UNIVERSITAT DE BARCELONA

Trabajo de Fin de Grado

GRADO EN INGENIERIA INFORMÀTICA

Facultad de Matemáticas e informática
Universidad de Barcelona

Simulación de un UAV en un entorno avanzado

Sergio Ferrando Gracia

Director: Eloi Puertas

Realizado en: Departamento de matemáticas e informática

Barcelona, 13 de Junio de 2023

Contenido

1. Introducción

Un cuadricóptero, comúnmente llamado dron, es un vehículo aéreo no tripulado (UAV) que puede ser controlado de forma remota, ya sea con un control a remoto o bien de forma autónoma, para seguir una ruta determinada. Normalmente estos dispositivos están dotados de muchos sensores, cámaras y otros instrumentos que le permiten realizar diferentes tareas de forma más sencilla.

Existen diferentes modelos de dron, desde pequeños modelos recreativos hasta algunos de mayor envergadura utilizados para tareas militares, comerciales o de investigación.

Los drones tienen muchísima versatilidad cosa que hace que sean útiles para gran variedad de campos, como la fotografía, las inspecciones de infraestructuras, cartografía, agricultura, búsqueda y rescate, vigilancia, entrega de paquetes, entre otros. En la *ilustración 1* se pueden ver dos claros ejemplos de utilización de drones en diferentes campos. A la izquierda un cuadricóptero utilizado para la agricultura, dicho dron es de la marca DJI y es utilizado para fumigar los campos de forma más eficiente. Por otro lado, en la imagen de la derecha de la *ilustración 1,* nos encontramos con un dron utilizado para fines recreativos, dichos drones suelen utilizarse también para la fotografía o filmografía debido a su gran calidad de la cámara.

Ilustración 1. Izquierda: dron agricultor (DJI Agriculture) Derecha: dron de uso recreativo (DJI Pahntom).

Fuente: Acre_admin (2022)

Hoy en día estos dispositivos están ganando mucha popularidad ya que son muy versátiles y que tienen la capacidad de acceder a lugares de difíciles accesos o

peligrosos para los seres humanos. Otro factor muy importante que les hace ganar popularidad es que actualmente las tecnologías de la Inteligencia artificial y la robótica están en auge. Cosa que hace que cada vez más cuadricópteros estén dotados de dichas tecnologías para ser utilizados de forma autónoma o bien para facilitar su uso a los humanos.

1.1. Motivación del Proyecto.

Los drones, una tecnología en auge, han despertado mi interés desde hace mucho tiempo. A pesar de tener diversos modelos de drones desde que era pequeño, nunca me había detenido a indagar sobre los procesos internos y el funcionamiento de estos, esto despertó mi curiosidad y se convirtió en la principal motivación para elegir este tema como mi Trabajo de Fin de Grado.

Mi objetivo personal era comprender cómo funcionan los drones y como programarlos. Siempre me preguntaba cómo eran capaces de realizar las funciones que hacen. Decidí sumergirme en este proyecto para adquirir un conocimiento profundo y poder simular drones, lo que me permitiría probar y desarrollar algoritmos sin incurrir en grandes gastos ni correr el riesgo de dañar un dron real.

Para ello se decidió orientar este Trabajo de Fin de grado en entender el funcionamiento el software de navegación, control y visión dentro de un UAV i aplicar dichos funcionamientos en una simulación utilizando un software, libre y disponible para todos, para en un futuro ser capaces de llevar estas simulaciones al mundo físico de la forma más sencilla y realista posible. Todo esto acompañado de un aprendizaje de las herramientas típicas utilizadas en la robótica, para así poder aprender más sobre la robótica y utilizar dichas herramientas para programar la simulación.

Además, este proyecto también representa un desafío académico significativo. Me entusiasmaba la oportunidad de integrar y aplicar los conocimientos adquiridos en diversas asignaturas a lo largo de mi carrera. Este Trabajo de Fin

de Grado me permitió unir conceptos de robótica, software distribuido e ingeniería de software, creando un enfoque interdisciplinario y demostrando una comprensión más profunda de los temas estudiados. Estas motivaciones me han permitido adquirir conocimientos técnicos, aplicar conceptos aprendidos y obtener una experiencia enriquecedora que me será útil tanto en mi desarrollo personal como profesional.

1.2. Objetivos

Para la realización de este trabajo se han dividido las diferentes tareas a realizar en función de su prioridad y en base a sus dificultades, poniendo las más dificultosas al principio juntamente con las triviales. En el siguiente aparatado se explicará cuáles han sido estos objetivos que se han ido siguiendo y su prioridad.

Dentro de los objetivos podríamos decir que se separan en función a su importancia, cada uno explicado a continuación.

1.2.1. Objetivos globales

Como objetivos cruciales, definiríamos los objetivos que sin ellos no se puede realizar ningún tipo de avance en los demás objetivos. Estos serían aquellos que acogen la base de nuestro trabajo y sobre los cuales trabajaremos para mejorarlo y ampliarlo.

Estos objetivos serían principalmente los de conseguir una simulación de un dron usando Gazebo, Ros y Rviz.

Seguidamente de esto, el siguiente objetivo sería añadirle al dron los diferentes sensores y la cámara. Una vez incorporados al modelo del dron, comprobar que funcionen correctamente y tomen medidas coherentes con lo que sucede en el entorno simulado.

El ultimo objetivo crucial es la elaboración de un código que sea capaz de hacer que le dron se mueva por el entorno simulado. En este objetivo decidí que para

poder manejar el dron se utilizarían entradas de teclado de tal forma que se podría mover el dron por el entorno simulado en tiempo real.

Una vez finalizados estos objetivos que serían los pilares del trabajo ya se podrían definir más objetivos con tal de ampliarlo y mejorar los códigos ya implementados.

1.2.2. Objetivos específicos

Una vez acabados los objetivos esenciales para este trabajo, se definieron una serie de objetivos que implementaran funcionalidades específicas de la programación con drones. Estos objetivos principalmente fueron los siguientes:

- Utilizar la cámara del dron para detección de objetos utilizando el algoritmo YOLO.
- Crear una Web capaz de mostrar la información de los diferentes sensores en tiempo real.
- Realización de un mapa del entorno del dron.
- Auto navegación del dron utilizando un mapa del entorno.
- Análisis de los costes de la implementación real de un dron aplicando los conceptos del trabajo.

1.3. Contenido de la memoria

En esta memoria se explicarán todos los procedimientos que se han seguido para la elaboración de este trabajo. Se detallarán los procedimientos utilizados, la planificación utilizada y las técnicas necesarias para la elaboración de los diferentes objetivos. Para esto primero se introducirán los conceptos claves de este proyecto de forma conjunta con las diferentes aplicaciones y sistemas utilizados.

Una vez introducidos estos conceptos se pasará a detallar las diferentes fases del desarrollo del trabajo junto con las técnicas utilizadas y diferentes diagramas para entender a la perfección la evolución del trabajo.

Al acabar la fase de desarrollo se explicarán las diferentes pruebas que se han realizado para validar que los objetivos funcionaban de forma correcta y se mostraran unas breves demostraciones de estos.

Finalmente, una vez acabada la fase de implementación de códigos y de pruebas, se pasó a analizar los diferentes componentes que harían falta para llevar este proyecto simulado a la vida real, para ello primero se detallaran los componentes escogidos y después un desglose con el coste que esto tendría.

2. Planificación

Para la planificación de este Trabajo de Fin de Grado sobre la simulación de drones con Gazebo y ROS he seguido una estructura organizada que permitirá garantizar un desarrollo sistemático de las diferentes funcionalidades y así poder ser más eficiente a la hora de trabajar. A continuación, se detalla el enfoque utilizado en la planificación del proyecto:

1. **Definición del alcance del proyecto:** En primer lugar, se realizó una definición clara del alcance del trabajo. Se identificaron y establecieron las tareas necesarias para llevar a cabo el proyecto, considerando su relevancia y dificultad. Esto se logró a través de una investigación exhaustiva que permitió comprender cómo se podrían abordar los diferentes objetivos del trabajo.

2. **Separación de objetivos:** Los objetivos se dividieron en dos grupos principales, los objetivos primordiales y los objetivos secundarios. Los objetivos primordiales estaban directamente relacionados con el desarrollo de la simulación del dron en un entorno virtual y su interacción con dicho entorno. Estos objetivos se consideraron fundamentales para el proyecto. Los objetivos secundarios, por otro lado, abordaron funcionalidades adicionales y mejoras complementarias.

3. **Definición de criterios de aceptación:** Se establecieron criterios de aceptación para cada objetivo, es decir, se definieron los requisitos mínimos que debían cumplir estos objetivos para considerar que se habían completado con éxito. Estos criterios garantizaban que las tareas fueran funcionales en su totalidad y que permitieran tener una forma de verificar que el objetivo se había implementado con éxito.

4. **Priorización y orden de desarrollo de los objetivos:** Los objetivos se ordenaron según su importancia y dificultad en función del impacto que tendrían en el proyecto. Se otorgó prioridad a los objetivos primordiales, ya que constituían la base del trabajo. Los objetivos secundarios se

planificaron en función de su relevancia y del tiempo estimado para su realización.

5. **Implementación progresiva de los objetivos:** Se inició la implementación de los objetivos primordiales, asegurando que cada uno cumpliera con los criterios de aceptación establecidos. Una vez acabada la base del proyecto se pasó a implementar los demás objetivos. En caso de encontrar dificultades o retrasos significativos en la realización de algún objetivo en concreto, se pasaba a trabajar en otros con el fin de avanzar en el proyecto de manera eficiente y evitar bloqueos.

6. **Documentación y redacción del informe:** Durante todo el proceso, se llevó un registro detallado de los avances, decisiones tomadas y resultados obtenidos. Esto permitió tener una documentación básica del trabajo realizado. Al finalizar los objetivos, se redactó este informe técnico que describe en detalle el desarrollo del proyecto, incluyendo la metodología empleada, los resultados obtenidos y las conclusiones alcanzadas.

Finalmente, se ha hecho un diagrama de Gantt, tal y como se puede apreciar en la ilustración 2, con el tiempo que se le ha dedicado a cada objetivo.

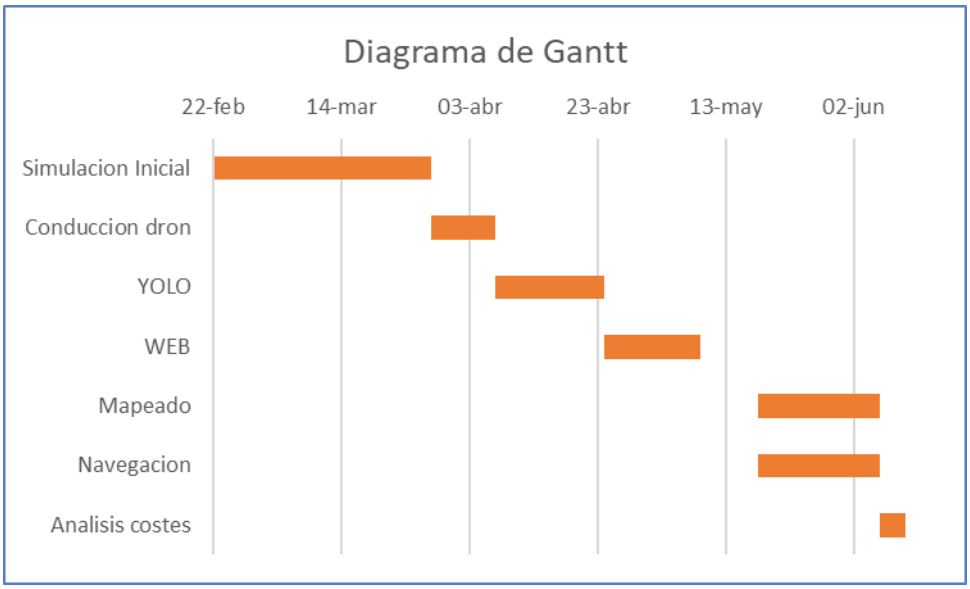

Ilustración 2. Diagrama de Gantt del proyecto.

Fuente: Elaboración propia.

3. Arquitecturas robóticas

Una arquitectura robótica es el conjunto de elementos que integran de forma genérica los robots. En el caso de nuestro proyecto esta estructura estaría simulada, ya que sería la forma de hacer pruebas sin asumir grandes costes de materiales y tener una idea aproximada de como funcionaria en un entorno real.

En la mayoría de los robots estas arquitecturas suelen seguir el mismo modelo base. Este modelo consta de un módulo de control, un módulo de procesado y los sensores y actuadores. El módulo controlador es el encargado de leer la información de los sensores y de controlar los actuadores. Para ello, envía la información de los sensores al módulo procesador que es el encargado de recolectar dichos datos y decir que acciones ejecutar para afrontar el objetivo deseado. Una vez procesada la información, el módulo de procesamiento envía las acciones a realizar al módulo controlador, este último es encargado de hacer llegar dicha información a los actuadores para que estos puedan ejecutar las acciones escogidas por el módulo procesador.

