensajes de estado del robot, éste recibirá mensajes cuando el robot haya decidido ir a hacer una entrega, o haya encontrado una caja por el camino, etc.

En la siguiente figura se muestra el resultado final de la página del administrador y sus funciones (los datos mostrados en las tablas e inputs son aleatorios y no datos reales):

Panel de Administración

¡Bienvenido, admin!

Robots

Modelo	Número de serie	Añadir Robot
asd	asd	

Trabajadores

DNI	Nombre	Apellido	Email	Mesa	Acciones
00000001A	h	h	h@gmail.com	5	Cabina
00000002A	p	p	p@gmail.com	6	Cabina
00000000A	admin	admin	admin@gmail.com	0	Cabina

Cabinas

ID	Longitud	Latitud	Añadir Cabina
1	12.000000	12.000000	
2	13.000000	13.000000	
3	14.000000	14.000000	
4	15.000000	15.000000	
5	13.000000	-7.000000	
6	9.000000	-11.000000	

Actividad

Horas: 19:58:28
 Tipo: documento
 Estado: Pendiente
 Origen: 00000001A
 Destino: 00000002A
 Robot ID: 1

[4:58:03 PM] He terminado la entrega

© Copyright RoboEntrega

Figura 5.3.4.1: Capturas del navegador en la página del administrador

Trabajadores

DNI	Nombre	Apellido	Correo electrónico	Contraseña	Mesa - 1	Acciones
1	asd	asd		asd		■

Cabinas

ID	Longitud	Latitud	Añadir Cabina
1	12.000000	12.000000	■

Figura 5.3.4.2: Captura de éxito en crear usuario desde admin

Trabajadores

DNI	Nombre	Apellido	Email	Mesa	Acciones
00000001A	h	h	DNI: 12345678B	5	■ ■ Cabina
00000002A	p	p	Nombre: Pedro	6	■ ■ Cabina
00000000A	admin	admin	Apellido: Garcia	0	■ ■ Cabina
12345678B	Pedro	Garcia	pedro@gmail.com	1	■ ■ Cabina

Cabinas

ID	Longitud	Latitud	Añadir Cabina
1	12.000000	12.000000	■
2	13.000000	13.000000	■

Figura 5.3.4.3: Captura de edición de datos de usuario desde admin

5.4. Caja física

El sistema de recogida está diseñado para facilitar la recolección y el almacenamiento de paquetes o cartas mientras el robot está en funcionamiento. El sistema consta de dos partes principales:

1. Caja de almacenamiento:

Una caja situada en la estructura del robot sirve como compartimento de almacenamiento temporal de los

paquetes recogidos. Está diseñada de forma modular, permitiendo su extracción rápida y sencilla en caso de mantenimiento o sustitución del robot.

2. **Recolector motorizado:**

El mecanismo de recogida está compuesto por un motor controlado por una placa Arduino. Al accionar un botón, el motor activa el sistema de recolección, sea tanto subir el recogedor para poder obtener los paquetes o entregarlos, como bajar los paquetes a la caja .

Este módulo también ha sido diseñado con un enfoque de fácil instalación y desmontaje. En caso de fallo o necesidad de reparación, el recolector puede ser retirado sin necesidad de herramientas especiales, facilitando así la manipulación y el mantenimiento general del robot.

