Universidad Rey Juan Carlos

GRADO EN INGENIERÍA ROBÓTICA DE SOFTWARE

Escuela de Ingeniería de Fuenlabrada

Curso académico 2019-2023

Trabajo Fin de Grado

Ejercicios con el drone Tello real en la plataforma RoboticsAcademy

Autor: Guillermo Bernal Ruiz

Tutor: Jose María Cañas Plaza

Agradecimientos

A mis padres,
resto de la familia y amigos

Resumen

Uno de los campos que más ha crecido y más crecimiento se le augura es el de la robótica. La robótica es un campo en constante evolución y por ello es imprescindible educar a los nuevos alumnos tanto en ingenierías relacionadas con este campo como adolescentes que cursan educación secundaria sobre robótica, para que sean protagonistas de este crecimiento. Por ello, la robótica educativa toma un papel primordial en el desarrollo de la robótica misma, para seguir formando a los futuros ingenieros robóticos que puedan afrontar este futuro próximo.

El objetivo de este TFG parte de esta premisa, y trata de aportar dos nuevos ejercicios en la plataforma sobre aprendizaje en robótica aérea, Robotics Academy. Estos dos nuevos ejercicios hacen uso del dron Tello como robot que servirá como agente para resolverlos, con ellos se acercarán al usuario dos aspectos: la familiarización con robots reales y acercamiento de la robótica aérea. Esto permitirá aumentar los conocmientos y autonomía del alumno en robótica aérea. Para conseguir este objetivo habrá que conseguir primero poder manipular mediante *drivers* este dron y posteriormente integrarlo a la arquitectura de Robotics Academy.

Han sido desarrollados los drivers necesarios que permitan una comunicación con el dron usando el *middleware* ROS2 en su versión más reciente, Humble. Tras eso se ha integrado dentro de la versión más reciente de Robotics Academy, RADI 4, creando así su plantilla python de funciones, página web y documentación necesaria para el usuario que acepte el reto de resolver los ejercicios.

Estos ejercicios propuestos son dos; el objetivo del primero es que el usuario despegue el dron, dibuje un cuadrado con su trayectoria y vuelva a aterrizar en el mismo sitio del que despegó. El segundo ellos tiene como objetivo ser capaz de seguir a una persona a través de la cámara incorporada en el dron.

Acrónimos

TFG *Trabajo Final de Grado*

ROS *Robotics Operating System*

YOLO *You Only Look Once*

IMU *Inertial Measurement Unit*

UDP *User Datagram Protocol*

SDK *Software Development Kit*

DOM *Document Object Model*

QoS *Quality of Service*

RADI *Robotics Academy Data Infrastucture*

DDS *Data Distribution Service*

CUDA *Compute Unified Device Architecture*

HAL *Hardware Abstraction Layer*

HAL *Hardware Abstraction Layer*

GUI *Graphical User Interface*

Índice general

Índice de figuras

Listado de códigos

Listado de ecuaciones

Capítulo 1
Introducción

En este primer capítulo vamos a describir tanto el contexto histórico como el estado actual del campo de la Robótica, que es el marco que engloba a este TFG y el grado cursado. Al tratarse este proyecto de la implementación de ejercicios de drones dentro de una plataforma educativa, hablaremos de los dos campos que intersecan en la realización de este; la robótica aérea y la robótica educativa. Estudiaremos y hablaremos de la plataforma donde estos ejercicios serán añadidos, Robotics Academy.

1.1. Robótica

La robótica es la disciplina que abarca áreas como ingeniería mecánica, informática, electrónica y la inteligencia artificial para diseñar, construir y operar con robots. Estos robots son sistemas autónomos o semiautónomos que pueden realizar tareas en entornos controlados o no controlados, a menudo replicando habilidades humanas. Desde sus primeras etapas en el siglo XX, cuando surgía la problemática de la automatización de tareas industriales, la robótica ha evolucionado para abordar una amplia gama de aplicaciones que incluyen desde robots quirúrgicos y drones de entrega hasta exploradores espaciales y asistentes de inteligencia artificial.

La robótica se ha beneficiado enormemente de los avances en tecnología de sensores, actuadores y algoritmos de aprendizaje automático. Los sensores permiten a los robots recopilar información sobre su entorno, mientras que los actuadores le confieren la movilidad y capacidad de interactuar con ese entorno. El aprendizaje automático y la inteligencia artificial añaden una capa adicional de complejidad al permitir que los robots se adapten y mejoren su rendimiento en función de la retroalimentación de sus acciones. Esto ha llevado a una mayor autonomía y capacidad de toma de decisiones, lo que permite a los robots realizar tareas cada vez más complejas y sensibles.

El impacto de la robótica en la sociedad tiene muchas ramas de evolución: medicina, exploración, seguridad y educación por ejemplo.

1.1.1. Evolución histórica

En sus inicios, durante la primera mitad del siglo XX, los robots eran principalmente sistemas mecánicos diseñados para tareas muy específicas, como la automatizacion de líneas de ensamblaje en la industria automotriz.

Ejemplo de ello es el Unimate, el primer robot industrial, que se introdujo en una fábrica de *General Motors* en 1961 para automatizar tareas peligrosas como la manipulación de piezas fundidas a alta temperatura.

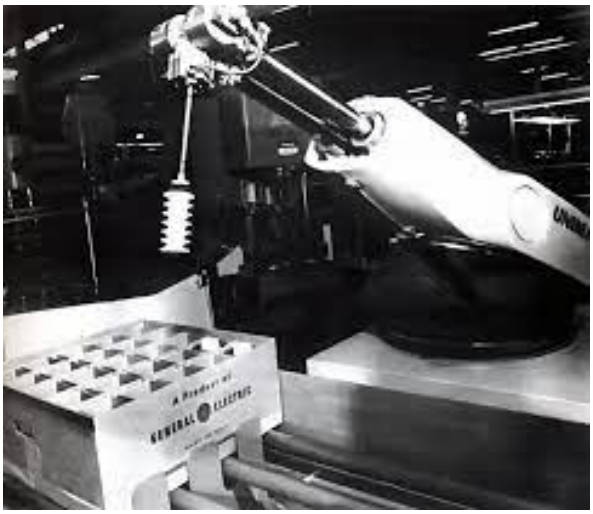

Figura 1.1: Robot Unimate

Otro de los robots de la misma década es el robot Tortuga, un robot educativo diseñado por Seymour Papert, este robot era capaz de dibujar formas geométricas y ayudó a sentar las bases para la educación en robótica.

Figura 1.2: Robot Tortuga

Con el advenimiento de la microelectrónica y la informática en las décadas de los 70 y 80, los robots se volvieron más verátiles y autónomos. Se desarrollador robots

móviles como el Stanford Cart, que puede navegar a través de una habitación llena de obstáculos en 1979.

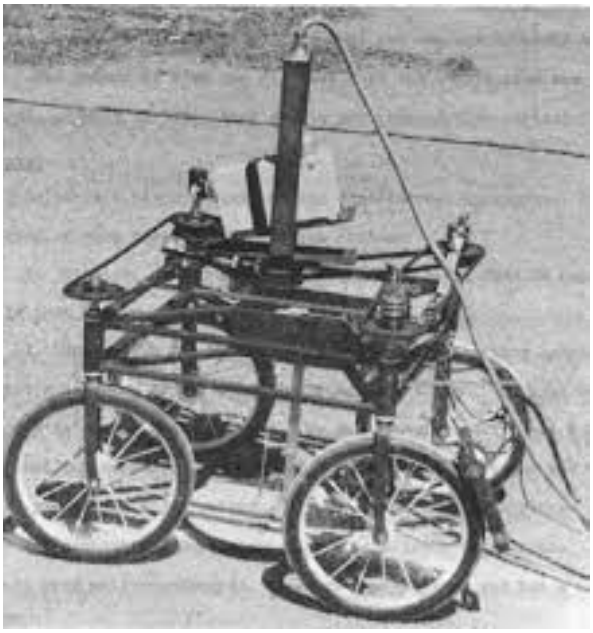

Figura 1.3: Robot Stanford Cart

El robot PUMA (Programmable Universial Machine for Assembly) se convirtió en un pilar en laboratorios de investigación y fábricas.

Figura 1.4: Robot PUMA

En el campo de la medicina, el robot *da Vinci* hizo su debut en el año 2000, permitiendo a los cirujanos realizar procedimientos complejos con mayor precisión y control.

Figura 1.5: Robot da Vinci

En la década de 2010, también surgió un avance en la robótica de servicio con robos como el Roomba para la limpieza del hogar.

Figura 1.6: Robot Roomba

Y con más actualidad robots como el Pepper diseñado para interactuar con humanos o los robots modelo de Boston Dynamics, como el Atlas, que puede realizar una gran variedad de movimientos y acrobacias.

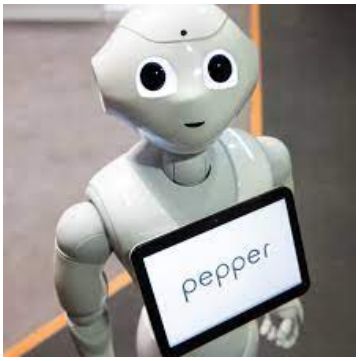

Figura 1.7: Robot Pepper

1.1.2. Contexto actual

El contexto actual de la robótica está muy protagonizado por el aprendizaje automático y cada vez más ramas de exploración están en desarrollo, algunas de estas

son:

1. Conducción Autónoma: Esta rama está avanzando rápidamente con empresas como Tesla y Waymo a la vanguardia. El desarrollo de algoritmos sofisticados y sensores de alta precisión está llevando a la conducción autónoma más cerca de ser una realidad cotidiana.

Figura 1.8: Coche Tesla

2. Robots domésticos y de almacén: No solo encontramos robots como Roomba en tareas de limpieza doméstica, sino que también se están implementando en atención al cliente y en logística. Empresas como Amazon utilizan robots para la selección y empaquetado de productos en sus almacenes.

Figura 1.9: Robot de almacén de Amazon

3. Robótica Industrial: Esta rama ha sido fundamental en la automatización de tareas de fabricación y producción. Los avances en inteligencia articial están permitiendo a estos robots tomar decisiones más elaboradas y realizar tareas más complejas.

1.2. Robótica Aérea

La robótica móvil es una subdisciplina que se centra en el desarrollo de robots capaces de moverse y navegar en diferentes entornos. Este campo ha experimentado un auge considerable en los últimos años, gracias a los avances en sensores, algoritmos de navegación y tecnologías de comunicación. Los robots móviles se utilizan en una amplia gama de aplicaciones que van desde la exploración espacial, como los rovers de Marte, hasta tareas más cotidianas como la limpieza de pisso con robots como Roomba.

Una de las áreas es la de los robots móviles para tareas de búsqueda o seguimiento, ya sea para rescate o seguridad. Estos robots deben ser capaces de tener la suficiente autonomía como para moverse sólo por entornos sin prácticamente seguridad, así como permitir gran rango de movimiento que nos lleven casi a cualquier lugar y sean capaces de realizar un seguimiento robusto y óptimo ayudándose de sensores como la cámara. Ejemplo de ello es la robótica aérea.

La robótica aérea se centra en el desarrollo y aplicación de drones o vehículos aéres no tripulados. Esta rama ha recibido un gran impulso de debido a sus avances en miniaturización de componentes, algoritmos de control y técnicas de fusión sensorial. Los drones se han vuelto más accesibles y versátiles, lo que ha llevado a crear máquinas capaces de volar de manera autónoma o semi-autónoma.

Uno de los requisitos condicionales de la robótica aérea que la diferencia del resto de campos de la robótica, es la gestión de energía, dado que la duración de la batería es a menudo una limitación en operaciones prolongadas.

Aun así, la versatilidad y movilidad de los drones abren la puerta a numerosas aplicaciones, tanto en ambientes interiores como exteriores. Se están desarrollando algoritmos avanzados de navegación y detección para pemitir vuelos más seguros y eficientes. Además de ello, se está haciendo uso de la inteligencia artificial para que tareas como la de este TFG, de seguimiento, sea posible y sobretodo más robusta que programada con algoritmos clásicos de visión artificial, por ejemplo.

1.3. Robótica Educativa

Otros de los pilares de este TFG es la robótica educativa. Esta se centra en el uso de robots como herramientas pedagógicas para facilitar el aprendizaje y el desarrollo de habilidades en estudiantes de todas las edades. Esta área utiliza los robots para enseñar conceptos que van desde las matemáticas y ciencia hasta la programación y pensamiento lógico. La idea es que la interacción con robots puede hacer que el aprendizaje sea más atractivo y práctico, lo que desemboca en un aprendizaje y comprensión más profundos.

La gama en robots educativos es muy amplia, se ofrecen herramientas de una forma interactiva para que el alumno reciba conceptos complejos pero esenciales de forma práctica de: mecánica, electrónica o desarrollo de *software*. Además la robótica educativa fomenta el trabajo en equipo y la comunicación entre compañeros con el fin de conseguir un objetivo común.

Para intersecar los puntos que se unen para la creación de este TFG, vamos a hablar de ejemplos de plataformas web de robótica educativa dedicada a la robótica

aérea, algunos de estos ejemplos son:

- `TheConstruct`[1]: Cuenta con cursos sobre robótica empezando sobre las bases de la misma en la actualidad. Cursos de ROS2, Python3 para robótica, ROS2 Navigation o sobre el simulador Gazebo. También cursos intermedios sobre ROS2 como navegación avanzada, aplicaciones distribuidas, o entendimientos de las transformadas. Y cursos avanzados como árboles de comportamiento, manipulación y percepción y sobre seguridad en ROS2. Pero también ofrece contenido teórico sobre el que se sustentan todos estos cursos, matemáticas para robótica, dinámica, kinemática o filtros de Kalman.

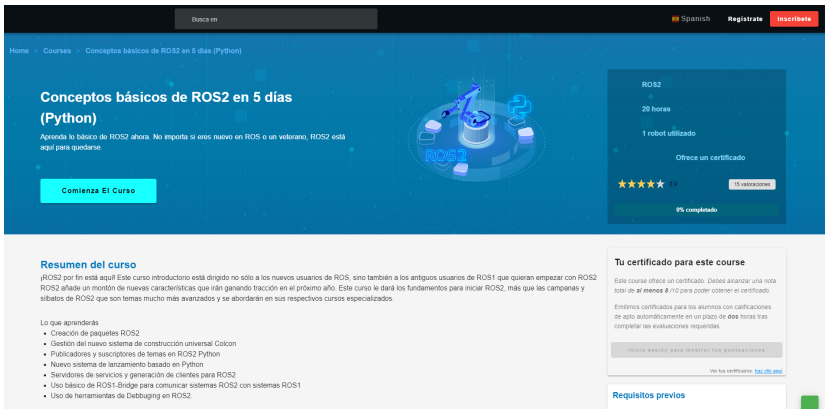

Figura 1.10: Muestra de curso en TheConstruct

- `Riders.ai`[2]: Esta no es solo una plataforma para aprender, sino para competir mediante la programación de robots. Mediante la competición fomenta aprender sobre programación y sobretodo robótica.

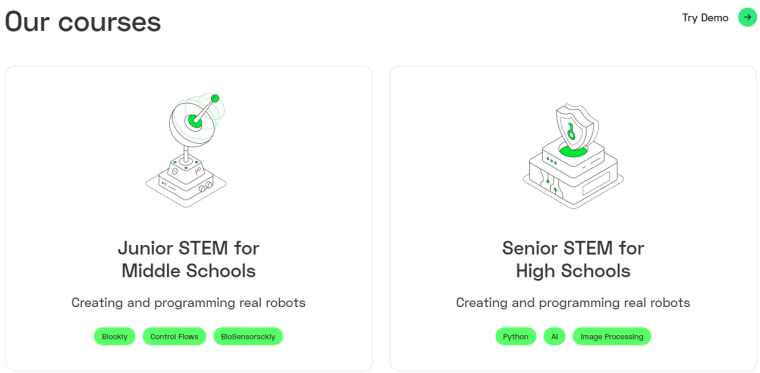

Figura 1.11: Muestra de cursos disponibles en Riders

[1] `https://www.theconstructsim.com/robotigniteacademy_learnros/ros-courses-library/`
[2] `https://riders.ai/en`

- Curso de drones de TUM[3] (Universidad Técnica de Munich): Son cursos sobre robótica creados por la universidad de Munich para que sus alumnos puedan aprender sobre robótica aérea. Cuenta con diferentes cursos de robótica aérea como son el *UAV Operations Lab* o *MAVs Micro Aerial Vehicles*.

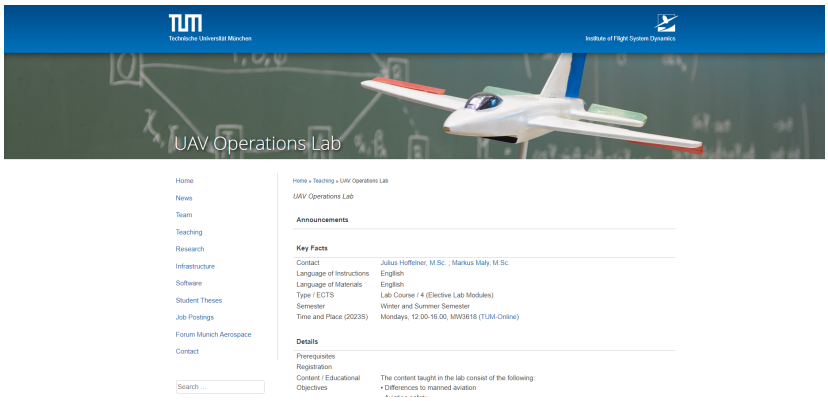

Figura 1.12: Muestra de curso en TUM

1.3.1. Robotics Academy

Robotics Academy es un repostorio[4] mantenido por la asociación JdeRobot donde se encuentran distintos ejercicios dentro del entorno de la robótica educativa. Cuenta con un gran número de ejercicios donde, a través de la educación, se exploran diferentes ramas de la robótica, como la robótica móvil o la robótica de servicios.

Robotics Academy, al ser un repositorio público de GitHub, es una colección de ejercicios de código abierto, por lo tanto, cualquier desarrollador interesado en la robótica educativa puede unirse a este desarrollo y seguir ampliando o mejorando la gama de ejercicios. Además, ofrece una gran facilidad para cualquier usuario que se preste a desarrollar un ejercicio o una solución a este, esto es debido a que corre en un entorno virtualizado. Todo Robotics Academy puede ser encapsulado dentro de un contenedor Docker, esto quiere decir que todos los robots (tanto simulados como físicos), sus drivers, simuladores, entornos de ejercicios, plantillas y páginas web de cada uno, se encuentra encapsulado dentro de una imagen Docker. Todas las dependencidas necesarias para empezar a tomar papel como desarrollador o alumno de Robotics Academy se resuelven a través de una simple imagen Docker, haciendo que el usuario no tome el papel de instalarse nada pero que a su vez, todo ejercicio corra en su máquina de manera nativa.

[3] `https://www.fsd.ed.tum.de/teaching/uav-operations-lab/`
[4] `https://jderobot.github.io/RoboticsAcademy/`

La página web de cada ejercicio en Robotics Academy incorpora básicamente 3 elementos: un editor de texto sobre el que pogramar, una visualización auxiliar de la cámara del robot y una terminal que permita depurar el código fuente mediante una conexión VNC con el contenedor.

Además, la página no solo reduce la instalación del entorno a una única instrucción, la página trata de reducir la complejidad de la programación de robots lo máximo posible. Encapsula toda la complejidad relacionada con los sensores o actuadres del robot a únicamente dos módulos:

- HAL: Una API que ofrece una capa de abstracción para acceder al *hardware* del robot. Tanto a sus sensores como actuadores.
- GUI: Una interfaz gráfica para que el usuario pueda visualizar aspectos relevantes del ejercicio, esto puede ser por ejemplo, un mapa.

1.3.2. Unibotics

Unibotics es la plataforma web en línea donde se incorpora toda la gama de ejercicios de Robotics Academy. Al tratarse de una página web, da pie a usuarios de todo el mundo a disfrutar y aceptar el desafío de cada uno de los ejercicios siguiendo su misma filosofía de poder jugar con ellos sin instalarse nada.

Unibotics cuenta con dos modos de funcionamiento principales: RADI 3, donde sus ejercicios estarán basados en ROS Noetic y RADI 4, donde existen ejercicios nuevos y antiguos migrados a la nueva versión de ROS, ROS2 Humble. Recientemente, cuenta con la opción de la ejecución en remoto, por lo que no sería necesario descargar y correr la imagen Docker de Robotics Academy en local. La idea de los nuevos ejercicios propuestos en este TFG es incorporarlos en un futuro a RADI 4, haciéndolo más sostenible en el tiempo en la plataforma.

Capítulo 2
Objetivos y Metodología

En la introducción se ha hablado del contexto en el que se encuentra este trabajo de final de grado compuesto principalmente: la robótica, la inteligencia artificial y las plataformas educativas sobre los dos anteriores mencionados. Ahora se abordará que objetivo o meta que tiene sobre estas bases.