Dentro de los sensores, nos podemos topar con diferentes tipos y cada uno de ellos con una funcionalidad especifica a la hora de recabar información del entorno. Un ejemplo de estos sensores pueden ser cámaras, sensores de la presión, sensores de proximidad, sensores giroscópicos, entre otros. Estos tienen la función de recabar la máxima información posible para así hacer que las tareas se realicen de la mejor forma posible.

Por otro lado, dentro de los actuadores, están todos aquellos elementos que son cruciales para realizar movimientos de los robots. Un ejemplo de actuador podría ser el un motor que mueva una rueda, altavoces que reproduzcan sonidos, luces, entre otros. Estos son los encargados de ejecutar las operaciones que el módulo procesador dicte.

El módulo controlador, es el encargado de comunicarse directamente con los sensores y con los actuadores, ya sea para enviar información como para recibirla. Este módulo en los UAV se le llama controlador de vuelo.

Por último, el módulo procesador, el más importante en esta arquitectura, ya que como hemos dicho antes es el que se encarga de la toma de decisiones y el procesamiento de los datos. Dicho modulo en los robots programados con ROS, corre un Sistema Operativo con ROS. Este Sistema Operativo es el encargado de ejecutar los diferentes algoritmos creados por el usuario para realizar las tareas que se deseen. Estos algoritmos harán uso de la información que reciben de los sensores para procesarla y tomar la decisión a realizar por los actuadores. Dentro de este sistema operativo también se podrían ejecutar otros programas que ayuden a la hora de saber el estado del robot en cada momento, como podría ser Rviz. En la *ilustración 3* se puede apreciar un esquema genérico con los componentes mencionados con anterioridad.

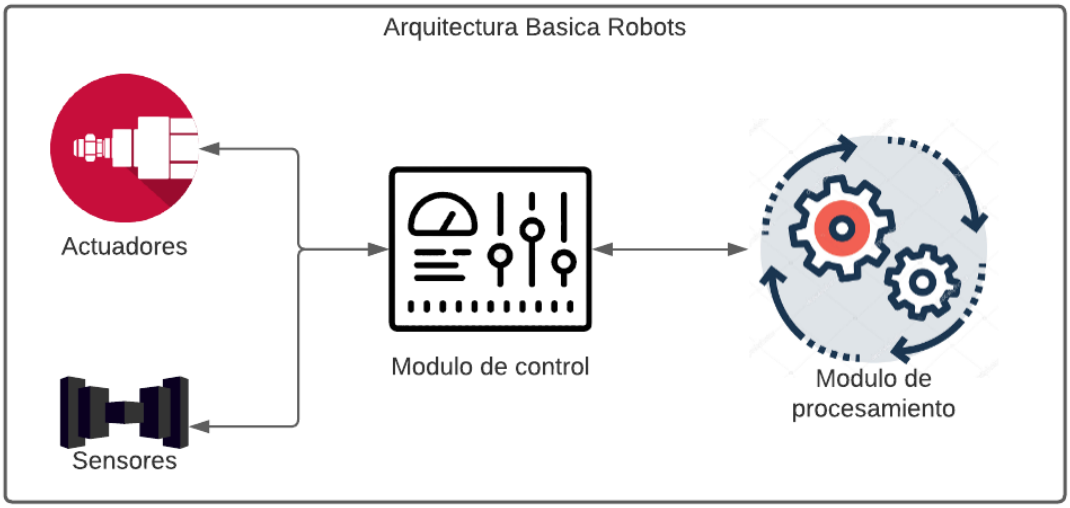

Ilustración 3. Diagrama arquitectura básica de hardware de un dron.

Fuente: Elaboración propia.

En el caso de nuestro trabajo al no existir estos módulos físicos mencionados anteriormente lo que se hará es simularlos, para ello se utilizara Gazebo, que es una herramienta de simulación de entornos físicos y entornos robóticos. En el siguiente aparato se comentarán al detalle las diferentes tecnologías utilizadas para la elaboración del proyecto más en detalle.

3.1. Tecnologías utilizadas para la simulación

Estas tecnologías que se han utilizado a la hora de hacer el trabajo simularían la arquitectura robótica comentada anteriormente. Esta arquitectura robótica se simularía utilizando las siguientes herramientas.

3.1.1. Linux/Docker

Para la realización de este trabajo se ha tenido que utilizar Ubuntu como sistema operativo, ya que tanto ROS como Gazebo únicamente existen para dispositivos Linux. Como mucha gente no dispone de un subsistema Linux instalado en su ordenador se propone como alternativa el uso de un subsistema Linux dentro de Windows utilizando WSL. Durante la realización del trabajo se fue viendo que a medida la complejidad del código iba augmentando, utilizando la simulación se necesitaban más recursos del ordenador para realizar las distintas operaciones, por lo tanto, se optó por instalar Linux nativo en el ordenador. Tanto en una opción como en la otra se utilizará la versión 20.04 (Focal), que es la última versión de Ubuntu que admite ROS-Noetic.

Tanto utilizando el sistema instalado como nativo o un subsistema Linux el código funcionará, la única diferencia es la velocidad de compilación y el realismo de la simulación, que en este caso será mejor utilizando una instalación nativa de Ubuntu. Ambas opciones estarán detalladas más adelante y se explicara como poder utilizar cualquiera de las dos opciones mencionadas.

Este sistema en un robot real se ejecutaría sobre el módulo controlador y es el que se encargaría del procesamiento de los datos y el posterior envió de información a los actuadores.

3.1.2. ROS

ROS, también conocido como Robot Operating System, es un sistema operativo de código abierto utilizado para robots. Dicho sistema está dotado de librerías y herramientas que nos facilitaran el desarrollo de nuestro robot. ROS nos dota de

una estructura distribuida y modular que permite crear grandes sistemas para robots, está diseñado para ser flexible, reutilizable y escalable. Una de sus ventajas es su comunidad que al ser de código abierto hay muchos recursos disponibles sobre este. Este sistema operativo robótico no ha parado de evolucionar de forma progresiva y actualmente cuenta con su versión ROS 2. Para la realización de este trabajo utilizaremos la versión de ROS Noetic ya que es una versión estable para Ubuntu 20.04 (Focal).

Previamente a la utilización de ROS se ha hecho una investigación de las funciones básicas de dicho software y a cómo utilizarlas. Para ello es necesario saber que ROS se organiza siguiendo un esquema de paquetes, dichos paquetes contienen todo los ejecutables, bibliotecas, conjuntos de datos, archivos de configuración, o cualquier cosa que sea útil de forma conjunta.

A los ejecutables mencionados con anterioridad dentro de la nomenclatura de ROS se les llama nodos, ROS internamente funciona a través de ellos. Estos nodos se pueden comunicar entre si gracias a los llamados tópicos (*tópicos*), para ello los nodos se subscriben o publican en dicho tópico de información. En el caso de que quieran publicar información en un tópico, para que uno o diversos nodos lo lean a posteriori, actúan como *publishers* y en caso de que quieran leer información que le vaya llegando al tópico se llaman *subscribers*. Esto permite compartir la información entre diversos nodos y que ambos funcionen de forma independiente.

Para que esto sea posible tiene que existir un nodo maestro que es el que controla todos los demás, que es el encargado de iniciar los nodos con unos ciertos parámetros definidos. En la *ilustración 4* se puede apreciar un esquema genérico de un subsistema ROS y cómo funciona internamente.

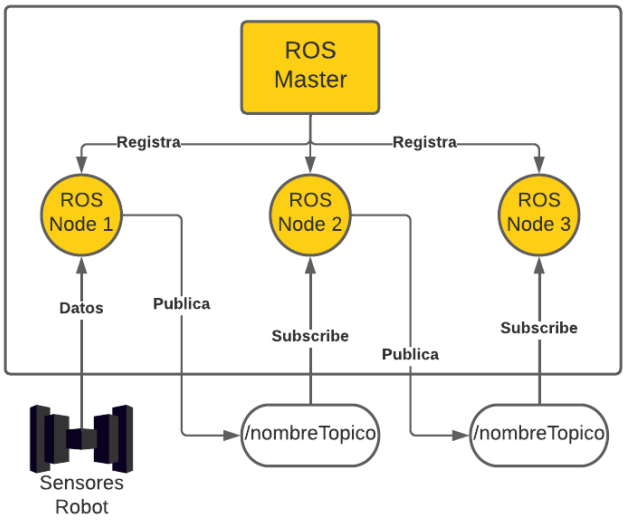

Ilustración 4. Arquitectura software ROS.

Fuente: Elaboración propia.

Este sistema operativo robótico se ejecutaría en el módulo controlador, en concreto en el sistema Ubuntu mencionado en el apartado anterior. Una de las ventajas que tienes ROS es que define a la perfección la arquitectura robótica, de esta forma los sensores serian nodos y los actuadores igual, dichos nodos en el caso de los sensores publicarían información de la información que perciben y en caso de los actuadores esperarían a que el módulo de procesamiento los mandara acciones a realizar.

3.1.3. Gazebo

Gazebo es un software que se encarga de hacer simulaciones de entornos reales en 3D utilizado en sistemas autónomos y diseño de robótica. En parte está programado a base de paquetes de ROS cosa que hace que sea más fácil utilizarlos conjuntamente. Por otra parte, ROS, vera la ejecución del Gazebo como un nodo más dentro del sistema, con esto facilitara la conexión entre ellos.

Esta herramienta es utilizada para recrear entornos reales con robots de forma precisa, con la posibilidad de añadir gravedad, fricción, colisiones y fuerzas externas. Esto hace que no haga falta que tengas físicamente el robot, sino que puedas hacer pruebas en el entorno simulado y si el resultado es satisfactorio probarlo en la vida real.

También, dicho software es totalmente configurable y personalizable, de esta forma los usuarios pueden hacer uso de el para simular edificios, personas, objetos, entre otros elementos. También permite la simulación de sensores, cámaras, láseres o radares, lo que facilita el desarrollo de nuevas funcionalidades.

Dichas simulaciones pueden ejecutarse en tiempo real, así los desarrolladores pueden ir viendo como interactuará el algoritmo/código en tiempo de ejecución.

Este proyecto estará basado únicamente en la simulación de un dron por motivos económicos y de permisos de vuelo. Debido a esto aprovecharemos el potencial de Gazebo para crear entornos de simulación que se asimilen a la realidad de la mejor manera posible y así probar las diferentes funcionalidades programadas en el dron.

Este software únicamente es utilizado para la simulación física, de esta forma si se implementara el trabajo en la vida real, no haría falta hacer uso de esta aplicación. En la *ilustración 5* se puede ver un ejemplo de simulación haciendo uso de Gazebo.

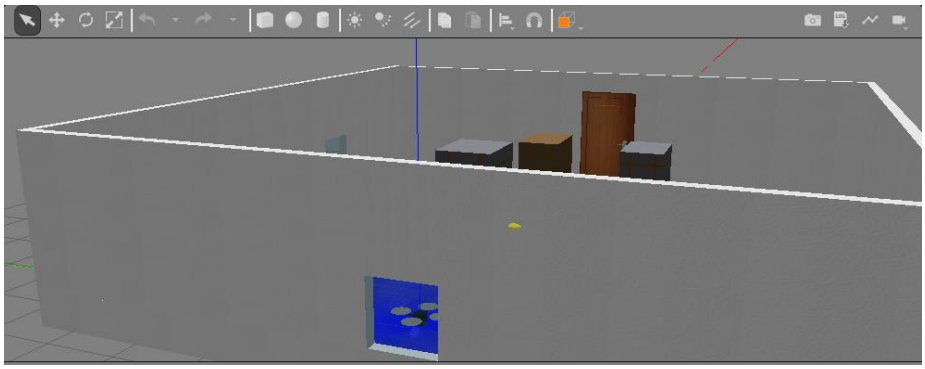

Ilustración 5. Simulación en Gazebo.

Fuente: Elaboración propia.

3.1.4. RVIZ

Rviz es una herramienta de visualización tridimensional que forma parte de ROS. Esta herramienta permite visualizar gráficamente los datos generados por los sensores y actuadores del robot, lo que facilita la comprensión de la información que el robot maneja. Proporciona una representación visual en tiempo real de cómo el robot percibe su entorno y cómo interactúa con él.

La integración de Rviz en el entorno de ROS la convierte en una herramienta muy útil, ya que podemos aprovechar las funciones y bibliotecas disponibles en ROS para desarrollar y probar nuestros códigos de manera más eficiente. Al mostrar una representación visual fiel a la realidad de la visión del robot sobre el entorno, Rviz nos ayuda a depurar y probar nuestros códigos para verificar su correcto funcionamiento.

Al utilizar Rviz, podemos verificar visualmente si nuestros algoritmos se comportan como se espera y si el robot interpreta y responde adecuadamente a los datos de los sensores. Esta capacidad de visualización en tiempo real es valiosa para el desarrollo y análisis de algoritmos.

A diferencia del Gazebo, este programa a la hora de pasar a un modelo físico real se podría utilizar esta herramienta y nos sería muy útil ya que nos permitiría ver en tiempo real el estado de los diferentes nodos y la información que proporcionan. Dicho programa residiría en la unidad de procesamiento.

