



## ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

DOBLE GRADO EN INGENIERÍA DE SISTEMAS  
DE TELECOMUNICACIÓN Y ADMINISTRACIÓN Y DIRECCIÓN DE  
EMPRESAS

### **TRABAJO FIN DE GRADO**

# Nuevas Prácticas Docentes de Robótica en el Entorno JdeRobot-Academy

Autor: Pablo Moreno Vera

Tutor: José María Cañas Plaza

Curso académico 2018/2019

# Agradecimientos

Me gustaría aprovechar esta parte del trabajo para dar a conocer un poquito de mí, de mi esencia y qué la ha causado. Para dar a conocer por qué he conseguido llegar hasta donde estoy y por haber conseguido llegar siendo quién soy.

Quiero dar las gracias, en primer lugar, a mis padres, quienes nunca han dudado de mí, ni por un segundo. Me han apoyado en todo momento y me han llevado de la mano hasta donde estoy. Me han demostrado que hay que pelear cada paso, porque nadie te va a regalar nada y menos si algo que merece la pena. Han conseguido sacar una sonrisa de los momentos más dolorosos y demostrarme que, a veces, sólo es necesario sonreír. Me han enseñado la importancia de la familia y sé que sin ellos no lo habría logrado. Hemos pasado malos momentos, de todo tipo, y hemos salido de ello juntos. Juntos hemos derrotado a una enfermedad que se lleva millones de vidas en el mundo y que, en muchas ocasiones, no tiene cura como es el cáncer y, aun teniendo que estudiar junto a la camilla de mi padre en el hospital hemos conseguido derrotarlo y aprobar mis asignaturas. Este título lo hemos conseguido los 3 juntos, gracias por todo.

También quiero dar las gracias a mi tutor José María, por mostrarme el camino hacia el éxito. Por enseñarme que sin esfuerzo y estudio no se alcanza ningún objetivo propuesto y por mostrarme la robótica. La ciencia que me ha mostrado un lugar de interés perpetuo, un campo en continuo desarrollo y en el cual, tras un año en el desarrollo de este Trabajo de Fin de Grado, sólo tengo conocimiento de una infinitésima parte. Te agradezco aquellas palabras de interés por mí al terminar tu examen para que decidiera unirme a ti para que mostrases la robótica. Has hecho de mí una persona trabajadora y constante durante todas las semanas de reuniones contigo.

Quiero agradecer a mi familia los momentos de tranquilidad y locura que me aportáis. Sólo con aparecer hacéis que desconecte de todo y me dé cuenta de que no estoy solo, que tengo gente que me quiere a mi alrededor y no es fácil de encontrar en personas que no eliges sino que forman parte de ti al nacer. Aun así estaré siempre agradecido por pertenecer a ese pequeño sitio del mundo en el que estáis vosotros.

Gracias a mi pareja, Vida, por enseñarme que la felicidad puede estar en unos ojos. Por hacerme sonreír con una mirada y hacer que olvide todo lo que me rodea. Sin el descanso que me

das no habría aguantado la presión y no lo habría conseguido. Me has ayudado con asignaturas de las que ni siquiera comprendías lo que te contaba y aún así, sin ti, no habría llegado aquí. Eres mi lugar de descanso al que acudir cuando todo lo demás tiembla.

Me gustaría dar las gracias a todas las personas que habéis aportado un poquito en mi vida, que habéis conseguido hacer de mí una persona que está escribiendo un Trabajo de Fin de Grado para terminar una carrera de 4 años en la que muchos no confiaban pudiese finalizar. Muchas gracias a ti, Gabriel, por enseñarme que hay personas que te acompañan por el camino y no te abandonan. A todos vosotros, gracias.

La última mención es para las personas que me han guiado desde las estrellas todos los días de mi vida, que me dan su amor desde el cielo y que me han visto convertirme en quién soy. Gracias a mis abuelos que me cuidan a cada paso que doy y que, aunque no puedan decirme cómo se sienten al verme, me dan fuerzas para enfrentarme a cualquier dificultad que he encontrado y que me encontraré durante el camino que me queda por recorrer hasta verlos de nuevo. Y sobre todo a ti abuela que, aunque el Alzheimer haya apartado tu conciencia de mí y el azar se haya llevado tu vista, sé perfectamente que tu alma ya está con el abuelo y con ella puedes verme y darte cuenta lo mucho que os echo de menos.

Gracias a todos.

# Resumen

Debido al auge de la robótica en la actualidad, cada vez encontramos productos desarrollados mediante la robótica a nuestro alrededor. Por ello se acrecienta la necesidad de especialistas en este campo. Es por esto que surgen plataformas y entornos dedicados a llevar el campo de la robótica a estudiantes de distintas edades. En claro ejemplo es el JdeRobot-Academy, el cual pone a disposición de los alumnos universitarios y pre-universitarios un conjunto de ejercicios que representan un problema específico de la robótica, de una manera sencilla, completa y eficaz.

Este Trabajo de Fin de Grado se ha centrado en el desarrollo de una nueva práctica para el entorno de JdeRobot-Academy, así como una completa reestructuración y optimización de una de las prácticas ya presente en el entorno. Para cada práctica se ha desarrollado una solución de referencia.

Para la nueva práctica incluida, llamada *Chrono*, ha sido necesario el completo desarrollo del nodo académico, así como su interfaz gráfica, la conexión de sensores y actuadores del robot con el nodo académico y la sincronización del simulador con las grabaciones procedentes de *ROS-Kinetic* para la visualización del robot F1 a vencer. Esta práctica permite al alumno enfrentarse al problema de captación y procesado de imágenes, además de preparar un algoritmo de control de movimiento del robot.

Para la optimización y mejora de la práctica llamada *Follow Road*, fue necesario una reestructuración de su interfaz gráfica para introducir una visualización de la imagen procesada por el alumno y la integración de una pausa académica, así como el desarrollo de un algoritmo nuevo de conexión de sensores y actuadores para dar soporte a los *drivers* proporcionados por *ROS-Kinetic*. Para esta práctica se ponen a disposición del alumnos todos los materiales para que pueda centrarse exclusivamente en los problemas de captación y procesado de imágenes y de la realización de un movimiento controlado por parte del dron para que siag de una manera eficiente la carretera.

# Índice general

<b>Índice de figuras</b>	<b>X</b>
<b>1. Introducción</b>	<b>1</b>
1.1. Robótica . . . . .	1
1.2. Software Robótico . . . . .	4
1.2.1. Middlewares robóticos . . . . .	5
1.2.2. Simuladores robóticos . . . . .	6
1.2.3. Bibliotecas . . . . .	6
1.3. Docencia en robótica . . . . .	7
1.3.1. Robotics-Academy . . . . .	8
1.3.1.1. Nodos ROS . . . . .	10
1.3.1.2. Robotics-Academy-Web . . . . .	10
1.3.1.3. Ejercicios disponibles . . . . .	12
<b>2. Objetivos</b>	<b>17</b>
2.1. Objetivos . . . . .	17
2.2. Requisitos . . . . .	18
2.3. Metodología . . . . .	18
2.4. Plan de trabajo . . . . .	20
<b>3. Infraestructura</b>	<b>22</b>
3.1. Entorno ROS . . . . .	22
3.2. Simulador Gazebo . . . . .	23
3.3. Entorno JdeRobot . . . . .	25

3.4.	Editores de modelos 3D: SketchUP y Belnder	26
3.5.	Lenguaje Python	27
3.6.	Biblioteca OpenCV	27
3.7.	Biblioteca PyQt	28
3.8.	Entorno web Jupyter	29
<b>4.</b>	<b>Ejercicio de cronometraje de vueltas competitivas con Gazebo y ROS</b>	<b>32</b>
4.1.	Enunciado	32
4.2.	Infraestructura	33
4.2.1.	Modelo de robot F1	33
4.2.2.	Cámara	34
4.2.3.	Sensor Odométrico	35
4.2.4.	Circuito de Nürburgring	35
4.2.5.	Ficheros de configuración	36
4.3.	Nodo Académico	37
4.3.1.	Arquitectura software	38
4.3.2.	Interfaz de sensores y actuadores	39
4.3.3.	Interfaz gráfica	39
4.3.4.	Visor de imágenes	41
4.3.5.	Botones de control	42
4.3.6.	Mapa de odometría	42
4.3.7.	Lector de tiempos	43
4.4.	Soluciones de referencia	46
4.4.1.	Procesamiento de imagen	46
4.4.2.	Control de movimiento	49
4.4.2.1.	Filtrado de imagen y control de movimiento basado en píxeles	49

4.4.2.2. Filtrado de imagen y control de movimiento basado en el centro del contorno . . . . .	50
4.5. Experimentación . . . . .	51
4.5.1. Ejecución típica . . . . .	51
4.5.2. Ejecución estática . . . . .	53
4.5.3. Ejecución en movimiento . . . . .	56
4.6. Cuadernillo académico Jupyter . . . . .	58
<b>5. Ejercicio de seguimiento automático de carreteras con drones</b>	<b>63</b>
5.1. Enunciado . . . . .	63
5.2. Infraestructura . . . . .	64
5.2.1. Arquitectura . . . . .	64
5.2.1.1. Modelo dron “Iris” . . . . .	65
5.2.1.2. Px4 . . . . .	65
5.2.1.3. MAVLink . . . . .	67
5.2.1.4. MavROS . . . . .	67
5.2.1.5. Drivers . . . . .	69
5.2.1.6. Sensores y actuadores . . . . .	69
5.2.1.6.1. Cámara . . . . .	69
5.2.1.6.2. Odometría . . . . .	70
5.2.1.6.3. Rotores . . . . .	70
5.2.1.7. Escenario de Gazebo . . . . .	70
5.2.1.8. . . . .	71
5.3. Plantilla de nodo ROS . . . . .	74
5.3.1. Arquitectura software . . . . .	75
5.3.2. Interfaz de sensores y actuadores . . . . .	76
5.3.3. Interfaz gráfica . . . . .	76

5.4. Configuración de los ficheros . . . . .	78
5.5. Solución de referencia . . . . .	78
5.5.1. Procesamiento de imagen . . . . .	79
5.5.2. Control de movimiento . . . . .	81
5.6. Experimentación . . . . .	82
5.6.1. Ejecución de la práctica . . . . .	82
5.7. Plantilla del cuadernillo Jupyter . . . . .	86
<b>6. Conclusiones</b>	<b>92</b>
6.1. Conclusiones . . . . .	92
6.2. Trabajos futuros . . . . .	94
<b>Bibliografía</b>	<b>99</b>

# Índice de figuras

1.1.	Robots modernos . . . . .	2
1.2.	Robot DaVinci . . . . .	3
1.3.	Flota de robot de Amazon . . . . .	3
1.4.	Robot Atlas . . . . .	4
1.5.	Estructura de una práctica en Robotics-Academy . . . . .	9
1.6.	Visualización de las prácticas “Follow Line” y “Follow Road” en Academy-Web . . . . .	11
1.7.	Follow Line . . . . .	12
1.8.	Vacuum Cleaner . . . . .	13
1.9.	Drone-Cat-Mouse . . . . .	14
2.1.	Modelo de desarrollo en espiral . . . . .	19
3.1.	Ejemplo de mundo y modelo de Gazebo . . . . .	24
3.2.	Arquitectura de Jupyter . . . . .	30
4.1.	Infraestructura de Chrono . . . . .	33
4.2.	Modelos disponibles de coches . . . . .	34
4.3.	Modelo del circuito de Nürburging . . . . .	36
4.4.	Nodo Académico de Chrono . . . . .	38
4.5.	Interfaz Gráfica Chrono . . . . .	40
4.6.	Visor de imágenes de Chrono . . . . .	41
4.7.	Ilustración del botón de pausa académica . . . . .	42
4.8.	Ilustración del botón de comienzo académico . . . . .	42
4.9.	Mapa del GUI . . . . .	43

4.10. Lector de tiempos de Chrono . . . . .	46
4.11. Inicialización ROS y Gazebo . . . . .	52
4.12. Inicialización del nodo académico y el GUI . . . . .	53
4.13. Ejecución estática de la práctica . . . . .	54
4.14. Ejecución con velocidad fija . . . . .	55
4.15. Odometría en ejecución con teleoperador . . . . .	56
4.16. Ejemplo de procesamiento de la imagen en las curvas . . . . .	57
4.17. Ejecución con la solución . . . . .	57
4.18. Estructura de la práctica en Jupyter . . . . .	58
4.19. Fichero Chrono.ipynb . . . . .	59
4.20. Celda con la inicialización del mundo y los drivers . . . . .	60
4.21. Celda para importar el nodo académico . . . . .	61
4.22. Celda con el algoritmo del alumno . . . . .	61
4.23. Ejecución de la celda del algoritmo . . . . .	62
4.24. Pausa de la ejecución del algoritmo . . . . .	62
5.1. Infraestructura del ejercicio Follow Road . . . . .	64
5.2. Ilustraciones del dron “Iris” . . . . .	65
5.3. Relación entre MavROS, MAVLink y Px4 . . . . .	68
5.4. Escenario de Follow Road . . . . .	71
5.5. Plantilla de nodo ROS del ejercicio Follow_Road . . . . .	75
5.6. Interfaz Gráfica Folow_Road . . . . .	77
5.7. Inicialización ROS y Gazebo . . . . .	83
5.8. Inicialización del escenario y el GUI . . . . .	84
5.9. Ejecución de la práctica . . . . .	85
5.10. Ejecución con teleoperador . . . . .	85
5.11. Ejecución con la solución . . . . .	86

5.12. Estructura de la plantilla de Jupyter de la práctica Follow_Road . . . . .	87
5.13. Fichero follow_road.ipynb . . . . .	88
5.14. Celda con la inicialización del mundo y los drivers . . . . .	89
5.15. Importación de las clases “MyAlgorithm” y “FollowRoad” . . . . .	89
5.16. Despegue del dron . . . . .	89
5.17. Celda de importación del algoritmo . . . . .	90
5.18. Ejecución del código . . . . .	90
5.19. Ejecución del código . . . . .	90
5.20. Ejecución del código . . . . .	91
5.21. Ejecución del código . . . . .	91

# Capítulo 1

## Introducción

El Trabajo Fin de Grado (TFG) descrito a continuación se encuadra en el entorno educativo *Robotics-Academy*, para la enseñanza de la programación de robots. La intención principal de su desarrollo es extender este entorno docente con nuevos ejercicios que representen problemas atractivos en la robótica. En este capítulo se introducirá el contexto en el que se sitúa este proyecto y la motivación que he llevado a su desarrollo. Es preciso comenzar con una explicación a grandes rasgos sobre qué es la robótica y sus aplicaciones en la sociedad.