La siguiente imagen es un frame de un video de demostración

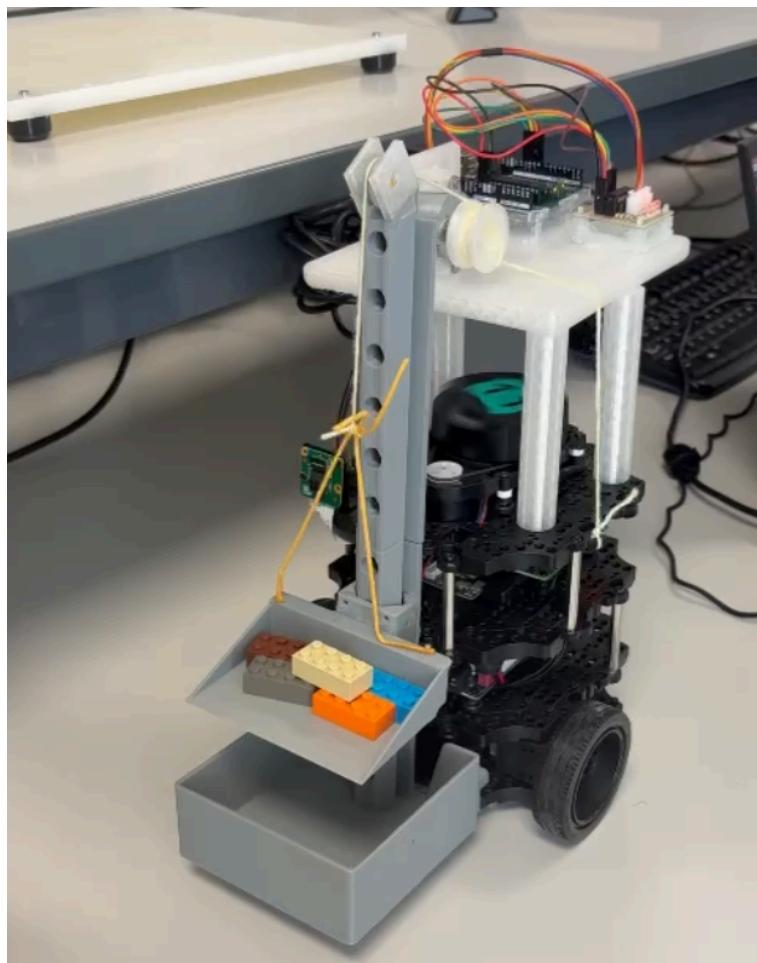


Figura 5.4.1: Frame del video de demostracion del motor fisico

6. Gestión del equipo

Somos el Equipo1 y nuestro grupo se llama RoboEntrega, aunque todos los miembros hemos trabajado en mayor o menor medida en casi todos los aspectos de este proyecto, cada uno ha tenido un papel predominante que hemos llamado de esta forma: (La descripción de cada papel es meramente orientativa, ya que no muestra fielmente el trabajo de cada miembro porque todos hemos trabajado en todo)

Diego Baldoví, **el administrador**; lleva las reuniones y gestiona la documentación y el flujo del proyecto.

Endika Matute, **el diseñador**; diseña el apartado visual y diagrama de flujo del servidor web.

Hao Xu, **el técnico backend**; diseña y hace el código de la api y la base de datos del servidor web.

Jaime Ferrer, **el técnico de imagen**; entrena los modelos de detección de objetos para ROS2.

Alejandro Roca, **el técnico de ROS2**; crea y hace el código de nodos importantes de ROS2.

Jordan Phillips, **el técnico en impresión 3d**; diseña e imprime las piezas necesarias para el robot.

Hemos seguido una organización basada en SCRUM, donde la entrega del proyecto se dividía en 4 entregas llamadas sprints y en cada sprint nos proponemos un aspecto a implementar:

Sprint 1: Navegación

Sprint 2: Web

Sprint 3: Visión artificial y procesado de imagen

Sprint 4: IA

Para organizarnos durante cada sprint utilizamos un [tablero de Trello](#)

Hemos trabajado en este repositorio [ProyectoRobotica_RoboEntrega](#) donde hemos gestionado el git de la siguiente manera tal como también se muestra en la figura 6.1::

Partimos de una rama main que solo contendrá el código definitivo de cada sprint, es decir el código que se presentará en cada sprint review. De esta rama main se creará una rama releaseXX donde XX es el número del sprint, por ejemplo 01 o

03. Por último, a partir de esta rama release se crearán tantas ramas como tareas hallan en el tablero de Trello, llamándose cada una QT-HY-ZZ donde Y representa el número de historia de usuario a la que pertenece la tarea, y ZZ el número de tarea designado en trello, esta rama es la única en la que deberían haber commits.