3.1.5. Modelo simulación UAV (Hector Quadrotor)

Para la realización de este proyecto se valoraron diferentes opciones de modelos de cuadricópteros de entre los que brinda ROS. De entre todos se decidió recurrir a el paquete de ROS llamado “*Hector quadrotor*”, este paquete contiene un modelo de dron genérico en 3D junto con su descripción. Dicho dron no existe en el mercado como tal pero que sí que contiene las características básicas que contienen la mayoría de los drones del mercado.

Este modelo ha sido modificado para añadirle diversos sensores entre ellos uno de rango laser capaz de detectar en un ángulo de 270 grados cualquier objeto u obstáculo, este sensor es muy útil para tareas de mapeo o de detección de obstáculos. Este sensor es de la marca Hokuyo y se trata del modelo “UTM-30LX”, este laser es utilizado como detector de obstáculos y es muy útil a la hora de saber el estado del dron respecto a su entorno. Este modelo también incluye una cámara, un sonar y 4 motores para poder realizar los movimientos. El sonar comentado anteriormente no es de ningún modelo en concreto, muchas placas controladoras de vuelo ya contienen un sonar interno capaz de tomar las medidas de altura del dron, más adelante se verá un ejemplo de ello.

Este paquete también contiene una serie de nodos, que juntamente con alguna librería de ROS, nos van a permitir programar y controlar el vuelo del dron, ajustar diferentes parámetros de vuelo y realizar pruebas de navegación.

Otras de las ventajas es que al ser una descripción genérica de los drones de hoy en día se podrían extrapolar los conceptos y códigos implementados para desarrollar nuestra simulación en un entorno real con un dron existente, por ello conseguiremos tener una implementación que sea parcialmente independiente al hardware que se vaya a utilizar en un futuro. En la *ilustración 6* se pude apreciar el modelo 3D del “Hector Quadrotor” en Gazebo.

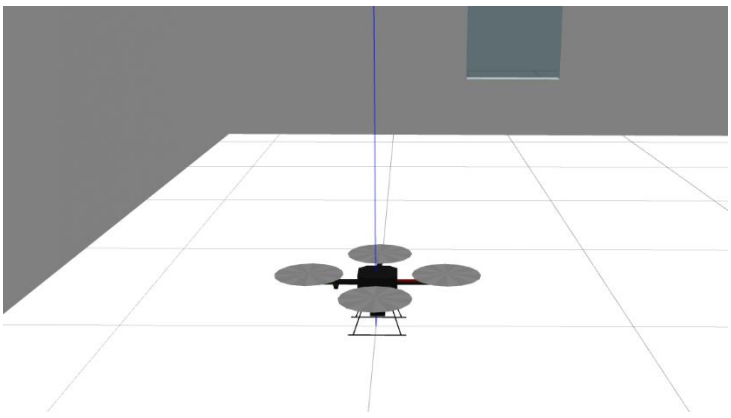

Ilustración 6. Hector Quadrotor en gazebo.

Fuente: Elaboración propia.

3.1.6. YOLO

YOLO (You Only Look Once) es un algoritmo de detección de objetos en imágenes y videos. A diferencia de otros enfoques de detección que dividen la tarea en múltiples etapas, YOLO es un enfoque unificado que realiza la detección y la localización de objetos en una sola pasada.

El entrenamiento de YOLO implica dos etapas principales, el entrenamiento de la red y el ajuste de los pesos.

En la etapa de entrenamiento de la red, se utiliza un conjunto de datos anotados que contiene imágenes etiquetadas con las ubicaciones y clases de los objetos que se desean detectar.

El algoritmo de entrenamiento se basa en una red neuronal convolucional (CNN) que se entrena para aprender características visuales y espaciales de los objetos. Durante el entrenamiento, se ajustan los pesos de la red para que la salida predicha se acerque lo más posible a las anotaciones reales.
Una vez entrenada la red, se utiliza un conjunto de datos diferente al de entrenamiento para poder validar el correcto funcionamiento de la red neuronal.

En cuanto a su funcionamiento, el algoritmo YOLO divide la imagen de entrada en una cuadrícula, y para cada celda de la cuadrícula, realiza predicciones sobre la presencia de objetos. Cada celda tiene la capacidad de predecir un conjunto fijo de cuadros delimitadores (bounding boxes) y las probabilidades de que cada cuadro delimitador contenga un objeto de una clase determinada.

Después de realizar predicciones en todas las celdas de la cuadrícula, se aplica un umbral de confianza para eliminar las detecciones de baja confianza. Luego, se utiliza un algoritmo llamado Non-Maximum Suppression (NMS) para eliminar detecciones redundantes y seleccionar las mejores detecciones. Un ejemplo de funcionamiento de YOLO podría ser el que se muestra en la *ilustración 7*.

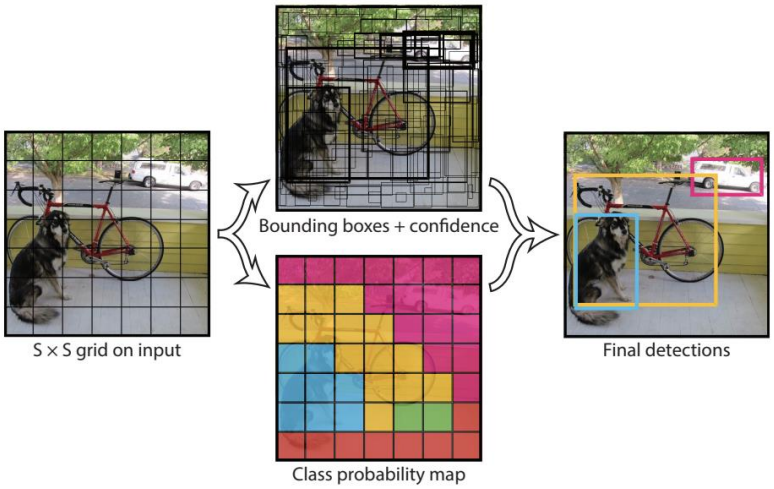

Ilustración 7. Esquema del funcionamiento YOLO.

Fuente: Velasco (2021)

Gracias a que YOLO es un algoritmo de detección de objetos rápido y preciso, se ha utilizado en este proyecto para poder procesar las imágenes que recibe el dron por la cámara y ser capaces de detectar múltiples tipos de objetos. Existen en la actualidad múltiples versiones de este algoritmo, el que se ha utilizado en este proyecto es el “yolo-v2-tinny”.

3.1.7. SLAM

El SLAM (Simultaneous Localization and Mapping) es una técnica comúnmente utilizada en la robotica para la realización de un mapa de un entorno y determinar la ubicación del robot dentro de este mapa. Esta herramienta permite la navegación de los robots sobre un entorno desconocido y que está en constante cambio.

Esta técnica hace uso de los sensores que tiene el robot para tomar datos del entorno y así poder utilizar dichos datos para la realización de un mapeo del entorno mientras el robot se mueve por él. A la vez que va realizando este mapeo, es capaz de actualizar su posición y orientación del entorno utilizando estimaciones sobre el mapa.

El SLAM utiliza diferentes algoritmos capaces de fusionar los datos de los diferentes sensores para realizar estimaciones de su posición y situar así el robot dentro del mapa del entorno. Los algoritmos más utilizados por el SLAM incluyen el filtro de Kalaman extendido (EKF), el filtro de partículas y métodos basados en grafos.

En este trabajo se ha hecho uso del SLAM para la poder mapear diferentes entornos y a posterior hacer uso de estos mapas para realizar una navegación por el entorno. Esto permitirá que el dron tenga la capacidad de llegar a un punto determinado evitando los obstáculos que encuentre por el camino y buscando la ruta óptima para realizar este recorrido hasta el punto indicado.

4. Desarrollo

Durante esta fase se fueron implementando los diferentes objetivos mencionados con anterioridad. A medida que iban siendo implementados se pasaban al siguiente, de esta forma se puede ver la progresión del desarrollo del trabajo en el entorno simulado. Para ello se ha decidido dividir este apartado explicando el desarrollo de cada objetivo de forma individual.

4.1. Desarrollo Inicial

El desarrollo inicial consistía en conseguir ejecutar la simulación del dron, para ello, se tuvo que investigar sobre los diferentes modelos de dron que incluye el paquete de “Hector quadrotor”. Cada uno usaba diferentes modelos de cámara y láseres de detección. Finalmente se decidió hacer uso del que utilizaba una cámara genérica, de esta forma si en un futuro se quisiera hacer la implementación en la vida real sería más fácil de adaptar. En cuanto al laser se utilizó el láser “hokuyo utm 30lx”, ya que es un modelo que funciona muy bien y es muy fácil de configurar. Tras la elección del modelo que se iba a utilizar, se investigaron los tópicos y nodos que incluía este modelo y que representaba cada uno. Para ello se ejecutó el código que abría el modelo que contenía todos los archivos de inicio del dron en el entorno simulado y también se ejecutó el comando “rqt_graph” que muestra un esquema con todos los nodos y tópicos juntamente con la relación que tienen entre ellos.

En la *ilustracion8* se pueden ver los tópicos y nodos que contenía el modelo utilizado en un inicio. Los nodos están representados por un círculo y los tópicos por rectángulos.

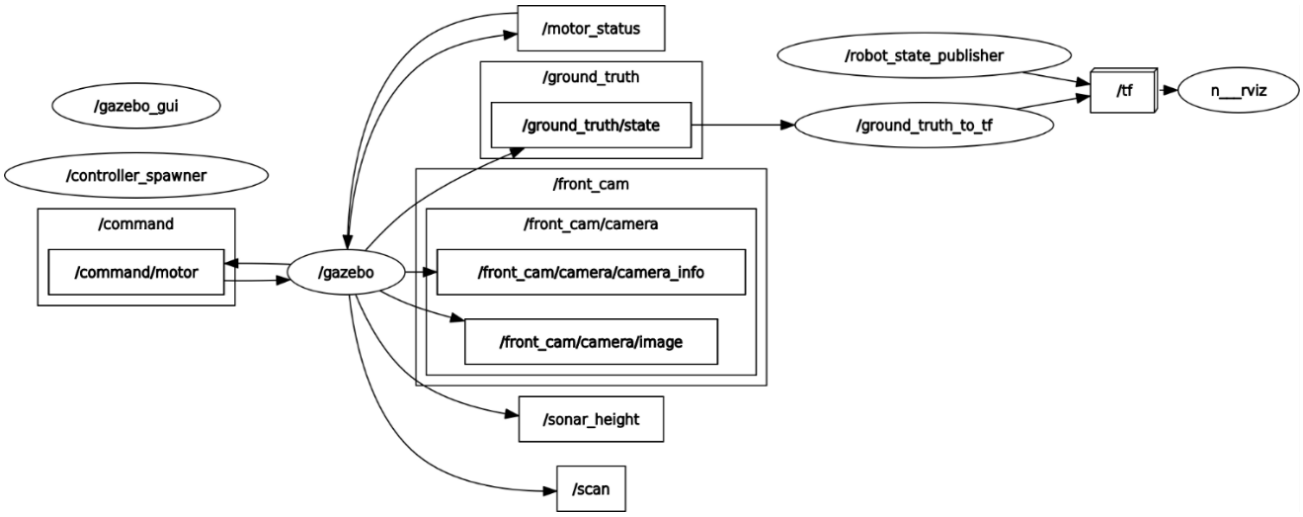

Ilustración 8. Diagrama inicial de tópicos y nodos ROS.

Fuente: Elaboración propia.

Como se puede ver en la *ilustración 8* los nodos, rodeados por un círculo, del modelo inicial del dron serían los siguientes:

- **Gazebo**: Este nodo es el encargado de hacer la simulación del entorno físico y del UAV, como se puede ver todos los tópicos salen de este ya que en este caso es el encargado de simularlos.
- **Gazebo_gui**: Es el que contiene la interfaz gráfica de Gazebo para que el usuario pueda interactuar con ella, para cambiar la configuración del entorno o modificar el entorno a simular.
- **Controller_spawner**: Es el nodo que se encarga de cargar y gestionar los controladores necesarios para controlar diferentes aspectos del comportamiento y movimiento de un robot.
- **Robot_state_publisher**: se encarga de publicar las transformaciones entre los marcos de referencia del robot, lo que permite a otros componentes y nodos del sistema acceder a la información actualizada del estado del robot.
- **Ground_truth_to_tf**: se utiliza para convertir la información de "ground truth" de un sistema en transformaciones de marcos de referencia (TF). Esto proporciona una forma de publicar la posición y orientación reales del sistema.

- **N__rviz**: es el nodo encargado de mostrar la información en el programa Rviz.