La parte más importante de la inteligencia de los robots viene suministrada por el software. Dentro del software destacaremos diferentes elementos como los simuladores, las bibliotecas de código y los middlewares de robótica. Dentro de la robótica, el contexto concreto relacionado con este TFG es la robótica educativa, y en particular, el propio entorno *Robotics-Academy*, desarrollado en la Universidad Rey Juan Carlos.

### 1.1. Robótica

A lo largo de la historia, la ciencia y la tecnología han sido utilizadas por el hombre para facilitarle la vida. Para ello, ha ideado, desarrollado y construido herramientas y máquinas empleándolas para reducir su carga de trabajo. La robótica es la rama de la tecnología basada en la utilización de la informática para el diseño y desarrollo de sistemas automáticos que faciliten la vida al ser humano e, incluso, llegan a sustituirle en algunas tareas determinadas. La robótica incluye conceptos de disciplinas diversas, como la física,

## CAPÍTULO 1. INTRODUCCIÓN

---

las matemáticas, la electrónica, la mecánica, la inteligencia artificial, la ingeniería de control, etc. Gracias a todas estas disciplinas involucradas unidas convenientemente se pueden diseñar máquinas que ejecuten comportamientos autónomos según el propósito para el que han sido desarrolladas. Estas máquinas autónomas se denominan Robots".

Desde 1950, estos sistemas autónomos han experimentado un crecimiento exponencial en cuanto a complejidad, versatilidad, autonomía y, sobre todo, en su incorporación a una gran diversidad de ámbitos. Los sistemas operados por el ser humano comienzan a incorporar un sistema de control específico programable que permiten el desarrollo de tareas repetitivas o con un gran riesgo para las personas, englobando tareas básicas y de difícil realización, hasta la actualidad, en la cual existe un gran marco de ejemplos en los que se integran la robótica y multitud de campos y tareas. Los robots comerciales e industriales realizan las tareas de una manera más exacta o más barata que las personas. También son utilizados en trabajos peligrosos, sucios tediosos para el ser humano. Gracias a esto, se trata de un campo en crecimiento constante.

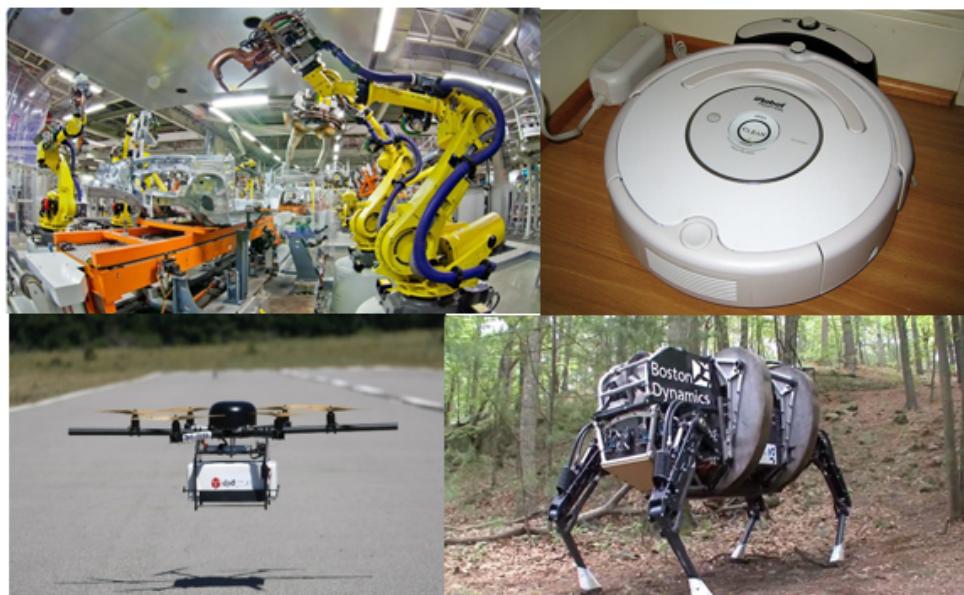


Figura 1.1: Robots modernos

Ya se ha comentado la importancia de los robots en la actualidad en el aspecto industrial, pero tal el crecimiento que está experimentando la robótica que está comenzando a cobrar una gran importancia en aspectos menos especializados como el entorno doméstico. Cabe destacar el desarrollo de robots para facilitar la vida al ser humano, un ejemplo está

ilustrado en la Figura 1.1. Las aspiradoras robóticas (Roomba, Dyson, Xiami, ...) han tenido un éxito rotundo en la realización de una actividad doméstica necesaria para la vida del ser humano.

Otro éxito de la robótica en la actualidad es el desarrollo de coches autónomos. Este hecho se ha conseguido paulatinamente mediante la incorporación de tecnología cada vez más sofisticada a los automóviles. En este aspecto cabe destacar los módulos de aparcamiento automático, el park assist o los asistentes de conducción autónoma (autopiloto Tesla), o los prototipos de coches autónomos (Apple o Google). Otro ámbito en el que la robótica ha sido introducida es el militar, donde se han incorporado robots de rescate o para la desactivación de bombas.

En la medicina se ha desarrollado el robot DaVinci (Figura ??), que permite operar desde cualquier parte del mundo con una precisión mayor a la humana. En el ámbito de la logística Amazon ha desarrollado una flota de robots de almacén que consiguen trasladar los pedidos a lo largo de sus almacenes (Figura ??). Tal es el punto de crecimiento de la robótica que se están desarrollando robots con “comportamientos inteligentes” como el robot Atlas de *Boston Dynamics* (Figura 1.4) que pueden interactuar con humanos, ya sea como asistente para el hombre o con fines experimentales como la locomoción bípeda.

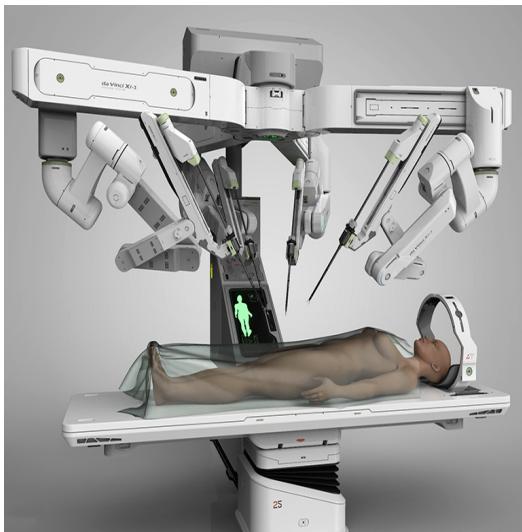


Figura 1.2: Robot DaVinci

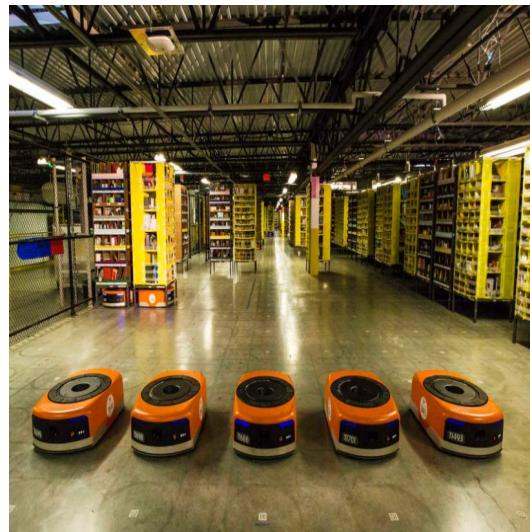


Figura 1.3: Flota de robot de Amazon

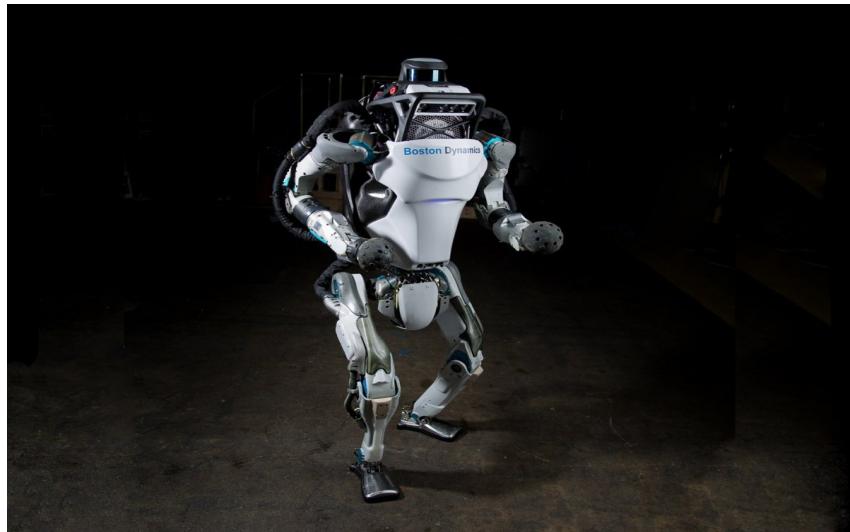


Figura 1.4: Robot Atlas

Los ámbitos de aplicación de la robótica son cada vez más extensos gracias al auge de esta ciencia. Algunos ejemplos son la agricultura de precisión mediante drones con análisis de imágenes térmicas y multiespectral para aumentar el rendimiento de las explotaciones agrícolas o el control de los productos industriales mediante el procesado de imágenes de la producción o la automatización de aplicaciones anestésicas de bajo nivel e, incluso, competiciones deportivas de robots.

## 1.2. Software Robótico

Para que los robots puedan ser controlados de una manera eficaz el comportamiento del software que los controla debe ser robusto. Para ello, el *software* de la robótica, se divide en distintas capas (*drivers*, *middleware* y aplicaciones), cuya arquitectura será distinta, típicamente, según su aplicación final.

Debido al gran desarrollo de la robótica, los robots actuales ya no precisan del control del ser humano para su funcionamiento, en la actualidad tienen comportamientos autónomos que les permiten realizar las tareas sin la mediación de terceros. Esto es posible gracias al minucioso desarrollo del software que compone los sistemas complejos del robot, algo parecido a una inteligencia autónoma. El desarrollo del software robótico parte de ciertas tareas o requisitos como son los circuitos de retroalimentación, control, búsqueda

de caminos, localización o filtrado de datos entre otras muchas.

En los últimos años se han creado una gran cantidad de plataformas de desarrollo de software para las aplicaciones robóticas, también llamados *middleware* robóticos. Los simuladores son otra herramienta importante para el desarrollo de software robótico ya que permiten realizar pruebas y depurar los fallos para programar una versión funcional del robot antes de ser fabricado.

### 1.2.1. Middlewares robóticos

Los *middleware* robóticos pueden definirse como entornos o *frameworks* para el desarrollo de software para robots. Se trata de software que conecta aplicaciones o componentes software para soportar aplicaciones complejas y distribuidas. Para controlar los sensores y actuadores de los robots estos entornos incluyen *drivers*, arquitectura software para las aplicaciones que se van a crear, bloques de funcionalidad robótica ya resuelta, además de simuladores, visualizadores... Por ello al *middleware* se le suele conocer como "pegamento para software". Una de las tareas del *middleware* es conectar el hardware, ya sea real o simulado, con la aplicación desarrollada. El *middleware* más extendido en el mundo es ROS.

**Robot Operating System (ROS)**<sup>1</sup> es una plataforma de software libre para el desarrollo software de robots que proporciona la funcionalidad de un sistema operativo en un clúster heterogéneo como el control de dispositivos de bajo nivel, mecanismos de intercambio de mensajes entre procesos y la abstracción del hardware, necesarios para el desarrollo de la robótica. Aunque el *framework* ROS se desarrolló para los sistemas UNIX, se ha adaptado para ser soportado en otros sistemas operativos como Fedora, Debian, Windows, Mac OS X, Arch, Slackware, Gento u OpenSUSE, llegando a permitir las aplicaciones multiplataforma. Gracias a esto el *framework* ROS se ha convertido en el más utilizado.

Existen otros *framework* interesantes como **Orocos**<sup>2</sup> que permite el control avanzado de máquinas y robots en C++, **Orca**<sup>3</sup> que está orientado a componentes por lo que

---

<sup>1</sup><http://www.ros.org/>

<sup>2</sup><http://www.orocos.org/>

<sup>3</sup>[http://orca-robotics.sourceforge.net//](http://orca-robotics.sourceforge.net/)

permite el desarrollo de aplicaciones más complejas y de *software* libre y **Urbi**<sup>4</sup> que es un *middleware* multiplataforma de código abierto en C++ que permite desarrollar aplicaciones en sistemas completos y complejos y trabaja de forma conjunta con ROS.

### 1.2.2. Simuladores robóticos

Debido al gran coste que supone la fabricación del hardware del robot es preciso depurar los posibles errores que contenga el código, así como el funcionamiento del hardware antes de su fabricación. Por ello debe probarse el código en un simulador orientado al tipo de aplicación que estemos desarrollando. Gracias a los simuladores es posible probar este código sin tener que fabricar previamente el hardware. De esta manera cualquier mal funcionamiento del código del robot puede ser solventado evitando la rotura del hardware. Algunos de los simuladores más utilizados son:

**Gazebo**<sup>5</sup>. Se trata del simulador 3D de código abierto más extendido. Funciona bajo la licencia Apache 2.0 y tienen gran importancia su motor de renderizado avanzado, sus motores de física y su soporte para *plugins* de robot y sensores, además de su amplio catálogo de robots con sus sensores y actuadores. Otro hecho importante es su soporte para ROS lo que permite probar el código real del robot en el simulador.