Una vez se ha terminado la tarea y se ha verificado, la rama se cierra dentro de release y una vez todas las tareas están terminadas y verificadas se hace una reunión con todos los miembros del equipo para asegurarnos de que todas las ramas se han cerrado correctamente y que el proyecto funciona con todos los objetivos de las tareas terminadas, si hubiera algún fallo, se arreglaría y se haría commit en la rama release (este caso es extraordinaria ya que se supone que se ha terminado y verificado en la rama de la tarea). Cuando se ha verificado que el código funciona la rama release se cierra en main y al llegar a la presentación del sprint review se vuelve a testear todo, de haber algún fallo o mejora de último momento se haría un commit con los cambios en la rama main asegurándose de que funciona totalmente y no inhabilita ninguna funcionalidad ya verificada.

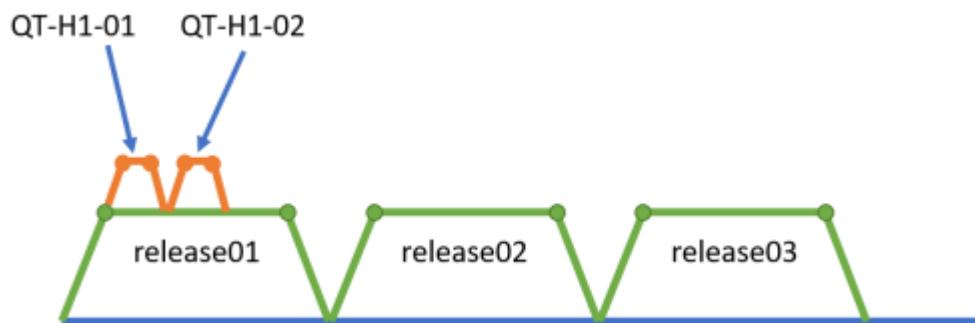


Figura 6.1: Diagrama de representación de los commits y ramas en git

Hemos seguido una organización basada en SCRUM, donde la entrega del proyecto se dividía en 4 entregas llamadas sprints

7. Reflexiones del equipo

A lo largo del desarrollo del proyecto, el equipo ha aprendido sobre diversas áreas tecnológicas. Este trabajo nos ha permitido enfrentarnos a varios retos relacionados con la integración de sistemas complejos, la programación de robots autónomos y la gestión del equipo para obtener unos resultados favorables.

Algo que creemos importante a destacar es que es fundamental una buena organización desde un inicio para poder llevar cada tarea a cabo sin complicaciones, ya que en este tipo de proyectos es frecuente tener problemas que ocasionan desajustes en los plazos principalmente asignados a cada tarea. Algunas herramientas que han sido utilizadas para mantener al equipo en contacto y bien organizado han sido Trello y Discord. También hemos tenido en cuenta la importancia de una documentación constante y el uso de control de versiones como base para un trabajo fluido.

A nivel más técnico, nos hemos familiarizado con entornos de simulación avanzados como Gazebo y RViz2, así como con herramientas de inteligencia artificial y diseño -D, lo que nos ha aportado una base del proceso de desarrollo de un sistema robótico funcional.

8. Conclusiones y propuestas de mejora

En conclusión, el proyecto ha sido una experiencia enriquecedora y hemos aprendido sobre el funcionamiento de nuevas tecnologías, pero somos conscientes de que no hemos podido cumplir todos los objetivos que nos planteamos en un principio y aún tenemos margen de mejora en unos cuantos aspectos relacionados con el proyecto. Sin embargo, hemos logrado un sistema funcional aunque hayan quedado aspectos por pulir o desarrollar con mayor profundidad. Con todo esto en cuenta, destacamos algunas propuestas de mejora para futuros proyectos similares, ya sea en la carrera o en nuestra vida profesional:

Mejor reparto de las tareas según perfiles y carga real de trabajo: En ciertos momentos, la carga de trabajo no estuvo bien distribuida entre los miembros del equipo. Sería beneficioso establecer mecanismos más precisos de seguimiento de contribuciones y ajustes en las responsabilidades de cada miembro.