Por otro lado, podemos encontrar los siguientes tópicos, marcados en la *ilustración 8* con un recuadro:

- **Front_cam:** Este tópico corresponde a la cámara que tiene el UAV, dentro podemos ver que tiene los tópicos siguientes:
  - Camera_info: se utiliza para publicar la información de calibración de una cámara frontal. Esta información es fundamental para realizar correcciones de distorsión y realizar procesamientos de imágenes.
  - Image: se utiliza para publicar las imágenes capturadas por una cámara frontal.
- **Ground_truth**: Este tópico proporciona información detallada sobre el estado actual del sistema, como la posición, orientación, velocidad, aceleración y otros atributos relevantes.
- **Sonar_height**: Este tópico contiene la información de la altura a la que se encuentra el robot gracias a las medidas del sonar.
- **Scan**: Este tópico contiene los datos de escaneo obtenidos por un sensor láser, en nuestro caso el “hokuyo utm 30lx”.
- **Motor_status**: Este tópico contiene la información del estatus actual de los motores.

Por último, podemos ver que hay un elemento en la *ilustración 8* que este marcado con un doble recuadro. Este se trata del “tf” que es una biblioteca que se utiliza para gestionar y realizar transformaciones geométricas entre diferentes marcos de referencia en sistemas robóticos.

Una vez identificados los tópicos, se empezó a desarrollar los objetivos primordiales, que eran conseguir que el dron se visualizara en un entorno simulados y que sus sensores tomaran las medidas pertinentes. Para ello una vez abierto el modelo del UAV en gazebo, se añadieron objetos delante del UAV para ver que tanto la cámara como el láser de detección de objetos funcionaban

correctamente. Para ello se comprobaron los datos con la ayuda del programa Rviz. A posteriori se implementaron unos códigos que se suscribían a los tópicos e imprimían su información por pantalla, para así ver como poder procesar los datos a la hora de desarrollar otros algoritmos más adelante.

4.2. Implementación controles del dron

Al validar que los sensores del dron funcionaban de forma correcta se empezó a desarrollar el siguiente objetivo, que era implementar un código para poder controlar los movimientos del dron. Este algoritmo, publicaba las diferentes velocidades angulares y lineales para que el dron se moviera en la dirección deseada.

Un problema que surgió durante la programación de este código fue que se tenía que hacer uso de un lector de las entradas por teclado que funcionara de forma no bloqueante, es decir que no se quedara la ejecución del código congelada a la espera de una entrada, ya que sino no se llegaba a leer la información de los diferentes tópicos. Para ello hice uso de la librería "*termios*" y "*tty*" que permiten leer entradas de teclado evitando la congelación de la ejecución del código. En este código de control de vuelo se utiliza la lectura del sonar suscribiéndose al tópico "/sonar_heigh" para poder hacer lecturas de la altura actual y así poder realizar un despegue controlado a una altura por defecto de 2 metros y también, poder hacer un aterrizaje controlado al nivel del suelo.

Las teclas utilizadas para la navegación son las siguientes:

- **"t":** Para hacer el despegue controlado a una altura de 2 metros, para ello vamos leyendo del tópico "*/sonar_heigh*" y publicamos una velocidad lineal en el eje de coordenadas z de 0,5 para augmentar la altura del dron hasta que las medidas del sonar exceden o son iguales a 2.
- **"y":** Para hacer el aterrizaje controlado, para ello vamos leyendo del tópico "*/sonar_heigh*" y publicamos una velocidad lineal en el eje de coordenadas z de -0,5 para disminuir la altura del dron hasta que las

medidas del sonar son iguales a las medidas la primera vez que encendemos el dron.

- **“w”:** Para mover el dron hacia adelante, para ello publicamos una velocidad lineal en el eje de coordenadas x de 0,5.
- **“s”:** Para mover el dron hacia atrás, para ello publicamos una velocidad lineal en el eje de coordenadas x de 0,5.
- **“d”:** Para mover el dron hacia la derecha, para ello publicamos una velocidad lineal en el eje de coordenadas y de -0,5.
- **“a”:** Para mover el dron hacia la izquierda, para ello publicamos una velocidad lineal en el eje de coordenadas y de 0,5.
- **“q”:** Para rotar el dron sobre su mismo eje hacia la izquierda, para ello publicamos una velocidad angular en el eje z de 0,5.
- **“e”**: Para rotar el dron sobre su mismo eje hacia la derecha, para ello publicamos una velocidad angular en el eje z de -0,5.
- **“ ”(espacio):** Para augmentar la altura del dron, para ello publicamos una velocidad lineal en el eje de coordenadas z de 0,5, cosa que hace que el dron suba.
- **“c”:** Para disminuir la altura del dron, para ello publicamos una velocidad lineal en el eje de coordenadas z de -0,5, cosa que hace que el dron descienda.
- **“Esc” (tecla escape):** Esta sirve para parar el código de control del UAV.

Las velocidades, como se puede ver se pusieron con los mismos valores, pero, en diferentes ejes y en velocidades lineares y angulares. Esto es debido a que las velocidades lineales son las que representan los movimientos del dron sobre una línea recta, ya sea en el eje X, Y o Z y las velocidades angulares son las velocidades de rotación alrededor de su eje. Por ello en un UAV hay mucha libertad de movimientos ya que se pueden llegar a combinar diferentes velocidades lineales y angulares que permiten dotar a estos robots de una alta capacidad de maniobrabilidad.

Como se puede ver en la *ilustración 9*, el dron para realizar los movimientos de subida, bajada, adelante, atrás, derecha e izquierda únicamente haría uso de las velocidades lineales, ya que estos movimientos no implican una rotación sobre su eje, estos movimientos están representados en la *ilustración 9* por los ejes coloreados en escala de grises. En cambio, a la hora de hacer uso de los movimientos que implican rotación del eje de coordenadas del dron, entonces se haría uso de las velocidades angulares, dichos movimientos están representados en la *ilustración 9* por los ejes coloreados de naranja, rojo y rosa.

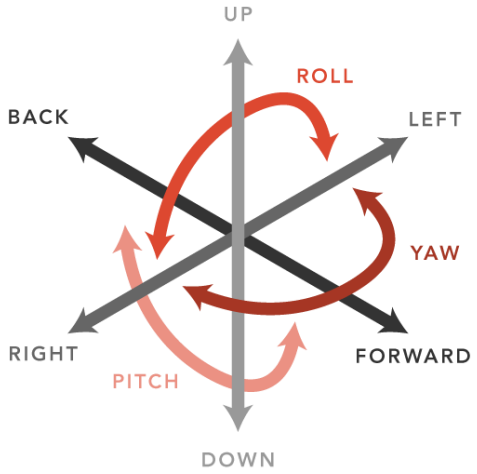

Ilustración 9. Ejes de movimiento de un dron.

Fuente: Del Drone, E. V. (2022).

Una vez implementado este código, que era el básico para poder realizar los principales movimientos del dron, se implementó una aplicación sencilla para poder controlar el UAV y poder ver su odometría en tiempo real. Para ello se utilizaron las librerías de "*tkinter*", utilizada para poder crear una interfaz gráfica sencilla. Esta interfaz lee de los tópicos "/ground_truth/state" y "/ground_truth_to_tf/pose" para leer los valores de la odometria del dron y mostrarlos por pantalla, también publica las velocidades necesarias para realizar los movimientos explicados con anterioridad. En los dos códigos hechos para el control del UAV una vez enviadas las velocidades, se envía una velocidad con todos los valores a cero. Esto nos permite que el dron no se quede permanentemente moviéndose con las velocidades anteriores y que por el contrario se quede quieto.

Para poder ejecutar el control del dron previamente se ha de ejecutar la simulación del dron para ambas opciones, cuando esta se esté ejecutando para poder iniciar la conducción por línea de comando se escribirá lo siguiente en el terminal “rosrun drone_basic conducir_drone.py”, por otro lado, si se desea ejecutar la comanda para hacer uso de la interfaz gráfica se ejecutara el comando “rosrun drone_basic ui_hector_quad.py”, esto hará que se inicie la interfaz y se puedan utilizar los diferentes botones para controlar el movimiento del dron.

Al finalizar esta implementación los nodos y tópicos quedaron tal y como se puede apreciar en la *ilustración 10.*

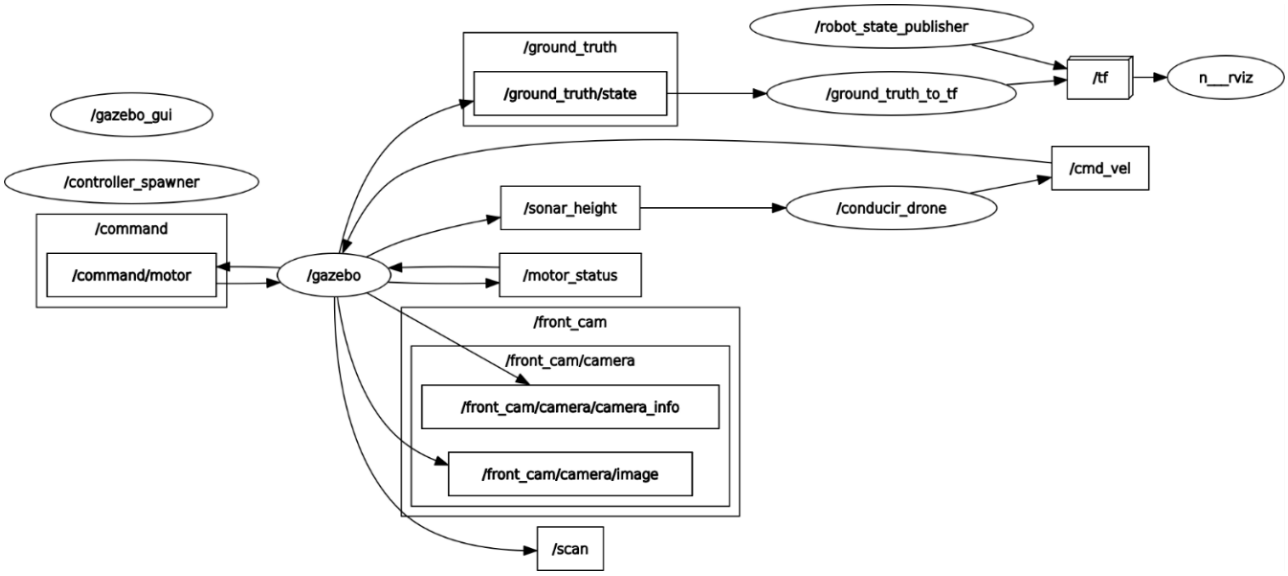

Ilustración 10. Diagrama tópicos y nodos con el objetivo conducción con comandos implementado.

Fuente: Elaboración propia.

La *ilustración 10* muestra los tópicos y nodos relacionados con la ejecución del primer Código implementado, el de entradas por teclado. Como podemos ver únicamente se ha añadido un nuevo nodo, respecto al esquema anterior, el nodo “conducir_drone” que este se subscribe al tópico “sonar_height” para poder leer la información que lee el sensor sobre la altura del robot para poder realizar el despegue y el aterrizaje controlado. También podemos observar que este nodo publica la información de las velocidades lineales y angulares en el tópico “cmd_vel” que este está relacionado directamente con Gazebo, que es el que lleva a cabo la simulación de los movimientos con las velocidades enviadas.

Por otro lado, la implementación con interfaz de usuario tiene otro esquema parecido de tópicos, la única diferencia es que esta también se subscribe al tópico “ground_truth_to_tf/pose” y “ground_to_tf/state” para mostrar por pantalla la información en tiempo real de la odometria del robot. Este esquema puede apreciarse en la *ilustración 11*.

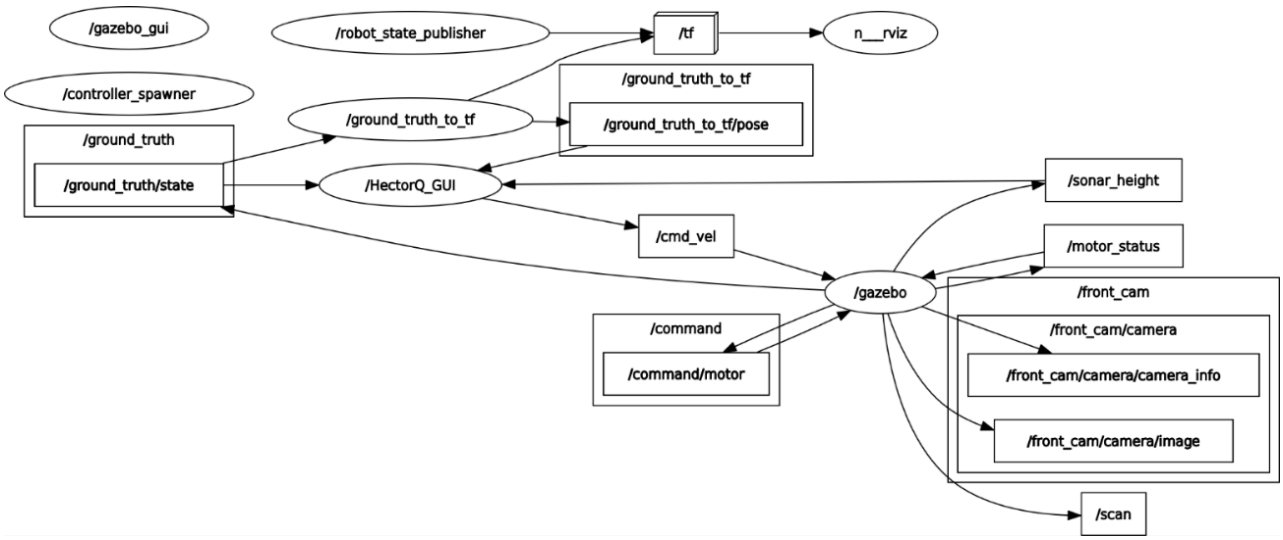

Ilustración 11. Diagrama tópicos y nodos con el objetivo conducción con interfaz gráfica implementado.