2.1. Objetivos

El objetivo general de este TFG es el uso de drones en la plataforma web de la educación robótica de Unibotics a través de dos ejercicios diseñados para diferentes niveles de conocimientos. Estos ejercicios a desarrollar se deben sustentar en una plataforma sobre la que todo alumno que quiera aprender cómo manejar un dron, pueda hacerlo fácilmente. Esta base de la que hablamos no es otra que la plataforma Robotics Academy. El primero de los ejercicios será un ejercicio introductorio y/o dedicado a las personas que quieren familiarizarse con la robótica aérea, consistirá en que el Dron Tello sea capaz de despegar, dibujar un cuadrado con su trayectoria y aterrizar en el mismo sitio del que despegó. El segundo aumenta en complejidad y conocimientos mínimos para resolverlo ya que tratará sobre la tarea de seguir a una persona. Para lograr esta meta, vamos a dividir el objetivo general en pequeños subobjetivos:

1. Adentrarse y entender la infraestructura de Robotics Academy y, desarrollar nuestros propios ejercicios con todos los pasos que lo componen.

2. Crear una infraestructura que soporte del robot Dron Tello en ROS2 Humble. Anteriormente no existía ningún soporte para ROS2 Humble funcional de este dron. Usar este soporte a través de un contenedor Docker. El dron Tello tiene su propio método de comunicación Wi-Fi y tenemos que enviarnos paquetes con él desde dentro del contenedor.

3. Crear los dos ejercicios en Unibotics, incluyendo su página web y su plantilla python.

4. Crear una solución referencia para ambos ejercicios. Esto servirá al alumno como demostración de que es posible resolver el ejercicio y le servirá como guía para intentar encontrar una solución.

2.2. Metodología

Con el fin de trabajar uniformememente en este TFG, resolver dudas lo más rápido posible y que quede constancia del trabajo realizado:

- Para el seguimiento del tutor sobre el TFG se han realizado reuniones semanales en las que se resolvían dudas, se comentaban avances respecto a la anterior semana y se planteaban próximas rutas para el desarrollo del TFG, sirviendo el tutor como retroalimentación para todas estas.

- Las reuniones fueron mediante videollamada o presencialmente. Pero para un contacto más rápido y con dudas más sencillas, se ha usado Slack. Esto también sirve para establecer método de comunicación con el tutor, sino con el resto del equipo de desarrolladores que compone Robotics Academy.

- Con el fin de dejar constancia sobre lo que se ha trabajo y como método para resolver viejos problemas ya abordados, he redactado un blog semanal donde se explicaba, con más detalle, todos los avances que ha habido semana a semana.

- Acorde con el blog, también se ha creado un repositorio de GitHub dentro del grupo de RoboticsLabURJC, donde subir todos los cambios en cuanto a código se refiere.

2.3. Plan de trabajo

Para poder desarrollar el TFG han sido necesarios pasos previos que sirvan como sustento para la idea final. La planificación sobre cómo ir escalando a lo largo del TFG ha sido la siguiente:

1. El primer paso a seguir es entender la arquitectura de Robotics Academy, cómo se comunican Webserver con el manager, cómo se integran nuevos ejercicios o como correr Robotics Academy en local a través de un miniRADI para poder probar las implementaciones.

2. Tras habernos familiarizado con el entorno de Robotics Academy como para entender su funcionamiento, pasamos al siguiente paso, participar en la migración de Robotics Academy de ROS1 Noetic a ROS2 Humble. Para este paso ha sido muy importante que el anterior quede bien afianzado.

3. Una vez conseguido el contexto sobre el que va a integrarse el ejercicio quedaba un paso más. Aprender a integrar un driver de ROS2 Humble que permita encapsular las comunicaciones de imágenes y comandos con el Tello a través de comandos de ROS2.

4. La implementación del ejercicio Dron-Cuadrado. Para ello hay que integrar esos drivers creados dentro del contenedor docker que compone Robotics Academy y crear el frontend y la plantilla de funciones.

5. La segunda implementación parte de la primera, es la del ejercicio Sigue-Persona con el Tello. En esta se añaden más funciones para comandar al Tello, además de la librería YOLO para permitir identifcar personas en una imagen. Y por último, se modifica el resto de la estructura de Robotics Academy para permitir más reactividad que sería necesaria para este nuevo ejercicio.

6. Para finalizar con la parte de desarrollo, falta poner en práctica todo lo integrado, y la mejor forma de comprobar que funciona correctamente es creando las soluciones referencia de ambos ejercicios.

7. Y por último, la redacción de la presente memoria del Trabajo de Fin de Grado.

Capítulo 3
Infraestructura

En este capítulo se explicarán todos los elementos que han sido necesarios para este TFG y las herramientas que usaremos para el desarrollo del mismo.

3.1. Tello

El dron Tello es uno de los múltiples robots de los que se dispone dentro de RoboticsLabURJC en la Universidad Rey Juan Carlos.

El Tello es una de las principales opciones a la hora de buscar robots aéreos y orientados a la educación. Si tomamos en cuenta que pesa apenas 80 gramos y sus dimensions 98x92x41 mm, es cómodo para usar y manipular. Además cuenta con la aplicación oficial distribuida por Ryze, su fabricante, donde se nos ofrece una interfaz para poder interactuar con el dron.

Nos ofrece diferentes modos de vuelo del Tello, con transmisión a tiempo real de lo que está viendo nuestro dron, además de permitir grabar todo aquello que ve y tenerlo como vídeo disponible.

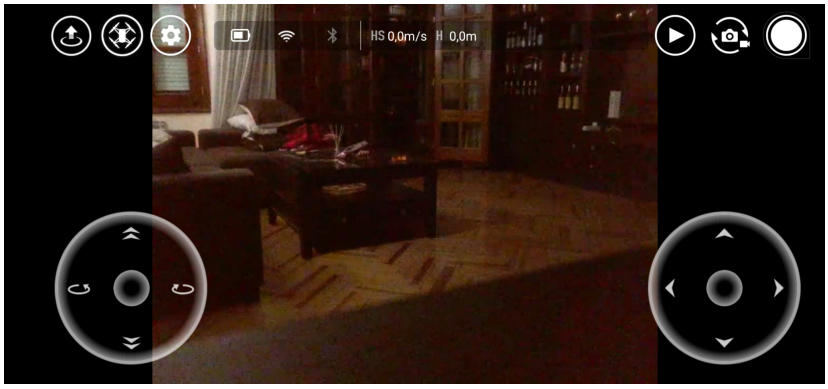

Figura 3.1: Aplicación oficial del Tello

Esta app además sirve como fuente principal para nuevas actualizaciones del Firmware o para calibrar la IMU (unidad de medición inercial) y el centro de gravedad

del dron.

En mi caso lo usé principalmente para familiarizarme con el uso de drones y para calibrar la IMU más de una vez.

Cuenta con una batería desmontable de 2.8V. Su duración es de unos diez a trece minutos dependiendo de como se use el dron y cuanto de ese tiempo esté volando. Su tiempo de carga es de unos treinta minutos, dependiendo de su uso.

3.1.1. Conexión Wi-Fi

La comunicación con el dron Tello se realiza a través de una conexión Wi-Fi. Desde el momento del encendido del Tello, se emite una señal Wi-Fi propia y permite a los dispotivos conectarse a él.

Utiliza como protocolo de comunicación el protocolo UDP. Esto significa que no garantiza la entrega de paquetes pero dentro de un dron se obtiene una ventaja fundamental, velocidad y eficiencia, lo cual permite rápida reacción a nuevos comandos para trabajar con el dron a tiempo real.

El Dron escucha por varios puertos:

- Puerto 8890: Desde este puerto se envía la información general del dron: duración de la batería restante, velocidad, estado, etc... El dron no para de usar este puerto para envíar información en todo momento.
- Puerto 11111: Este puerto es usado por el Tello para el envío del fujo de video. Con esto conseguimos dar pie a varias aplicaciones con visión por computadora como la que se usará en uno de los ejercicios
- Puerto 8889: Es el puerto dedicado a enviar y recibir comandos de control del Tello. Es decir, sobre este se comanda una acción para el Tello y este te responde con tres tipos de mensajes:
  - Valor de respuesta '1': El comando se ha recibido y se va a ejecutar con normalidad.
  - Valor de respuesta '2': Ha habido algún tipo de error con el comando, por ejemplo, comando desconocido.
  - Valor de respuesta '3': Avisa al usuario de que en este momento se encuentra ocupado y va a desechar el mensaje. Esto ocurre, por ejemplo, si le comandas al Tello un giro en medio de una operación de despegue.

3.1.2. Cámara

La cámara del Tello cuenta con un sensor de 5 megapíxeles con el que captura imágenes hasta una resolución de 2592x1936 píxeles. Cuenta con un ángulo de visión de 82.6 grados. Un aspecto interesante para abordar el ejercicio del Sigue-Personas es que también puede capturar imágenes a 720p (1280 x 720) a 30 fotogramas por segundo. Con esto tenemos ya la primera limitación a la que actuará el seguimiento de la persona. Pues lo máximo a lo que reaccionará el dron en base a lo que vea será a 30 hercios de frecuencia.

El dron Tello también permite grabar y guardar en vídeo todo lo que ve a través de comandos que se mencionarán en el siguiente apartado.

Figura 3.2: Aspecto del Tello

3.1.3. Comandos del Tello

Los motores del Tello son motores sin escobillas de corriente continua. El empuje máximo es de unos 1.8kg entre los cuatro motores. Cuenta con una velocidad de rotación de 10.000 revoluciones por minuto y controladores electrónicoas de velocidad incorporados.

Para controlar estos motores se usa principalmente el kit de desarrollo software (SDK) que permite a los programadores salirse del contexto de la aplicación móvil y enviar comandos directamente al dron para controlar su comportamiento. Dentro de las posibles versiones del SDK, usado para este TFG es el SDK 1.3. Existe hasta la versión 2.0 actualmente. Algunos de los principales comandos usando el SDK son:

- Comandos de control:
  - `takeoff`: Este comando hace que los motores se aceleren para generar suficiente empuje y levantar el dron del suelo.

- `up x`: Hace que el dron suba una distancia x en cm.
- `cw x`: Rota el dron en sentido horario x grados.
- `land`: Este comando desacelera los motores gradualmente y permite que el dron aterrice de manera segura.

- Comandos de lectura:
  - `speed`: Se recibe la velocidad actual del dron (cm/s).
  - `battery`: Se obtiene el porcentaje de batería restante.
  - `time`: Devuelve el tiempo de vuelo que lleva el dron.

3.2. Lenguajes de programación

Gran parte de trabajo que se ha llevado a cabo para la realización de este TFG ha sido por medio de la programación. Con el fin de establecer una infraestructura sobre la que tener los dos ejercicios que vamos a añadir y el desarrollo de estos, los dos lenguajes que más se han usado han sido Python y C++.

3.2.1. Python

Python[1]. es un lenguaje de programación de alto nivel que es ampliamente utilizado para una variedad de aplicaciones, desde servidores web y análisis de datos hasta inteligencia artificial y aprendizaje automático. Se caracteriza por su sintaxis clara y legible, lo que facilita el aprendizaje para los nuevos programadores. Python también cuenta con una amplia gama de bibliotecas y entornos que amplían sus capacidades, permitiéndote hacer más cosas con menos código.

```python
# Dos números para sumar
num1 = 5
num2 = 10

# Calcula la suma de ambos números
suma = num1 + num2

# Muestra el resultado
print("La suma de", num1, "y", num2, "es", suma)
```

Figura 3.3: Código de ejemplo de Python

[1] `https://www.python.org/`

El papel del desarrollo en Python para este TFG ha sido muy importante, esto es debido a que Robotics Academy, la parte de backend y comunicaciones, está desarrollada principalmente en este lenguaje. Algunas de las librerias principales usadas dentro de la página pueden ser: *sys, subprocess, json, os* o *pylint*. La versión usada de Python para este TFG ha sido la 3.10.

3.2.2. JavaScript y React

JavaScript[2] es un lenguaje de programación de alto nivel que se utiliza principalmente para el desarrollo web, especialmente para agregar interactividad y dinamismo a las páginas web. Aunque originalmente se diseñó para funcionar en el navegador web del cliente, su uso se ha expandido a servidores, dispositivos móviles y otros entornos gracias a plataformas como Node.js. JavaScript es versátil y cuenta con una amplia gama de bibliotecas y entornos que facilitan el desarrollo de aplicaciones más complejas.

```javascript
// Dos números para sumar
const num1 = 5;
const num2 = 10;

// Calcula la suma de ambos números
const suma = num1 + num2;

// Muestra el resultado en la consola
console.log("La suma de " + num1 + " y " + num2 + " es " + suma);
```

Figura 3.4: Código de ejemplo de JavaScript

React[3] es una biblioteca de JavaScript desarrollada por Facebook que se utiliza para construir interfaces de usuario, principalmente para aplicaciones web de una sola página. Lo que hace que React sea especialmente poderoso es su sistema de componentes reutilizables y su manejo eficiente del DOM (Document Object Model). Al utilizar React, puedes crear aplicaciones web interactivas con una experiencia de usuario fluida, gracias a su sistema de *Virtual DOM* que optimiza la renderización de componentes.

Dentro de este proyecto toma un papel principal, ya que toda la parte de Frontend está realizada en JavaScript usando React. Y no ha sido excepción para la creación de los nuevos ejercicios implementados en este TFG. La versión utilizada para JavaScript ha sido ECMAScript 6 y para React 0.72.2.

[2] https://developer.mozilla.org/es/docs/Web/JavaScript
[3] https://es.react.dev/

3.2.3. C++

C++[4] es un lenguaje de programación de propósito general que se extiende desde C, añadiendo características como el paradigma de programación orientada a objetos. Se utiliza en una variedad de aplicaciones que van desde sistemas embebidos y desarrollo de sistemas operativos hasta software para aplicaciones de escritorio, videojuegos y servidores. C++ es conocido por su rendimiento y ofrece un control muy preciso sobre los recursos del sistema, lo que lo hace popular para aplicaciones que requieren alta eficiencia.

```cpp
#include <iostream>

int main() {

    // Dos números para sumar
    int num1 = 5;
    int num2 = 10;

    // Calcula la suma de ambos números
    int suma = num1 + num2;

    // Muestra el resultado
    std::cout << "La suma de " << num1 << " y " << num2 << " es " << suma << std::endl;

    return 0;
}
```

Figura 3.5: Código de ejemplo de C++

Este lenguaje ha servido de soporte para el desarrollo de los ejercicios, para la realización del driver del Tello en ROS2 Humble o para programas complementarios para probar el funcionamiento del Tello con comandos a través de sockets o usando el driver de ROS2 implementado. La versión utilizada de C++ en este proyecto ha sido la 20.

3.3. Robot Operating System 2 (ROS2)

ROS2 (Robot Operating System 2) es un middleware para la robótica que proporciona una amplia variedad de funcionalidades esenciales para el desarrollo de software en sistemas robóticos. Esto incluye comunicación entre componentes del robot, abstracción de hardware, manejo de paquetes de software y una amplia variedad de herramientas y bibliotecas para tareas comunes en robótica.

La arquitectura de ROS 2 se centra en `nodos` que son entidades de procesamiento que realizan cálculos y pueden comunicarse entre sí. Estos nodos se organizan en un grafo de cómputo, que permite una comunicación flexible a través de un modelo de publicación-suscripción. En términos simples, algunos nodos publican mensajes

[4] `https://es.wikipedia.org/wiki`

en `topics`, mientras que otros nodos se suscriben a estos `topics` para recibir esos mensajes.

Para dar un ejemplo ilustrativo con el fin de entender este concepto, tenemos un nodo que mira personas detectadas en una cámara, el nombre del nodo puede ser `track_people_node`. Tenemos otro nodo que se preocupa de leer la información recibida de la cámara y si detecta una persona o más, hace saltar una alarma, este nodo es llamado `alarm_activation_node`. El topic sirve como lugar donde el primer nodo deja la información y el segundo coge esa información para actuar.

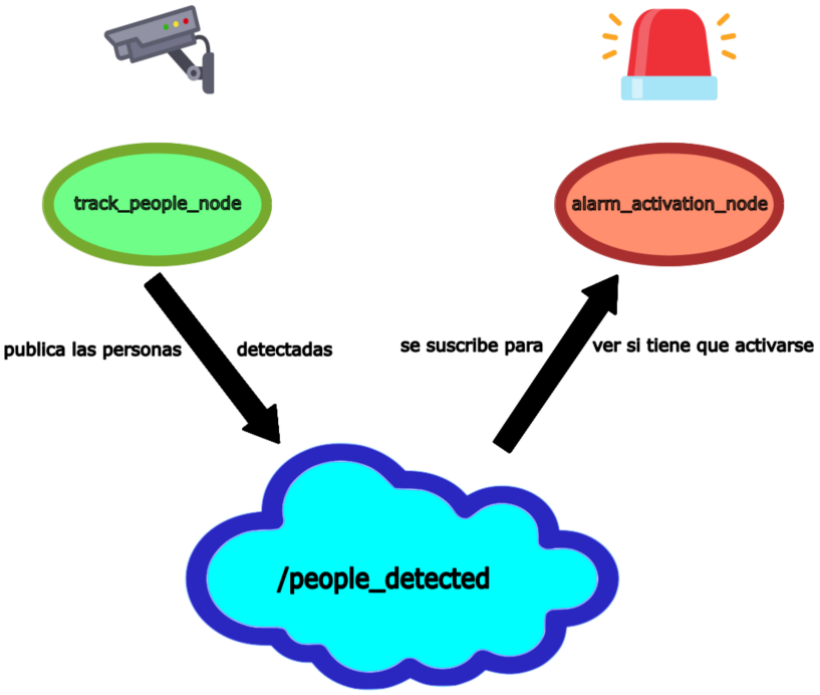

Figura 3.6: Esquema del funcionamiento de los topics

También en este TFG se hace uso de otra de las funcionalidades de ROS2, los servicios y clientes. En el contexto de ROS 2, los servicios y los clientes son mecanismos que permiten la comunicación síncrona entre nodos. Esto contrasta con la comunicación asincróna típica que se realiza a través de publicadores y suscriptores. En la arquitectura de servicios y clientes, un nodo ofrece un `servicio` que realiza alguna función específica y otro nodo actúa como `cliente` que solicita ese servicio. Cuando el cliente realiza una solicitud al servicio, espera una respuesta antes de continuar. Esta interacción de solicitud-respuesta es crucial para operaciones que necesitan ser atómicas o que requieren un cálculo antes de poder avanzar en el flujo de datos, como por ejemplo el despegue de un dron.

La comunicación entre nodos en ROS 2 se maneja mediante el middleware DDS (Data Distribution Service), que proporciona transmisiones robustas y eficientes para

la transferencia de datos. Esto permite, entre otras cosas, tener control sobre aspectos como la calidad del servicio (QoS) para ajustar la fiabilidad, durabilidad y otros parámetros de la comunicación entre nodos.

La programación de ROS2 puede ser realizada en dos de los lenguajes mencionados anteriormente, Python y C++. Con ellos se puede crear nodos, servicios o clientes.

Como es comprensible pensar, si hablamos de ROS2 es porque existe un ROS1. En el por qué se ha usado la primera opción de estas, una de las razones es porque se ha ligado al desarrollo del nuevo RADI para Robotics Academy, que está pensado para usarse con ROS2 Humble, siendo Humble la última versión y estándar actual de ROS2.

3.4. RVIZ2

RViz2 es la versión actualizada de RViz para ROS 2, y es una herramienta de visualización 3D que permite a los desarrolladores y usuarios ver y entender lo que está sucediendo dentro de un sistema robótico. Con RViz2, se pueden visualizar datos de sensores, estados del robot, planes de trayectoria, nubes de puntos y mucho más, todo ello en tiempo real. Esto es especialmente útil para la depuración, la monitorización y la representación de datos en sistemas robóticos desarrollados con ROS 2.

RViz2 retiene muchas de las funcionalidades que hicieron popular a su predecesor en ROS, pero también incorpora mejoras aprovechando la arquitectura y las capacidades de ROS 2. Por ejemplo, la comunicación entre RViz2 y otros nodos de ROS 2 se beneficia del middleware DDS, permitiendo una interacción más eficiente y confiable.

La herramienta ofrece una interfaz de usuario que permite a los usuarios personalizar la visualización mediante la adición y configuración de paneles y displays. Los paneles son esencialmente ventanas dentro de la interfaz que pueden mostrar diferentes tipos de información o controles, mientras que los displays son configuraciones que te permiten visualizar distintos tipos de datos, como imágenes de una cámara o la posición de un robot en un mapa.

En este proyecto se ha usado a menor escala como un método para poder ver en tiempo real el flujo de vídeo del Tello. Tras lanzar el `launcher` con el driver del Tello para ROS2, se elige dentro de RVIZ2 el topic que hace referencia a la imagen en crudo del Tello, y tras eso podremos verlo en una ventana nueva.