Más trabajo en equipo: A lo largo del proyecto cada miembro se ha centrado en sus propias tareas y cuando más hemos colaborado entre nosotros ha sido para juntar todo y para ayudarnos en algunas tareas. Algo a mejorar sería llevar un seguimiento mayor de lo que está haciendo cada miembro para aportar conocimientos o ayuda si se atasca en algún punto y así que todos los compañeros puedan avanzar en sus responsabilidades de igual forma.

Priorización de tareas principales: En algunos momentos del proyecto se dedicó tiempo a funcionalidades secundarias que resultaban más atractivas o interesantes, dejando en segundo plano tareas clave que debían haberse abordado con mayor antelación. Esto provocó cuellos de botella en fases finales del sprint y una acumulación de carga de trabajo que podría haberse evitado con una priorización más estratégica desde el inicio.

9. Programas y servicios utilizados

Hemos usado todos estos programas y webs para el desarrollo de este proyecto:

- **Oracle VirtualBox:** Para simular en nuestros ordenadores una máquina Linux Ubuntu
- **Visual Studio Code:** Para escribir prácticamente todo el código del proyecto
- **GitKraken:** Para utilizar git desde Ubuntu y controlar más visualmente los commits y las ramas
- **Blender:** Para diseñar en 3d las piezas físicas del robot
- **MySQL Workshop:** Para diseñar la base de datos
- **Arduino:** Para hacer el código del motor de la pala
- **Drawio.com:** Para realizar los diagramas de flujo y de nodos
- **Axure:** Para crear los diseños de la página web
- **Excel:** Para llenar los diagramas de burndown en cada sprint
- **Word:** Para crear los documentos de presentación y actas y la memoria final del proyecto
- **Photopea.com:** Para editar las imágenes .pgm del mapa y corregir imperfecciones y detalles
- **Canva:** Para crear las portadas de los documentos
- **Gazebo:** Para simular un espacio donde testear el robot
- **Rviz2:** Para visualizar el estado y características del robot durante la ejecución
- **Firefox:** Para acceder a la web dentro de la máquina virtual
- **DB Browser (SQLite):** Para comprobar el estado de la base de datos sin usar código ni funciones extra que complicarían la consulta.
- **ChatGPT y DeepSeek:** Para resolver dudas rápidas o ayudar a solucionar problemas en el código.
- **Discord:** Para hacer las reuniones y compartir notas y código importantes.

- **WhatsApp:** Para comunicarnos de manera fácil y sencilla unos con otros.
 - **Trello:** Para gestionar el flujo de trabajo y mantener registro de las tareas hechas y por hacer y sus estados.
 - **Roboflow:** Para los datasets de entrenamiento del modelo YOLO
-

10. Historial de contribuciones al documento

Versión	Autor	Descripción	Fecha
V1.0	Diego Baldoví	Versión inicial del documento, Estructura básica	25/05/2025
V1.1	Diego Baldoví	Puntos de concepción de la idea y descripción y funcionalidades básicas	27/05/2025
V1.2	Diego Baldoví	Diseño diagrama de nodos	28/05/2025
V1.3	Diego Baldoví	Programas y servicios usados	30/05/2025
V1.4.1	Hao Xu	Diseño, Implementación y verificación nav2 system y base de datos	30/05/2025
V1.4.2	Jaime Ferrer	Implementación y verificación detección imágenes cajas y manchas	31/05/2025
V1.4.3	Alex Roca	Implementación y verificación detección personas y caras y my_waypoint_follower	31/05/2025
V1.4.4	Endika Matute	Portada, Diseño, Implementación y verificación página del trabajador, admin, index e inicio sesión	01/06/2025
V1.4.5	Jordan Phillips	Diseño, Implementación y verificación del motor y piezas físicas	01/06/2025
V1.5	Diego Baldoví	Gestión de equipo	02/06/2025
V1.6	Diego Baldoví	Reflexiones de equipo, conclusiones y propuestas de mejora	02/06/2025