Fuente: Elaboración propia.

4.3. Implementación YOLO

Una vez finalizados los objetivos globales, se pasó a desarrollar los específicos. Para ello se decidió empezar con la incorporación de YOLO al proyecto para dotar a nuestro robot de detección de objetos. Para ello, se hizo uso de un paquete de ROS llamado “darknet_ros”, este paquete utiliza redes neuronales convolucionales para realizar la detección de objetos y proporciona los nodos de ROS necesarios para realizar la suscripción y publicación de los datos. Dentro de este paquete existían ya unas redes neuronales previamente entrenadas, que fueron las que se utilizaron. Una vez incorporado el paquete al trabajo se tuvieron que hacer una serie de modificaciones para que funcionara. Una de las modificaciones fue indicar cual era la fuente de entrada de las imágenes, que en nuestro caso era el tópico donde la cámara publicaba las imágenes (“/front_cam/camera/image”). Otra de las modificaciones fue cambiar algunos archivos para que no se utilizara el sistema CUDA de NVIDIA ya que la tarjeta

gráfica del ordenador donde se desarrolló el trabajo no era combatible con esta tecnología. La tecnología CUDA, permite utilizar las GPUs para acelerar aplicaciones y cálculos intensivos, cosa que provoca que al no poder incluirla en nuestro trabajo influya en la fluidez de la detección de imágenes y que a menudo aparezcan bajadas de FPS en la imagen de detección de salida.

Al acabar esta implementación los nodos y tópicos quedaron representados como se puede apreciar en la *ilustración 12*.

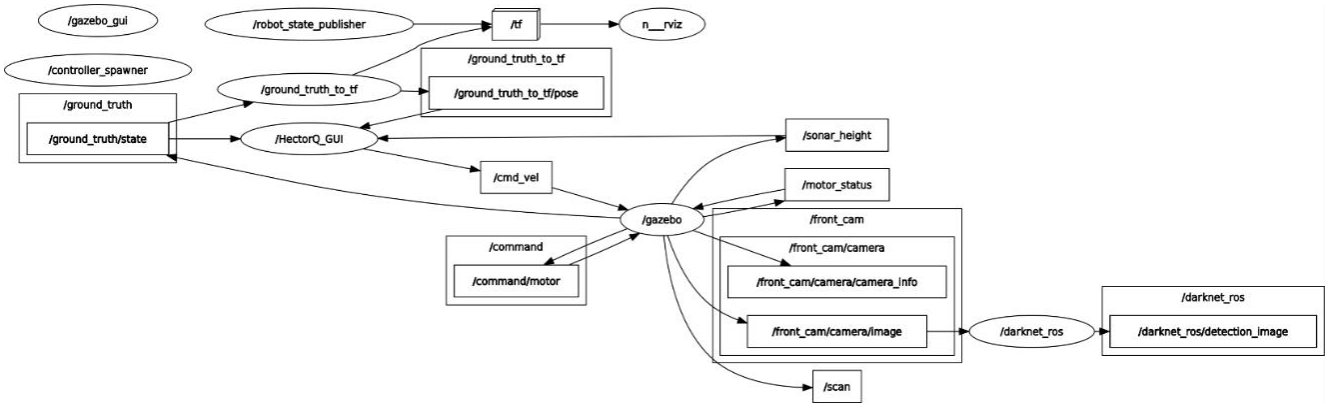

Ilustración 12. Diagrama tópicos y nodos con el objetivo YOLO implementado.

Fuente: Elaboración propia.

Como se puede ver en la *ilustración 12* el único cambio con el esquema anterior (*ilustración 11*) es que se ha creado un nuevo nodo llamado “darknet_ros” que se subscribe al tópico “front_cam/camera/image”, de donde coge las imágenes que capta el dron y las procesa para después publicar en el tópico “/darknet_ros/detection_image” la imagen recibida por la cámara resaltando los objetos detectados con un marco juntamente con una pequeña etiqueta del nombre del tipo de objeto detectado.

4.4. Implementación Aplicativo Web

Cuando acabe con la implementación de YOLO, se pasó a la implementación de la web para poder ver la información del sistema ROS y sus componentes de forma más gráfica y visible de cara al usuario. Para ello, investigando, se descubrió que existía un paquete en ros llamado ”rosboard” el cual permitía hacer esta visualización del entorno ROS. Así que se decidió integrarla en el

proyecto ya que era una herramienta que podía substituir con facilidad el Rviz y aparte, mostraba la información de forma más gráfica y expresiva. Para poder hacer uso de este paquete era necesario la instalación de algunas librerías de Python, estas librerías eran "*torando*", "*simplejpg*" y "*rospkg*". Una vez instaladas estas librerías se podía ejecutar el nodo de la web de forma que detectaría los nodos ejecutados en cada momento y aquellos que pudieran ser serializados para ser enseñados por pantalla se podrían visualizar. Esta web por defecto se levanta sobre la dirección de localhost utilizando los puertos 8888. De esta forma si se introduce la dirección "localhost:8888" en el navegador donde se esté ejecutando el código, aparecerá la información de los diferentes tópicos que se hayan seleccionado haciendo uso de la barra de navegación lateral. Un punto fuerte de este paquete es que este nodo de la web se iba subscribiendo a los tópicos a medida que los ibas seleccionando de esta forma el código era mucho más eficiente. Una vez integrado este sistema la gráfica de tópicos y nodos quedaba representada como se puede ver en la *ilustración 13.*

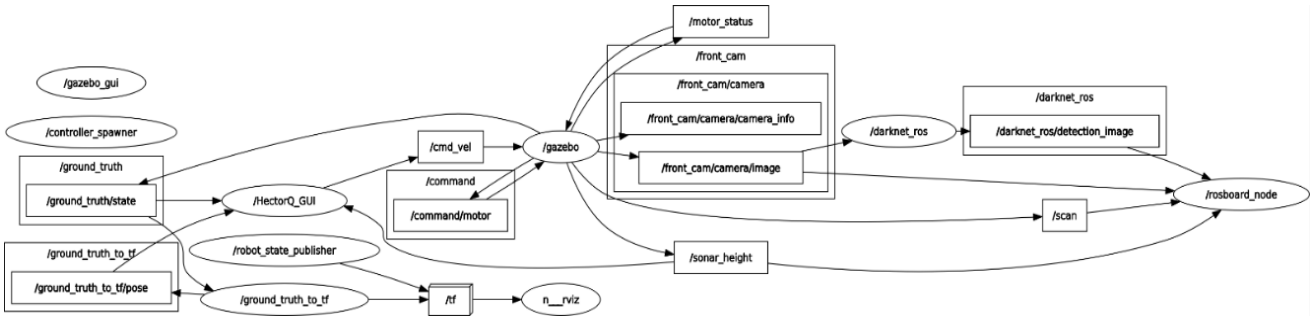

Ilustración 13. Diagrama tópicos y nodos con la web implementada.

Fuente: Elaboración propia.

Como se puede ver en la *ilustración 13*, en comparación al esquema de tópico anterior (*ilustración 12*) se ha añadido un nodo nuevo llamado "rosboard". Este nodo, como se ha explicado con anterioridad únicamente se suscribirá a los tópicos que el usuario seleccione desde la web, de esta forma ahorra en recursos y es más eficiente. Si nos fijamos en el gráfico de la *ilustración 13*, podremos ver que actualmente la web estaría mostrando la información de los tópicos "front_cam/camera/image", "sonar_height", "scan" y "darknet_ros/detection_image". Esto significaría que se estarían mostrando por la web las imágenes recibidas por la cámara del dron, las imágenes de detección,

la información de la altitud y la información del láser de escaneo de forma simultánea y en tiempo real.

4.5. Navegación Dron

Finalmente, y una vez acabados los demás objetivos se inició la implementación de la navegación del UAV. Dentro del objetivo de navegación se diferenciaron dos grandes tareas:

La tarea principal era la de crear un mapa del entorno donde se situaba el dron, con la ayuda de las lecturas del sensor laser. De esta forma, previamente a la navegación, se realiza un mapeo de todo el entorno. Dicho mapeo se hace utilizando el paquete de ROS llamado “Hector Mapping”, que es un paquete que dentro contiene todos nodos necesarios para realizar un mapa del entrono utilizando los diferentes sensores como una o varias cámaras y un sensor láser para recopilar datos del entorno. Estos datos se utilizan para estimar la posición y orientación del robot en tiempo real y construir un mapa 2D del entorno.

Al ejecutar esta tarea juntamente con las ya implementadas anteriormente, los nodos y tópicos quedaron representados como se puede apreciar en la *ilustración 14*.

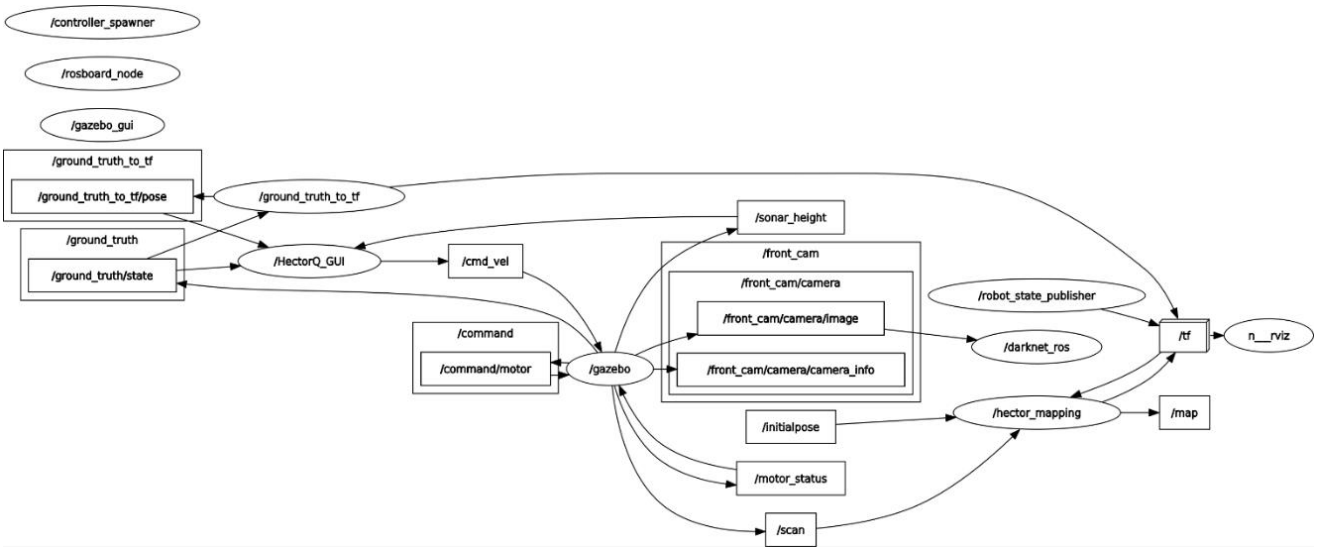

Ilustración 14. Diagrama tópicos y nodos con el objetivo del mapeo implementado.

Fuente: Elaboración propia.

En la *ilustración 14* se puede ver como a aparecido un nuevo nodo llamado “hector_mapping” que es el nodo encargado de hacer el mapeo del escenario, este nodo se subscribe a los tópicos “scan”, de donde obtiene las medidas tomadas por el láser para detectar los obstáculos y al tópico “InitalPose” para leer la posición de partida del robot. Haciendo uso de estas medidas y conjuntamente utilizando la librería “*tf*” para calcular la posición del robot respecto al mapa, publica en el tópico “map” la información que encuentra del entorno, para poder tener un mapa que sea fiel a la realidad se tendría que navegar por todo el entorno para así poder recabar los máximos detalles sobre este. Una vez finalizado el mapeo del escenario, quedaría un mapa en dos dimensiones, parecido al ilustrado en la *ilustración 15*.