**Stage**<sup>6</sup>. Es un simulador en dos dimensiones, integrable con ROS, que permite simular numerosos robots simultáneamente.

**Webots**<sup>7</sup>. Simulador de robótica avanzada en el que se pueden desarrollar modelos propios y su física, escribir sus controladores y hacer simulaciones a gran velocidad. Un ejemplo es su soporte para el humanoide Nao. Actualmente, se ha convertido en *software* libre.

### 1.2.3. Bibliotecas

En el desarrollo de software deben abordarse un gran abanico de problemas clásicas, básicas y específicas para cada aplicación. Esta tarea es muy tediosa si hay que abordarla

---

<sup>4</sup><https://github.com/urbiforge/urbi>

<sup>5</sup><http://gazebosim.org/>

<sup>6</sup><http://wiki.ros.org/stage>

<sup>7</sup><https://www.cyberbotics.com/>

desde cero y requeriría un gran tiempo enfrentarse a ellas. Para evitar la repetición de problemas a la hora de desarrollar una aplicación, existen bibliotecas de código. Estas bibliotecas ofrecen conjuntos de implementaciones de código que permiten solucionar de manera directa ciertos problemas contenidos en su código. Esto permite al programador ahorrar una gran cantidad de tiempo para solucionar problemas que ya han sido solucionados anteriormente. Las bibliotecas pueden vincularse a un programa o a otra biblioteca en distintos puntos del desarrollo (bibliotecas estáticas) o durante la ejecución del programa (bibliotecas dinámicas). Una de las bibliotecas más utilizadas es **OpenCV**<sup>8</sup>. Esta librería aborda, principalmente, problemas de visión y procesamiento de imágenes.

### 1.3. Docencia en robótica

La robótica con fines educativos está adquiriendo gran importancia en la actualidad en la enseñanza preuniversitaria. Esto es debido a que su aprendizaje está disponible para estudiantes de cualquier nivel, el único requisito para estudiar robótico es la motivación por el desarrollo de aplicaciones. La robótica en el campo de la docencia resulta especialmente interesante al ser una ciencia multidisciplinar, ya que incluye campos multidisciplinares como electrónica, informática, mecánica, física, ... Gracias a ello el estudiante adquiere una gran variedad de conocimiento de todas estas áreas. Además, proporciona una nueva visión del universo que le rodea, aprendiendo a distinguir los problemas y tomar una decisión al respecto.

En docencia primaria y secundaria se intenta despertar el interés del estudiante por la robótica, con la transformación de asignaturas teóricas tradicionales en asignaturas más prácticas e interactivas, ya que la robótica permite la recreación de problemas que les rodean y a través de los cuales pueden utilizar su creatividad y plasmar los conceptos teóricos que han adquirido. En los centros de enseñanza primaria y secundaria se imparte robótica mediante plataformas físicas como los robots LEGO (Mindstorms, RCX, NXT, Evo, WeDo), placas Arduino, los kits de SolidWorks, etc.

En la docencia universitaria se imparte la robótica en distinto Grados y Postgrados en las escuelas de ingeniería. En España, se puede cursar la docencia robótica en el “Grado en Ingeniería Robótica” de la Universidad de Alicante, en los Grados de

---

<sup>8</sup><http://opencv.org/>

## CAPÍTULO 1. INTRODUCCIÓN

---

“Electrónica industrial y automática” o “Ingeniería Electrónica, Robótica y Mecatrónica” en distintas universidades, además de grados que están en desarrollo como el “Grado en Ingeniería Robótica Software” que imparte la Universidad Rey Juan Carlos desde el mes de Septiembre. Sin embargo, la docencia en robótica se reserva, mayoritariamente, para los Postgrados, dado que se trata una ciencia muy especializada. Existen varios Másteres destacados en cuanto a la docencia de robótica como el “Máster de Visión Artificial”, el “Máster Universitario en Ingeniería Mecatrónica”, o el “Máster Universitario en Automática y Robótica”. Dentro del ámbito internacional pueden encontrarse distintas universidades orientadas a robótica como el MIT, Carnegie Mellon University, Standford o Geordia Institute of Technology. También existen asociaciones prestigiosas como ACM (Association for Computing and Machinery) y la IEEE-CS (IEEE Computer Society) que ven la robótica como una ciencia imprescindible en estudios de ingeniería, informática y sistemas inteligentes. En cuanto a la propia Universidad Rey Juan Carlos, cuenta con la plataforma docente JdeRobot, que consta de un entorno académico para la docencia de robótica llamado Robotics-Academy. Este entorno educativo se ha utilizado con éxito en distintas asignaturas como “Visión en Robótica” del Máster de Visión Artificial o en la asignatura Robótica del Grado de Ingeniería Telemática. Del mismo modo se han impartido talleres de aprendizaje para todos los públicos docentes, desde profesores de secundaria y primaria hasta estudiantes de secundaria pasando por trabajadores de distintas empresas. También se han impartido cursos de programación de drones para estudiantes universitarios.

### 1.3.1. Robotics-Academy

Forma un *toolkit* o conjunto de herramientas para la robótica robótica. Se trata de *middleware* robótico empleado para el desarrollo de aplicaciones robóticas y, sobre el que se respaldado este TFG para su realización.. Con este trabajo se ha pretendido extender las posibilidades de aprendizaje en este entorno, ampliándolo con dos nuevos ejercicios. Los ejes en los que se apoya Robotics-Academy son:

- a) Lenguaje Python (por su sencillez y potencia).
- b) Simulador Gazebo (con distintos modelos de robot, tales como drones, formula1, brazos, aspiradoras, etc.).

- c) Foco en el algoritmo en vez de en el middleware, ocultando al estudiante los detalles de la infraestructura.

Debido a la compatibilidad de ROS con Gazebo y con JdeRobot, se pueden utilizar los *plugins* de ROS para las simulaciones en Gazebo de las prácticas presentes en el entorno JdeRobot mediante el establecimiento de las conexiones de los sensores y actuadores de la práctica con los *plugins* de ROS en el nodo académico de la misma.

El entorno Robotics-Academy cuenta con elenco de prácticas que abordan distintos problemas clásicos de la robótica. Para cada práctica se dispone de un componente académico que resuelve tareas auxiliares como la conexión con sensores y actuadores necesarios, la temporización o la interfaz gráfica y aloja el código del algoritmo del estudiante. De esta manera el estudiante se puede centrar en la solución del ejercicio exclusivamente. Cada nodo académico está formado por una parte específica oculta y el algoritmo con la lógica del robot del estudiante que se rellena en un fichero plantilla.

Debido a esta estructura, pueden distinguirse distintas capas en la composición de la práctica. La capa de nivel más bajo se le facilita al estudiante que sólo se centra en la capa superior donde se aloja la lógica del robot. Aunque esta capa más baja le viene dada al estudiante, es necesaria su implementación para poder dar solución a la práctica. En este aspecto se incluyen las conexiones de los sensores actuadores del robot, la interfaz gráfica, la temporización, el desarrollo del modelo del robot y un escenario que lo contenga, los plugins del modelo, los *drivers* del mismo y la comunicación entre el simulador y el componente académico de alto nivel. Todo esto puede apreciarse en la Figura 1.4:

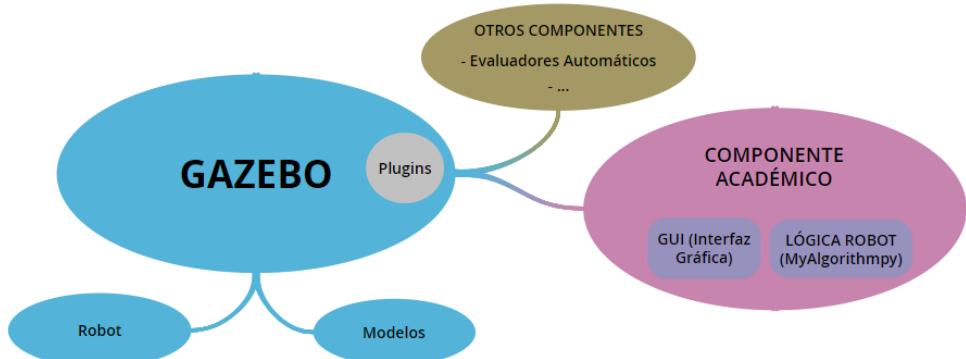


Figura 1.5: Estructura de una práctica en Robotics-Academy

Como se ha descrito anteriormente, en la Figura 1.4 se puede apreciar que la arquitectura software de las prácticas académicas facilita el desarrollo de las mismas por parte de los alumnos de manera que sólo se concentran en el desarrollo del algoritmo con la lógica del robot.

El entorno usual para la realización de las prácticas es el simulador Gazebo, aunque las prácticas se han desarrollado de manera que puedan ser soportadas por robots reales sin realizar ninguna modificación, con los correspondientes drivers del robot. Gracias a esto, el código puede ser probado en robots reales. El sistema operativo base sobre el que se han desarrollado las prácticas es Linux dado que presenta una interfaz más sencilla a la hora de programar, por ello Linux cuenta con toda la infraestructura de las prácticas.

### 1.3.1.1. Nodos ROS

Una de las posibilidades que ofrece este *toolkit* para realizar los problemas que ofrece es el acceso a su plataforma mediante los nodos ROS. De esta manera, basta con instalar el paquete de la plataforma y se puede acceder a cualquiera de las prácticas que lo componen, además de tener todas las herramientas necesarias para su desarrollo.

El componente académico es el encargado de cargar en el simulador el código desarrollado por el alumno desde el fichero *MyAlgorithm.py* y visualizar trazas que ayuden a la depuración del código como las imágenes procesadas, datos del láser o imágenes de la cámara integrada, dependiendo de la práctica.

También se puede acceder a las prácticas docentes con el navegador. De esta manera se convierte al entorno JdeRobot en multiplataforma dotándolo de mayor accesibilidad. Esto es gracias al desarrollo de la interfaz web de Gazebo para la simulación, a la plataforma *Jupyter* (ver 3.8) que, mediante sus cuadernillos, han permitido trasladar las prácticas de Robotics-Academy a sus cuadernillos y, de esta manera, dotar el entorno de un soporte multiplataforma.

### 1.3.1.2. Robotics-Academy-Web

Esta plataforma web proporciona un servidor remoto para el desarrollo y ejecución de las prácticas contenidas en el entorno *Robotics-Academy*. Esto supone una gran innovación y el paso final al soporte multiplataforma del entorno docente *JdeRobot*. Academy-Web

presenta dos formatos para el alumno, mediante nodos ROS y la versión web.

Para ello, se basa en el uso de Jupyter para cargar el nodo académico de las prácticas, así como de un script (incluido en el *Notebook* de Jupyter) para realizar las conexiones de los sensores y actuadores del robot con el nodo académico. También utiliza el soporte del simulador Gazebo en navegadores web para ofrecer una visualización del escenario de la práctica cargando un fichero *.world* donde se incluye la descripción del robot y el mundo. Para soportar esta carga computacional, el servidor de la plataforma está basado en el servidor *Apache*<sup>9</sup>, sobre el que se ha desarrollado un servidor en *Django*<sup>10</sup>. Ambas plataformas son de código libre y proporcionan el código necesario para el desarrollo de servidores web. Por último, utiliza *Dockers*<sup>11</sup> para ofrecer al estudiante todos los componentes necesarios para dar soporte a toda la infraestructura anterior. De esta manera, tanto JdeRobot, ROS-Kinetic, modelos, escenarios, *plugins*, *drivers*, etc. Están disponibles y son totalmente transparentes al alumno.

Mediante la integración de todas las plataformas anteriores se ha conseguido desarrollar el servidor web *Academy-Web* que proporciona al estudiante todas las herramientas necesarias para realizar las prácticas que contiene en cualquier plataforma de una manera muy sencilla. El servidor ofrece la visualización del escenario en Gazebo junto con el cuadernillo de *Jupyter* en el que está presente la celda en la que el estudiante desarrollará su código.

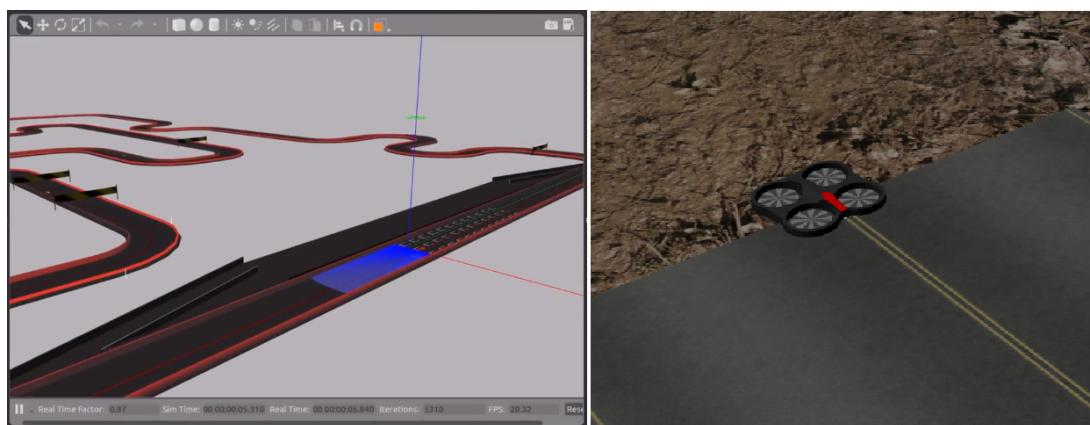


Figura 1.6: Visualización de las prácticas “Follow Line” y “Follow Road” en Academy-Web

---

<sup>9</sup><https://www.apache.org/>

<sup>10</sup><https://www.djangoproject.com/>

<sup>11</sup><https://www.docker.com/>

### 1.3.1.3. Ejercicios disponibles

Algunas de las prácticas que componen la plataforma son los siguientes:

*Follow Line*

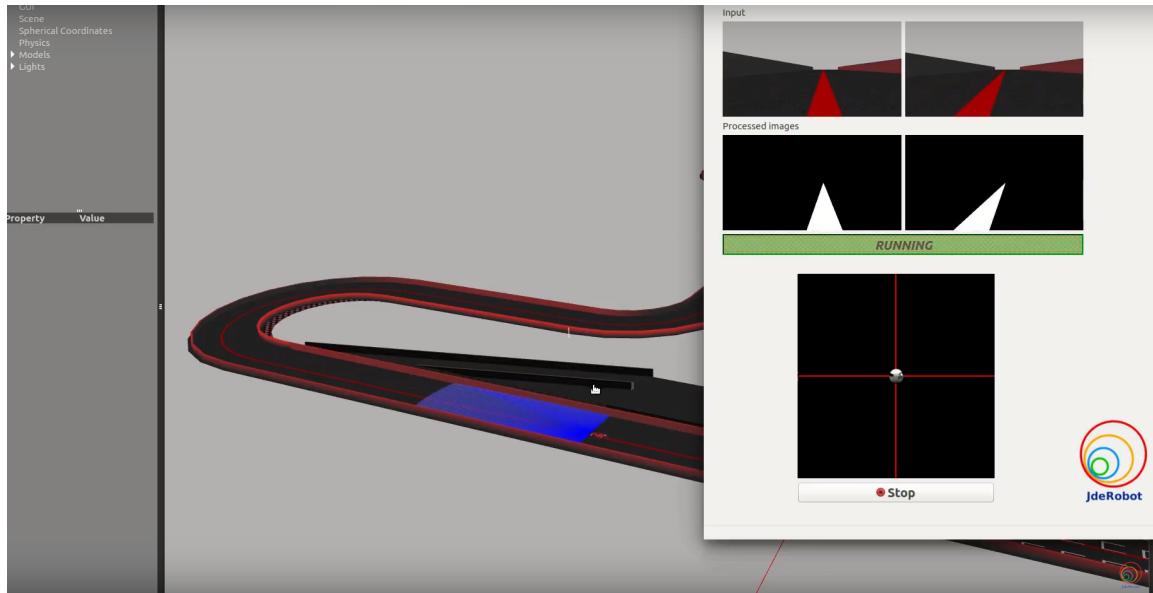


Figura 1.7: Follow Line

Este ejercicio *Follow Line* (Figura 1.7) trata de un robot coche Fórmula1 que consta de una cámara en su parte frontal por la que recoge imágenes. En su código se deben recoger las imágenes y procesarlas de manera que filtre la línea roja del circuito y la siga hasta que complete el circuito por completo<sup>12</sup>.

---

<sup>12</sup><https://youtu.be/QG09aoBVoA>

*Vacuum Cleaner*

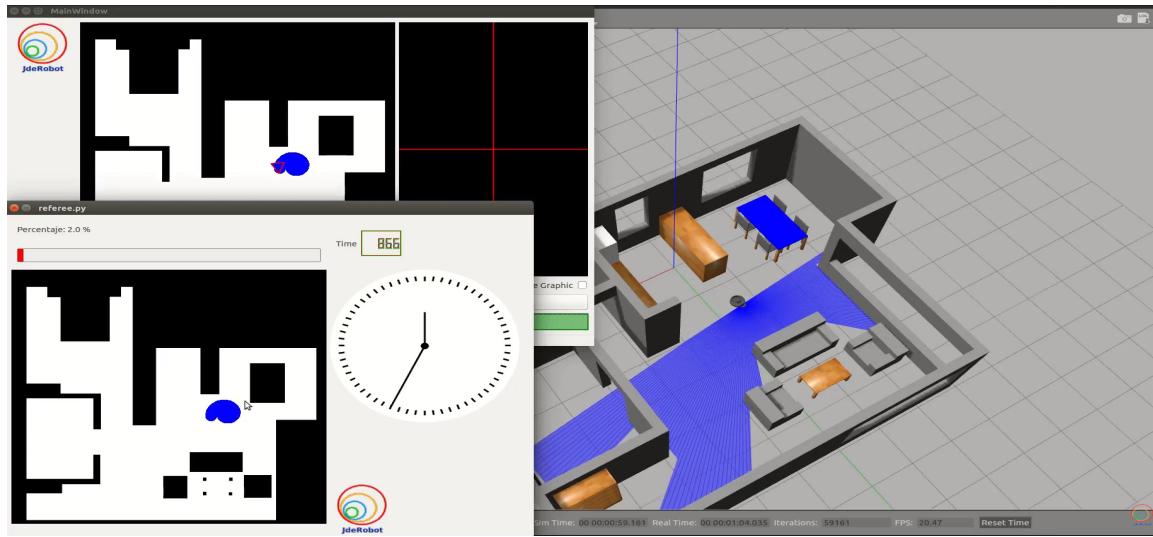


Figura 1.8: Vacuum Cleaner

En este ejercicio *Vacuum Cleaner* (Figura 1.8), el alumno debe recoger los datos del láser incluido en el modelo del robot aspiradora Roomba para que pase por el mayor área posible del escenario evitando la colisión con los obstáculos contenidos <sup>13</sup>.

---

<sup>13</sup><https://youtu.be/12muuY9JXLk>

*Drone-Cat-Mouse*

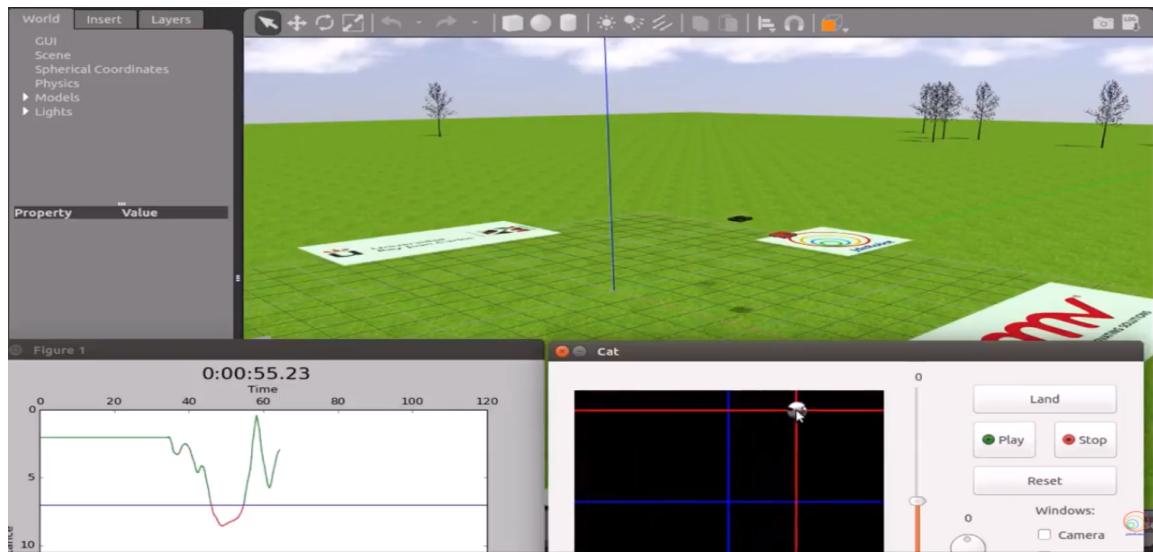


Figura 1.9: Drone-Cat-Mouse

*Drone-Cat-Mouse* es una de las prácticas más complejas del entorno Robotics-Academy. En ella el alumno debe dotar de la lógica necesaria a un dron (dron negro) para recoger las imágenes captadas por su cámara y filtrarlas para encontrar a un dron rojo. Una vez reconocido el dron rojo debe dotar de movimiento al dron para perseguirlo dado que el dron rojo está en movimiento. El objetivo es que el dron negro se acerque los más posible al dron rojo pero sin colisionar con él emulando el juego Gato-Ratón<sup>14</sup>. Esta práctica se ha utilizado en las dos ediciones del campeonato *Program-A-Robot*, la primera en la URJC y la segunda en las Jornadas Nacionales de Robótica. Se va a utilizar en la tercera edición que tendrá lugar en la conferencia internacional IROS<sup>15</sup>.

El contexto inmediato de este Trabajo de Fin de Grado consiste en una serie de ejercicios elaborados recientemente para enriquecer el contenido del entorno Robotics-Academy.

Entre los ejercicios elaborados más actuales caben destacar el TFG de Irene López Rodríguez "Nuevas Prácticas en el Entorno Docente de Robótica Robotics-Academy"<sup>[54]</sup> en el que se introdujeron dos prácticas nuevas llamadas Coche autónomo negociando un cruce y Aspiradora autónoma con autolocalización. El primer ejercicio, después llamado *Car Joint* trata sobre un coche autónomo que realiza un cruce por el que circulan coches.

<sup>14</sup><https://www.youtube.com/watch?v=DYD9oPawhWg>

<sup>15</sup><https://www.iros2018.org/competitions>

## CAPÍTULO 1. INTRODUCCIÓN

---

Para ello el coche debe filtrar las imágenes para reconocer una señal de Stop, así como los coches que circulan y las líneas de los carriles. Cuando no detecte ningún coche circulando debe tomar la intersección y escoger el carril correcto. La segunda práctica es similar a la práctica *Vacuum Cleaner* pero consta del mapa con el escenario y sensor de posición, de manera que la aspiradora debe saber en qué lugar del escenario se encuentra y no repetir áreas ya limpiadas.

Otro TFG destacado en este aspecto es el de Vanessa Fernández Martínez “*Nuevas Prácticas en el Entorno Docente de Robótica Robotics-Academy*”[52], en el cual se añadieron dos prácticas nuevas llamadas Aspiradora Autónoma o *Vacuum Cleaner* y Aparcamiento Automático, además de mejorar la práctica Tele Taxi con nuevos modelos, un mejor rendimiento del algoritmo GPP de navegación global y la inclusión de un evaluador automático capaz de medir el desempeño del algoritmo y proporcionar una nota. En cuanto al ejercicio desarrollado *Vacuum Cleaner* ya ha sido comentado en el apartado anterior. La segunda práctica llamada *Autopark* tiene como objetivo el aparcamiento de un coche autónomo mediante mediciones láser de los sensores frontales, laterales y posteriores.

También hay que mencionar el Trabajo de Fin de Grado desarrollado por Carlos Awadallah Estévez “*Nuevas Prácticas Docentes de Robótica en el Entorno JdeRobot-Academy*”[51], en el cual se incorporan dos nuevas prácticas al entorno JdeRobot-Academy. La primera de ellas llamada *Follow Face* trata de dar la lógica necesaria a una cámara pantilt de manera que procese las imágenes captadas por la cámara y reconozca la cara. Una vez hecho esto debe seguir el movimiento de la cara. La segunda práctica que se desarrolla en este TFG se llama *Laser Loc* en la cual mediante mediciones de los sensores laser y un mapa con el escenario es capaz de realizar estimaciones de posición mediante movimiento y odometría.

Siguiendo con esta filosofía, el presente Trabajo de Fin de Grado aporta una práctica totalmente nueva al entorno Robotics-Academy y una optimización y mejora global de una práctica obsoleta incluyéndola en el entorno Robotics-Academy-Web.

## CAPÍTULO 1. INTRODUCCIÓN

---

El objetivo de este TFG es ampliar el número de prácticas que forman el entorno Robotics-Academy desarrollando dos nuevas prácticas, una completamente nueva y otra actualizada y adaptada a *ROS*, además de aportar en el elenco de prácticas de Robotics-Academy-Web para acercar al entorno a dar soporte multiplataforma.

En los próximos capítulos serán abordados los elementos necesarios para conseguir este objetivo. Comenzaremos con el Capítulo 2, en el que se concretarán los objetivos marcados, así como el punto de partida de este TFG y la metodología que ha sido empleada. En la Capítulo 3 se abordará la infraestructura utilizada para realizar el proyecto. En los Capítulos 4 y 5 explicaremos las dos prácticas que se han abordado en este TFG. Y, por último, en el Capítulo 6, se expondrán las conclusiones obtenidas, además de las posibles líneas de mejora futuras.

# Capítulo 2

## Objetivos

Una vez introducido el contexto en que se ha desarrollado este trabajo, es hora de profundizar en los objetivos concretos que se han tratado de alcanzar, los requisitos para las soluciones desarrolladas y la metodología que se ha seguido para conseguirlos.

### 2.1. Objetivos

La meta planteada en este proyecto es la mejora de la plataforma educativa *Robotics-Academy* utilizando el *software* robótico ROS y el entorno web *Jupyter*, además de la creación de una práctica completamente nueva, llamada *Chrono*, y la actualización de una de las prácticas existentes en esta plataforma, llamada *Follow Road*, así como una actualización de sus *drivers* para que soporte *ROS* y su inclusión en la infraestructura web de *Robotics-Academy-Web*.

La primera práctica consiste en la competición de dos coches de F1 por un circuito que dispone de una línea roja que se debe seguir. El código del alumno competirá con el F1 del mejor tiempo registrado, de menor tiempo posible, para el circuito en el que esté compitiendo. De esta manera conseguiremos que el alumno pueda depurar su código de solución y tenga un estímulo para alcanzar la perfección en el desarrollo de su algoritmo.

En cuanto a la segunda práctica, trata de un dron con una cámara que debe seguir una carretera. El código del alumno deberá filtrar las imágenes para segmentar la carretera y dotar de un movimiento controlado al dron que le permita seguir la carretera. Esta práctica ha sido actualizada para que funcione con la nueva infraestructura *software* de

los drones (*ROS* y *MavROS*, además de realizar una versión del ejercicio con cuadernillo de *Jupyter*.

## 2.2. Requisitos

Además de lograr los objetivos, los requisitos necesarios que se van a exigir al software desarrollado son:

1. El Sistema Operativo empleado será Ubuntu 16.04 LTS.
2. Se utilizará el *middleware* robótico JdeRobot en su versión 5.6.2. El uso de este *middleware* robótico simplifica la programación de comportamientos en los robots.
3. Se usará *OpenCV3* para el procesamiento de las imágenes captadas por las cámaras en ambas prácticas.
4. Para dar soporte a los sensores y actuadores se utilizará *ROS-Kinetic*, que es el estándar de facto en la comunidad robótica.
5. Para mostrar el comportamiento de los robots se utilizará el simulador *Gazebo*, uno de los más completos y utilizados en la actualidad.
6. El lenguaje de programación utilizado para el desarrollo de ambas prácticas será *Python* en su versión 2.7.12, por compatibilidad con el *middleware* robótico *ROS-Kinetic*.
7. Las soluciones desarrolladas deben ejecutar algoritmos en tiempo real, por lo que deben ser eficientes y realizar movimientos suaves.