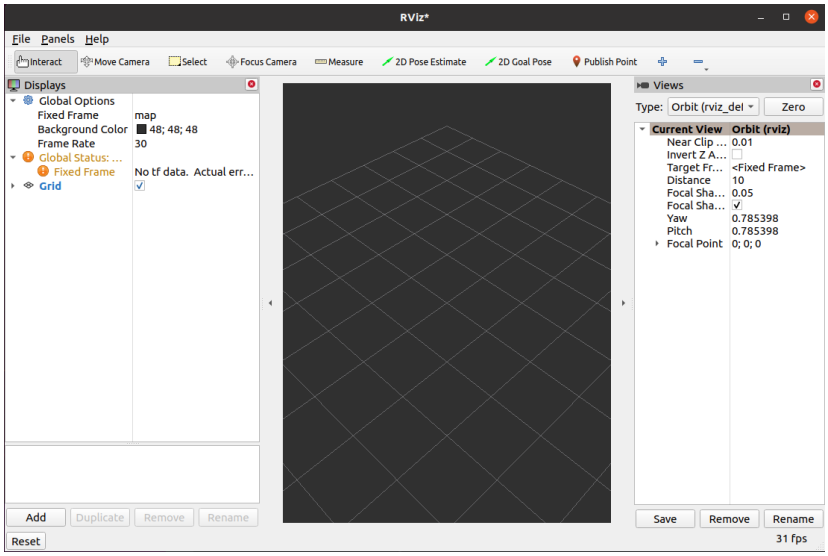

Figura 3.7: Interfaz de RVIZ2

3.5. Docker

Docker[5] es una plataforma abierta orientada al desarrollo de entornos virtualizados donde se añade una capa de abstracción extra al entorno en el que se trabaja. Propone separar el entorno habitual en el que se trabaja para poder desarrollar software en un entorno controlado donde especificas tú mismo el contexto en el que trabajas.

Para el TFG esto ha sido importante, ya que, mediante contenedores Docker se ha encapsulado toda la funcionalidad del dron y sus dependencias dentro de un contenedor. Esto sirve para reducir al mínimo posible los paquetes necesarios para usar el dron del modo que se hace en este TFG.

Otra ventaja es su portabilidad, una persona puede correr una aplicacion del mismo modo que lo harías tú tan solo usando su imagen docker o el fichero y construiselo por sí mismo posteriormente. En el contexto de Robotics Academy esto es muy importante y es una de las principales bases de su filosofía. Permite que cualquier persona, sin importar el sistema operativo en el que esté, pueda ejecutar el ejercicio del dron sin necesidad de instalarse nada directamente. Es decir, el usuario tan solo debe bajarse la imagen del contenedor del ejercicio, correrla y ya puede usar el ejercicio como si hubiera instalado en local todos los paquetes necesarios para su funcionamiento. Y cuando consideres que has terminado con ese ejercicio, tan solo borras el contenedor Docker y tu sistema queda limpio de todo rastro que el ejercicio pudiera haber creado. La versión de Docker utilizada en este proyecto ha sido la 24.0.2.

[5] `https://www.docker.com/`

3.6. OpenCV

OpenCV[6] o Open Source Computer Vision Library, es una biblioteca de software libre y una herramienta esencial para aquellos que trabajan en el campo del procesamiento computacional de imágenes y la visión por computadora. Desde su creación en 1999, OpenCV ha experimentado una notable adopción en la comunidad académica, investigadora y comercial, en parte debido a su licencia abierta y, en parte, a su rica funcionalidad que abarca una amplia variedad de tareas en visión por computadora.

Dentro de OpenCV, encontramos una serie de módulos y funciones que facilitan la manipulación y análisis de imágenes. Estas operaciones van desde las más básicas, como el cambio de formato, ajuste de contraste o filtrado, hasta técnicas más avanzadas como la detección de características, seguimiento de objetos y reconstrucción 3D. Una de las fortalezas de OpenCV radica en su capacidad para ofrecer estas funcionalidades de manera eficiente, permitiendo su aplicación en sistemas en tiempo real o en aplicaciones con grandes volúmenes de datos.

A lo largo de los años, OpenCV se ha ampliado para incluir algoritmos más complejos. Por ejemplo, la biblioteca cuenta con módulos dedicados a la inteligencia artificial y aprendizaje automático que facilitan la integración de modelos de reconocimiento y clasificación en aplicaciones de visión por computadora.

Para el desarrollo de este TFG ha sido muy importante, pues todo el procesamiento y tratamiento de imágenes ha sido usando esta librería. Desde dibujar cuadrados y texto dentro de las imágenes que capta el Tello para que el usuario pueda ver claramente dónde se encierra la persona que se detecta hasta usar su librería para detectar palmas de una mano en una imagen para controlar el Tello en el experimento del que hablaremos más tarde. La versión de OpenCV usada en este proyecto ha sido 4.5.4.

3.7. NumPy

NumPy[7] es una biblioteca fundamental para la computación científica en Python. Proporciona estructuras de datos para matrices multidimensionales y una amplia colección de funciones matemáticas de alto nivel para operar con estas matrices. NumPy no solo ha definido el ecosistema de computación numérica en Python, sino que también ha impulsado la eficiencia y capacidad de programas y sistemas que dependen del

[6] `https://opencv.org/`
[7] `https://numpy.org/`

análisis y manipulación numérica de datos.

La estructura de datos principal que NumPy introduce es el array o ndarray, que es una matriz n-dimensional homogénea. A diferencia de las listas en Python, los arrays de NumPy están optimizados para operaciones numéricas y permiten la ejecución de operaciones matemáticas en grandes cantidades de datos de manera rápida y eficiente. Estos arrays son flexibles y pueden manejar datos desde vectores y matrices hasta tensores con muchas dimensiones.

Un aspecto esencial de NumPy es su capacidad para realizar operaciones de manera vectorizada. Esto significa que, en lugar de iterar sobre los elementos individuales de los arrays, las operaciones se aplican a los arrays enteros a la vez. Esta característica no solo simplifica el código y lo hace más legible, sino que también aprovecha las optimizaciones a nivel de hardware, lo que resulta en un rendimiento considerablemente mejorado.

Este librería se entrelaza con OpenCV ya que participa en el tratamiento de imágenes. Todas las imágenes se vectorizan con esta librería para poder adaptarse a la librería de YOLO. La versión de NumPy utilizada en este TFG ha sido la 1.24.3.

3.8. YOLO

YOLO[8] (You Only Look Once) es un algoritmo de detección de objetos en tiempo real que representa un enfoque innovador en el campo de la visión por computadora. A diferencia de los métodos tradicionales de detección de objetos que suelen realizar múltiples pasadas sobre una imagen para identificar y ubicar objetos, YOLO realiza ambas tareas en una sola pasada. Esto lo convierte en una de las soluciones más rápidas y eficientes disponibles para aplicaciones en tiempo real.

Hace uso de una única red neuronal convolucional para detectar objetos. Esta red dividde la imagen en regiones, prediciendo cuadros de identificación y probabilidades por cada región.

A la hora de su uso, YOLO recibe como entrada una imagen y devuelve la imagen modificada y etiquetada como salida. Dentro de esa imagen detecta distintos objetos comunes dentro de una imagen, los encierra en una caja que englobe dónde está y se le añade un número entre 0 y 1 (con decimales) que es la confianza que tiene YOLO en que el objeto que ha detectado sea ese realmente.

[8] `https://www.section.io/engineering-education/introduction-to-yolo-algorithm-for-object-detection/`

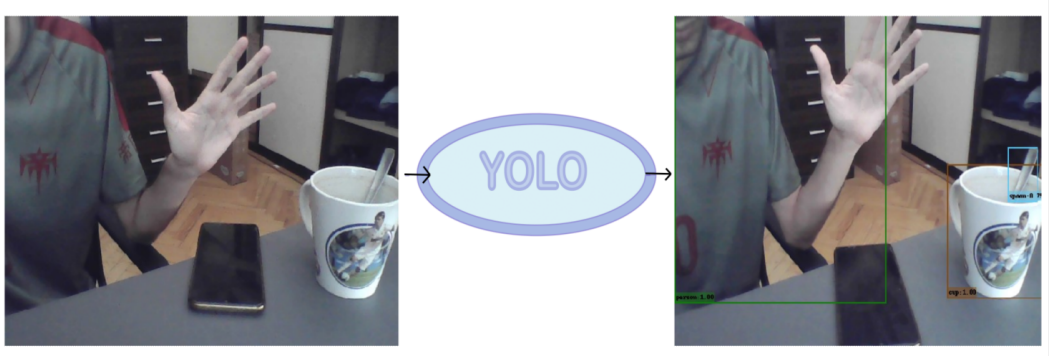

Figura 3.8: Esquema YOLO

La detección de objetos se basa en una localización más una clasificación. La velocidad de respuesta de este algoritmo lo convierte en un método conveniente para detectar objetos, ya que, aunque depende del hardware, ronda los 30 fotogramas por segundo.

Dentro de las 3 principales implementaciones de YOLO (Darknet, AlexeyAB, Darkflow) se ha hecho uso de una implementación modificada de Darknet. Esta es la implementación hecha por los mismos creadores de YOLO, hace uso de CUDA. CUDA es una plataforma de computación en paralelo que incluye compilador y un conjunto de herramientas de desarrollo creadas por Nvidia que permiten a desarrolladores usar algoritmos en GPU de Nvidia. Por lo tanto, Darknet está pensado para usarse con GPU y sobretodo con una Nvidia. Este TFG se ha desarrollado usando una GPU Nvidia GeForce RTX 3060, con la versión del driver gráfico: 535.86.05 y la de CUDA Version: 12.2.

Para usar YOLO es importante tener una coordinación de versiones de drivers gráficos y la de CUDA, ya que deben ser compatibles. Por ello existe una lista por parte de Nvidia con tablas comparativas de compatibilidad entre versiones.

CUDA Toolkit	Linux x86_64 Minimum Required Driver Version	Windows Minimum Required Driver Version
CUDA 12.x	>=525.60.13	>=527.41
CUDA 11.x	>= 450.80.02*	>=452.39*

CUDA Toolkit	Linux x86_64 Minimum Required Driver Version	Windows MinimumRequired Driver Version
CUDA 10.2	>= 440.33	>=441.22
CUDA 10.1	>= 418.39	>=418.96
CUDA 10.0	>= 410.48	>=411.31

Figura 3.9: Tabla de compatibilidad de CUDA y Nvidia

Para el desarrollo del TFG ha servido como parte de la plantilla de funciones para el ejercicio sigue-personas. La versión usada de YOLO para este TFG ha sido el YOLOv3.

Capítulo 4
Soporte de Dron Tello en ROS2 Humble

Ahora abordaremos otro de los pasos esenciales en este TFG, hablaremos del desarrollo del soporte en ROS2 Humble del dron Tello, paso esencial para hacer funcionar el dron con ROS2 y más concretamente, dentro del RADI 4.

4.1. Driver del Dron Tello físico

Ambos ejercicios estan pensados para ejecutarse en el dron real, con lo cual, hay que diseñar todos los drivers con el fin de que se comunique realmente y no de manera simulada.

4.1.1. Conexión al Dron Tello

Lo primero para lograr el desarollo de un driver en ROS2 que sea capaz de comunicarse con el Dron, es establecer conexión con este. Este paso se logra a través de una conexión Wi-Fi con el dron. Como se ha descrito antes, Tello crea una red Wi-Fi y permite al resto de dispositivos conectarse a él (libremente o de manera privada con una contraseña). Tras la conexión establecida con el dron, mediante sockets o envío de paquetes al dron se le pueden enviar mensajes a una dirección IP y puerto concretas para la comunicación de comandos para que el Tello pueda recibirlos. Estos comandos pueden ser de pregunta o de acción. Luego el Tello utiliza este mismo puerto para responder con los mensajes `rc` que han sido descritos.

Asímismo, el Tello envía mediante otro puerto las imágenes que captura para que podamos recibirlas y almacenarlas mediante otra conexión preparada para recibir flujo de datos en lugar de envío y recepción con la anterior. Esto es, el puerto por el que se reciben las imágenes capturadas por el Tello no está preparado para enviar otra cosa y será ignorado en caso de que se haga.

Con esto ya tenemos la información del Tello almacenada y la capacidad de enviarle comandos de acción a bajo nivel. El siguiente paso a seguir es enlazarlo con ROS2 para que pueda hacer lo mismo usando los comandos de ROS2 que añaden una capa de abstracción por encima de todos estos comandos de bajo nivel y de envío de imágenes del dron.

4.1.2. Sensores, Actuadores y ROS2 Topics

Para poder añadir la capa de abstracción que nos separará el Hardware y comunicación de bajo nivel y las herramientas Software de alto nivel, utilizaremos el middleware de ROS2.

En el punto anterior recogimos la información ofrecida por la cámara, así como establecimos un canal de comunicación por el que mandar órdenes que serán seguidas por el Tello. El siguiente paso es hacer uso de los topics y servicios de ROS2 método de comunicación entre Tello y nuestra máquina.

Para poder conectar el dron a ROS2 lo primero es crear un launcher del driver, algo que al ejecutar permita relacionar los comandos de ROS2 con los comandos que se le pasarán a Tello. Para ello existe el launcher `teleop_launch.py` donde se establecerá conexión con el Tello y se lanzará una ventana externa con la visualización de la cámara del Tello. Y a partir de aquí se van a poder usar los topics y servicios siguientes:

Siguiendo la convención de ROS con respecto al comandar velocidades, se ha creado el topic `/cmd_vel` en el que se publican mensajes del tipo Twist, este tipo de mensajes tienen los siguientes campos que pueden ser rellenados

- linear.x, linear.y, linear.z
- angular.x, angular.y, angular.z

Las unidades comandadas por los mensajes lineales está en cm/s mientras que los angulares están en grados/s. Cabe resaltar que este tipo de mensaje, Twist, es muy usado en robots móviles, por ejemplo también hace uso de este el Turtlebot2. En el caso del dron no nos hacen falta todos los campos del mensaje, ya que, el dron no permite girar sobre sí mismo en el eje x o en el eje y, es por eso que los campos angular.x y angular.y los va a ignorar.

Estos mensajes de velocidades lineales y angulares se traducen a comandos del tipo rc que van directos al Tello, que tienen la siguiente estructura del comando `rc a b c d`, donde:

- 'a' es velocidad lineal en el eje y (que el dron vaya a derecha/izquierda) el cual corresponde al campo de linear.y

- 'b' es la velocidad en el eje x (que el dron avance o vaya hacia atrás) el cual corresponde al campo de linear.x
- 'c' es la velocidad en el eje z (que el dron ascienda o descienda) el cual corresponde al campo de linear.z
- 'd' es la velocidad angular en el eje z (que el dron gire sobre sí mismo) el cual corresponde al campo angular.z

Este comando 'rc a b c d' se manda como string mediante sockets al Tello por el canal establecido en el anterior punto. Así que un ejemplo del trayexto completo desde que el usuario comanda velocidades al robot y son enviadas a este, es así:

- El usuario ejecuta el comando

```
ros2 topic pub --once /cmd_vel geometry_msgs/msg/Twist "{linear: {x:
    20.0, y: 0.0. z: 0.0}, angular: \x: 0.0. y: 0.0, z: 30.0}}
```

Código 4.1: Ejemplo de publicación para que el Tello avance

  Esto publica en el topic `/cmd_vel` un mensaje tipo `Twist` donde se rellena el campo *linear.x* = 20 y el *angular.z* = *30.0*.

- El driver internamente coge la información de estos campos y los transforma a un comando rc: 'rc 0 20 0 30'.
- El driver envía por sockets el string 'rc 0 20 0 30'.
- El Tello recibe este mensaje y sí todo ha ido correctamente, avanzará a 20 cm por segundo a la vez que gira a 30 grados por segundo, dibujando un círculo.

De esta manera se definen y actúan el resto de topics implementados en este driver:

- `tello_response`: donde el Tello publica la respuesta a un comando enviado por `/cmd_vel`, mensajes del tipo `std_msgs/String`
- `flight_data`: donde el Tello publica varios datos sobre el vuelo actual, como por ejemplo velocidad y orientación actual en los 3 ejes, batería, tiempo de vuelo o temperatura. Mensajes del tipo `tello_msgs/FlightData`, que es el primer tipo de mensaje que no existe como estándar de ROS2 y se ha tenido que implementar.
- `image_raw`: Es por donde el Tello envía lo que está viendo por su cámara. Mensajes del tipo `sensor_msgs/Image`

- `camera_info`: Información detallada de la cámara del Tello, como sus dimensiones de imagen, matrices intrínsecas y extrínsecas de la camara, etc. Mensajes del tipo `sensor_msgs/CameraInfo`.

Además de estos topics existe un servicio llamado`/tello_action` donde se publican mensajes del tipo `tello_msgs/TelloAction`. Estos mensajes envían una cadena de texto que es la que se envía directamente al Tello, sin manipular antes. Esto quiere decir que le envíamos nosotros directamente una cadena de texto con 'rc a b c d' al servicio y este al Tello. En este caso, la capa de abstracción que se obtiene con el servicio es distinta a la que se ofrece mediante los topics. En este servicio no hay traducción de comandos de ROS2 – Tello. Pero esto puede dar a pensar que no tiene sentido seguir con esta capa de abstracción ya que se acaban comandando acciones del mismo modo que si se hace con Tello directamente. La ventaja es que Tello, mediante QoS y su propia manera de orquestar el intercambio de mensajes entre servidores y clientes, decidirá cuándo es el mejor momento para que se pueda enviar esta acción con tal de no saturar el buffer de entrada del dron.

Es importante comprender el concepto de no saturar el buffer de entrada del dron. Tello tiene una cola de acciones de longitud concreta, es decir, puede almacenar un número predeterminado máximo de próximas acciones a realizar y todo nuevo comando que entre será desechado ya que no cabe dentro de esta cola de futuras acciones.

El `launcher` del driver ofrece por la salida información sobre esto y sobre cuando la cola que recibe acciones está saturada y no acepta más. Que a bajo nivel no deja de ser una respuesta de rc=2 por parte del dron como respuesta a una acción pedida como se ha explicado anteriormente.

Por otra parte, el Tello ignorará cualquier acción que se le comande por este servicio o los topics si antes no ha despegado. Es decir, que sin importar cuántas acciones se le pasen al dron, estas nunca serán encoladas ya que para que puedan ser encoladas lo primero es que el dron esté volando. El punto inicial del dron siempre es el despegue y se hace mediante este servicio. Este comando es el `takeoff`. Ejemplo de despegue:

```
ros2 service call /tello_action tello_msgs/TelloAction "{cmd:
    'takeoff'}".
```

Código 4.2: Ejemplo de ejecucion del servicio del Tello en ROS2 para despegar

4.2. Comunicación Robotics Academy - Dron Tello

En este apartado abordaremos el método de conexión que comunica Robotics Academy con el Dron Tello. Dependiendo de si este ejercicio se usa de manera nativa, o integrado dentro del contenedor RADI, esta comunicación cambia drásticamente, así que las similitudes entre ambos mecanismos no son muchas.

Partamos de la base de que todo ejercicio que se ejecuta dentro de Robotics Academy tiene la estructura de comunicación base donde el browser/web ejecuta el código que ha desarrollado el alumno y manda instrucciones a la máquina local de acciones que debe realizar el robot simulado que corre en nativo en el ordenador de la persona que está haciendo ese ejercicio.

El Manager juega un papel de intermediario entre estos dos puntos, pero desde fuera la comunicación es uno a uno.

Tras esta estructura más común de los ejercicios, hay que meter una variable nueva, el Tello. El Tello crea una red Wifi y no deja de ser otro punto de acceso que se añade a la máquina de la persona y su browser. Al igual que en el caso común el ordenador del usuario es el punto final de acción ya que es ahí donde es supuesto que ejecutarán las instrucciones al robot simulado, aquí es distinto, pues el punto final donde se acaban ejecutando las acciones no es otro que el propio dron. Así dependiendo de como se use el ejercicio.

Pero para ambos casos el esquema de comunicación no es otro que: el usuario introduce su código en el editor del servidor, luego esto es recibido por el Manager y envíado a la máquina en local. La máquina local recibe las funciones de la API de HAL en orden para ser ejecutadas. En la máquina local se producen dos traducciones de comandos con el fin de que el dron Tello las entienda, la primera es pasar de las funciones de plantilla HAL a ROS2, y la segunda es pasando de comandos de ROS2 a comandos SDK que el Tello es capaz de interpretar, definido en el apartado del desarrollo del driver del soporte del dron Tello en ROS2 Humble. Tras esto, el Tello ejecuta la acción propuesta por la máquina local.