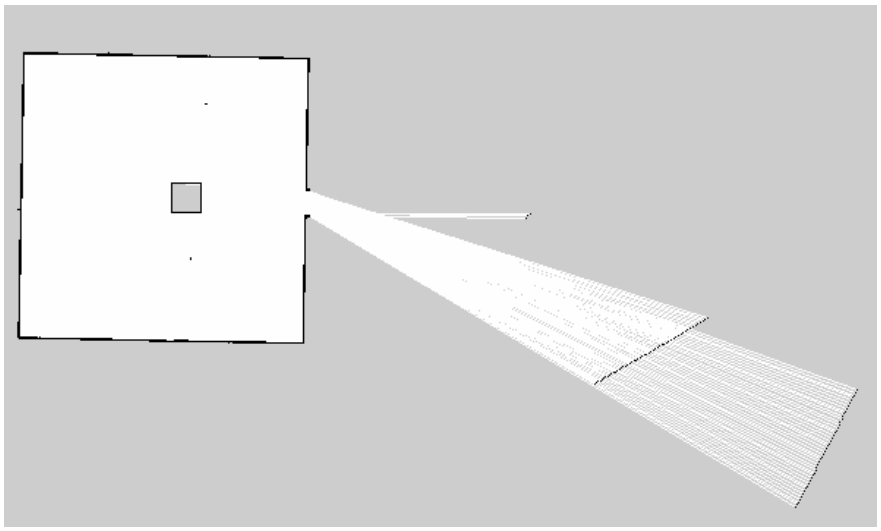

Ilustración 15. Mapa de un entorno creado gracias a la implementación.

Fuente: Elaboración propia.

Este mapa corresponde al mundo con nombre school2 de dentro del paquete. Como se puede ver detecta de forma correcta la caja situada en medio de la sala y las dos señales a su lado.

La segunda y última tarea era, utilizando este mapa del entorno creado con anterioridad, **enviar al dron un punto de destino dentro de este mapa** para que navegara hasta él. De esta forma se consiguió que el UAV tuviera la libertad para llegar a cualquier punto de forma autónoma. Para ello se hizo uso de forma conjunta de los paquetes “amcl” y “Move Base”.

El paquete AMCL (Adaptive Monte Carlo Localization) se utiliza para la localización y el mapeo del entorno. Este paquete utiliza una técnica llamada filtro de partículas para realizar la estimación de la posición actual del robot. Para ello, inicialmente, se distribuyen varias partículas en el espacio del mapa y se les asigna una probabilidad de ser la posición real del robot. A medida que se obtienen datos de percepción de los diferentes sensores, las partículas se ponderan y recalculan según la similitud con las mediciones reales del entorno. Este proceso se va repitiendo a medida que van llegando nuevos datos, cosa que hace que cada vez vaya con más precisión. Por otro lado, el paquete “Move base” permite la recepción de un objetivo de destino en el mapa y planificar una ruta óptima para llegar al destino, para ello utiliza algoritmos de planificación como podrían ser A* o Dijkstra, que son algoritmos basados en búsqueda en grafos que sirven para obtener la mejor ruta posible para alcanzar un objetivo.

Ejecutando esta implementación, se puede ver en la *ilustración 16* como quedo la gráfica de tópicos y nodos.

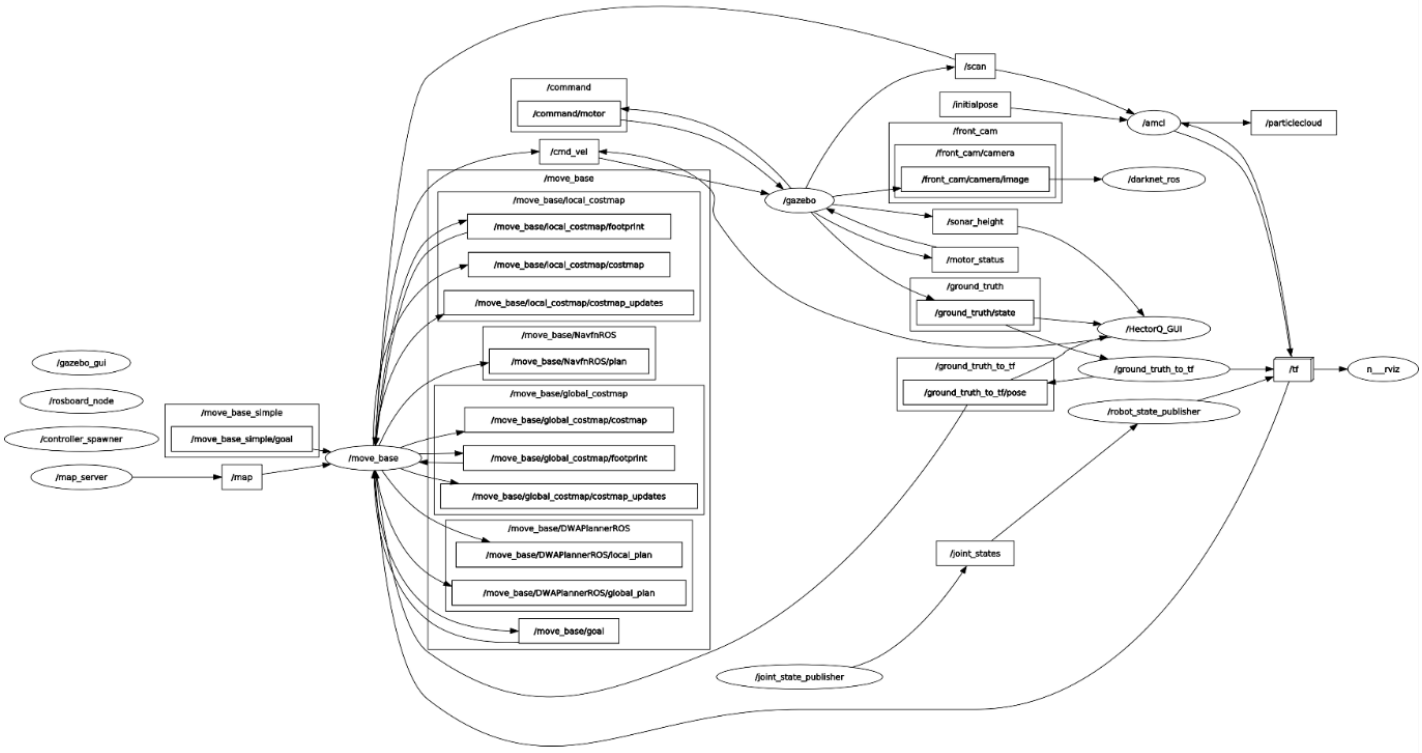

Ilustración 16. Diagrama tópicos y nodos con el objetivo de la navegación implementada.

Fuente: Elaboración propia.

En el diagrama de la *ilustración 16*, se puede ver como a aparecido el nodo “map_server” que es el encargado de cargar el mapa en el tópico “map” para que a posteriori se subscriba el nodo “move_base” a él para poder realizar la navegación. El nodo “move_base” es el que se encarga de realiza la navegación, para ello inicialmente lee la información del tópico “move_base_simple/goal”, en este tópico es donde el Rviz publica el punto al que se quiere navegar. Una vez publicado el punto hace uso de las lecturas de la librería “tf” para ubicar el robot en el mapa, estas librerías se ayudan del algoritmo “amcl” mencionado anteriormente que se subscribe al tópico “sonar” con tal de realizar la ubicación. Paralelamente el nodo “move_base” tiene diversos tópicos dentro de el:

- **“move_base/local_costmap”:** Este tópico es el encargado de guardar el mapa de costes, es decir guarda la información de lo que costaría llegar a ciertas posiciones leyendo la información de los sensores.
- **“move_base/NavfnROS”:** Es el encargado de calcular la ruta óptima para que el robot llegue hasta la trayectoria.
- **“move_base/global_costmap”:** Es el tópico que se encarga de ir actualizando el mapa de costes y el que planifica la primera ruta para que el robot llegue a su destino.
- **“move_base/dwaPlannerROS”:** Es el encargado de calcular todas las posibles rutas para alcanzar el objetivo y utiliza el algoritmo Dynamic Window Approach (DWA) para calcular dichas trayectorias.

Todos estos tópicos y algoritmos son los utilizados por el nodo “move_base” para poder realizar la navegación. Para poder realizar la navegación, este nodo publica en el tópico “cmd_vel” la información de las velocidades lineales y angulares necesarias para alcanzar dicho objetivo.

5. Pruebas y resultados

Mientras se iba realizando el desarrollo de este trabajo se fueron haciendo diversas pruebas, estas pruebas consistían en validar cada objetivo realizado con tal de asegurarse de que los criterios de aceptación de cada objetivo se cumplían y darlo como finalizado.

En los objetivos esenciales se validó que el dron y todos sus componentes funcionaban de forma correcta y precisa, para ello se creó un entorno con algunos obstáculos para ver si el láser era capaz de detectar los diferentes obstáculos. Para validar esto se hizo uso de la herramienta Rviz, al principio se observó que la distancia de detección del láser estaba acotada a un rango muy pequeño, investigando en la documentación del láser se vio que la distancia máxima de detección que aceptaba el láser era de 30 metros y se estableció esta como la máxima por defecto.

También se vio que la distancia de detección máxima del sonar era de 3 metros, este parámetro se dejó así ya que el sonar únicamente se utiliza en las implementaciones para el despegue y el aterrizaje, por lo tanto, un despegue a 3 metros se consideró que estaba bien.

Como se puede ver en la *ilustración 17*, al hacer estos cambios, en el Rviz se puede observar que tanto la cámara, el láser y el sonar median de forma correcta y por lo tanto pasaban el criterio de aceptación del objetivo.

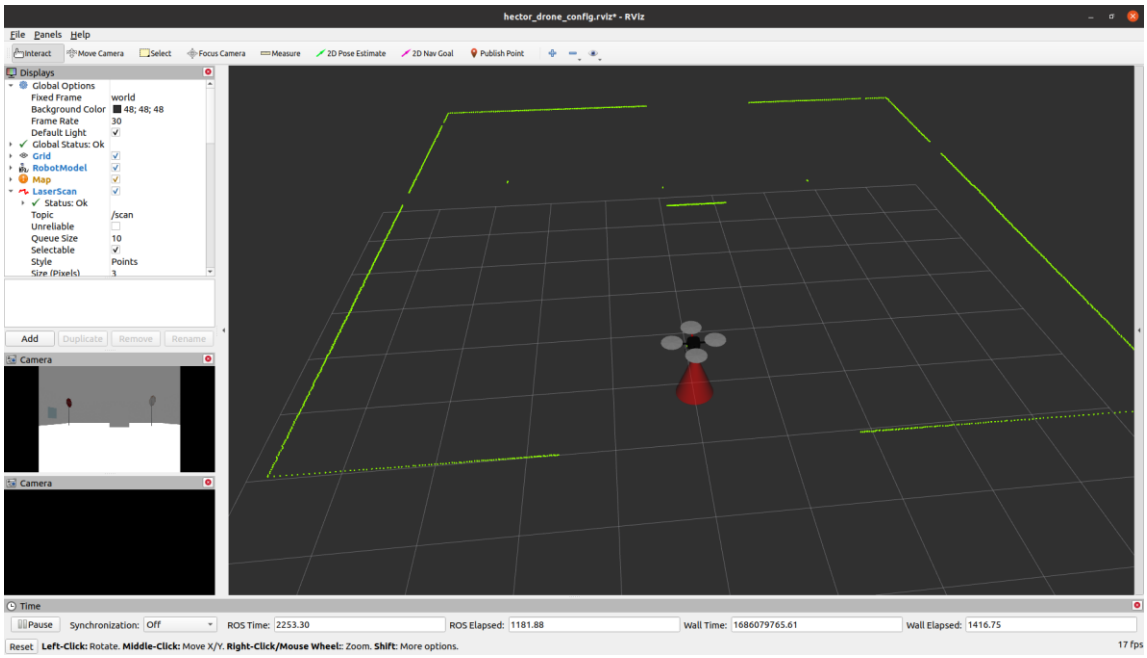

Ilustración 17. Simulación en Rviz para validar los sensores.

Fuente: Elaboración propia.

Una vez validado el correcto funcionamiento de la simulación del dron y sus componentes, se realizaron unas pruebas para validar la conducción del UAV. Para ello, se inició el dron en el entorno simulado y se verifico que con las entradas por teclado el dron hacia los movimientos esperados, una vez validado esto, se validaron los movimientos haciendo uso de la interfaz gráfica. El aspecto de dicha interfaz gráfica puede verse representado en la *ilustración 18*.

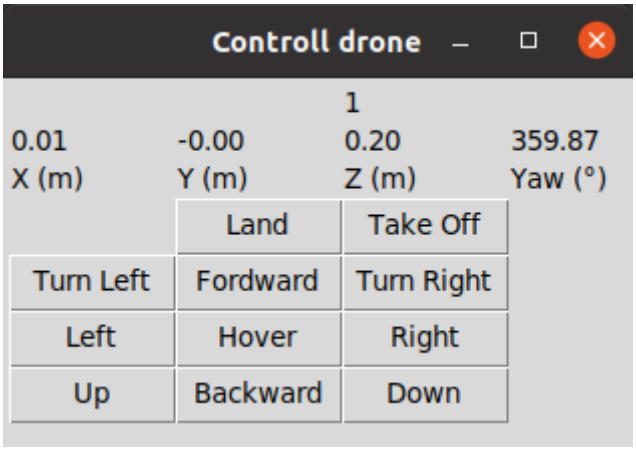

Ilustración 18. Interfaz gráfica creada para controlar el dron.

Fuente: Elaboración propia.