## 2.3. Metodología

El desarrollo de este Trabajo de Fin de Grado puede descomponerse en un conjunto de iteraciones con distintas fases. Cada fase está formada por una reunión semanal con el tutor para determinar los objetivos a abordar, la planificación de cómo abordarlos, intentar solucionar los problemas que vayan a surgir anticipadamente y la consecución de los objetivos durante la semana. De esta manera se ha conseguido un desarrollo fluido y

## CAPÍTULO 2. OBJETIVOS

---

completo, asentando los conocimientos y despejando las dudas que surgían durante los meses dedicados a este desarrollo.

Además de las reuniones semanales, se han utilizado herramientas de apoyo como la bitácora semanal en la Wiki de JdeRobot<sup>1</sup>, donde se redactaban los avances obtenidos acompañados de vídeos demostrativos e imágenes. Además, el código creado se desarrollaba, progresivamente, en la plataforma Github, en un repositorio personal<sup>2 3</sup>, a los cuales el tutor tiene acceso para dar realimentación y orientar el proceso.

El modelo de desarrollo escogido ha sido el modelo creado por Barry Boehm. Al tratarse de un modelo en espiral, se adaptada a la perfección a nuestras necesidades, permitiendo disponer de flexibilidad ante cambios en los requisitos semanales, algo común mientras avanzaba el desarrollo. En paralelo, nos permitía separar el objetivo final en varias subtareas más sencillas. Con esto se ha conseguido una subsanación de los riesgos temprana y la definición de una arquitectura en las fases iniciales del desarrollo, todo ello dotado con un control de calidad continuo.

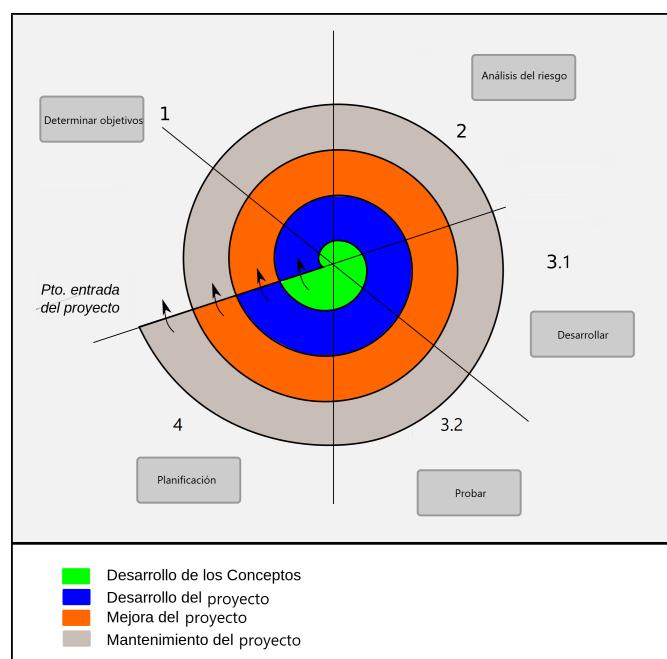


Figura 2.1: Modelo de desarrollo en espiral

La ventaja de este ciclo de vida es que permite la obtención de prototipos funcionales

<sup>1</sup><https://jderobot.org/Pablomoreno-tfg>

<sup>2</sup><https://github.com/RoboticsURJC-students/2017-tfg-pablo-moreno>

<sup>3</sup><https://github.com/PabloMorenoVera/Academy>

en una etapa temprana, la optimización progresiva del prototipo desarrollado y, en última instancia, pulir los detalles para abarcar la totalidad de los requisitos especificados (Figura 2.1). De esta manera el trabajo se desarrolla de manera incremental con cuatro fases bien definidas:

- **Determinar objetivos:** Esta primera fase del ciclo está formada por la definición de las metas.
- **Análisis del riesgo:** Se evalúan los posibles problemas iniciales al desarrollo y las soluciones a los mismos.
- **Desarrollar y probar:** En esta tercera fase se procede al desarrollo del trabajo propiamente dicho, junto con una serie de pruebas para verificar su funcionamiento.
- **Planificación:** En esta última fase del ciclo se valoran los resultados obtenidos y se planifican las siguientes etapas del proyecto.

## 2.4. Plan de trabajo

Para la consecución de los objetivos descritos, se han seguido seis etapas de trabajo:

- **Estudio de JdeRobot y el entorno educativo *Robotics-Academy*:** una vez descargado e instalado tanto el software, dependencias y bibliotecas como el simulador, se tomará un primer contacto con el entorno JdeRobot mediante la modificación y readaptación de algunas prácticas existentes, como sus interfaces gráficas.
- **Estudio de *ROS-Kinetic* y del simulador Gazebo:** esa etapa se ha dedicado al desarrollo de algunos modelos en el simulador, estudiando ejemplos disponibles en la web<sup>4</sup> y en JdeRobot, así como modificándolos y desarrollando algunos modelos nuevos. En esta etapa también se han estudiado el funcionamiento básico de los *plugins* que dispone Gazebo para el control de sus robots, sensores y actuadores. Esto ha supuesto una toma de contacto con el lenguaje de programación C++ utilizado, también, para comprender los *plugins* de ROS-Kinetic.

---

<sup>4</sup><http://gazebosim.org/tutorials>

- **Actualización de la infraestructura del drone sigue-carretera:** Esta práctica estaba bastante obsoleta y se procedió a renovar por completo su interfaz gráfica, el escenario utilizado incluyendo un nuevo dron que soportaba ROS y una nueva conexión de los sensores y actuadores del propio dron. Además se desarrolló una optimización global del nodo académico como la inclusión de una pausa académica. Además se creó una versión de la práctica para la plataforma Jupyter y se incluyó en el elenco de prácticas soportadas en *Robotics-Academy*.
- **Desarrollo de una solución de referencia para la práctica drone sigue-carretera:** Mediante la utilización de filtros de color y control PID.
- **Desarrollo de la infraestructura del ejercicio F1-chrono:** Se desarrolló el modelo del F1 y el circuito para competir en *Blender* y *SketchUp* para conformar el escenario de Gazebo. Se utilizaron los drivers del robot F1 ya existentes para dar soporte en ROS-Kinetic de los motores, la cámara y el láser. Se creó el nodo académico de la práctica para alojar el código del estudiante y la versión de la práctica para Jupyter.
- **Desarrollo de una solución de referencia para F1-chrono:** Con algoritmos de filtros de color y control PID.

# Capítulo 3

## Infraestructura

En este capítulo se presentan todos los componentes y software que han servido de apoyo en el desarrollo del TFG. En este punto se dará una explicación introductoria de las plataformas en las que se cimienta el trabajo (ROS y JdeRobot), en el simulador Gazebo, en los editores de modelos, en las librerías usadas más importantes (OpenCV y PyQt) y en el proyecto Jupyter.

### 3.1. Entorno ROS

ROS (Robot Operating System)<sup>1</sup> proporciona a los desarrolladores de software robótico los componentes necesarios para el desarrollo de aplicaciones robóticas. Entre ellos destacan la abstracción hardware, bibliotecas, intercambios de mensajes, administración de paquetes, controladores de dispositivo y visualizadores. Esta plataforma se distribuye en código abierto bajo una licencia BSD.

Una de las características más importantes de ROS es su integración con el simulador Gazebo, con el que se comunica a través de paquetes llamados *gazebo\_ros\_pkgs*<sup>2</sup>. Mediante estos paquetes, ROS es capaz de proporcionar las interfaces necesarias para simular un robot en Gazebo usando *ROS Messages*, servicios y reconfiguración mecánica.

Esta plataforma se conforma como una colección de nodos o procesos que suponen una computación. Los nodos se combinan en un gráfico y se comunican entre sí mediante *topics*

---

<sup>1</sup><http://www.ros.org/>

<sup>2</sup>[http://ros.org/wiki/gazebo\\_ros\\_pkgs](http://ros.org/wiki/gazebo_ros_pkgs)

de transmisión, servicios RPC y el Servidor de Parámetros. Un sistema de control de un robot se formará por la integración de distintos nodos, cuanto mayor sea la funcionalidad de la que se dote al robot, mayor número de nodos tendrá. Existen nodos de control de láser, vista gráfica del sistema, motores de ruedas, odometría, cámaras, etc. La existencia de nodos de ROS en el robot proporciona beneficios para el sistema robótico como tolerancia adicional a fallos soportados por cada nodo de manera individual, de esta manera el fallo se concentra en un solo nodo. Además, la complejidad del código se reduce con estos sistemas monolíticos.

Los *topics* de ROS actúan como forma de comunicación, de esta manera se definen como buses sobre los que los nodos intercambian mensajes. Gracias a la semántica de publicación y/o suscripción anónima de los *topics*, se desacopla la producción de información de consumo. Debido a esto los nodos no saben con quién se están comunicando. Por otra parte, los nodos interesados en un *topic*, se suscriben a él para recoger la información que se publique por el mismo y, por otra parte, los nodos que generen datos pertenecientes a ese *topic*, transmitirán la información por él. Es importante destacar que puede haber varios editores o generadores y varios suscriptores del mismo *topic*.

Existen una gran cantidad de *plugins* de ROS que proporcionan una enorme diversidad de funcionalidad para el desarrollo de robots<sup>3</sup>. Entre ellos destacan el plugin que controla el láser, llamado *libgazebo\_ros\_laser* o el que controla una cámara, llamado *libgazebo\_ros\_camera*. Ambos plugins serán usados en las prácticas contenidas en este Trabajo de Fin de Grado.

### 3.2. Simulador Gazebo

Gazebo<sup>4</sup> es un simulador de robótica que permite emular escenarios tridimensionales para robots autónomos (Figura 3.1). Es apropiado para comprobar algoritmos basados en visión artificial y elusión de objetos. Al desarrollar algoritmos de control de robots es necesaria la realización de pruebas del software para confirmar la validez del código escrito. Es por ello que Gazebo adquiere una gran importancia, dado que permite probar

---

<sup>3</sup>[http://wiki.ros.org/gazebo\\_plugins](http://wiki.ros.org/gazebo_plugins)

<sup>4</sup><http://gazebosim.org/>

la eficacia del código sin necesidad de hardware real, evitando dañarlo. Esta es una de las razones por las que los simuladores son importantes en la robótica, permiten abaratar costes evitando los daños en el hardware del robot.

El simulador utilizado en el presente Trabajo de Fin de Grado es Gazebo 7, al ser de código abierto, versátil (capaz de simular objetos, robots y sensores en entornos complejos de interior y exterior), al poseer una interfaz de gran calidad y un robusto motor de físicas (pueden describirse componentes como la masa, rozamientos, inercia, amortiguamiento, etc.). Fue elegido para soportar el DARPA Robotics Challenge de 2012 a 2015 y está mantenido por la Fundación Open Robotics<sup>5</sup>

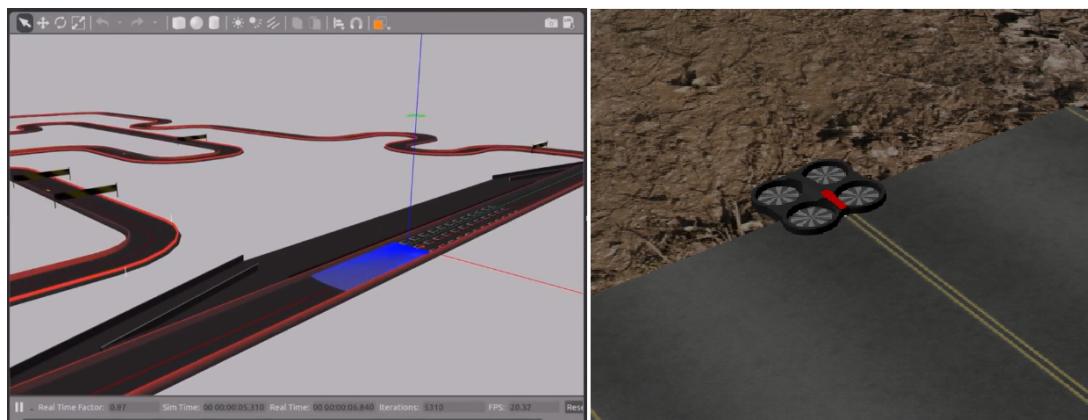


Figura 3.1: Ejemplo de mundo y modelo de Gazebo

Los escenarios de Gazebo se describen en fichero con extensión ".world", que son ficheros escritos en XML (Extensible Remarkable Language) de descripción de documentos, definidos en el lenguaje de simulación SDF (Simulation Description Format), donde se recogen todos los elementos del escenario:

- Escena: Iluminación, propiedades del cielo, sombras, etc.
- Mundo: Representación del mundo como conjunto de modelos, *plugins* y propiedades físicas.
- Modelo: Componentes que forman el robot, como articulaciones, objetos de colisión, sensores, etc.

---

<sup>5</sup><https://www.openrobotics.org/>

- Físicas: Gravedad, inercia, rozamiento, colisiones, motor físico, tiempo, etc.
- *Plugins*: Pueden incluirse en el mundo, el modelo o un sensor. Pueden incluirse *plugins* disponibles en la red como el que da soporte completo a la funcionalidad del robot Roomba, llamado *libroombaplug.so*.

Cada elemento del escenario cuenta con una etiqueta propia que lo distingue del resto. Cualquier propiedad descrita dentro de su etiqueta tiene que ir marcada con la etiqueta de la propiedad correspondiente. En la versión 7 de Gazebo se ha incluido un editor de modelos básico para desarrollar modelos y escenarios básicos en 3D. A partir de los modelos que se importen o desarrollen en Gazebo, es necesario adjuntar un *plugin* que los dote de la funcionalidad necesaria, de otra manera no serían más que simples objetos inanimados.