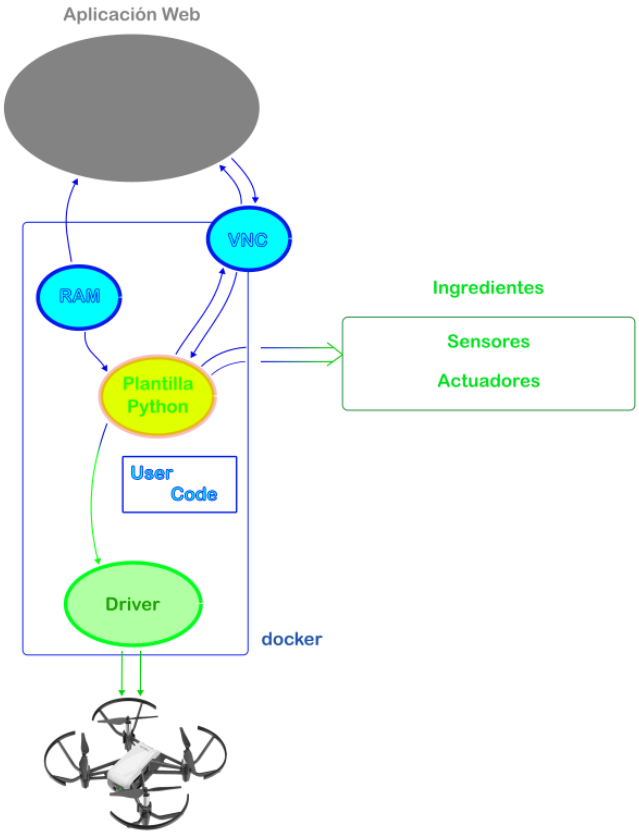

Figura 4.1: Esquema comunicación general

4.2.1. Nativo

Para la ejecución del ejercicio en local hay que tener en cuenta que se basa en tres puntos:

1. Browser: El browser corre en la red local del ordenador. Para lanzar la interfaz y el servidor en local de Robotics Academy hace falta ejecutar dos acciones que permiten la comunicación entre servidor y el cliente, que es nuestra máquina local y a la vez levantar el propio servidor. Estas dos acciones son:

   - `npm install && npm run dev`: En primera instancia, npm install busca un archivo llamado package.json en la carpeta donde estás ejecutando el comando. Este archivo contiene una lista de todas las bibliotecas, módulos y otros paquetes de software que Robotics Academy necesita para funcionar correctamente.

     El comando npm install descargue todos estos paquetes desde el repositorio de npm, los colocará en una carpeta llamada node_modules dentro de Robotics Academy. Además, se asegurará de que todas las versiones sean las correctas según lo que se especifique en package.json.

En segunda instancia, npm run dev realiza dos acciones, inicia la comunicación con el servidor de desarrollo, es decir el servidor en local e inicia las bases de datos que recogen todos los ejercicios y sus datos para funcionar correctamente.

- `python3 manage.py runserver`: inicia el servidor de desarrollo en Django, Django buscará en su configuración y preparará un servidor web de desarrollo en tu máquina local. Además, observa cambios realizados dentro del RADI que corre en local y reinicia automáticamente el servidor cada vez que se modifica algo. Esto hace que sea más fácil ver los efectos de tus cambios sin tener que detener y reiniciar el servidor manualmente. Y sin olvider que muestra errores y mensajes de diagnóstico en la terminal y en la página web que te ayudan a depurar problemas que haya tenido Robotics Academy a la hora de lanzar su servidor.

Una vez realizadas estas dos acciones preeliminares, se pasará al estado ya conocido donde el usuario desarrolla su código en el editor insertado en el frontend de la página. Tras el desarrollo y cuando vea que es el momento de realizar la primera prueba, pulsará el botón de 'Play'. Tras esto, el código será envíado al Manager y este mandará instrucción por instrucción a seguir a la parte local de la máquina, donde pasaremos al siguiente punto.

2. Máquina en local: El Manager tras recibir el código por parte del browser, se convertirá en el siguiente punto de comunicación y actuará de "puente.^[en]tre el dron Tello y el browser del servidor lanzado en local. Este concepto de puente quiere decir que después del envío por parte del Manager a la parte local, esto acaba activando determinadas funciones dentro de hal.py, quién está esperando a que se le de la orden de ejecutar alguna de las funciones de la plantilla. Este a su vez llama a motors.py que es el que interactúa directamnete con el driver del Tello en ROS2, llamando a los servicios y/o publicadores que sea necesario para accionar el dron. Tras esto, el driver de ROS2 empezará el envío de paquetes con el dron Tello lo que nos llevará al último punto de esta comunicación a tres bandas.

3. Dron tello: Este es el punto final del recorrido desde que una instrucción es escrita por el alumno en el visor del editor de código del browser de Robotics Academy corrido en un servidor en local, pasando por la máquina local en sí que serviŕa como punto intermediario entre funciones del HAL y comandos que sí entienda

el dron Tello. Para el Tello le es indiferente todo el proceso desde la creación del código en el browser hasta el driver de ROS2, para él le siguen llegando comandos en forma de paquetes a través de su red Wi-Fi.

En resumen, en la máquina en nativo el usuario está conectado a una sola red Wi-Fi. Esta red Wi-Fi no es otra que la red que el dron Tello crea y de la que es el punto final de acceso de todos los comandos para que sean recibidos y ejeutados por el dron. Por lo tanto, al ejecutar el servidor en la red local, usar el ordenador como puente entre servidor/browser y que el punto final sea el Tello, quién acaba recibiendo las acciones comandadas por el driver de ROS2 desde la máquina en nativo, es el resumen de cómo se realiza la comunicación entre los tres diferentes puntos de conexión, a la hora de ejecutar un ejercicio de Robotics Academy en nativo.

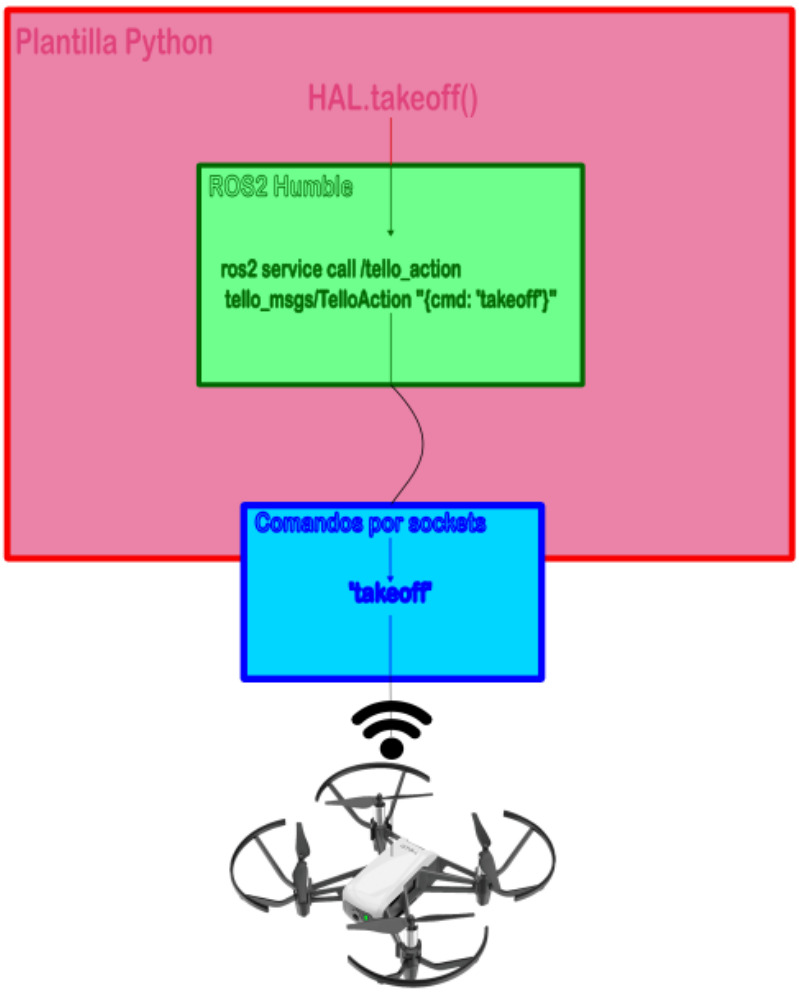

Figura 4.2: Esquema recorrido de una función

4.2.2. Desde el contenedor Docker

El sistema de comunicación usando un esquema online sufre un cambio drástico en cuanto cómo se comunican los tres puntos por lo que pasa todo el proceso a la hora de usar Robotics Academy en su página web. Y es que, ya no es necesario levantar un servidor en local, lanzar una aplicación que sea capaz de levantar el frontend del ejercicio.

Pero antes de pasar a detallar estos dos conceptos en el momento en el que se describa cómo funcionan los tres puntos del proceso, vamos a mencionar la principal innovación en la arquitectura comunicativa y uno de los problemas a los que se enfrenta el desarrollador cuando crea un ejercicio dentro de Robotics Academy usando un robot real que para funcionar crea su propia red Wi-Fi.

El principal problema es el siguiente: salvo arquitecturas de red más caras y contemporáneas, el ordenador solo puede conectarse a una red Wi-Fi a la vez. Y sin embargo, y sobre el papel, para poder desarrollar e intentar resolver el ejercicio de un dron Tello real dentro de Robotics Academy, es necesario tener dos interfaces de red activas simultáneamente, o dicho coloquialmente, estar conectados a dos redes Wi-Fi a la vez. El por qué no nos vale con una sola es sencillo, si tan sólo contamos con la red Wi-FI del Tello no tendremos acceso a Internet y por lo tanto no podremos acceder al ejercicio para resolverlo ya que no hay nadie que envíe comandos al dron si el primer punto de esta cadena de comunicación desaparece. Por otro lado, si tan solo contamos con la red Wi-Fi del lugar en el que estamos y con la que sí contamos con acceso a Internet, no hay modo de que esa red conecte con la del dron Tello de ninguna manera, ya que este solo funciona creando su propia red y no conectándose a otras. Al final este problema tiene que resolverse de algún modo, estamos obligados a ello, sino el objetivo y meta definida en el inicio de la memoria del TFG no tendría sentido. Buscamos implementar un ejercicio con el dron real en la web de Robotics Academy. Que pueda ser usado por cualquiera desde sus casas tan solo contando con el dron Tello a su disposición.

Las posibles soluciones pueden separarse en dos caminos:

- Software: Lo ideal es que la solución pueda conseguirse mediante software desarrollado por mí o inclusive por otros. Pero al fin y al cabo que sea software con el cometido de que lo que se implemente forme parte del RADI y del ejercicio y que el usuario no vea esta capa ni tenga que lidiar con ella lo más mínimo, se sigue buscando la mayor comodidad del alumno para programar y aprender sobre robótica. Dentro de las posibles soluciones que han sido probadas mediante

Software a esta problemática son:

- El primer método fue usando el comando iw, más concretamente la página de las interfaces virtuales sobre este. Básicamente este comando añadía una nueva interfaz de red virtual sobre la que se podía conectar a otra red Wi-Fi:

```
sudo iw dev wlan0 interface add wlan1 type station
```

Código 4.3: Ejemplo de ejecución del servicio del Tello en ROS2 para despegar

- El segundo método era modficar el fichero de configuración de las interfaces de red a mano, y programar la del Tello conociendo sus puertos y dirección IP. Así como añadir la máscara de red, la interfaz ethernet a usar y el resto de configuración para una red.
- Usar programas externos que ayuden a ello. Programas como Speedify dan soporte a esta problemática y pueden servir como solución a tener en cuenta.

  Todas estas tienen un problema que hacen imposible el uso de ninguna de ellas como solución, y es que, como ha sido mencionado al inicio, se necesita hardware moderno para poder soportar dos interfaces de red, en concreto una tarjeta de red más actualizada. Por lo tanto, y ya que no esperamos que los usuarios tengan que optar a gastar o usar hardware caro con el fin de poder realizar el ejercicio, quedan descartadas estas opciones.

- Crear un Manager que intercale entre las dos redes. Es decir, crear como una especie de Manager que asigne tiempos a cada red para ser usada por el ordenador y vaya cambiando la conexión de red continuamente de una a otra. De tal forma que se retendrían los mensajes envíados desde la web en nuestra máquina local mientras se está usando la red Wi-Fi con acceso a Internet y que sea este Manager de conexiones el que detecte cuando pasamos a que nuestra máquina local se trate de comunciar con el dron Tello para que cambie a la red que el dron crea y que sea posible tanto la comunicación como el control del Tello.

  Esta solución parece tener sentido pensado rápidamente. Pero la realidad es que el tiempo de demora en el intercambio de una red a otra es demasiado como para ser útil y viable integrarlo en Robotics Academy. Contamos con que, cuando nos contectamos al dron Tello, tarda unos cinco segundos en

que este detecte un nuevo dispositivo en su red, y tras ello, habría que lanzar el launcher que lanza el driver del Tello en ROS2, lo que nos retrasaría unos segundos más hasta que se inicialicen de manera correcta todos los topics y servicios.

Por lo tanto, todas las opciones exploradas para poder integrar una comunicación entre tres puntos usando dos redes Wi-Fi a la vez han sido descartas así que hay que pasar al otro lado de las posibles soluciones. Las soluciones Hardware.

- Hardware: Tras la utópica idea de poder resolver este problema mediante solo Software, vamos a explorar las opciones de Hardware adicional al que cuenta el usuario normalmente que nos permitan conseguir conexión a dos redes Wi-Fi simultáneamente.

  - Nueva tarjeta de red: La primera de las soluciones Hardware nos la hemos topado antes al intenta explorar vías Software, y es que estas tenían un techo que no permitía que sea viable, las tarjetas de red, normalmente, solo cuentan con una interfaz de red disponible. Por lo tanto, una de las posibles soluciones es que el usuario adquiera una tarjeta de red que cuente con dos interfaces de red para poder abordar el ejercicio.

    Por ejemplo, la Intel PRO/1000 PT Dual Port Server Adapter, cuyo valor ronda los ochenta euros, nos ofrece esta solución.

  - USB externo TP-Link: Por unos diez euros, el usb TP-Link ofree dos puertos de red Ethernet, lo que permite a la máquina conectarse a dos redes diferentes simultáneamente. Esta solución parece mucho más viable que la anterior y tras pruebas con ella, y usándose con el fin de comunicarse con el Tello (que tiene mucha menos carga en su red), tenemos una solución sólida y barata.

  Al final nos decantamos por esta última opción y será el punto de apoyo que nos permitirá conectarnos a dos interfaces de red simultáneamente para que los tres puntos que componen la arquitectura de comunicación puedan comunicarse fluidamente, el USB TP-Link.

Con este problema ya solucionado pasamos a ver cómo sería la comunicación entre los tres puntos:

- Browser: Esta vez el browser está dentro de un servidor web, así que la comunicación con el Manager es vía Internet, además, necesitamos acceder a

este a través de una de las dos interfaces de red que tenemos disponibles, preferiblemente haremos uso de la red Wi-Fi con la que se cuente en el lugar en el que estemos porque va a estar mejor preparada que el TP-Link para el intercambio de información continúo y rápido.

- Máquina en local: La máquina en local funciona de la misma manera que en el anterior punto, pues sigue en su rol de puente y desde este punto de vista, no hay gran diferencia en que existan dos interfaces de red o que Robotics Academy esté en un servidor externo y no local. La única diferencia es que es aquí donde se conecta el TP-Link y el acceso a Internet por medio de la red Wi-Fi que tengamos ya.

- Dron Tello: Al dron Tello se le debe asignar al TP-Link para que se conecte a él ya que cuenta con menos capacidad que la otra interfaz de red y no se espera gran volumen de envío de paquetes, ya que son comandos muy cortos. Este punto también es igual al anterior, pues el dron solo ve lo que le llega del driver de ROS2 actuando el resto de la comunicación encapsulada como caja negra para él.

  [figura explicativa de esto]

4.3. Plantilla Python de los ejercicios, acceso a sensores y actuadores

Para poder ejecutar el código fuente que introduce el estudiante como solución a un ejercicio dentro de Robotics Academy es necesario coordinar cuatro ficheros entre ellos de manera específica para que los diferentes publicadores, suscriptores y servicios sirvan como base para la futura plantilla de los ejercicios. Esto es la integración del driver de ROS2 Tello en Robotics Academy, en RADI4 (basado en ROS2) y como ejemplo de cómo se hace uso de esta integración, vamos a explicarlo a través del comando de despegue. El comando que hace que el Tello avance determinado número de centímetros, o dentro de Robotics Academy, *HAL.forward()*.

Para poder entender los cuatro ficheros y como se comunican entre ellos, vamos a explicar primero el concepto de SharedValue.

El objeto SharedValue está diseñado para compartir un valor de punto flotante entre diferentes procesos. Para hacer esto, SharedValue utiliza un segmento de memoria compartida y un semáforo. La memoria compartida es como un tablero de anuncios que cualquier proceso con permiso puede leer o modificar. Sin embargo, si varios procesos

intentan cambiar el tablero de anuncios al mismo tiempo, pueden ocurrir errores o inconsistencias. Ahí es donde entra en juego el semáforo.

Un semáforo es básicamente una señal de tráfico para procesos. Garantiza que solo un proceso pueda acceder al "tablero de anuncios"(memoria compartida) en un momento dado. Cuando un proceso quiere leer o escribir en la memoria compartida, primero debe adquirir el semáforo, realizar su operación y luego "liberarël semáforo para que otros procesos puedan usarlo.

Al instanciar un objeto SharedValue, se intenta conectar a una región de memoria compartida y un semáforo ya existentes; si no existen, los crea. Este objeto tiene tres métodos principales:

- get(): Es utilizado para recuperar el valor de punto flotante que está almacenado actualmente en la memoria compartida. Antes de acceder a este valor, el método adquiere el semáforo para bloquear temporalmente el acceso a otros procesos. Una vez que ha obtenido el control exclusivo, lee el valor almacenado en la memoria, libera el semáforo y devuelve el valor. Este mecanismo garantiza que el valor leído sea coherente y no se corrompa debido al acceso simultáneo de múltiples procesos.

- add(value): Se encarga de almacenar un nuevo valor de punto flotante en la memoria compartida. Similar al método get(), este método adquiere el semáforo antes de realizar cualquier operación para asegurar que tenga acceso exclusivo a la memoria compartida. Una vez que el semáforo se adquiere, el nuevo valor se almacena en la memoria. Después de esto, el semáforo se libera para que otros procesos puedan acceder a la memoria compartida.

- close(): Se encarga de la limpieza de recursos. Bloquea el semáforo para evitar que otros procesos accedan a la memoria compartida y luego cierra y desvincula la memoria. Finalmente, libera el semáforo y lo cierra. Este método asegura que los recursos se liberen de manera adecuada, minimizando la posibilidad de fugas de memoria o bloqueos de recursos.

En resumen, SharedValue es una clase que facilita compartir un valor de punto flotante entre múltiples procesos de manera segura y sincronizada. Los procesos que se comunican entre sí con el browser, es decir, la página donde se programa y lanza el ejercicio, y el ordenador en local, que es donde acaba corriendo el ejercicio que se programa en el browser. Estos dos procesos comparten la memoria que se mira en la clase SharedValue. También existe la instancia SharedImage que en vez de compartir

un valor de punto flotante, compare un objeto numpy que hace referencia a una imagen. Es decir, trata de compartir imágenes entre procesos.

Estos cuatro ficheros se dividen en dos y dos. Los dos primeros: user_functions.py y brain.py son de la parte usada por parte del alumno dentro de la página.

- `user_functions.py`: Este fichero define las funciones HAL y GUI que va a usar el alumno y que forman parte de la plantilla. Se definen SharedValue para función de la plantilla que explicaremos más a fondo en el diseño de los ejercicios, y SharedImage para la compartición de imágenes.

  Por lo tanto, para el ejemplo de forward sería algo así:

```
self.shared_forward = SharedValue("forward")
```

Código 4.4: Ejemplo de uso de SharedValue

Esto hará que cuando se quiera escribir o leer, lo hará en la zona de memoria reservada para el tema "forward". En este caso solo escribiremos en esa zona memoria, es decir, solo haremos add().

También definiremos una variable que sirva como referencia a la respuesta del dron a cualquier acción que le comandemos:

```
self.tello_response = 0
```

Código 4.5: Variable de tello_response

Donde el significado de sus posibles valoes (1-3) será similar al rc enviado por Tello como respuesta.

Y esta variable será actualizada con otro SharedValue:

```
self.shared_response = SharedValue("response")
```

Código 4.6: Ejemplo de uso de SharedValue

Será usado como método de lectura de valores, es decir, tan solo haremos get().

Ahora definiremos la función a la que llamará el usuario cuando ejecute *HAL.forward()*:

Esta función recibe por parte del usuario una distancia (distance) y añade con un add() el valor a la zona de memoria reservada a "forward", este valor es el

```
def forward(self, distance):
    self.shared_forward.add(distance)
    while self.tello_response < 1:
        self.tello_response = self.shared_response.get()
    self.tello_response = 0
```

Código 4.7: Metodo forward en user_functions.py

inputado por el usuario directamente. Tras eso, hace una espera activa esperando la respuesta del tello responda, esta respuesta se recoge con el get(). Mientras que el valor de tello_response no se actualice seguiŕa esperando, que será el caso en el que Tello esté en medio de otra acción y no pueda llevar a cabo el forward.