A continuación, y como los criterios de aceptación de los objetivos esenciales se cumplían, se fueron validando, a medida que se iban realizando, los objetivos específicos. Como el primer objetivo finalizado fue la implementación del algoritmo YOLO utilizado la detección de imágenes, se decidió crear un entorno donde el robot residía en medio de dicho entrono y estaba rodeado de objetos, con esto se quería validar si el algoritmo era capaz de detectar dichos objetos y con que precisión para así poder adaptar el umbral de confianza para evitar las falsas detecciones de objetos. Una vez adaptado el umbral de detección se probó que el algoritmo funcionara de forma correcta y que pasara los criterios de aceptación. En la *ilustración 19* se ve como el algoritmo es capaz de detectar la señal de stop y mostrar la imagen con el recuadro de donde se encuentra el objeto.

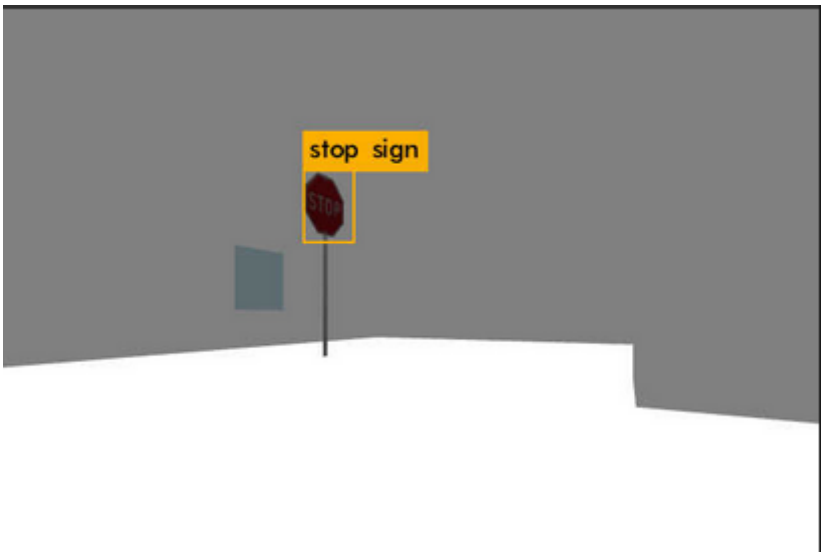

Ilustración 19. Imagen de detección YOLO detectando un stop.

Fuente: elaboración propia.

El siguiente objetivo para realizar fue el de la implementación de una web capaz de visualizar toda la información de los nodos y tópicos que contenía ROS en tiempo real de ejecución, para validar este objetivo, se añadieron diferentes tópicos a la visualización de la web utilizando el menú lateral que se ve representado en la *ilustración 20*, en la esquina superior izquierda con el icono de las tres rayas. Una vez añadidos algunos tópicos se validó que la información que se mostraba por la web era la misma que la que se mostraba en el Rviz.

De esta forma se estaba validando que la web mostraba de forma correcta la información de dichos tópicos y que se actualizaban los datos en tiempo real. La web en cuestión tenía el aspecto que se muestra en la *ilustración 20*.

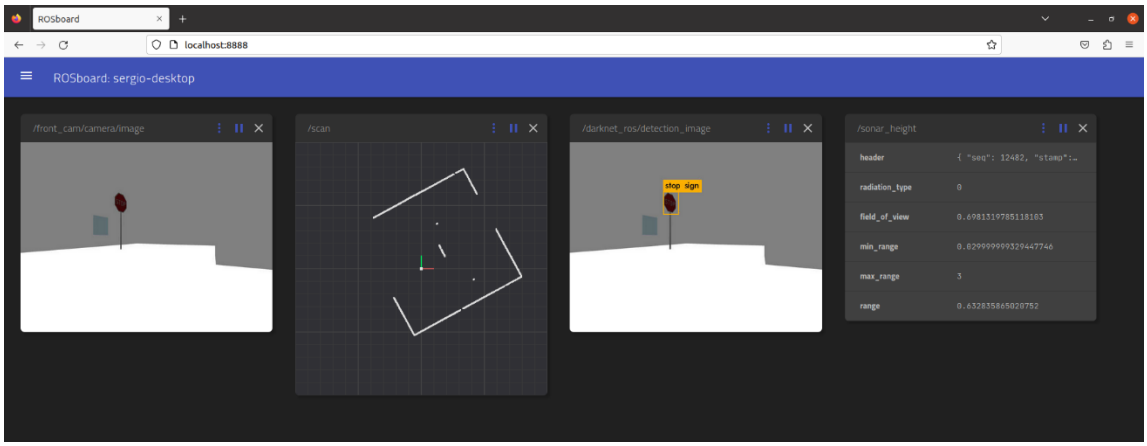

Ilustración 20. Web creada para visualizar los nodos.

Fuente: Elaboración propia.

Finalmente, para la implementación de la navegación en dos dimensiones del UAV, primero se validó que haciendo uso de diferentes entornos simulados, el dron era capaz de hacer un mapeo de forma precisa de cada uno de ellos. Para ello se ejecutaban los algoritmos necesarios para la creación del mapa y se navegaba por todo el entorno para ver si detectaba todos los obstáculos de forma correcta.

Una vez validado el mapeo se pasó a verificar que el algoritmo de navegación funcionara de forma correcta, para ello se utilizaron diferentes mapas creados con anterioridad para ir probando a asignarles diferentes puntos de destino, cada uno de ellos con obstáculos por el medio, para ver si el algoritmo funcionaba de forma correcta y era capaz de navegar hasta el punto y por el camino era capaz de esquivar los obstáculos que se le presentaban.

Una vez acabadas todas las pruebas y validado el correcto funcionamiento de todos los objetivos, se realizó una serie de videos de demostración de las pruebas. Dichos videos se pueden ver en el siguiente enlace: "https://www.youtube.com/playlist?list=PL_9UO3ombEuM2Jg7vq8Btsar7kZXjuMIj".

6. Análisis de componentes hardware

Una vez acabadas todas las implementaciones de la simulación del dron, se decidió realizar un análisis de los costes que tendría la implementación de la simulación en la vida real. Para ello hacía falta detallar todos los componentes físicos que el dron necesitaba para realizar las tareas implementadas con anterioridad y estos componentes tenían que ser lo más parecidos posibles a los simulados para que la adaptación de los códigos fuera mínima. Por esta razón, se tenían que escoger de forma meticulosa en función a su tamaño y su peso.

6.1. Elección de componentes

En esta primera fase, se investigaron los diferentes componentes necesarios para la implementación en la vida real. Para ello primer se buscaron los componentes estructurales y sensores para después poder calcular la potencia que necesitarían los motores para levantar este peso inicial.

Los elementos estructurales y los sensores escogidos fueron los siguientes:

- **Controladora de vuelo:** Este componente es el que estará conectado con todos los demás componentes, para ello se utilizara una "px4" de la marca Pixhawk ya que es muy potente y combatible con el sistema operativo ROS.

- **Ordenador con ROS:** Para ello se ha escogido una "Rasberri Pi 4", en ella residirá el entorno ROS y se comunicará directamente con la controladora de vuelo.

- **Laser de detección de obstáculos:** En la simulación se ha hecho uso del láser "Hokuyo utm 30lx", que funciona muy bien, la única pega que tiene este laser es que es muy costoso, por lo tanto si no se quiere asumir el coste de dicho laser, podría utilizarse alguno que le fuera sustituto como el "RPLIDAR" que permitiría bajar el coste del dron, bajando el rango de escaneo a 12 metros, pero ganando un ángulo de escaneo de 360 grades enfrente 270 que tiene el "Hokuyo utm 30lx". Por este motivo se hará uso del RPLIDAR ya que es una opción más asequible y con prestaciones similares.

- **Sonar:** Para el sonar no haría falta comprarlo, ya que la controladora de vuelo "px4" incorpora un barómetro capaz de medir la altura a la que se encuentra el dron. Únicamente habría que adaptar un poco el código para que dicho sensor publique la altura en el mismo tópico que lo hacía en la simulación.

- **Cámara:** Para la cámara sería interesante hacer uso de alguna que tuviera una buena calidad, para que así nuestro algoritmo de YOLO fuera capaz de detectar con facilidad los diferentes objetos. Para ello he seleccionado una "RunCam Robin" ya que tiene muy buena calidad de video.

- **Fuselaje dron:** En cuanto al fuselaje se podría utilizar cualquiera que permitiera integrar todos los demás componentes de forma espaciosa. Para ello se recomienda el fuselaje "DJI F450" ya que es bastante grande.

Una vez seleccionados estos componentes, se procedió a escoger los motores, variadores y batería capaces de levantar el peso de los componentes ya seleccionados, para ello se realizó un desglose de los pesos acumulados hasta ahora.

El desglose de pesos se puede ver representado en la *ilustración 21*.

Componentes	**Peso (gramos)**
Controladora px4	15,8
Rasberri PI 4	45
RPLIDAR	370
Camara RunCam Robin	5,5
Fuselaje DJI F450	340
Cables y demas	20
Peso TOTAL	**796,3**

Ilustración 21. Desglose de pesos inicial.

Fuente: Elaboración propia.

Como se puede ver en la *ilustración 21,* el peso total de los componentes seleccionados era de 796,3 gramos, por lo tanto, se tenían que buscar unos motores capaces de levantar dicho peso, pero teniendo en cuenta también su propio peso, el de los variadores y el de la batería. Para ello cada fabricante de

motores en las especificaciones del producto dan un informe detallado sobre la cantidad de peso que el motor puede levantar en función a el porcentaje de velocidad al que están funcionando.

Según el artículo publicado en Drone 24 hours, es recomendable que el empuje generado en gramos con los motores funcionando a su potencia máxima tiene que ser el doble al peso total del dron (2018).

Al escoger los motores, también se debía tener en cuenta su peso, el de los variadores y el de la batería a escoger.

Teniendo esto en cuenta finalmente se decidió escoger los siguientes elementos:

- **Motores y hélices**: Se escogieron los motores "Racerstar Racing Edition 2306" en su versión de 2700 KV, ya que cada uno es capaz de levantar un kilo funcionando a máxima potencia, por lo tanto, serían capaces de levantar hasta 4 kilos. Cada uno de ellos tiene un peso de 33,5 gramos y haría uso de las hélices del modelo "TC5045".

- **Variadores**: Para los variadores, que son los encargados de pasar la señal de la controladora a información de sentido y velocidades de giro para el motor. Hemos escogido los "Cyclone 20A BLHeli_S" ya que admiten hasta baterías de 4s es decir de 14,8 voltios. Estos variadores pesan cada uno 5 gramos y serían necesarios cuatro, uno por motor.

Antes de escoger la batería se debe tener en cuenta el peso hasta ahora del dron, ya que si se supera el peso que pueden levantar los motores el dron no se levantara del suelo.

Para ello se ha restructurado la tabla de pesos anterior añadiendo los componentes nuevos.

Dicha tabla quedo representada como la *ilustración 22* muestra.

Componentes	Peso (gramos)
Controladora px4	15,8
Rasberri PI 4	45
RPLIDAR	370
Camara RunCam Robin	5,5
Fuselaje DJI F450	340
Cables y demas	20
Motores (x4)	134
Variadores (x4)	20
Peso TOTAL	**950,3**

Ilustración 22. Desglose pesos sin batería dron.

Fuente: Elaboración propia.

Como se puede ver en la *ilustración 22*, el peso actual del dron con todos los componentes detallados seria de 950,3 gramos. Por lo tanto, se tendría que comprar una batería que no superará los 550 gramos, ya que así contaríamos con un margen de empuje que nos permitirá que el dron fuera más ligero y ágil.

También hemos de tener en cuenta que los variadores y motores escogidos son para una batería de 14,8 voltios. Por estos dos motivos se ha escogido la batería "El grafeno Turnigy 4000mAh 45C Lipo 4S", esta batería pesa 484 gramos por lo tanto esta entre nuestros límites y es capaz de alimentar todos los componentes de forma correcta.

Finalmente teniendo todos los componentes escogidos se decidió comprobar que el dron tendría la suficientemente fuerza como para levantar todos los componentes. Para ello se calculó el peso total definitivo del dron que era de 1434,3 gramos y también se calculó el empuje máximo que tendría el dron.

Para ello, se hizo uso de la ficha técnica de los motores, dicha información técnica puede verse representada en la *ilustración 22*.

MODEL	KV (rpm/V)	Voltage (V)	Prop	Load Current (A)	Pull (g)	Power (W)	Efficiency (g/W)	Lipo Cell	Weight (g) Approx
BR2306S	2400	11.1	5045	23.7	660	250	2.6	2-4S	33.5
		14.8		33.0	980	458	2.1		
	2700	11.1	4045	16.5	480	175	2.7		
		14.8		24.5	760	345	2.2		
		11.1	5045	25.0	705	262	2.7		
		14.8		35.0	1070	490	2.2		

Ilustración 23. Tabla especificaciones serie motores Racerstar Racing Edition 2306 2400/2700kv.

Fuente: RC Groups.