### 3.3. Entorno JdeRobot

La plataforma *JdeRobot*<sup>6</sup> es un *middleware* abierto para desarrolladores de robots y visión artificial. Fue creada por el Grupo de Robótica de la Universidad Rey Juan Carlos en 2003 y está licenciada como GPLv3<sup>7</sup>. La estructura de esta plataforma ha sido desarrollada en C y C++, aunque tiene componentes escritos en Python y JavaScript. El entorno ofrecido es mediante componentes, los cuales son ejecutados como procesos que interoperan entre sí mediante *middleware* de comunicaciones como ICE o *ROS-Messages*, que permiten la interoperación de componente en un entorno multilenguaje.

JdeRobot facilita los *drivers* necesarios para la funcionalidad de sus robots. De esta manera, los *drivers* están asociados al hardware del robot proporcionando interfaces de acceso, por lo que simplifica la comunicación de las aplicaciones con los actuadores del robot que se realiza mediante una función mediante los interfaces ICE o ROS.

Los dispositivos que se pueden encontrar en JdeRobot son muy diversos, destacan el cuadricópteros como el Ardrone de Parrot, operativo con ICE, o el SoloDrone de 3DR, operativo con ROS, varios coches Fórmula1 simulados que incluyen modelos de la mayoría de las escuderías, operativos con ROS y modificados en este TFG para incluirlos en la plataforma.

---

<sup>6</sup><http://jderobot.org>

<sup>7</sup><https://www.gnu.org/licenses/quick-guide-gplv3.html>

JdeRobot incorpora librerías de software libre para su uso como OpenCV para visión, Eligen para álgebra o PCL para manejo de nubes de puntos. Al ser compatible con ROS, en específico con ROS-Kinetic, las aplicaciones de la plataforma pueden incorporar nodos de ROS y conectarse a ellos de manera fluida.

Las prácticas que componen este Trabajo de Fin de Grado se han desarrollado en la versión de JdeRobot 5.6.2, última versión estable.

### 3.4. Editores de modelos 3D: SketchUP y Belnder

Para el desarrollo del modelo 3D de robots y escenas en el simulador se ha trabajado con dos editores de modelos, SketchUp 2018<sup>8</sup> y Blender v2.80<sup>9</sup>. Estos editores son necesarios para importar los modelos de robots generados en ellos al simulador Gazebo.

Existe un almacén web<sup>10</sup> desde donde es posible descargarse una gran variedad de modelos y escenarios, además de desarrollar los propios. Una vez desarrollado el modelo o escenario, estos editores exportan el modelo o escenario en formato ".dae"(Digital Assets Exchange), formato expresado en el lenguaje XML, y los correspondientes texturas en imágenes con formato ".JPG". Una vez obtenidos estos ficheros, ya son importables por el simulador Gazebo, pero es necesario una modificación para dotar al modelo o escenario de colisiones, inercias, gravedad, etc.

Los escenarios y modelos generados por estos editores son creados mediante la intersección de líneas, generando los distintos tipos de objetos. Es posible adjuntar una textura o color a cada cara que forma el objeto. Además, el editor Blender, al ser más complejo, permite la introducción de iluminación y trabajar con formas geométricas en tres dimensiones directamente. En cambio el editor SketchUp trabaja con líneas, aunque es más sencillo de utilizar.

El editor de modelos Blender es de código abierto pero el editor de modelos SketchUp es de pago, pero tiene la ventaja de contar con su almacén de modelos en el que puedes descargar una gran variedad de modelos.

En este TFG se han creado los escenarios de Gazebo de ambas prácticas con estos

---

<sup>8</sup><https://www.sketchup.com/>

<sup>9</sup><https://www.blender.org/>

<sup>10</sup><https://3dwarehouse.sketchup.com/?hl=es>

editores. Con SketchUP se han desarrollado los escenarios y con Blender se han editado las líneas de contornos para mejorar la iluminación del escenario. Una vez finalizado el editado de los modelos se han exportado a Gazebo.

### 3.5. Lenguaje Python

Python<sup>11</sup> es un lenguaje de programación orientado a objetos, interpretado y de alto nivel con semántica dinámica. Es un lenguaje de fácil aprendizaje y comprensión debido a su apariencia intuitiva. Su creador fue Guido van Rossum, un investigador holandés que trabajaba en el centro de investigación CWI (Centrum Wiskunde & Informática). La primera versión de este lenguaje surgió en 1991, pero no fue publicado hasta tres años después. El nombre que recibió este lenguaje fue dado por su creador en honor a la serie de televisión *Monty Python's Flying Circus*.

La combinación del tipado y el enlace dinámico con sus estructuras de datos integradas de alto nivel permiten un desarrollo rápido de aplicaciones, scripting o ser lenguaje de interconexión de componentes existentes. La sintaxis fácil y simple enfatiza su legibilidad y reduce el coste de mantenimiento del código. Además, Python admite módulos y paquetes, por lo que fomenta la modularidad del programa y la reutilización de código.

El intérprete de Python y la extensa biblioteca de paquetes están disponibles en formato binario o en código fuente de manera gratuita para las plataformas principales y pueden ser distribuidas libremente.

La última versión de Python Software Foundation es la 3.6.5. En este Trabajo de Fin de Grado hemos utilizado la versión 2.7.12, compatible con JdeRobot 5.6.2 y con ROS-Kinetic. Las dos prácticas desarrolladas en este trabajo están escritas en esta versión.

### 3.6. Biblioteca OpenCV

OpenCV<sup>12</sup> (Open Source Computer Vision Library) es una librería de código abierto destinada al procesamiento de imágenes y el aprendizaje máquina. Fue desarrollada por Intel y publicada bajo licencia de BSD. El propósito de esta librería es facilitar el desarrollo

---

<sup>11</sup><https://www.python.org/>

<sup>12</sup><https://opencv.org/>

de programas de visión por computador en tiempo real.

Se trata de una librería multiplataforma con soporte para MacOS, Linux, Android y Windows. Además, existen versiones en Java, Python y C# a pesar de que era, originalmente, una librería en C/C++. También existen interfaces en desarrollo para Ruby, Matlab y otros lenguajes. La librería OpenCV implementa algoritmos para técnicas de detección de rasgos, clasificación de acciones humanas en vídeos, reconocimiento, segmentación de objetos, calibración, seguimiento de caras, análisis de la forma y movimiento, reconstrucción 3D...

Los algoritmos que componen esta librería están basados en estructuras de datos flexibles acoplados a estructuras IPL (*Intel Image Processing Library*), utilizando la arquitectura de Intel respecto a la optimización de la mayoría del paquete. También aprovecha la aceleración de cómputo gracias al uso de tarjetas gráficas avanzadas (GPUs). OpenCV fue desarrollado para tener una alta eficiencia computacional. Está escrito en el lenguaje de programación C y puede aprovechar las ventajas de los procesadores *multicore* de nueva generación. Tales son las ventajas que aporta que las grandes compañías como Google, Yahoo, Microsoft, Intel, IBM, Sony, Toyota u Honda utilizan esta librería, que se ha convertido en el estándar de facto en su campo.

Para este trabajo se ha utilizado la librería OpenCV en la versión 3.2 y ha sido empleada en toda la parte del código relacionada con tratamiento de imágenes.

### 3.7. Biblioteca PyQt

PyQt<sup>13</sup> es un conjunto de enlaces Python utilizado para el conjunto de herramientas Qt, un *framework* multiplataforma orientado a objetos y escrito en C++ que permite el desarrollo de interfaces gráficas. Incluye sockets, hilos, bases de datos SQL, Unicode, etc. Combina todas las ventajas de Qt y Python empleando todas las funcionalidades de Qt con un lenguaje de programación sencillo como es Python. Fue desarrollada por Riverbank Computing Ltd y tiene soporte multiplataforma con versiones para Windows, Linux, Mac OS X, iOS y Android.

Para este proyecto se ha utilizado la versión 5 de PyQt. Se trata de un conjunto de enlaces Python para Qt5, con soporte para Python 2.x y Python 3.x. Incluye más de 6000

---

<sup>13</sup><https://pypi.org/project/PyQt5/>

funciones y 620 clases y métodos. Dispone de una licencia dual, es decir, puede elegirse una licencia comercial para usuarios o una licencia GPL (General Public Licence) para desarrolladores.

Las clases de PyQt5 se dividen en módulos: QtCore, QtGui, QtWidgets, QDom y QSql, entre otros. Para las dos prácticas desarrolladas, se han utilizado los siguientes módulos:

- QtGui: contiene clases para la creación de interfaces gráficas y el desarrollo de gráficos en 2D, imágenes, texto y desarrollo de ventanas.
- QtCore: incorpora las clases principales no relacionadas con la interfaz gráfica. Se utiliza para trabajar con archivos, hilos, datos, procesos, urls, etc.
- QtWidgets: está formado por clases que proporcionan distintas funcionalidades a la interfaz del usuario.

### 3.8. Entorno web Jupyter

Jupyter Notebook<sup>14</sup> se trata de una aplicación web de código abierto que permite al usuario desarrollar y compartir documentos que contengan código empotrado, ecuaciones, textos y visualizaciones. Proporciona muchas ventajas entre las que destacan limpieza, simulación numérica, modelado estadístico, visualización y transformación de datos, aprendizaje automático, etc. Inicialmente fue desarrollado como IPython 3.0 pero se renombró como Jupyter.

El cuadernillo de trabajo o *Notebook* está compuesto por celdas en las que se inserta el código, en lenguaje Python, o distintos elementos de texto enriquecido como párrafos, ecuaciones, enlaces, figuras, etc (Figura 3.2). El resto de tipos de celdas son legibles para los humanos como figuras, tablas o texto y contienen los análisis y resultados del trabajo, además de documentos ejecutables para la ejecución del análisis. El *Notebook* está formado por una sucesión lineal de celdas. Hay cuatro tipos básicos:

- **Celda de código:** se trata de *input* y *output* de código que se ejecuta en el kernel del cuadernillo a tiempo real.

---

<sup>14</sup><http://jupyter.org/>

- **Casillas de reducción:** son celdas de texto con ecuaciones en LaTex empotradas.
- **Encabezado de celdas:** formado por 6 niveles de organización jerárquica y su formato.
- **Celdas sin formato:** se trata de texto sin formato que se incluye en el cuadernillo sin ningún tipo de modificación cuando los cuadernillos son convertidos a formatos distintos mediante *nbconvert*.

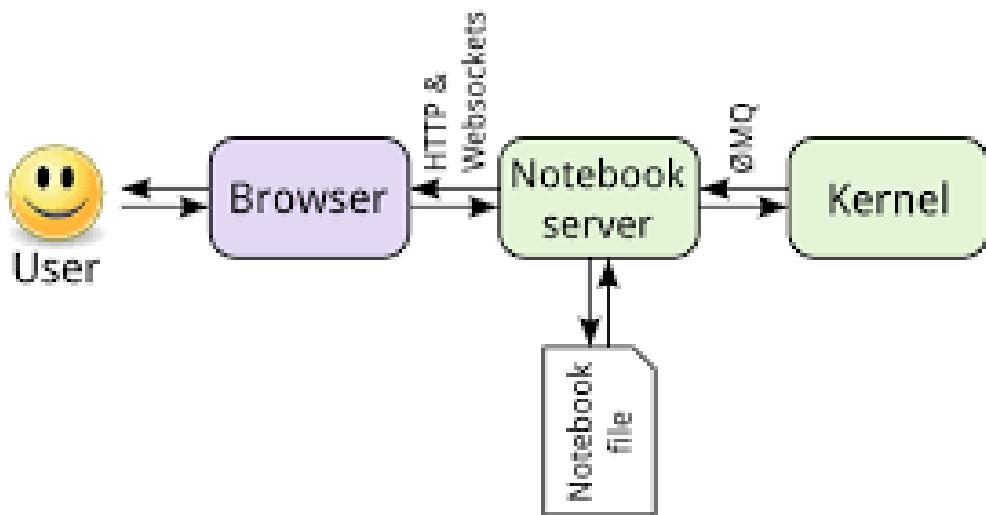


Figura 3.2: Arquitectura de Jupyter

La aplicación tiene un modelo cliente-servidor que permite la ejecución y edición de los *Notebooks* mediante un navegador web. La aplicación Jupyter puede ejecutarse desde un escritorio local sin necesidad de disponer de conexión a Internet o instalarse en un servidor remoto y acceder a ella a través de internet. Además de estas características, Jupyter dispone de un *Panel de control* o llamado *Dashboard* con el que permite la apertura, guardado y cierre de los archivos y los núcleos del *Kernel*.

Estos *kernels* son motores computacionales que ejecutan el código contenido en el *Notebook*. Existen multitud de *kernels* oficiales que dan soporte a distintos lenguajes como Python, Julia, R, Ruby Haskell, Scala, ...), incluso versiones distintas de *kernels* para un mismo lenguaje. Al abrir el *Notebook*, el *kernel* se inicializa automáticamente. De este modo, al ejecutar una celda del cuadernillo se reproduce el código contenido en ella y, a

continuación, de muestran los resultados. Es importante tener en cuenta que, dependiendo del código contenido en la celda, el *kernel* puede consumir una gran cantidad de recursos CPU y RAM, que están limitados por el navegador.

Los cuadernillos, así como el resto de tipos de documentos (ficheros de código auxiliar, fichero de texto, imágenes, etc.) pueden guardarse y son almacenados en el sistema de ficheros local del usuario. El *Notebook* será almacenado con una extensión *.ipynb*.

Gracias a esta aplicación, se han desarrollado prácticas análogas a las existentes en la plataforma *Robotics-Academy* con nodos ROS. De esta manera, *Robotics-Academy* se acerca a dar soporte multiplataforma gracias al uso del navegador web y Jupyter para la interacción con su entorno docente. Para ello se ha empotrado el nodo académico de las prácticas en celdillas de un *Notebook* de Jupyter y se ha proporcionado una celdilla para que el alumno escriba el algoritmo de solución en él y sólo tenga que ejecutar esa celdilla para ver los resultados. Los *Notebooks* utilizados para la recreación de las prácticas han sido desarrollados en la versión 2.7 de Python, por lo que las prácticas y la solución que desarrollean los alumnos deben ser en esta versión.