- `brain.py`: Este fichero cumple muchas funciones en la orquesta de procesos que es Robotics Academy, pero para esta integración del driver del Tello en ROS2, solo haremos hincapié en una parte del código. Lo mencionado antes de que en user_functions se definen las funciones que usará el usuario era una verdad incompleta, pues realmente usarán un pseudónimo. Dentro de `brain.py` hay una función que genera los módulos HAL y GUI para el usuario. Aquí generamos un pseudónimo bajo el módulo de HAL para las funciones de `user_functions`, pero el funcionamiento será exactamente igual ya definido.

  Este proceso de cambiarle el nombre se hace del siguiente modo:

```
hal_module.HAL.forward = self.hal.forward
```

Código 4.8: Creacion del modulo HAL en brain.py

Donde la parte izquierda hace referencia al pseudónimo que usará el usuario y la función de la derecha hace referencia a la función de forward definida anteriormente. De ahí que al inicio se mencione que el usuario acabará usando *HAL.forward*, este nace exactamente de aquí. Profundizaremos con más amplicación de este fichero más tarde cuando se hable de la creación de la plantilla de los ejercicios.

En resumen, estos dos ficheros recogen la función de la plantilla (bajo un pseudónimo que se hace para generar el módulo HAL) que use el usuario y añade a la zona de memoria compartida la información que considere relevante respecto a esa función, y espera una respuesta por parte del Tello para poder seguir ejecutando las siguientes instrucciones.

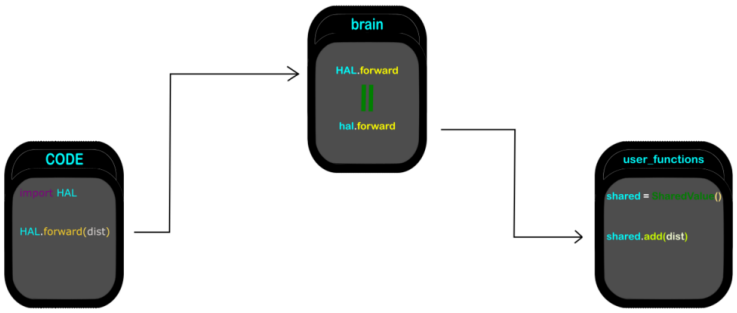

Figura 4.3: Esquema sobre Shared Value

Los otros dos ficheros a mencionar forman parte del proceso que corre localmente en la máquina del alumno. Estos son:

- `hal.py`: En este fichero vuelven a crearse objetos del tipo SharedValue. Se necesita coger el valor programador por el alumno para saber cuántos centrímetros ha de avanzar, por ello se crea la variable:

```
self.shared_forward = SharedValue("forward")
```

Código 4.9: SharedValue de forward

Está variable solo la usaremos para leer lo que se introduce desde el código fuente del usuario, es decir, solo usaremos el método get(). Es en este fichero donde se envía la respuesta del dron que sucede tras comandarle una acción. Esto es otra variable del tipo SharedValue que actualizará, al igual que en `user_functions.py`, la variable `self.tello_response`. El valor que recoge esta variable es la que se enviará al otro proceso como respuesta:

```
self.shared_response = SharedValue("response")
```

Código 4.10: SharedValue de response

En local se empieza a tener contacto directo con ROS2 y el driver desarrollado. Se ha de crear un suscriptor al topic `/tello_response` que reciba la respuesta del Tello de la que acabamos de hablar y se la asigne a `self.tello_response` una vez Tello publique ahí.

También en necesario crear un objeto del tipo Motors, que va a ser último paso y en el que vamos a profundizar en el siguiente fichero que me queda por mencionar. Todo esto nos da la siguiente función:

```
def forward(self):
    distance = 0
    distance = self.shared\forward.get()
    if distance > 0:
        self.motors.forward(distance)
        while self.tello_response != 1:
            rclpy.spin_once(self.node, timeout_sec=0.5)
            self.motors.forward(distance)
        self.tello_response = 0
        self.shared_response.add(1)
        self.shared_forward.add(0)
```

Código 4.11: SharedValue de response

En esta función se espera que el usuario use la función y añada un valor distinto a cero y manda a los motores a avanzar esa distancia. Se vuelve a realizar una espera activa para recibir la respuesta por parte del Tello a este forward. Pero en lugar de un get() esperamos a que la variable `self.tello_response` obtenga un valor distinto al predeterminado que viene a significar que no se encuentra en ningún en estado en concreto y por lo tanto va a esperar. Mientras realiza esta espera va preguntando a ROS2 por esta variable y si ha cambiado su valor y sigue comandando la acción al Tello por si la hubiera ignorado la primera vez que se le ha llamado.

Tras tener una respuesta del dron, volvemos a dejar la variable `self.tello_response` en estado indeterminado y mandamos al código fuente del usuario el valor para salir del bucle en el que él está. Y vacíamos la cola que comanda velocidades al SharedValue de la zona de memoria de forward para eliminar memoria basura.

Más tarde profundizaremos en cómo se llama a esta función, con que frecuencia y que es el `update_hal`.

- `motors.py`: Este es el fichero que se comunica con el dron mediante ROS2 después de todo el largo proceso. Es donde se crea el objeto Motors del que hablamos antes. Motors necesita como argumentos un topic sobre el que actuar, es decir, el topic donde va a plasmar los valores inputados por el usuario. En este fichero se crean el publicador con el topic pasado por argumento, que será `cmd_vel`. Y el servicio para comandos que no tengan que ver con la velocidad y sean acciones únicas, como es el caso del ejemplo del forward que estamos usando.

  El publicador:

```
self.pub = self.create_publisher(Twist, topic, 10)
```

Código 4.12: Creacion de un publicador en motors.py

El servicio:

```
self.cli = self.create_client(TelloAction, 'tello\_action')
self.req = TelloAction.Request()
```

Código 4.13: Creacion de un cliente de servicios en motors.py

La variable self.cli es la que va a recibir la respuesta del servicio `tello_action` y `self.req` la que solicitará ejecutar las distintas acciones.

La función que se llama en `hal.py` de `self.motors.forward(distance)`, es la siguiente:

```
def forward(self, d):
    msg = "forward " + str(d)
    self.send_request(msg)
```

Código 4.14: Envio de solicitud de accion en motors.py

Como hemos mencionado antes, el servicio recibe solicitudes en forma de string único, por lo tanto tendremos que ceñirnos al tipo de mensaje de los comandos de SDK para poder enviarle acciones. Esto aquí es la solicitud de avanazar más la distancia que se desea avanzar. Y lanzamos la solicitud para que se realice usando esta función.

```
def send_request(self, cmd):
    self.req.cmd = cmd
    self.future = self.cli.call_async(self.req)
```

Código 4.15: Funcion de envio de solicitud de accion en motors.py

Donde se recibe un comando y el cliente hace la solicitud espernado una respuesta por parte de ROS2 de que el Tello la ha recibido correctamente.

Para resumir lo que han hecho estos dos ficheros, `hal.py` espera a que se le llame a una de sus funciones y una vez hecho manda a los motores ejecutar ese comando esperando la respuesta para poder continuar con la ejecución. Por dentro, los motores comandan mediante un servicio/topic de ROS2 aquello que le ha sido enviado.

Con esto tenemos ya la integración del driver del Tello en ROS2 Humble dentro de Robotics Academy. Para resumir y que se pueda entender todo este proceso se deja un esquema del proceso completo:

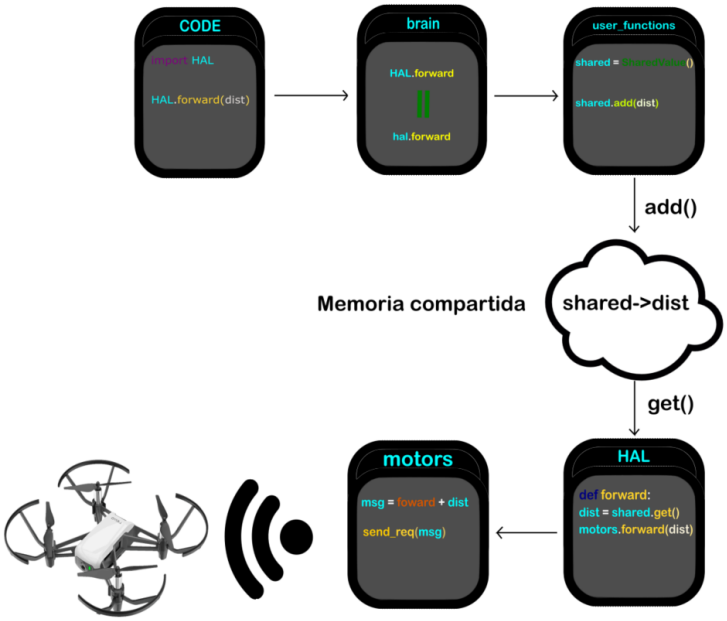

Figura 4.4: Esquema funcionamiento HAL

Capítulo 5
Ejercicios Dron en Robotics Academy

Tras haber descrito el soporte e integración del Dron Tello en ROS2 dentro de RADI4, pasamos a describir la realización y desarollo de los dos ejercicios propuestos para integrarse en Robotics Academy

5.1. Arquitectura de los ejercicios

Dentro de la complejidad que supone integrar un nuevo ejercicio en Robotics Academy, así como desarollar un driver que permite la comunicación de ROS2 con el dron, el desarrollo de la arquitectura que sostiene los ejercicios desarrollados consta de menos pilares que desarrollar.

La arquitectura de ambos ejercicios es igual ya que se busca la total libertad del usuario para poder resolverlos como mejor se lo proponga, por ello, la plantilla goza de una extensa gama de funciones a usar por el mismo usuario permitiendo flexibilidad y comodidad para el desarrollo de una solución.

Otra parte de la arquitectura de estos ejercicios es la del `launcher`. Es `launcher` es un fichero que se encuentra dentro de la arquitectura del ejercicio que se ejecuta nada más entrar al ejercico. Esto sirve para establecer, en la mayoría de casos, escenario y robot.

En otros ejercicios de la misma plataforma, como el sigue-líneas, el `launcher` se dedica a lanzar el escenario que sostiene este ejercicio, su escenario y robot:

- Escenario: Mundo en el que el robot va a desarrollar el comportamiento y el contexto donde se va a realizar el ejercicio. En el caso del sigue-líneas es un circuito de carreras con una linea de color diferente al de la pista para que el robot la siga. Los escenarios son normalmente entornos simulados en Gazebo.

- Robot: Va a ser el agente que vamos a programar y que seguirá nuestras instrucciones en pos de realizar el ejercicio desenvolviéndose en el escenario. En el caso del sigue-líneas, al ser un ejercicio de robot simulado, es un robot simulado de un pequeño coche de carrreras.

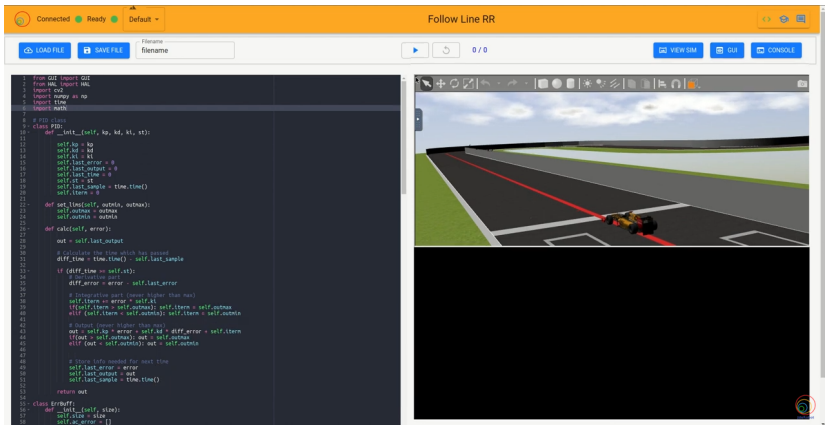

Figura 5.1: Ejercicio sigue-líneas en Unibotics

El `launcher` de nuestro ejercicio es un caso es especial:

- El escenario no es necesario ser lanzado: No estamos tratando con un entorno simulado, estos dos ejercicios tienen como base usar el dron Tello real. Es por eso que el entorno o escenario sobre el que el robot va a efectuar sus acciones no es otro que la propia realidad o el lugar donde se encuentre el dron en físico en ese momento. No será necesario lanzar ningún escenario simulado pero sí tener en cuenta sobre qué escenarios puede actuar el dron. El dron Tello es un robot de tamaño reducido, pensado para ser un soporte educativo para la introducción de drones a la persona que lo posea. Pero este dron es frágil, el entorno donde debe operar, y el escenario al que se le debería asignar es un escenario grande y amplio, esto es para que el dron pueda permitirse despegar con total libertad, calibrar su posición y poder actuar y moverse sin limitación alguna. También el escenario debe contar con gran fuente de luz o iluminación, el dron Tello sufre mucho para poder estabilizarse en una posición tras el despegue si no cuenta con una gran iluminación sobre la que apoyarse. Es por eso que el mejor escenario para la ejecución de este ejercicio es al aire libre. Pero como hemos adelantado, esto no corre a cargo de la arquitectura en sí, sino del usuario que quiera realizar el ejercicio.

- El robot es el propio dron: Al tratarse de un ejercicio orientado al uso con el robot real, no es necesario modelar ni diseñar un dron Tello simulado. Este `launcher`

ha de lanzar el driver del robot real. El `launcher` tiene que lanzar el driver que interconecte la red de Tello y su comunicación entre comandos a ROS2 para que pueda ser usada la integración que se hace del driver en RADI 4. El `launcher` del driver no solo enlaza ROS2 y Tello, sino que lanza todos los topics y servicios de ROS2 asociados al Tello donde se encuentra, por ejemplo, la visión de la cámara en tiempo real.

5.2. Plantilla Python

La plantilla python de los ejercicios no es otra cosa que todas las funciones de las que dispone el usuario: del módulo HAL para comunicarse con los actuadores del robot, básicamente los motores e imágenes, y del módulo GUI para que pueda trabajar con las imágenes.

Ambos ejercicios comparten la misma plantilla si hablamos exclusivamente de la parte de HAL y GUI, es decir, disponen de la misma gama de funciones a usar para dotar de total libertad al usuario así como todas las posibles funciones y comandos que acepta Tello con el fin de que pueda resolver ambos ejercicios de la manera que mejor le parezca.

5.2.1. Módulo HAL

Sabemos que el módulo HAL es una capa de abstracción que nos permite no tratar con directamenmte con el *hardware* del dron. Este módulo está preparado para ser lo más amplio posible y con funciones concretas y más generales, por ejemplo, existe la función `HAL.setVX(x)` para comandar al dron una velocidad constante en el eje x del sistema de coordenadas del Tello y otro que es `HAL.setVelocities(x,y,z)` para comandar velocidades constantes en todos los ejes del dron en una misma llamada.

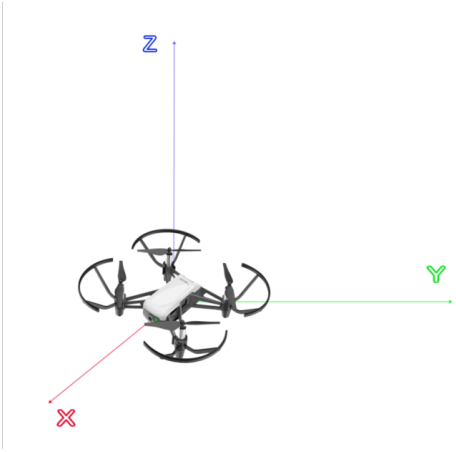

Figura 5.2: Sistema de coordenadas del Tello

Pasaremos a explicar los principales grupos de funciones que se tienen en este módulo HAL:

- `getImage()`: Sabemos que el Tello publica imágenes de su cámara y lo que ve dentro del topic `/image_raw`. En la integración del driver dentro de Robotics Academy se creó un suscriptor a este topic para que cada vez que Tello mande una imagen, este suscriptor la reciba y tras ello, en `hal.py` se añada esta imagen al SharedImage para que cuando el usuario pregunte por la imagen más reciente con el get() de SharedImage se le devuelva la imagen que menor tiempo lleve añadida. En `user_functions.py` (`HAL.getImage()`) se hace este `shared_image.get()` y posteriormente se devuelve al usuario la imagen que este ha pedido para que pueda manipularla con total libertad.

- Funciones de velocidad lineal constante en un solo eje: Este tipo de funciones actuan comandando velocidades constantes en un solo eje, siendo posible un rango [-100,100] cm/s para ser introducido como argumento en cada una de ellas. Las velocidades negativas tendrán como efecto que el dron recorra ese mismo eje en dirección contraria con la velocidad inputada. Las funciones que componen este grupo son: `HAL.setVX`, `HAL.setVY` y `HAL.setVZ`. Estas funciones acaban llamando al topic `/cmd_vel` donde se le comanda al tipo de mensaje Twist velocidades lineales en x, y, z.

- Funciones de velocidad angular constante en un solo eje: Estas funciones tienen la misma idea que el tipo anterior, ofrecer una velocidad constante en un solo eje pero de giro, es decir, velocidad angular. Este grupo lo compone una única función, `HAL.setW(w)` ya que el driver de ROS2 solo permite que gire en el eje z. Su rango es el mismo al anterior apartado [-100, 100] siendo las velocidades

positivas un giro sobre sí mismo en el sentido horario visto el dron desde arriba y las velocidades por debajo de cero acaban actuando con un giro antihorario. Esta función acaba publicando en el mismo topic, `/cmd_vel`, pero rellenando el campo de la velocidad angular en el eje z.

- Funciones de velocidad constante en varios ejes: Con el fin de compactar el resto de funciones y pensado para comportamientos más sofisticado para el dron, se han implementado estas funciones. Todas comparten el ya conocido rango [-100, 100] cm/s donde serán ignoradas las llamadas que usen como parámetro una velocidad que se salga de este rango. Las dos funciones que componene este grupo son:

  - `setVelocities`: Esta función llama a los 3 ejes de velocidad lineal del dron y comanda velocidad a cada uno de ellos en una sola llamada. `HAL.setVelocities(vx,vy,vz)` donde vx es la velocidad lineal en el eje x, vy la velocidad en el eje y y vz la velocidad en el eje vz. Por dentro se sigue llamando a la función que publica en el topic `/cmd_vel` rellenando los tres campos de velocidad lineal.
  - `setRC`: Aquí vamos a diferir de la tónica habitual por la cual se llama al topic `/cmd_vel`, innovaremos y llamaremos al servicio, ya que la idea es que la función `HAL.setRC(a,b,c,d)` sea lo más parecido al comando 'rc a b c d' para aquellos que están más familiarizados con el dron Tello y su plantilla SDK 1.3. Por recordar los argumentos del comando rc: 'a' hace referencia a la velocidad en el eje y del dron, 'b' hace referencia a la velocidad en el eje x, 'c' es para la velocidad en el eje z y 'd' se refiere a la velocidad angular, el giro sobre sí mismo del dron.

- Funciones bloqueantes de estado: Este tipo de funciones llaman tambien al servicio de `/tello_action`, esto es, porque son funciones que son de un solo comando y no se puede realizar mediante topics. Estas funciones son: `HAL.takeoff()`, `HAL.land()`, `HAL.pause()` y `HAL.emergency()`. Son funciones bloqueantes, es decir, hasta que no se termine la acción ignorará todas aquellas que le llegue. La plantilla está preparada para que no se ignore ningún comando del código del usuario y hasta que no se ejecute una de estas acciones bloqueantes no pasará a la siguiente instrucción que se deba ejecutar. `HAL.takeoff()` y `HAL.land()` está clara su función pues ya sido mencionada como ejemplo en apartados anteriores, pero las otras dos funciones son las siguientes:

- `pause`: Pausa toda acción que haya impuesta a los motores que no sea el despegue. Es decir, toda velocidad comandada a cualquiera de los ejes, linear o angular, será removida y a efectos prácticos el dron se quedará quieto en el sitio en el que esté, pero volando, sin variar su posición.
- `emergency`: Este estado es un estado de alerta en peligro total y que no debería ser usado salvo caso extremo con riesgo de choque o de que el dron pueda salir dañado de alguna acción. Esta función para los motores esté donde esté el robot y tenga lo que tenga en su cola de ejecución pendiente de ejecutar. Esto es muy peligroso porque al pararse los motores inmediatamente, el robot caerá sin importar donde esté pudiendo resultar en daños al dron. Aun así, con la filosofía de ofrecer total libertad al usuario, le ha sido dispuesta esta función.