Como se puede ver en la *ilustración 23*, la versión del motor 2700 KV (Kilovoltio) con una batería de 14,8 voltios y las hélices 5045 que son las escogidas tiene un empuje máximo de 1070 gramos, por lo tanto, si hacemos uso de 4 motores de estos, tendremos un empuje total de 4,28 Kilogramos, cosa que haría que los motores fueran capaces de levantar el dron y todos sus componentes de forma eficaz y con un margen. Este margen serviría por si en algún futuro se quisiera añadir algún otro componente se pudiera hacer sin problema.

6.2. Desglose de costes de los componentes

Una vez definidos los mejores componentes para realizar el trabajo en el mundo físico, se hizo un cálculo del costo total que tendría la elaboración del dron.

Para ello se creó una tabla con los precios actuales de cada componente y un sumatorio final donde aparece el precio que costaría la implementación de la simulación en la vida real. Dicha tabla está representada en la *ilustración 24*.

Componentes	Precio (€)
Controladora px4	129,89
Rasberri PI 4	60,57
RPLIDAR	119,99
Camara RunCam Robin	24,95
Fuselaje DJI F450	31,86
Cables y demas	10
Motores (x4)	52,2
Variadores (x4)	19,92
Bateria	57,67
Helices	5,19
Precio TOTAL	512,24

Ilustración 24. Desglose precios dron.

Fuente: Elaboración propia.

Como se puede ver en la *ilustración 24*, el coste total del dron seria de 512,24 € con los precios en la actualidad de dichos componentes. Si alternativamente a la utilización de ROS se quisiera poder manejar el dron con un a control a remoto se tendrá que añadir un receptor de señales y una emisora capaz de comunicarse entre sí.

También sería buena idea adquirir otra batería así de esta forma mientras se está utilizando una la otra se está cargando y poder así ampliar las horas de vuelo del dron.

7. Conclusiones y trabajo futuro

En conclusión, hemos podido abordar los diferentes objetivos planteados durante la elaboración del trabajo de forma satisfactoria. Dichos objetivos han ido solucionándose a medida que se iban realizando los anteriores.

Se empezaron abordando los objetivos que eran puros de la simulación, es decir aquellos que se basaban en la visualización del dron en el entorno simulado y los que permitían que el dron se pudiera mover y por lo tanto interactuar con su entorno.

Una vez abordados y realizados estos objetivos se pasó a realizar los objetivos que eran más específicos, es decir que estaban más orientados en hacer una tarea en concreto para mejorar las características del dron. Tras realizar los objetivos mencionados y dando por finalizado el trabajo de las implementaciones, se llevó a cabo una investigación exhaustiva de los componentes necesarios para poder implementar este proyecto a la vida real.

Por otro lado, como trabajo futuro de este proyecto, estaría la implementación de estos algoritmos y este esquema de ROS en un UAV real, ya que al utilizar un modelo genérico para programar dichos algoritmos la adaptación a la vida real seria simple. Para poder realizar este trabajo futuro habría que adquirir los componentes mencionados en el análisis y montarlos.

Una vez montados todos los componentes se realizarían las mismas pruebas que se han realizado en simulado para verificar que el dron funcionara de forma correcta. Seguramente durante la ejecución de las pruebas aparecieran algunas necesidades de adaptar el código para que funcionara en real, o bien puede ser que se decidiera implementar alguna otra funcionalidad. Para ello es recomendable que una vez teniendo el dron funcional, se probaran las nuevas funciones en el simulador, para validarlas y después probarlas en el dron. De esta forma se podrían probar las nuevas implementaciones sin correr ningún tipo de riesgo.

Otra de las posibles mejoras de cara al futuro seria la incorporación de un GPS al dron capaz de localizarlo de forma precisa para así poder hacer uso de sus

coordenadas para definir puntos de navegación, ya que como está planteado actualmente en caso de tener un escenario que sea muy extenso nos será muy complicado realizar el mapeo de forma completa.

La eficiencia de los algoritmos implementados es bastante optima, únicamente se quedaría un poco justa cuando se hace uso del YOLO, ya que al no tener una gráfica combatible con el sistema CUDA de NVIDIA, provoca retardos en la detección de objetos y en la imagen con los objetos remarcados por recuadros. Así que como mejora se podría hacer uso de algún ordenador que soportara el sistema CUDA.

Gracias a la elaboración de este proyecto, he conseguido aprender más sobre los algoritmos y estructura de un dron y a familiarizarme con el entorno ROS y sus componentes. También, he sido capaces de hacer uso de los conocimientos dados durante el grado para abordar los diferentes objetivos y construir una arquitectura robusta y dividida en diferentes módulos.

8. Anexos

Para poder realizar la ejecución de este trabajo primero deberemos tener un sistema Ubuntu corriendo en su versión 20.4[1]. Para ello podemos hacer uso de Docker o tener un subsistema de Ubuntu instalado de forma nativa en nuestro ordenador.

Si se ha decidido utilizar una imagen Docker hará falta que se instale algunos componentes previamente. El primer componente para instalar es el Wsl[2], necesario para poder ejecutar Ubuntu en Windows y el programa XLaunch[3] para poder visualizar la información. Una vez instalados estos dos programas, se tendrá que instalar Docker Desktop abrirlo y después ejecutar los siguientes comandos en la terminal:

1. *docker pull osrf/ros:noetic-desktop-full*
2. *docker run --name TFG_DronSimulator -e DISPLAY=host.docker.internal:0.0 -it osrf/ros:noetic-desktop-full*

(Cada vez que se ejecute el código hay que tener ejecutando el XLaunch).

Una vez se ha instalado una versión 20.4 de Ubuntu, tanto en Docker como en nativo, previamente a la ejecución del código hay que instalar una serie de paquetes y librerías que serán necesarias para la ejecución del trabajo. Los comandos de instalación de los diferentes paquetes, por orden de ejecución, son los siguientes:

1. *sudo apt update*
2. *sudo apt upgrade*
3. *sudo apt install python3*
4. *sudo apt install -y python3-pip*
5. *sudo apt intsall -y git*
6. *sudo apt install ros-noetic-desktop-full*
7. *echo “source /opt/ros/noetic/setup.bash" >> ~/.bashrc*

[1] Ubuntu 20.4: https://releases.ubuntu.com/focal/
[2] Guía instalación Wsl: https://learn.microsoft.com/en-us/windows/wsl/install
[3] Guía instalación XLaunch: http://www.straightrunning.com/XmingNotes/

8. *source ~/.bashrc*
9. *sudo apt install python3-rosdep python3-rosinstall python3-rosinstall-generator python3-wstool build-essential*
10. *sudo apt get ros-noetic-geographic-msgs*
11. *sudo apt get ros-noetic-hector-mapping*
12. *sudo apt-get install python3*
13. *sudo apt-get install python3-pip*
14. *pip install getch*
15. *sudo pip3 install tornado*
16. *sudo pip3 install simplejpeg*
17. *sudo pip3 install rospkg*

Una vez instaladas todas las librerías necesarias, se tendría que crear un nuevo workspace (entorno de trabajo) de ROS. Para ello ejecutaremos los siguientes comandos de forma ordenada y con la ruta que deseemos que resida nuestro proyecto, dicha ruta se sustituirá en las siguientes comandas donde aparezca <YOUR-WORKSPACE-ROUTE>:

1. *mkdir <YOUR-WORKSPACE-ROUTE>/src*
2. *cd <YOUR-WORKSPACE-ROUTE>*
3. *catkin_make*
4. *cd ~/src*
5. *git clone https://github.com/SergioFerrando/DroneSimulator_TFG.git*
6. *cd <YOUR-WORKSPACE-ROUTE>*
7. *catkin_make*
8. *echo “source <YOUR-WORKSPACE-ROUTE>/devel/setup.bash”> ~/.bashrc*

Una vez ejecutados estos comandos, primero, deberíamos cerrar el terminal abierto y después fijarnos en si tenemos una estructura de carpetas igual a la mostrada en la *ilustración 25*.

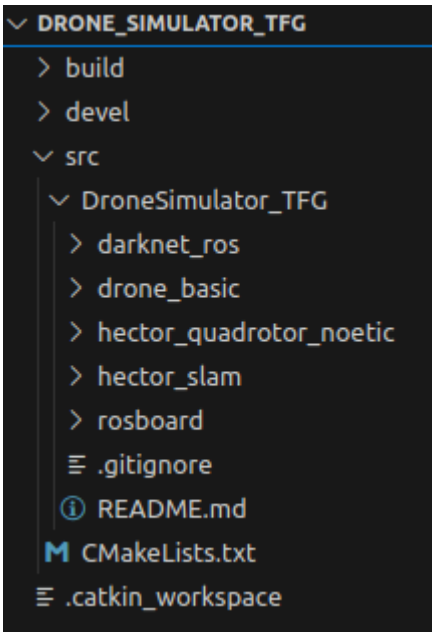

Ilustración 25. Estructura espacio de trabajo creado.

Fuente: Elaboración propia.

Si tenemos la estructura de la *ilustración 25,* podríamos pasar a la fase de ejecución, para esta fase existen múltiples opciones dependiendo de cuál de las funcionalidades implementadas se quiera ejecutar. Se pueden ejecutar cada una por si sola o de forma conjunta. Para ello se han creado diferentes archivos launch que nos permitirán facilitar estas ejecuciones.

A continuación, se explicarán las diferentes formas que existen de ejecutar las funcionalidades y como ejecutar cada una de ellas.

Comandos de ejecución

Ejecutar inicio simulación (necesario para los demás comandos):

roslaunch drone_basic bring_hector_drone.launch

Ejecutar conducción dron:

Conducción por teclado -> *rosrun drone_basic conducir_drone.py*

Conducción por Interfaz gráfica -> *rosrun drone_basic ui_hector_quad.py*

Ejecutar YOLO:

roslaunch darknet_ros darknet_ros.launch

Ejecutar WEB:

rosrun rosboard rosboard_node

Ejecutar Mapeo (no ejecutar el comando de inicio de simulación):

roslaunch hector_slam hector_slam.launch

Para mover el dron utilizar las implementaciones de conducción mencionadas con anterioridad.

Una vez el mapa este creado guardar con el comando:

rosrun map_server map_saver -f <FilePath>

Ejecutar navegación (no ejecutar el comando de inicio de simulación):

roslaunch hector_slam hector_navigation.launch

Antes de publicar un punto elevar el dron utilizando las implementaciones de conducción mencionadas anteriormente. Para cambiar el mapa a utilizar. Cambiar la ruta en el archivo *hector_navigation.launch* de dentro del espacio de trabajo.

El código donde están todas las implementaciones del trabajo realizadas juntamente con una pequeña guía de comandos puede encontrarse en el siguiente enlace: https://github.com/SergioFerrando/DroneSimulator_TFG.

9. Bibliografía

Drone24Hour (2018, 27 abril). COME SCEGLIERE I MOTORI BRUSHELLES PER IL DRONE - D24H. Drone24Hours. https://www.drone24hours.com/Blog/C%C3%B3mo-elegir-motores-para-el-dron-de-carreras-quadcopter/?lang=es

Velasco, K. R. (2021, 10 diciembre). YOLO (You Only Look Once) - Towards Data Science. *Medium*. https://towardsdatascience.com/yolo-you-only-look-once-17f9280a47b0

Acre_admin. (2022, 8 abril). *Tecnología dron en agricultura de precisión. DJI AGRICULTURE | Grupo Acre España*. Grupo Acre España. https://grupoacre.es/tecnologia-dron-en-agricultura-de-precision-dji-agriculture/

RC Groups - View Single Post - Racerstar Racing Edition 2306 BR2306S 2400/2700kv 2305 BR2305S 2400/2600kv. (s. f.). https://www.rcgroups.com/forums/showpost.php?s=8bf157d31999d847a7c5335a7f30d87b&p=37474096&postcount=3

Del Drone, E. V. (2022). Cómo volar un drone: guía práctica para principiantes. El vuelo del Drone. https://elvuelodeldrone.com/blog-de-drones/como-volar-un-drone/

Tu-Darmstadt-Ros-Pkg. (s. f.). GitHub - tu-darmstadt-ros-pkg/hector_quadrotor: hector_quadrotor contains packages related to modeling, control and simulation of quadrotor UAV systems. GitHub. https://github.com/tu-darmstadt-ros-pkg/hector_quadrotor

Leggedrobotics. (s. f.). GitHub - leggedrobotics/darknet_ros: YOLO ROS: Real-Time Object Detection for ROS. GitHub. https://github.com/leggedrobotics/darknet_ros

Dheera. (s. f.). GitHub - dheera/rosboard: ROS node that turns your robot into a web server to visualize ROS topics. GitHub. https://github.com/dheera/rosboard

Documentation - ROS Wiki. (s. f.). http://wiki.ros.org/

Osrf. (s. f.). Gazebo : Tutorial : Installing gazebo_ros_pkgs (ROS 1). https://classic.gazebosim.org/tutorials?tut=ros_installing&cat=connect_ros

Redmon, J. (s. f.). YOLO: Real-Time Object Detection. https://pjreddie.com/darknet/yolov2/