# **Capítulo 4**

## **Ejercicio de cronometraje de vueltas competitivas con Gazebo y ROS**

En este capítulo del TFG ya se han definido el contexto, los objetivos y las herramientas utilizadas para la consecución de este proyecto. En este capítulo abordaremos todo lo relacionado con el desarrollo de una de las prácticas llamada *Chrono*. Esta práctica forma parte del conjunto de prácticas de *JdeRobot*. Se explicará la infraestructura de soporte de la práctica, el software y el funcionamiento de la práctica.

### **4.1. Enunciado**

El objetivo de esta práctica, enfocado al estudiante, es el desarrollo de un algoritmo que dote de la inteligencia necesaria a un modelo de robot F1 que compite con el récord grabado para el circuito. De esta manera, no solo se permite el desarrollo de un algoritmo funcional, sino que se propone un avance en el grado de dificultad exigiendo al alumno que optimice su algoritmo para que consiga completar una vuelta al circuito sin colisionar y con un tiempo inferior al ofrecido.

Para desarrollar la solución del algoritmo, el alumno deberá abordar distintos problemas relacionados con la programación. El primero de ellos es programar un algoritmo para realizar un filtrado de la imagen que capta el robot F1 por su cámara y escoger la línea roja central del circuito. Tras esto, el alumno deberá desarrollar un algoritmo que establezca un avance controlado del robot F1 para que siga la línea roja

que ha sido filtrada por el algoritmo anterior.

Como puede observarse en los problemas abordados, además de tener que afrontar los problemas, es necesario que sean abordados conjuntamente, ya que el movimiento controlado del modelo depende del filtrado de las imágenes. Para añadir dificultad a la práctica, es necesario que el algoritmo de control de movimiento del vehículo sea eficaz dado que, de otro modo, será más lento que la solución proporcionada y llegará en segundo lugar.

## 4.2. Infraestructura

En esta sección se abordarán los distintos elementos involucrados en la práctica (Figura 4.1).



Figura 4.1: Infraestructura de Chrono

### 4.2.1. Modelo de robot F1

El robot que se ha utilizado para esta práctica es un modelo de robot terrestre móvil que cuenta con 4 ruedas. El chasis elegido corresponde a un modelo de F1, específicamente del modelo RedBull. Además de este modelo se han incluido un gran conjunto de modelos

de F1 correspondientes a las principales escuderías que participan en la Fórmula 1. Esto supone un total de 12 modelos de coches de escuderías reales (India, HRT, Lotus, McLaren, Mercedes, RedBull, Renault, Tororoso, Virgin y Williams) y un modelo sin marca comercial. Además, para cada modelo de coche ha sido necesario el desarrollo de dos modelos distintos, uno que incorpora una cámara, para esta práctica, y otro modelo que tiene un láser para otras prácticas de *JdeRobot*. Gracias a estos modelos, en la práctica actual puede usarse indistintamente el modelo preferido por el estudiante (Figura 4.2).

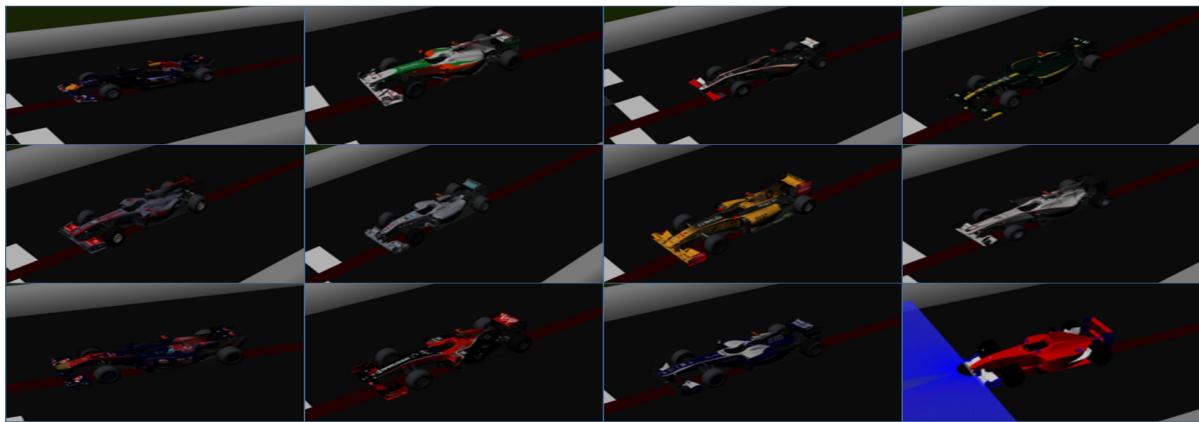


Figura 4.2: Modelos disponibles de coches

Cualquiera de los modelos de la Figura 4.2 utiliza los *plugins* de ROS para dotar al modelo de movimiento, captación de imágenes y odometría. Estos *plugins* son ofrecidos por las librerías de *ROS Kinetic*:

- *libgazebo\_ros\_camera* Para la captación de imágenes.
- *libgazebo\_ros\_planar\_move* Para el control de motores y obtener la odometría.

Tras crear el modelo, basta con importarlo en el mundo de Gazebo para su utilización.

#### 4.2.2. Cámara

En la parte frontal del modelo se ha incluido una cámara para poder captar imágenes.

Los *plugins* de la cámara dan soporte a una cámara conectada por USB con una velocidad de refresco de las imágenes de 20 fps. Gracias a esto, el modelo puede recoger

imágenes a una velocidad suficiente y no perder de vista la línea. La velocidad de refresco de la cámara es un parámetro importante debido a que, dependiendo de este parámetro, entre otros, la velocidad del coche tendrá que adaptarse.

Las imágenes captadas por el *plugin* son recogidas por el nodo académico que, mediante un API sencillo, proporciona las imágenes captadas y soporta la visualización de los filtros que se le apliquen a la imagen.

#### 4.2.3. Sensor Odométrico

El sensor odométrico del modelo es imprescindible para esta práctica, ya que el nodo académico recoge la odometría del robot y la publica en un mapa con el modelo del circuito. La odometría utiliza sensores de movimiento para determinar la posición incremental del robot con respecto a su posición inicial.

Si el robot conoce el diámetro de sus ruedas, puede conocer su posición incremental contando el número de vueltas de las mismas. El conteo de vueltas se realiza mediante *encoders*, que emiten un número fijo de pulsos por revolución. En el caso de los modelos de robots F1 utilizados, el refresco de la odometría se realiza a un ratio de 20. Gracias a este ratio, pueden contarse el número de vueltas y giros dados y, de esta manera, saber el recorrido realizado.

#### 4.2.4. Circuito de Nürburgring

Para esta práctica se ha desarrollado un modelo del circuito de Nürburgring acortado. Mediante el uso de los softwares *Blender* y *SketchUp*, se ha modelado el circuito con una línea de salida, una grada, la carretera, paredes para evitar que el robot se salga del recorrido y césped de adorno (Figura 4.3).

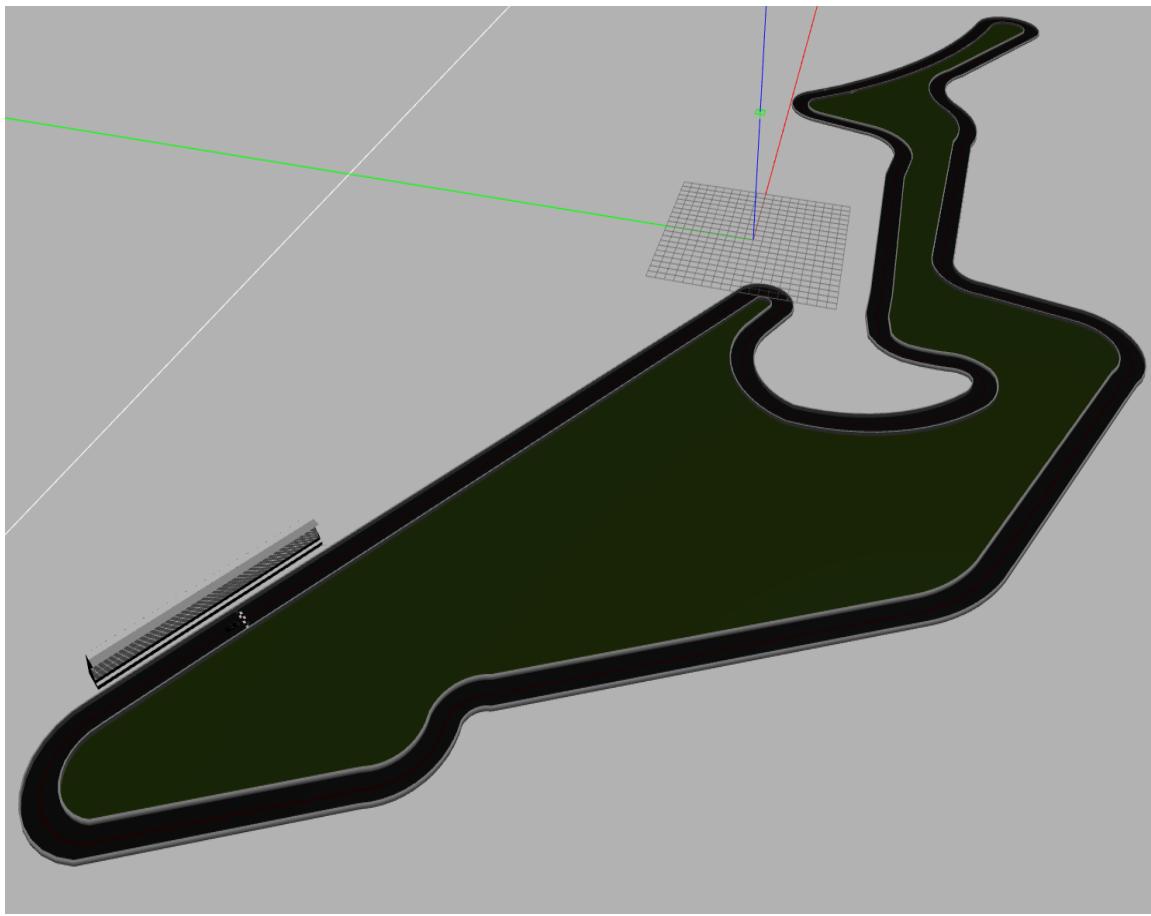


Figura 4.3: Modelo del circuito de Nürburgring

#### 4.2.5. Ficheros de configuración

Para la incorporación del modelo del circuito y del robot, es necesario la creación de un modelo de configuración que importe en *Gazebo* los elementos de los que consta el escenario y su localización. Este fichero tiene la extensión *.world* y *Gazebo* es capaz de leerlo y mostrar el escenario al iniciarse. El código del fichero es el siguiente:

```
<?xml version="1.0"?>
<sdf version="1.4">
  <world name="default">
    <include>
      <uri>model://sun</uri>
    </include>
    <include>
      <uri>model://nurburgrinLine</uri>
```

```
<pose>70 -47 0 0 0 0</pose>
</include>
<include>
  <uri>model://f1ROS</uri>
  <pose>0.05 -0.44 0 0 0 0.9</pose>
</include>
</world>
</sdf>
```