- Funciones bloqueantes de giro: Al igual que la función `HAL.setW()`, son funciones que se preocupan del giro del dron en el eje Z. La diferencia es que el rango de estas funciones, con el fin de que sea más claro, es de [0-360] ya que a las funciones `turn_left` y `turn_right`, que son las funciones de este apartado, funcionan en grados, se le comandan unos grados concretos a girar y el dron ignora el resto de acciones para poder girar lo inputado por el usuario. Como antes, la plantilla está preparada para que ningún mensaje sea ignorado y se ejecutará toda instrucción escrita por el usuario.

- Funciones bloqueantes de distancia: Las funciones bloqueantes tanto de distancia como de giro, están pensadas para que sean más sencillas de entender y de usar, ya que se sentirán más cómodos con ellas gente que no está acostumbrada con la programación de drones ya que es más seguro, es decir, al comandar velocidades hay que tener cuidado de que el dron no choque, con las funciones bloqueantes de distancia es fácil conseguir que no lo haga. Por lo tanto, se intenta concretar lo máximo posible las funciones, es por eso que las funciones son: `HAL.forward(distance)`, `HAL.left(distance)`, `HAL.right(distance)`, `HAL.back(distance)`, `HAL.up(distance)` y `HAL.down(distance)`. Todas ellas tiene de rango [0-100] cm posibles a avanzar.

Con el fin de que la reacción del Tello sea lo más rápida y controlada posible desde que un usuario manda a ejecutar una función HAL hasta que el Tello la ejecuta, vamos a hablar del hilo y tiempo de comprobación de qué se debe ejecutar.

La parte en la que el usuario programa una función y esta acción llega a la máquina local no tiene apenas latencia quitando la que tenga la red o el tiempo

despreciable de las llamadas a las funciones de SharedValue, ya que, si el usuario desea hacer `HAL.forward(distance)`, por dentro, tan solo se hace un `shared_forward.add(distance)` añadiendo ese número entero a la memoria compartida y ya ha terminado tu tarea.

El siguiente paso es como hal.py percibe que se ha hecho un `HAL.forward(distance)`, como lo detecta y como acaba ordenando al Tello directamenmte que lo haga a través de `motors.py`

En `hal.py` existe un hilo de ejecución ajeno al hilo de ejecución principal. Este hilo únicamente trata de estar esperando y monitorizando las llamadas HAL que se hagan para ejecutarlas lo antes posible, este se llama ThreadHAL.

Esta clase corre únicamente un bucle que llama a una función que comprueba cada función del móduo HAL para ver si debe ejecutar, esto es, que su get() del SharedValue tenga un valor mayor a cero. En otro caso, esta función que monitoriza todas las funciones de HAL, pasará a comprobar si debe ejecutar la siguiente función de la lista.

Con el fin de no sobresaturar al Tello de comandos y que acabe desechando o descartando posibles comandos claves como un `HAL.pause()`, en el desarrollo de `hal.py` y en la clase `ThreadHAL` se puede definir un tiempo de ciclo para entrar a la función que comprueba que función HAL debe ejecutar, se han hecho varias pruebas en este caso:

- Con tiempos inferiores a 25ms: Tello acaba descartando funciones que pueden ser clave para seguir a la persona o para realizar el cuadrado. No tiene sentido usar tan poco tiempo.
- Con tiempos por encima de 50ms: La respuesta del Tello ante cualquier comando acaba siendo muy lento y diferencial. Esto se puede notar sobretodo a la hora de seguir a una persona con la cámara mientras esta está en movimiento, acaba perdiéndola fácilmente.

He llegado a la conclusión de que los tiempos ideales son entre 25 y 50 ms de espera hasta la comprobación de qué función se va a usar. Actualmente en el ejercicio está establecido a 25ms.

Por otra parte, al igual que en el apartado HAL, GUI cuenta con su propio hilo de ejecución distinto al principal que se encarga de actualizar las imágenes que nos da el Tello y mostrarlas por pantalla con el showImage lo más actualizada posible. Como SharedImage recibe solo la imagen más reciente y descarta las que ya tuviera guardadas con el fin de agilizar todo el proceso, este hilo de ejecución cuenta con un

tiempo de espera entre vueltas de buclue que llaman a la función de comprobación de llegada de nuevas imágenes menor al del HAL, actualmente establecido a 10ms. Más de eso puede resultar tosco o inviable para generar un comportamiento lo más reactivo posible. Si ponemos más estaríamos reaccionando a imágenes del pasado.

5.2.2. Percepción visual

El otro módulo principal dentro de estos dos ejercicios y de Robotics Academy en general es el módulo GUI. Este módulo sirve para el tratamiento de imágenes únicamente y consta de:

- `showImage(img)`: Este método recibe como parámetro una imagen, tenga como origen `GUI.getImage()`, otro origen no mencionado o creada desde cero. La finalidad es mostrar la imagen que se le ha pasado por una ventana llamada GUI dentro del Frontend de Robotics Academy. Por dentro, esto hace otro add() del tipo SharedImage y desde `GUI.py` es donde se hace el get() para poder mostrarlo por la pantalla. Aquí no hay interacción con el Tello realmente.

La única diferencia entre ambos es que en el ejercicio de sigue-personas es necesaria la inclusión YOLO a la percepción visual para poder reducir la complejidad de la detección de personas. En caso de no optar a esta vía, el usuario se vería obligado:

- Desarrollar YOLO desde cero: Esta opción es claramente inviable pues requiere un entreno previo de una red para que esta infiera personas en una imagen así como otros tantos objetos como YOLO nos ofrece.
- Usar métodos de visión artificial clásica: Es una solución que se puede abordar pero requiere de altos conocimientos de visión artificial y un largo desarrollo. El usuario debería ser capaz de detectar una persona a través de los siguientes métodos:
    - Detección de Haar: Es un método usado para la detección de caras. Esto solo detecta caras, no tiene en cuenta la totalidad de la persona y no daría una respuesta efectiva en el caso de que la persona no estuviera mirando directamente al dron.
    - Histograma de Gradientes Orientados (HOG): Calcula mediante descomposición en cuadrados y el cálculo de los gradientes de cada uno de ellos, objetos en una imagen. Con ello se podría alimentar un clasificador y así diferenciar personas.

- Filtros de partículas: Suele ser mayormente efectivo en movimiento de personas en vídeo. Los filtros de partículas estiman el estado futuro de la persona basándose en su estado actual y en observaciones de su movimiento. A pesar de que este método es más robusto, requiere una computación intensa que no es ni de cerca lo que se busca para el ordenador de un alumno.
- Contornos y Segmentación: Esta técnica busca dividir una imagen en regiones o segmentos basados en características como el color, intesidad o textura. Esta técnica tiene problemas con todo aquello que no sean escenarios simples.

Además, cabe destacar que estos métodos son muy poco robustos y menos precisos que los algoritmo de aprendizaje profundo modernos, especialmente es casos donde las escenas cuentan con ruido o están muy pobladas de posibles soluciones a inferir.

- Usar funciones de la libreria de OpenCV: Es la única librería dedicada a la visión artificial que está incluída en Robotics Academy. Pero, tras investigar, usa un clasificador de caras basado en la detección de Haar así que no deja de ser inviable.

Es por eso que YOLO parece la mejor opción, ya que, dentro de esta arquitectura, ofrece mejor tiempo de reacción a la par de resultados más robustos y preparados para todo tipo de posiciones y/o orientación de la persona, así como la detección de una parte corporal de la persona, es decir, detectar una persona cuando solo se le ve el brazo y el hombro:

Figura 5.3: Diferencia de precisión de YOLO y OpenCV

A pesar de la diferencia, el resto de la arquitectura de Robotics Academy dedicada a los dos ejercicios es para ambos ejercicios.

Con esto tenemos la página python del Tello completa para que el usuario pueda resolver los dos ejercicios con total libertad sobre la manera de hacerlo.

5.3. Página Web

Igual que es imprescindible el uso de una página python para poder comunicarnos con el robot, también es necesaria la inclusión de una página web que permita al usuario acceder y usar esta página python en primera instancia. A través de aprendizaje superficial de HTML, CSS y JavaScript basándome en las páginas web de otros ejercicios, ha sido posible diseñar la mía propia para los dos ejercicios a desarrollar.

Como ambos usan el robot real y no necesitan de ninguna extensión extra, ambos ejercicios cuentan con la misma página web. Esta página web está compuesta de un editor de texto que utilizará el alumno para programar el dron. Una ventana del RVIZ2 con el topic `/image_raw`, esta ventana de RVIZ2 está editada con un fichero de configuración que nos permita lanar RVIZ2 con la ventana de un tamaño concreto y con el topic de la imagen del dron preconfigurado para salir automáticamente. También es necesario una consola de comandos para depurar las soluciones de los usuarios y una ventana adicional que nos permita visualizar las imágenes que el usuario desee a través de `GUI.showImage()`.

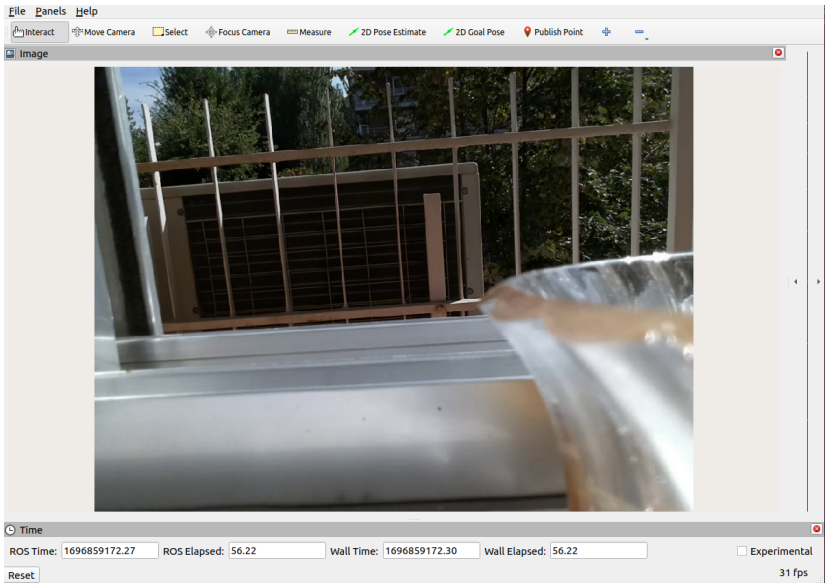

Figura 5.4: Sistema de coordenadas del Tello

Por otra parte, la arquitectura tambíen cuenta con su lado de Frontend, debemos disponer el ejercicio de tal modo que se nos permita observar en él código, consola, visualización de imagen con la GUI y con RVIZ2.

La parte de la visualización de código y consola es similar en todos los ejercicios. Todos ellos usan estas interfaces y la tienen como parte esencial para la realización de este, ya que el código es la parte más básica para resolver el ejercicio pues es donde se introducirán las instrucciones que posteriormente el robot, o en este caso el Tello,

ejecutará. Mientras que por parte de la consola sirve como parte de apoyo para las siguientes cuestiones:

- Visualizar trazas que ayuden al desarrollo de ejercicio: Toda aquella persona que haya programado algún ejercicio o programa tiene como punto de apoyo la consola para colocar trazas que ayuden a encontrar un error en su código que no puede ser detectado, o al menos de manera visual y clara, y por lo tanto se apoya en esta consola para poder imprimir por pantalla variables que componene el programa con el fin de encontrar que hay erróneo en el comportamiento del algoritmo implementado.

- Comprobar que Tello se ha lanzado correctamente: Como método de ayuda al usuario, y ya que la comunicación con el Tello (la cual abordaremos en el siguiente apartado) no es especialmente sólida, el launcher que lanza el driver activa una espera a que el dron termine de ejecutar su inicialización e imprimir por pantalla si esta ha tenido éxito o no. El dron a veces no conecta bien o pierde la conexion y por lo tanto no despegará ni realizará ninguna de las acciones posteriores que se le comanden, así que para eliminar dudas de si es error de código y de la implementación de la solución por parte del usuario, se ofrece esta ayuda.

- Usar la terminal: Una vez se lanza el driver del dron correctamente, y para probar funciones de la plantilla que se han implementado, la consola ha sido un apoyo fundamental para lanzar comandos de ROS2 directos al dron. Esto es, que en pruebas del PID que abordaremos en la solución, o de comandos de velocidad, ha habido que ajustar variables en el proceso, así que el mejor y más rápido método para que el dron no acabe en un accidente fatal en medio de esta calibración, se han usado comandos de aterrizar o entrar en estado de emergencia como soporte adicional.

- Usar ficheros del RADI: En ocasiones y para salidas y trazas del programa que sean de gran información y que sobrecarguen la salida de la pantalla de la consola de comandos, mediante código se puede redirigir estas trazas a ficheros. Se puede navegar con la consola a lo largo de la imagen docker del RADI y explorar la salida de estos ficheros.

Ya mencionando aspectos que no tienen por qué compartir todos los ejercicios, son los dos métodos de visualización de la cámara del dron:

- A través de RVIZ2: Uno de los botones de la interfaz gráfica te ofrece la visión constante del flujo de vídeo enviado por el Tello a través del topic dedicado a

ello. RVIZ2 mediante búsqueda de ese topic en concreto, activa la visualización. De forma predeterminada, RVIZ2 ofrece la imagen del topic seleccionado a un tamaño pequeño, modificando el launcher que lanza el drive del Tello, se modifica la inicialización de RVIZ2 cargando un fichero de configuración previamente diseñado por mí que cargue automáticamente el topic que va obteniendo las imágenes enviadas por el Tello, así como el tamaño de ventana para que se ofrezca una imagen nítida de esta visualización.

- A través del GUI: Este es un tema a desarrollar más ampliamente cuando se hable de la plantilla python diseñada para este ejercicio, pero uno de los métodos de esta plantilla es crucial para la arquitectura. Y es que se ofrece al usuario otro método de visualización de imágenes pero esta vez a su gusto, es decir, con GUI.showImage(img), por argumento, se le puede pasar cualquier imagen de tipo NumPy que este la plasmará en la interfaz de la web. Esto tiene el potencial que le quiera dar el usuario, ya que, puede realizar la misma función que se obtiene a través de la visualización por RVIZ2, es decir, observar las imágenes que se obtienen a través de la cámara del Tello, o puede servir como otro tipo de apoyo, por ejemplo: visualizar las imágenes del Tello pero después de ser aplicada la inferencia hecha por YOLO y se visualicen todas las cajas e ID's que se han detectado después del procesamiento.

5.4. Ejercicio Dron-Cuadrado

El primero de los ejercicios propuestos para usar el dron físico dentro de Robotics Academy es el ejercicio Dron-Cuadrado. La intención principal de este ejercicio es servir como ejercicio introductorio para que el usuario se familiarice con el control del Tello usando la plantilla y con el Tello en sí si no lo ha usado antes. Por ello, el cometido es tan sencillo como que un dron pueda despegar, dibujar un cuadrado con su trayectoria y aterrice idealmente en el mismo sitio del que partió.

Otros ejercicios de Robotics Academy cuentan con un objetivo visible para el usuario el cual alcanzar. Por ejemplo, en el ejercicio del Vacuum Cleaner[1], te tiene una pantalla en la que se puede ver el recorrido que lleva completado la aspiradora inteligente dentro de la casa, así como una puntuación de cuantas casillas lleva recorridas (y por lo tanto limpiadas) y las totales que quedan por recorrer.

En esta ocasión no es necesario una interfaz extra para el ejercicio, pues visualmente se ve el comportamiento del robot en vivo.

[1] `https://www.youtube.com/watch?v=-Uwl4TQNVpU`

5.5. Ejercicio Sigue-Personas

El segundo de los ejercicios incrementa el nivel con respecto al anterior y está destinado a usuarios que ya estén familiarizados con la robótica. El dron, tras su despegue, debe ser capaz de mantener a una persona en su cámara y seguirla, no solo con la cámara, que vaya se intente acercar a ella.

El dron debe seguir a esa persona sin importar tamaño, físico o color de su indumentaria. Tampoco tiene que importar que esa persona esté sentada, tumbada, de pie frente a la cámara, de lado o dada la vuelta. En resumen, la idea es que el sistema de detección sea robusto.

Además de que haga un seguimiento natural, es decir, que la persona pueda realizar acciones cotidianas, moviéndose a una velocidad estándar y que el dron sea capaz de seguirla con la cámara y acercarse a ella.

Otro aspecto importante es que el dron debería parar de seguir a esa persona cuando se encuentre cerca de ella, para evitar daños del dron a la persona o viceversa. Esta también es otra cuestión a abordar.

Lo que nos resumen el ejercicio en tres problemas distintos: cómo seguir a la persona con la cámara del dron, teniéndola en objetivo todo el rato, cómo acercarse a ella continuamente sin importar si esa persona se mueve y como punto final, cuando parar de seguir a esa persona, a qué distancia y cómo hacerlo.

5.5.1. Uso de YOLO como parte de la plantilla python

Esto se hace con el fin de ofrecer ya una librería controlada y acotada por la plantilla para el problema de la detección de una persona. Esto no quiere decir que el usuario no sea libre de usar el método que quiera para conseguir resolver este primer problema. El módulo YOLO cuenta con la siguiente forma y funciones:

- `YOLO()`: El constructor que inicializa YOLO y su red para inferir nuestras imágenes que nos vienen desde la cámara del Tello.
- `__call__(img)`: Esta es una función que permite a los programadores escribir clases donde las instancias se comportan como funciones y se les puede llamar como a una función.

```
    class Product:
    def __init__(self):
        print("Instance Created")

    # Defining __call__ method
    def __call__(self, a, b):
        print(a * b)

# Instance created
ans = Product()

# __call__ method will be called
ans(10, 20)
```

Código 5.1: Ejemplo de uso del metodo __call__

En este ejemplo se puede ver como la propia instancia de la clase obra como función con el fin de calcular un producto. En el caso de nuestro módulo YOLO hemos hecho algo así para llamar a su constructor:

```
person_tracker = YOLO()
```

Código 5.2: Instancia de YOLO

Para la función __call__ su llamada será siguiente:

```
bbox, cls_conf, cls_ids = person_tracker(img)
```

Código 5.3: Metodo __call__ de YOLO

Y es que esta función recibe una imagen y devuelve tres cosas distintas como salida. La variable `'bbox'` contiene las coordenadas de cajas que encierran a los objetos detectados por la red neuronal de YOLO, estas coordenadas tienen el aspcecto de [x1, y1, x2, y2] donde las coordenadas (x1, y1) harán referencia a las coordenadas de la esquina superior izquierda de una de las cajas que es un objeto detectado mientras que (x2, y2) serán las coordenadas de la esquina inferior derecha. La variable `'cls_conf'` contiene la confianza de que sea cierto su estimación, por ejemplo: la confianza de que el objeto encerrado en el área = [0,0,20,20] sea una taza. Este número tiene un rango posible [0.0-1.0], con decimales incluidos, donde 1 es la máxima confianza. Y `cls_ids` contiene los identificadores de aquello que detecta, esto tiene como ID's números enteros

donde cada número hace referencia un tipo de objeto, por ejemplo el ID = 0 se refiere a una persona, el ID = 67 a un teléfono móvil. Existen 79 identifadores disponibles a detectar por YOLO. En este ejercicio solo necesitamos el ID = 0, puesto que nuestro objetivo es detectar únicamente personas.

Capítulo 6
Soluciones de referencia

Después de la integración de la estructura de ambos ejercicios en RoboticsAcademy, vamos a describir una solución para ambos con el fin de probar tanto el driver del Tello en ROS2 Humble propuesto como de las plantillas realizadas para la creación de los mismos.

6.1. Solución de referencia Drone-Cuadrado Físico

Recordemos que el propósito de este ejercicio sirve como introducción a la robótica aérea y más en concreto al dron Tello donde se trata de conseguir que el dron sea capaz de dibujar un cuadrado con la trayectoria que realiza y aterrizar posteriormente.

Vamos a describir dos modos de resolver este ejercicio, la solución por posición y la solución por velocidad.

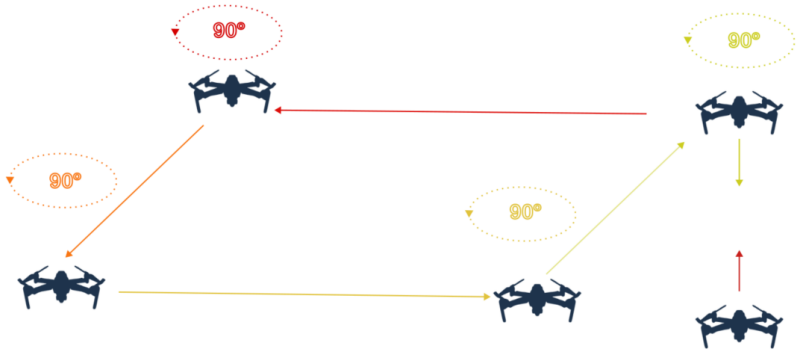

Figura 6.1: Esquema ejercicio dron-cuadrado

6.1.1. Solución por posición

Para la solución de este ejercicio por posición es necesario ejecutar una serie de acciones sencillas ofrecidas por la plantilla de python que hemos realizado. La solución

pasará por tres estados principalmente.

- Despegue: A pesar de ser sencillo no deja de ser esencial, que lo primero que debe hacer el dron para poder realizar este ejercicio es despegar. Esto lo conseguimos a través de la función `HAL.takeoff()`.

- Bucle de avance-giro: En el caso del ejercicio de realizar un cuadrado, este bucle ejecutará cuatro veces. Se basará en que cada vuelta del bucle avance una distancia determinada y gire 90 grados hacia un lado predeterminado.

  Con el fin de extrapolar el desarrollo del código y lograr una solución más portable y ágil, el desarrollo de la solución ha previsto realizar cualquier tipo de polígono cerrado, no solo un cuadrado. El dron así podrá realizar cualquier polígono con su trayectoria, incluído el cuadrado que se busca.

  Para dibujar un polígono regular usando el dron, necesitas saber cuántos grados debe girar el Tello entre cada segmento. Un polígono regular es aquel que tiene todos sus lados y ángulos iguales. Para calcular el ángulo interno de un polígono regular, se puede usar la fórmula:

$$Angulointerno = \frac{(n-2) \times 180}{n} \tag{6.1}$$

  Ecuación 6.1: Ángulo interno de un polígono regular

  Mientras que la del ángulo externo, que será el de giro del dron, es la siguiente fórmula:

$$Anguloexterno = 180 - Angulointerno \tag{6.2}$$

  Ecuación 6.2: Ángulo externo de un polígono regular

  Donde, trás simplicar la función con unos pocos pasos se obtiene la fórmula final:

$$Anguloexterno = 180 - \frac{(n-2) \times 180}{n} \tag{6.3}$$

  Ecuación 6.3: Ángulo externo de un polígono regular (desarrollado)

  Con esto obtenemos el ángulo de giro que tendrá el dron con el fin de dibujar cualquier polígono con su trayectoria, donde n, es el número de lados con el que contará el polígono.

  Las funciones a usar de la plantilla python creada serán las siguientes:

$$Anguloexterno = \frac{360}{n} \tag{6.4}$$

Ecuación 6.4: Ángulo de giro que tendrá el dron

- `HAL.forward(distance)`: La distancia a recorrer hacia adelante es lo de menos del ejercicio, esto solo es para determinar la longiutd de cada lado del polígono. Esto va a gusto de cada usuario, en la solución se ha usado un distance = 20. Lo que quiere decir que se avanzan 20 centímetros.
- `HAL.turn_left(degrees)`/`HAL.turn_right(degrees)`: Cualquiera de las dos funciones nos valen, usaremos por ejemplo la función de `HAL.turn_left` porque soy zurdo. Los degrees que se usan como parámetro de la función es el ángulo externo calculado en la última ecuación 360/n. Donde recordemos que n es el número de lados, en este caso, cuatro.

  Tras realizar este bucle n veces, en este caso cuatro, pasaremos al último estado.

- Aterrizaje: Tras realizar el cuadrado acabamos aterrizando. La idea es que la posición desde la que se despega y la posición donde se aterriza sea la misma, si estas dos posiciones son iguales habremos completado con éxito el ejercicio. Esta acción se llevará a cabo con función `HAL.land()`

6.1.2. Solución por velocidad

En el caso de desarrollar por velocidad, la solución vamos a tener en cuenta el factor del tiempo, y por ello a usar funciones de la plantilla que manipulen la velocidad del Tello. Seguimos usando la misma máquina de estados donde el despegue y el aterrizaje no van a sufrir ningún tipo de modificación, pues no son funciones por posición o por velocidad, son funciones de estado bloqueantes. En el caso del estado que nos queda:

- Bucle de avance-giro: Como hemos mencionado, la fórmula de la velocidad es la siguiente:

$$v = \frac{d}{t} \tag{6.5}$$

Ecuación 6.5: Fórmula para la velocidad, donde v es la velocidad, d es la distancia y t es el tiempo.

Con esto podemos pasar a usar la variable del tiempo con la librería time de python. Con esta librería podemos usar la función `time.time()`. Esta función nos devuelve el

tiempo en segundos en epoch actual. De esta manera podemos medir el tiempo pasado obteniendo el tiempo actual en dos instantes de tiempo distintos. Si queremos seguir la regla de la anterior solución y realizar el polígono de una distancia igual a 20, nos apoyaremos en la fórmula de la velocidad para obtener que con una distancia igual a 20 y una velocidad escogida a dedo, por ejemplo 20, necesitaremos comandar velocidad de 20 centímos por segundo al Tello durante un segundo. Para ello usaremos la función `HAL.setVX(20)`.

Recordemos que las funciones de set dejan la velocidad que se le comande hasta que otra velocidad en el mismo eje la sobreescriba. Con lo cual, cuando pase ese segundo deberemos hacer un `HAL.setVX(0)` para que pare su avance en el eje X del Tello. En el momento que empecemos cada vuelta de este bucle, cogeremos la marca de tiempo actual, lo siguiente que haremos será comandar la velocidad igual a 20 centímetros por segundo. Tras esto, haremos espera activa esperando a que pase un segundo desde que cogimos la primera marca de tiempo, esto se consigue cogiendo marcas de tiempo constantemente y haciendo la diferencia con la primera marca que conseguimos.

En código de python para que se entienda mejor:

```
# Establecer la velocidad inicial a 20 cm/s
HAL.setVX(20)

# Tomar una marca de tiempo inicial
tiempo_inicial = time.time()

# Establecemos el avance
avance_activo = True

while avance_activo:
    # Obtener el tiempo actual
    tiempo_actual = time.time()

    # Calcular el tiempo transcurrido
    tiempo_transcurrido = tiempo_actual - tiempo_inicial

    # Comprobar si ha pasado un segundo
    if tiempo_transcurrido >= 1.0:
        # Detener la velocidad
        HAL.setVX(0)
        # Terminamos el estado de avance
        avance_activo = False
```

Código 6.1: Ejecucion de HAL.setV en un tiempo determinado

Una vez realizado el avance hacia adelante, pasaremos al giro, seguiremos usando

la velocidad como filosofía así que usaremos la función `HAL.setW(w)` para girar. Por recordar, esta función hace que el dron gire sobre sí mismo a una velocidad w medida en grados por segundo, con lo cual, tambien necesitaremos el tiempo aquí.

La fórumla a usar es la misma, es decir, elegida una velocidad de giro, en la solución usada una velocidad de 45 grados por segundo y sabiendo que tenemos que girar 90 grados (o 360/numero_de_lados si lo extrapolamos a la solución general para cualquier polígono) tendremos que mantener la velocidad durante 2 segundos. Tras esto y como el dron necesita sobre escribir la velocidad de giro para poder parar, necesitaremos llamar a la función `HAL.setW(0)`. El código en python sería algo así:

```
# Establecer la velocidad de giro inicial a 45 grados/s
HAL.setW(45)

# Tomar una marca de tiempo inicial
tiempo_inicial = time.time()

grados_girados = 0
giro_activo = True

while giro_activo:
    # Obtener el tiempo actual
    tiempo_actual = time.time()

    # Calcular el tiempo transcurrido
    tiempo_transcurrido = tiempo_actual - tiempo_inicial

    # Calcular los grados girados
    grados_girados = 45 * tiempo_transcurrido

    # Comprobar si ha girado 90 grados
    if grados_girados >= 90:
        # Detener el giro
        HAL.setW(0)
        giro_activo = False
```

Código 6.2: Ejecucion de HAL.setW en un tiempo determinado

Y con el sistema de avance y de giro realizados, tan sólo habrá que llamarlos 4 veces para realizar el cuadrado correctamente y tras esto aterrizar. En el caso de querer extrapolar la solución a un polígono de n lados, ejecutar el bucle n veces.

6.2. Solución de referencia Sigue-Persona Físico

Ahora vamos a describir la solución desarrollada al ejercicio Sigue-Persona. Recordemos que el ejercicio se basa en que el dron, una vez despegado del suelo, sea capaz de seguir a una persona y acercarse a ella indefinidamente y deje de seguirla si distancia a la persona es demasiado corta.

Podemos separar el problema a resolver en pequeñas tareas que ir resolviendo poco a poco.

6.2.1. Detección de objetos en la imagen

Tras despegar, tenemos que detectar personas en la imagen. Aquí tenemos dos opciones realmente:

- Mantener un estado de pausa o de espera hasta que la cámara detecte a una persona, es decir, que una persona entre dentro del objetivo de la cámara de una persona.

- Buscar a persona por medio de un giro constante. Esto quiere decir que mientras no existan personas detectadas por la cámara, comandar al dron un giro constante con el que facilite el objetivo de buscar una persona a la que empezar a seguir.

En el caso de mi solución finalmente me decanté por la primera opción. Ahora pasaremos a que, tras tener una persona en el objetivo de la cámara, ser capaces de saber qué ese conjunto de píxeles representa a una persona para poder decir realmente que la hemos detectado.

Para esto haremos uso de la librería de YOLO implementada como plantilla para este ejercicio, esto nos permitirá detectar varios objetos en la imagen, todos de los que dispone YOLO realmente.

Usaremos YOLO del modo que se recomienda, a YOLO le pasaremos las imágenes que vaya recogiendo la cámara del Tello y este nos devolverá todo aquello que detecte en forma de listas, confianza e identificadores como ya hemos descrito en la plantilla python de este ejercicio.

Tras esto y con todo detectado, tendremos que filtrar esa lista de cajas en las que se encierran objetos, la confianza e identificar de cada una de ellas para solo tener en cuenta las que sean referentes a personas. Las 3 variables que se devuelven son listas que tienen el mismo orden de detección, es decir:

- La primera caja de la lista de cajas encierra un área de una cama.

- El primer elemento de la lista de confianza de detección hará referencia a la confianza de detección de que la cama sea una cama.
- El primer elemento de la lista de identificadores tendrá el número que hace referencia al identificador de una cama.

Sabiendo que están en el mismo orden todas las listas, para poder filtrar todas las cajas por personas, tan sólo tendremos que saber cual es el identificador de una persona. Esto se menciona en la documentación del ejercicio para que el usuario no tenga que comprobar todos los objetos que es capaz de detectar el Tello uno a uno. El identificador de la persona es el 0. Esto quiere decir que si en la lista de identificadores encontramos un elemento de valor 0, esa misma posición en la lista de identificadores hará referencia en el resto de lista a la caja donde se encierra a esa persona y la confianza de su detección.

Con lo cual ya tendríamos filtradas a las personas del resto de objetos de la imagen, sabiendo su posición estimada por la red neuronal de YOLO y la confianza de que sea una persona.

Como sabemos el rango de confianza es de [0.0-1.0] donde 0.0 es que se lo acaba de inventar que es una persona y 1.0 es que está cien por cien seguro de que lo es. Para evitar falsos positivos y seguir a objetos en vano, tendremos solo en cuenta personas detectadas con un mínimo de 0.6, es decir, estar seguros al sesenta por ciento de que se trata de una persona.

Con el fin de ejemplificar que se ha detectado una persona, pondremos una traza que hará uso de la consola donde pondremos un mensaje avisando que hemos detectado a la persona. Además, con la función GUI.showImage(img) mostraremos una imagen alterada del fotograma de la cámara del robot.

Sobre esta imagen que nos pasa el dron, dibujaremos la caja que encierra a la persona para que se vea visualmente donde se encuentra, así como una etiqueta con la confianza de la decisión de que es una persona.

Con la persona y su posición detectada pasamos al siguiente punto.

6.2.2. Controlador PID rotatorio

El siguiente paso es que, una vez detectada la persona, sabiendo que es una persona realmente, es no perderla de vista más. El hecho de seguir a una persona pasa por primero seguirla con la vista.

Para conseguir esto, hemos implementado un sistema que es capaz de mantener a la persona en el centro de la imagen para posteriormente seguirla. Para ello vamos a

aplicar la teoría de control, en concreto vamos a usar un PID para no perder de vista a la persona, un PID para girar sobre sí mismo.

Para implementar un PID debemos decididr qué vamos a usar como entrada para el PID y sobre qué se va a calcular el error y cual va a ser la salida en base a ese error, con el fin de corregirlo y acercarlo lo máximo posible a un error nulo, error igual a cero.

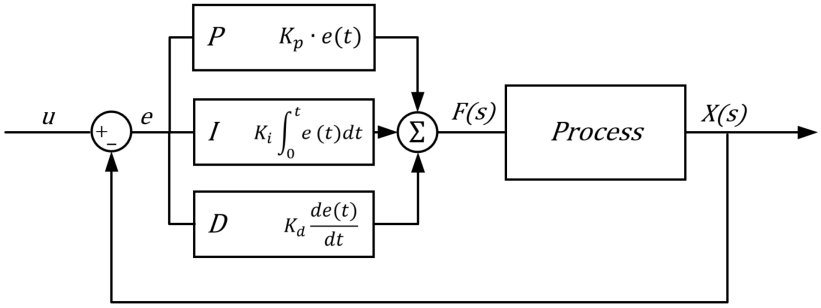

Figura 6.2: Esquema controlador PID

Recordemos la fórmula del PID que va a usarse en el código de la solución:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{d}{dt} e(t) \tag{6.6}$$

Para definir cada uno de los componentes en el desarrollo del código:

- `u(t)` hace referencia a la salida del controlador PID. En el código esta salida no será otra cosa que no sea los grados a girar para el dron. Tanto salidas positivas (giro a la derecha), como salidas negativas (giro a la izquierda) serán admitidas, con el rango siendo [-360, 360] grados. Buscamos que el giro tenga que ser cero, por eso elegir el giro como salida es una opción inteligente, estamos buscando que el dron no tenga que girar, es decir, que tengamos a la persona centrada en la imagen y por lo tanto un giro no sea necesario.

- `e(t)` hace referencia al error sobre el que se calcula la salida. En el código que estamos desarrollando este error se basará en la diferencia de la posición de la persona con respecto al centro de la imagen. Esto se consigue del siguiente modo: De la caja que encierra a la persona extraeremos el centro, creando así una función auxiliar que nos ayude a ello. Con el centro de la caja obtenido cogeremos su diferencia horizontal con el centro de la imagen, esto es, la diferencia con la mitad del ancho de la imagen.

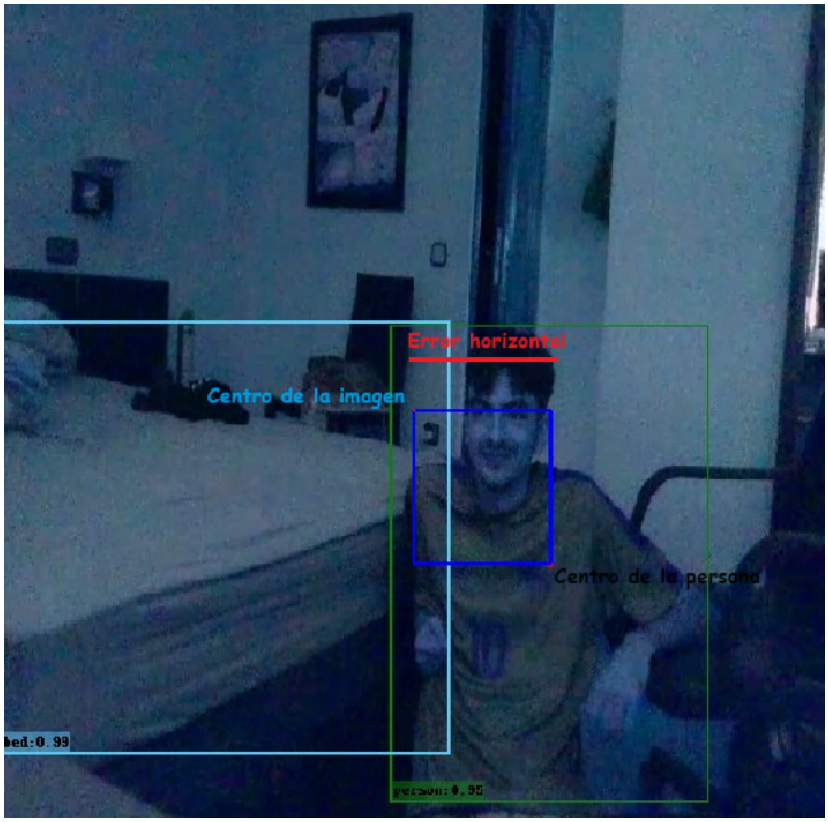

Figura 6.3: Error horizontal del PID

A partir de este error calcularemos la salida.

- La constante proporcional: Esta constante es la que multiplica al error que acabamos de definir y decide su importancia a la hora del cáculo de la salida. El peso que se le ha asignado en la solución es de 0.2.

- La integral del error: Es un concepto que llevado a código se puede simplificar de manera sencilla. Trataremos esta integral como lo que es realmente, una suma. Se sumarán todos los errores y los acumularemos en una variable. Esto es literalmente lo que hace la función ya que suma los errores en suma desde el instante 0, el inicio del programa, hasta t, que es el instante de tiempo en el que nos encontremos en el momento del cálculo de la salida del controlador PID.

- La constante integral: Es la constante que mide el peso o importancia que tendrá el error acumulado en el cálculo de la salida. Esta constante la hemos definido a 0.01. Es decir, tendrá un peso no muy alto a la hora de calcular cuantos grados debemos de girar.

- La derivada del error: la derivada no deja de ser una diferencia entre instantes de tiempo, así que tomaremos este concepto y lo llevaremos al código de modo que

la derivada del error será igual a la diferencia entre el error actual y el error en el instante anterior. Este error del instante anterior se refiere al error resultante de esta misma diferencia en la anterior vuelta del bucle.

- La constante derivativa: Esta constante define el peso de la derivada del error sobre la función que calcula la salida. Se le ha asigando un valor de 0.15.

Para mostrar cómo sería este controlador en pseudocódigo en python:

```
# Parametros PID
Kp = 0.2
Ki = 0.15
Kd = 0.01

# Inicializacion de variables
error_anterior = 0
suma_errores = 0
valor_deseado = 100 # Este es el valor que queremos alcanzar

while True:
    # Leer el valor actual del sistema
    valor_actual = leer_valor_actual()

    # Calcular el error
    error = valor_deseado - valor_actual

    # Actualizar la suma de errores para el termino integral
    suma_errores += error

    # Calcular la derivada del error
    derivada_error = error - error_anterior

    # Calcular la senal de control usando la formula PID
    u = Kp * error + Ki * suma_errores + Kd * derivada_error

    # Aplicar la senal de control al sistema
    aplicar_control(u)

    # Guardar el error actual para el proximo ciclo
    error_anterior = error
```

Código 6.3: Pseudocodigo de un controlador PID en python

Tras esto ya tendremos el giro en grados que debe realizar el dron para intentar mantener a la persona detectada lo más en el centro posible así que pasaremos al siguiente punto, mantener el dron a una altura igual a la mitad de la persona.

6.2.3. Controlador PID vertical

Mantener una altura determinada es esencial, esto es porque si el dron detecta a una persona pero está muy por encima de donde debe, por ejemplo, a la altura de la cabeza de la persona, en cuanto el dron se acerque a ella va a perderla de vista de la cámara

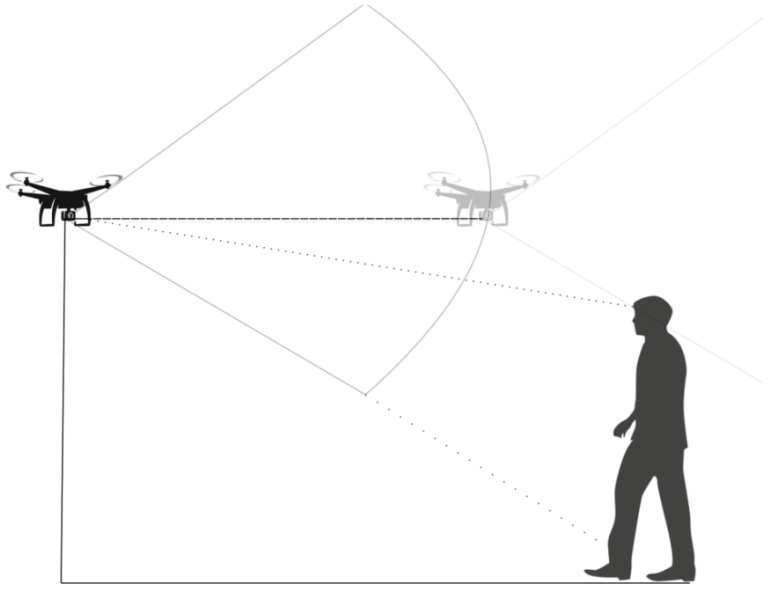

Figura 6.4: Ejemplo de mal seguimiento 1

Por debajo de una altura determinada tampoco es lo más lógico y lleva a problemas fácilmente, ya que si volamos muy bajo, llegará a un punto donde solo le detectaremos los pies a la persona y con mucha probabilidad el dron dejará de detectarla como persona.

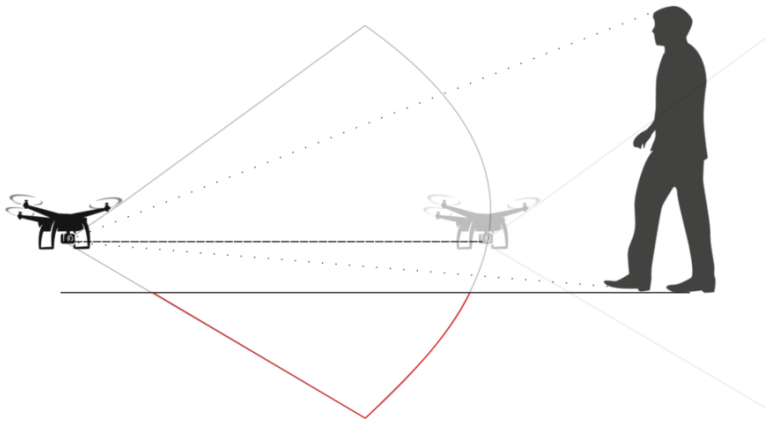

Figura 6.5: Ejemplo de mal seguimiento 2

Por eso lo ideal es mantener al dron a la altura igual a la mitad de la persona idealmente. Esto hará que según se acerque menos partes de la persona se salgan de plano y por lo tanto el dron va a seguir detectándolo durante mucho más tiempo.

Por lo tanto ya tenemos la entrada y salida para nuestro siguiente PID establecidas. En este caso no es tan vital mantener a una persona en altura exacta como si mantenerla en el centro de la imagen, por lo tanto no haremos uso de la parte integral del PID, así que no estará previsto un error cero en este caso, lo dejaremos como PD.

$$u(t) = K_p e(t) + K_d \frac{d}{dt} e(t) \tag{6.7}$$

Las partes de nuestro nuevo controlador serán las siguientes:

- `u(t)` hace referencia a la salida del controlador PD. En este caso, la salida ideal es la que haga que el dron se mantenga en el centro de la persona, es decir, que el centro de la cámara del dron apunte al mismo sitio que el centro de la persona. Por lo tanto la salida será la más lógica, será el control de velocidad del dron en el eje Z. Valores en un rango [-100, 100] centímetros por segundo, permitiendo valores negativos para poder hacer al dron descender y valores positivos para hacerlo ascender.

- `e(t)` hace referencia al error sobre el que se calcula la salida. Como sabemos queremos mantener a la persona en el centro horizontalmente, pero ahora también verticalmente, es decir, que el centro de la persona coincida con la mitad vertical de la imagen. Por ello vamos a hacer uso de la función auxiliar que hemos creado antes que nos da el centro de la caja que encierra a la persona pero esta vez

no nos quedaremos con las coordenadas horizontales de ese punto, sino con las verticales. De este modo, el error será la diferencia de esta coordenada vertical del punto y la mitad del alto de la imagen del Tello.

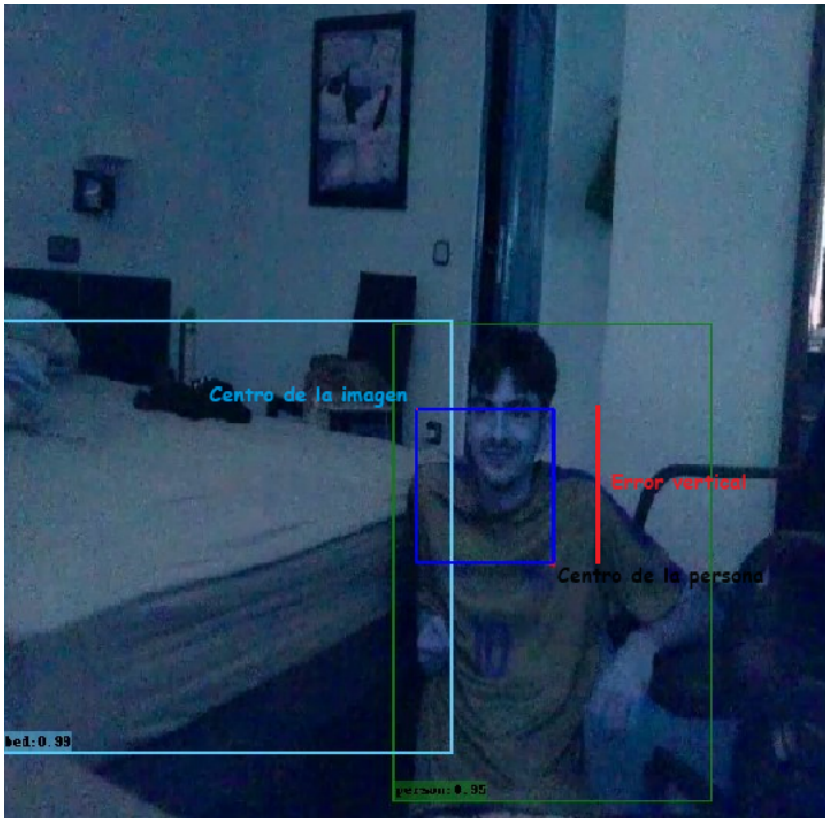

Figura 6.6: Error vertical del PID

A partir de este error calcularemos la salida.

- La constante proporcional: Esta vez la constante proporcional tendrá un valor de 0.15.
- La derivada del error: seguimos con el concepto de que la derivada es la diferencia entre errores en dos instantes de tiempo y por ello será la diferencia entre el error actual y el error anterior.
- La constante derivativa: Esta constante define el peso de la derivada del error sobre la función que calcula la salida. Se le ha asigando un valor de 0.15.

Con esto ya tenemos un método para mantener a la persona en el centro de la imagen vertical y horizontalmente, así que pasaremos al último paso, acercanos a ella.

6.2.4. Acercamiento a la persona

Pasamos a explicar el método seguido para poder acercanos a la persona. Esto puede tener soluciones más complejas como asignar una velocidad en función de la altura de la persona: Esto es, asignar una función que comande velocidades lineares en el eje X al Tello más grandes si el área que encierra a la persona es más pequeña y velocidades más bajas cuando estemos cerca de ella. Esto parece ser una muy buena idea y puede usarse perfectamente.

Si no se ha acabando usando este método es por las limitaciones que yo mismo me he impuesto al usar YOLO y su red neuronal ya que esta varía mucho el tamaño de las cajas que encierran a las personas aunque esta se encuentre a una misma distancia, por lo tanto, esta irregularidad ha hecho que descartemos esta opción.

Idealmente ya tenemos a la persona centrada totalmente en la imagen, de tal modo que el centro de la caja que encierra a la persona sea prácticamente igual al del centro de la imagen. Así que la problématica de acercarse de la persona puede simplificarse hasta convertirlo en un simple número constante. Esto quiere decir que, podemos usar una velocidad constante con la que nos acercaremos a la persona siempre y cuando la tengamos detectada, sin importar si está realmente centrada en la imagen, ya que, si nuestro controladores son buenos y sólidos, la persona estará centrada o muy cerca de estarlo todo el tiempo. Por ello se ha usado una velocidad en el eje X constante igual a veinte centímetros por segundo.

En el código de la solución, todo control que ha sido definido aquí; giro sobre sí mismo del dron, cambio en la altura del dron y velocidad constante en el eje X, ha sido encapsulada en la función `HAL.setRC(a,b,c,d)` donde, por recordar:

- a: Esto es el movimiento en el eje Y del dron, es decir su movimiento a izquierda o derecha. En esta solución se ha despreciado y se ha dejado en todas las llamadas a cero.
- b: Esto es el movimiento en el eje X del dron. Esta es la velocidad constante de acercamiento que se ha establecido por la que el dron va hacia adelante siempre que tiene a una persona detectada, por lo tanto su valores solo pueden ser 0 o 20.
- c: Esto es el movimiento en el eje Z del dron. Su valor es la salida de nuestro PID vertical.
- d: Esto es el giro sobre sí mismo del dron en el eje Z. Su valor es la salida de nuestro PID horizontal.

Con esto ya tendríamos todos los componentes para seguir y acercarse a la persona.

Tan sólo falta describir cuando para el dron, es decir, cuando detecta que está demasiado cerca de la persona y debe parar para evitar choques dron-persona innecesarios.

Al igual que YOLO nos ha limitado a la hora de elegir una velocidad con la que acercarse a la persona, esta vez nos va a ayudar, ya que, cuando la cara y extremidades de la persona salen de plano, este deja de detectar a la persona y por tanto para. En efectos prácticos, el dron para a unos quince centímetros de la persona, impidiendo el choque. Tras esto la persona puede alejarse para que el dron siga con el seguimiento o enseñar la cara, que es el método donde más rapido detecta YOLO a una persona.

Tras todos los puntos expuestos, cómo han sido desarrollados y por supuesto, programados, tenemos la solución referencia del ejercicio sigue-persona terminada.

Capítulo 7
Conclusiones

Para finalizar la memoria de este Trabajo Fin de Grado, vamos a describir un resumen con los objetivos que han sido logrados junto con el aprendizaje que se ha adquirido mediante el desarrollo del TFG. Además, se van a proponer posibles vías futuras que lleven más lejos las metas conseguidas en este TFG.

7.1. Conclusiones y aprendizaje

7.1.1. Objetivos cumplidos

- Hemos logrado adentrarnos en el contexto e infraestructura de Robotics Academy con el fin de poder implementar un nuevo ejercicio. Como aporte adicional hemos aprendido de su funcionamiento tanto para el RADI 3, donde se usa ROS1, hasta participar en el desarrollo de RADI 4 cuya implementacion ya está basada en ROS2.

- El siguiente de los objetivos era implementar un soporte del Dron Tello en ROS2 Humble. Esto se ha conseguido partiendo del desarrollo de uno existente en ROS2 Foxy, creando así un nuevo soporte funcional y usado en este TFG como prueba de su robustez para ROS2 Humble.

- Hemos aprendido sobre todos los pasos para implementar un nuevo ejercicio, pasando por la plantilla python y la página web y la creación de las diferentes interfaces para comunicar y controlar el dron Tello con Robotics Academy mediante las plantillas python.

- Hemos introducido este driver, y todo lo necesario para usar el Tello dentro de un contenedor Docker para encapsularlo junto al resto de ejercicos de Robotics Academy superando el problema de que el Tello debe comunicarse utilizando únicamente su red Wi-Fi que él mismo establece.

- Se han creado satisfactoriamente dos ejercicios en base a estos pasos; el dron-cuadrado y el sigue-personas. Y crear soluciones referencias para ambos ejercicios sirviendo como demostración de que se pueden resolver ambos junto a la documentación que completa a ambos ejercicios.

7.1.2. Competencias adquiridas

Tras el desarrollo de este TFG las competencias adquiridas han sido:

- Conocimiento avanzado de Robotics Academy (frontend y backend): creación de dos nuevos ejercicios y participación en el desarrollo en el nuevo RADI 4.
- Aprendizaje básico sobre React, HTML, CSS y JavaScript
- Conocimiento sobre Docker: creación de imágenes complejas y uso de ellas.
- Conocimiento sobre la comunicación hardware del Tello con el fin de crear el driver de ROS2.
- Profundización en ROS2 Foxy y Humble.
- Aprendizaje general sobre YOLO y métodos para detectar personas.

7.2. Líneas futuras

Las metas de este TFG han sido las establecidas en Objetivos, pero la investigación y exploración de llevarlas más allá es muy interesante:

- Sería posible profundizar y acabar implementando el soporte del mencionado experimento de manos para añadir más complejidad al ejercicio y más funcionalidad para ayudar al usuario.
- Crear un ejercicio simulado que se base en la misma premisa y objetivo que el segundo ejercicio creado. Creando así un Tello simulado para ROS2 Humble y un entorno en Gazebo con una persona moviéndose sobre el que actuar.
- Implementación con varios drones Tellos simultáneamente: Esto sería posible si la máquina local sirviera como un switch, así se conseguiría una patrulla de Tello capaces de comunicarse y organizarse para seguir a diferentes personas.
- Sería muy interesante crear una librería/ejercico de mapeo basado en la cámara de Tello. Para crear mapas tan sólo usando el único sensor que el Tello nos ofrece, su cámara.

Bibliografía

[, 2023] (2023). Docker documentation. https://docs.docker.com/.

[Alberto, 2023] Alberto, D. (2023). Sistema para la automatización de tareas de inspección de estructuras físicas utilizando un dron. https://repositorio.uniandes.edu.co/handle/1992/69233.

[Arias, 2022] Arias, P. (2022). Infraestructura de programación de robots aéreos y aplicaciiones visuales con aprendizaje profundo. Master's thesis, Universidad Rey Juan Carlos.

[Bradski, 2023] Bradski, G. (2023). Opencv tutorial. http://roswiki.autolabor.com.cn/attachments/Events(2f)ICRA2010Tutorial/ICRA_2010_OpenCV_Tutorial.pdf. Stanford University.

[Caminero, 2022] Caminero, C. (2022). Ejercicios sigue-persona para la plataforma académica robotics academy, usando un robot real y simulado en ros2. Master's thesis, Universidad Rey Juan Carlos.

[Craig, 2023] Craig, J. J. (2023). Introduction to robotics: Mechanics and control. https://books.google.es/books?hl=es&lr=&id=hRzOp_qdxG8C&oi=fnd&pg=PR5&dq=robotics&ots=fR9g91THZs&sig=0v8k6yKRnaWFmJB4qjVcK4yGd9I#v=onepage&q=robotics&f=false.

[Feron, 2023a] Feron, E. (2023a). Aerial robotics. https://www.researchgate.net/profile/Eric-Johnson-67/publication/225247482_Aerial_Robotics/links/02e7e539a0c5a42b25000000/Aerial-Robotics.pdf.

[Feron, 2023b] Feron, E. (2023b). What is educational robotics? https://ebotics.com/what-is-educational-robotics/.

[GIAMARCHI, 2023] GIAMARCHI, F. (2023). Mobile robots. https://www.researchgate.net/publication/355067102_Context_Representation_and_Reasoning_in_Robotics-An_Overview.

[Gómez, 2023] Gómez, S. (2023). Herramientas software para ejecución y evaluación de planes de robots aéreos en misiones de inspección. https://oa.upm.es/75177/.

[Hatzilygeroudis, 2023]
Hatzilygeroudis, I. (2023). Context representation and reasoning in robotics-an overview. https://www.researchgate.net/publication/355067102_Context_Representation_and_Reasoning_in_Robotics-An_Overview.

[Knospe, 2023] Knospe, C. (2023). Pid control. https://www.researchgate.net/profile/Carl-Knospe/publication/3207700_PID_control/links/542c19bf0cf29bbc126b32f6/PID-control.pdf?_sg%5B0%5D=started_experiment_milestone&origin=journalDetail&_rtd=e30%3D.

[Kumar, 2023] Kumar, V. (2023). Robotics: Aerial robotics. https://www.coursera.org/learn/robotics-flight.

[Liu, 2023] Liu, C. (2023). Object detection based on yolo network. https://ieeexplore.ieee.org/abstract/document/8740604/.

[Nehmzow, 2023] Nehmzow, U. (2023). Mobile robots, a practical introduction. https://www.google.es/books/edition/Mobile_Robotics/hrjkBwAAQBAJ?hl=es&gbpv=1&dq=robotics+movil&printsec=frontcover.

[Ospennikova, 2023] Ospennikova, E. (2023). Educational robotics as an inovative educational technology. https://www.sciencedirect.com/science/article/pii/S1877042815059431.

[Simrock, 2023a] Simrock, S. (2023a). Control theory. https://cds.cern.ch/record/1100534/files/p73.pdf.

[Simrock, 2023b] Simrock, S. (2023b). Ros2 documentation: Humble documentation. https://docs.ros.org/en/humble/index.html.

[Tentone, 2023] Tentone (2023). Dji tello ros2. https://github.com/tentone/tello-ros2.

[Zhang, 2023] Zhang, F. (2023). Mediapipe hands: On-device real-time hand tracking. https://arxiv.org/abs/2006.10214.

Apéndice A
Experimento con el uso de manos

Con el fin de experimentar nuevas fuentes de interacción con el dron, y un método claro para que el dron pueda reaccionar no solo a la persona sino a qué esté haciendo o que gesto haga, se ha experimentado el uso de manos dentro de la plantilla de Robotics Academy.

La librería MediaPipe, para python, nos ofrece soporte para este experimento. MediaPipe es una biblioteca de procesamiento desarrollada por Google. Se utiliza principalmente para la construcción de aplicaciones de visión artificial, procesamiento de señales de audio, y algunas aplicaciones de aprendizaje automático. MediaPipe ofrece una estructura flexible para conectar diferentes componentes de procesamiento de datos, lo que permite a los desarrolladores construir aplicaciones complejas de manera más sencilla.

La biblioteca ofrece soluciones preconstruidas para una variedad de tareas, como reconocimiento de gestos, seguimiento de objetos, estimación de pose, segmentación de imagen, y muchas más. Estas soluciones son altamente eficientes y están optimizadas para funcionar en tiempo real en diversos dispositivos, incluidos móviles y computadoras.

Dentro de nuestro ejercicio, queremos centrarnos en el uso de la detección de manos por parte de MediaPipe. Esta librería trata de encerrar una mano detectada en una caja y a partir de ella estimar los puntos claves de esa mano, esto se hace para poder diferenciar entre muñeca, dedos y sus posiciones y orientaciones.

En la plantilla, estas funciones descritas, se ha probado con la siguiente implementación:

- Dedos índice y anular hacia arriba: En estado de reposo del robot, en vez de usar `HAL.takeoff()`, se añade más protagonismo a la persona que va a seguir y siempre y cuando el Tello esté observando a la persona desde el punto en el que está en reposo, el dron tras reconocer este gesto despegará y empezará a seguir

a la persona.

- Palma en signo de STOP: Para que el seguimiento se pause, se ha experimentado esta funcionalidad. El tello al detectar la palma de la persona considerará que debe de parar de seguir a la persona y pasar a estado de pausa, es decir, lo que haría `HAL.pause()`, para toda velocidad comandada hasta el siguiente gesto.

- Puño cerrado: Para que el seguimiento se reanude, el dron debe ver a la mano de la persona que pretende seguir con la mano cerrada, en forma de puño. Ahí volverá a moverse intentando seguir y acercarse a la persona.

- Dedos índice y anular hacia abajo: Este estado es algo parecido al fin del seguimiento total o al menos eso se pretende. Ya que este ejercicio no cuenta con una meta que alcanzar con la que se acaba el ejercicio, sino que sigue a la persona indefinidamente, se ha experimentado con que, tras hacer este gesto, el dron aterrice hasta que el usuario realice el primer gesto descrito.

A pesar de que sería un añadido interesante al ejercicio, no se ha podido implementar en Robotics Academy esta funcionalidad basada en Law por falta de tiempo. Aun así aquí hay algunas imágenes de los experimentos realizados para que se pudiera implementar.

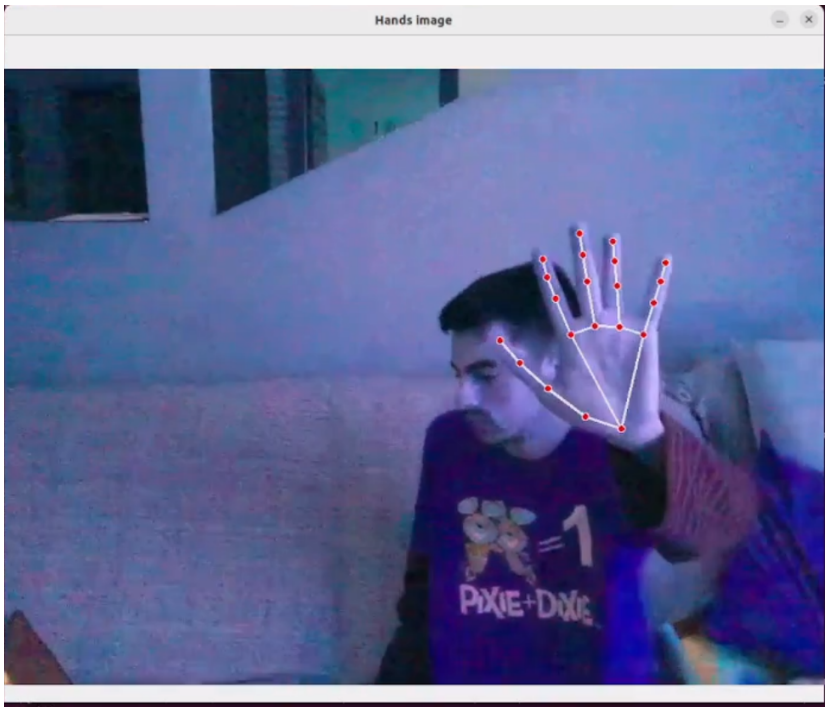

Figura A.1: Ejemplo libreria MediaPipe