Además de este fichero de configuración, es necesario un fichero complementario que importe los *plugins* y *drivers* de ROS-Kinetic. Este tipo de fichero tienen la extensión *.launch*. En este fichero se pasan a *Gazebo* argumentos como el nombre del fichero de configuración con el escenario, establecer el tiempo que se va a utilizar en el escenario, la posible implementación de un GUI y otras opciones de depuración. El fichero es el siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<launch>
  <!-- We resume the logic in empty_world.launch, changing only the name of the world to be
       launched -->
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="nurburgrinLineROS.world"/> <!-- Note: the world_name is
      with respect to GAZEBO_RESOURCE_PATH environmental variable -->
    <arg name="paused" value="false"/>
    <arg name="use_sim_time" value="true"/>
    <arg name="gui" value="true"/>
    <arg name="headless" value="false"/>
    <arg name="debug" value="false"/>
    <arg name="verbose" default="false"/>
  </include>
</launch>
```

### 4.3. Nodo Académico

En esta sección se trata el desarrollo del nodo académico de la práctica (Figura 4.4). Este elemento ha sido desarrollado específicamente para la práctica ofreciendo al estudiante todas las herramientas necesarias para un desarrollo del algoritmo sencillo

(Figura 4.4).



Figura 4.4: Nodo Académico de Chrono

#### 4.3.1. Arquitectura software

Esta práctica tiene dos hilos de ejecución para aliviar la carga computacional de la práctica. De esta manera se aumenta la velocidad la que puede trabajar el simulador *Gazebo*.

- Hilo de sensores: este hilo se encarga de la actualización de los datos de los sensores del robot. Este hilo se comunica con *Gazebo* para recoger datos de la odometría y la cámara y para publicar datos de control del motor para mover el robot.
- Hilo de la interfaz gráfica del usuario (GUI): este hilo se encarga del refresco del GUI de la práctica. En esta práctica tiene una carga computacional elevada dado se encarga de refreshar las imágenes obtenidas por la cámaras, el procesamiento visual sobre esa imagen que ha realizado el alumno, un mapa del circuito con la posición actualizada del robot y del robot fantasma a vencer, y una lectura controlada de tiempos para sincronizar ambos coches.

Gracias a esta interfaz gráfica el estudiante puede servirse de algunos elementos de depuración que veremos en profundidad en la sección 5.3.2 y 5.3.3. De esta manera el

alumno solo tiene que centrarse en el desarrollo del algoritmo. El nodo académico dispone de una función reservada para que el alumno escriba su algoritmo en ella y pueda ver los resultados, todo ello viene indicado en fichero *README.md* de la práctica.

### 4.3.2. Interfaz de sensores y actuadores

El nodo académico proporciona un API de sensores y actuadores al programador para facilitar la interconexión con los mismos. El API del robot es el siguiente:

- *self.pose3d.getPose3d()*: con esta función podemos obtener los datos de odometría y posición del robot.
- *self.camera.getImage()*: con esta función se recogen las imágenes obtenidas por la cámara del robot.
- *self.motors.senV()* o *self.motors.sendW()*: con esta función publicamos la velocidad y giro del robot.

### 4.3.3. Interfaz gráfica

La interfaz gráfica del usuario (GUI), se utiliza para representar información relacionada con los sensores del robot. Esta información es muy útil para la depuración del algoritmo del estudiante ya que permite la visualización de las publicaciones que realiza el algoritmo al robot.

Esta GUI (Figura 4.5) está formada por cuatro *widgets*, uno para el visionado del robot, otro para su comportamiento, otro para su odometría y otro para la sincronización de la grabación de ROS con el nodo académico.

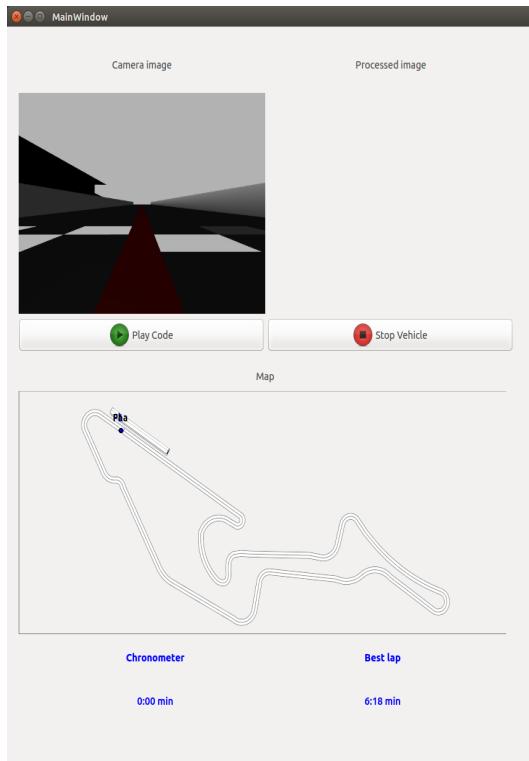


Figura 4.5: Interfaz Gráfica Chrono

El *widget* superior recoge las imágenes captadas por la cámara del robot y las muestra en una ventana del GUI. Al lado de las imágenes captadas por la cámara se muestra otra ventana con el filtrado que realice el alumno de esas imágenes. Esto es muy útil para el estudiante a la hora de enfrentarse al problema de visión que supone la primera parte de la solución de la práctica.

El tercer *widget* está formado por un mapa del circuito, en este caso el circuito de Nürburgring, en el que se pueden visualizar las posiciones del robot y el robot fantasma con el récord del circuito.

El último *widget* se llama *Chrono Widget* y se trata de un algoritmo de sincronización del tiempo de simulación de Gazebo con la reproducción a tiempo real de *ROSbag*.

Además de estos *widget*, en el GUI se incluyen distintos botones de control para hacer una pausa académica del algoritmo y para reiniciar la posición del teleoperador. El primer botón se llama *pushButton* y consiste en un botón interactivo para iniciar el código programado por el alumno y para parar el código. El segundo botón se llama *ResetButton* y se utiliza para reiniciar el teleoperador de la interfaz gráfica y hacer que el

robot no se mueva.

La parte inferior del GUI tiene un lector de tiempos en el que se muestra el tiempo simulado de *Gazebo* y, por lo tanto, el tiempo que está necesitando el robot para completar la vuelta. A su derecha se muestra el tiempo del récord del circuito para que el alumno tenga una idea de la optimización que necesita el código de control de movimiento del robot para que sea más eficiente.

#### 4.3.4. Visor de imágenes

En este primer *widget* se pueden visualizar las imágenes captadas por la cámara que incorpora el robot. Gracias a ella, el alumno puede tener una idea de la visión del robot y programar una solución de una manera más sencilla.

A la derecha de la ventana del visor de imágenes de la cámara, se ha incluido otra ventana de visualización. Esta ventana se encarga de mostrar el procesamiento de la imagen desarrollado por el alumno. Gracias a esta ventana, el alumno puede hacerse una idea del algoritmo de procesamiento de imagen que ha desarrollado.

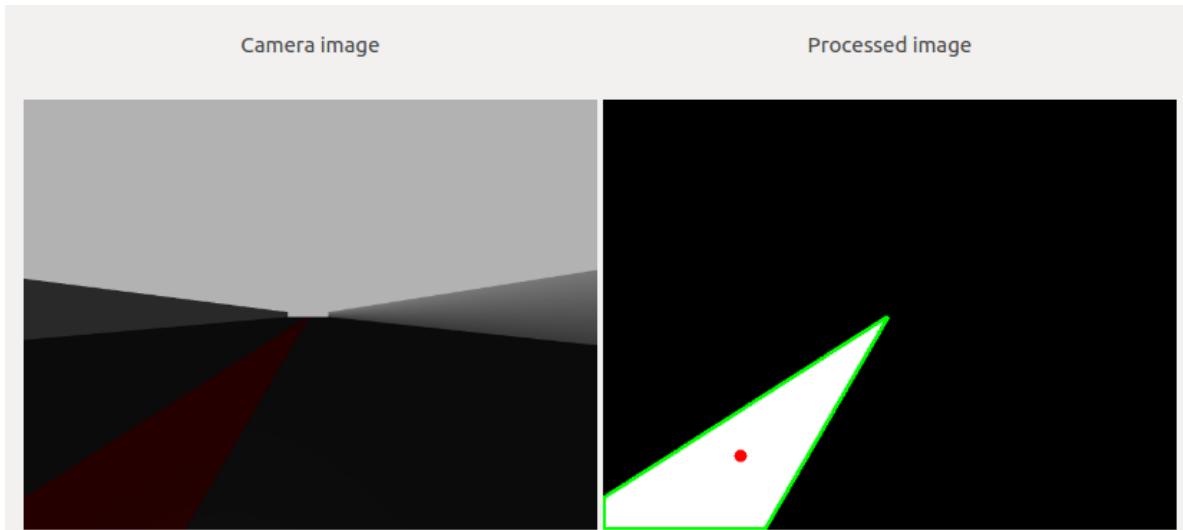


Figura 4.6: Visor de imágenes de Chrono

#### 4.3.5. Botones de control

Otra función incluida en el GUI son los botones de control. Existen dos botones de control “Push Button” y “Reset Button”. Con el primer botón de control, se puede parar la ejecución de la práctica y volver a reanudarla. Con esto se permite un control en la depuración del código del alumno, ya que puede detener la ejecución en cualquier momento para visualizar los datos que se están recogiendo.

El segundo botón es complementario al primero, pues su funcionalidad es la de reiniciar las publicaciones de datos que se envían a los motores del robot. Es decir, con este botón se puede parar el robot y después con el botón “Push Button” se puede parar la ejecución.



Figura 4.7: Ilustración del botón de pausa académica



Figura 4.8: Ilustración del botón de comienzo académico

#### 4.3.6. Mapa de odometría

Ya se ha descrito la funcionalidad del mapa en la sección anterior, en esta sección nos vamos a centrar en el algoritmo que dota de esta funcionalidad al mapa.

El nodo académico, concretamente el hilo de ejecución encargado del interfaz gráfico (GUI), se encarga de cargar la imagen del mapa en el GUI. Una vez hecho esto, recoge los datos de odometría del sensor *Pose3D* y dibuja su posición en el mapa. Adicionalmente, recoge los datos de odometría del fichero de grabación de ROS para extraer la posición del coche fantasma y dibujarla en el mapa también.

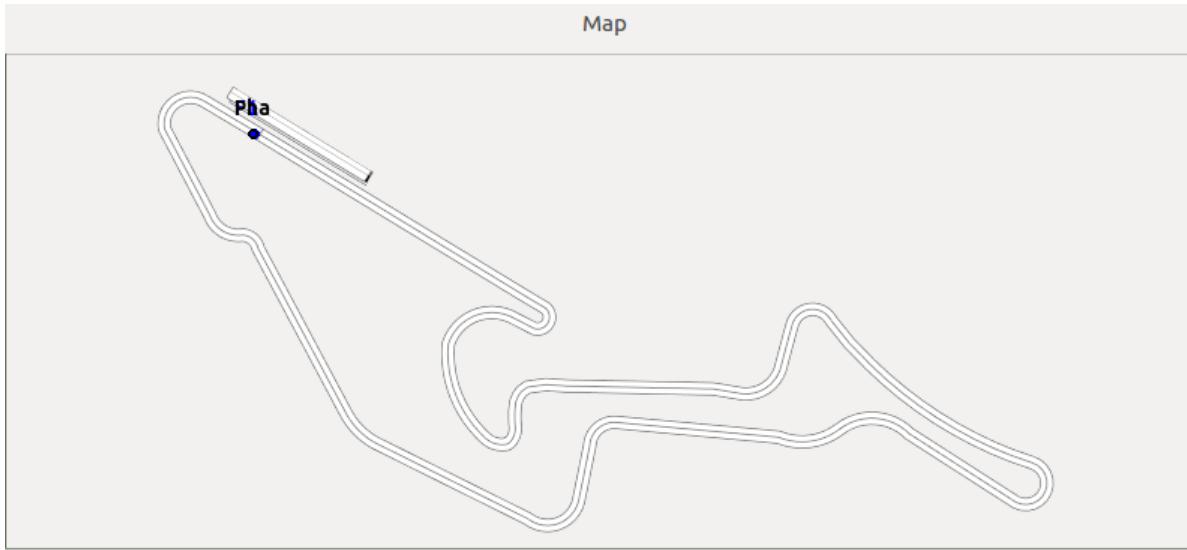


Figura 4.9: Mapa del GUI

Esta ejecución es bastante pesada dado que requiere de una actualización constante para reflejar una posición lo más exacta posible, tanto del coche fantasma como del robot. Es por ello que se utiliza el hilo de ejecución proporcionado por el GUI. Esto alivia a la ejecución principal de una gran carga computacional.

#### 4.3.7. Lector de tiempos

Este ha sido el principal reto de esta práctica por su complejidad a la hora de recoger los datos, tanto de *Gazebo*, como de *ROSbag* y sincronizarlos.

El problema que se planteaba en la sincronización de tiempos era el hecho de que los tiempos de *Gazebo* se sincronizan mediante el tiempo de simulación del propio simulador pero los tiempos grabados de *Gazebo* mediante *ROSbag*, se reproduce mediante el tiempo real. Esto supone una desincronización muy grande ya que, en el tiempo de simulación es, aproximadamente, un 0,15 veces el tiempo real. Adicionalmente existe el problema de que el tiempo de simulación depende de la potencia de cómputo de cada ordenador, por lo que la reproducción no se puede realizar a un ritmo constante porque sería un hándicap muy grande para los ordenadores potentes que se verían afectados a reproducir de una manera muy lenta.

En este aspecto, debían abordarse los dos problemas propuestos de manera simultánea.

Por un lado sincronizar el tiempo de reproducción con el tiempo de simulación, y por otro llevar un ritmo del tiempo de simulación adaptado para la carga de cómputo que pueda soportar cada ordenador. Adicionalmente se encontró un problema adicional con la reproducción de la grabación de ROSbag por el cual, las etiquetas de tiempo de las que consta, no se inicializan desde el comienzo sino que se graban con el tiempo simulado actual. Es por esto que la reproducción comienza con una etiqueta temporal distinta de cero.

Para solucionar sendos problemas ha sido necesaria la definición de diversas medidas de tiempos:

- *Tiempo de grabación*: este tiempo representa el ritmo al que se reproduce la grabación de ROSbag.
- *Tiempo simulado*: este tiempo representa el tiempo al que se refresca el simulador *Gazebo*.
- *Tiempo de reproducción*: este tiempo se basa en el tiempo de simulación pero restándole el offset del comienzo de la práctica. Por ello empieza cuando el estudiante ejecuta la práctica en lugar de cuando se inicia el simulador.

El algoritmo de simulación comienza recogiendo el instante de tiempo en el que el alumno ejecuta su código mediante el comando:

```
initime = rospy.Time.from_sec(rospy.get_time()).to_sec()
```

Una vez obtenido el tiempo inicial, es necesario recoger el instante de tiempo en el que estamos refrescando la sincronización:

```
sim_time = rospy.Time.from_sec(rospy.get_time()).to_sec()
```

Además de estos dos tiempos, es necesario conocer el tiempo de la primera etiqueta de la grabación de ROSbag para comenzar a leer las etiquetas desde ese offset:

```
rep_start = str(bag).split('start:      ')[1].split(' ')[4].split()[0][1:-1]
```

Con el tiempo de simulación actual, el tiempo inicial y el tiempo de inicio de la reproducción, podemos obtener el tiempo con el que vamos a comparar las etiquetas temporales de la grabación para saber si tenemos que leer la etiqueta temporal y actualizar

la posición del coche fantasma en el mapa del GUI o seguir devolviendo la misma posición porque es pronto.

```
t_sim_unif = sim_time - initime + float(rep_start)
```

Gracias a esta sincronización se pueden devolver los valores de odometría grabados para la solución con el récord de la vuelta para el coche fantasma y actualizar su posición en el mapa. A continuación se describe el código utilizado para la sincronización:

```
def synchronize(self):
    global posx, posy, cursor

    t_sim_unif = sim_time - initime + float(rep_start)
    if initime != 0.0 and t_sim_unif != 0.0:
        for (topic,msg,t) in bag.read_messages(start_time=rospy.Time(t_sim_unif-0.05)):
            t = t.to_sec()
            if t_sim_unif > t:
                try:
                    posx = str(msg).split('x: ')[1].split()[0]
                    posy = str(msg).split('y: ')[1].split()[0]
                    return float(posx), float(posy)
                except IndexError:
                    pass
            else:
                return float(posx), float(posy)

    else:
        return float(posx), float(posy)
```

A continuación se muestra una imagen (Figura 4.10) del *widget* en el GUI con la práctica iniciada. A la izquierda se puede visualizar el cronómetro con la duración de la ejecución y a su derecha se encuentra el registro con el la duración de la vuelta para la grabación que se esté reproduciendo.

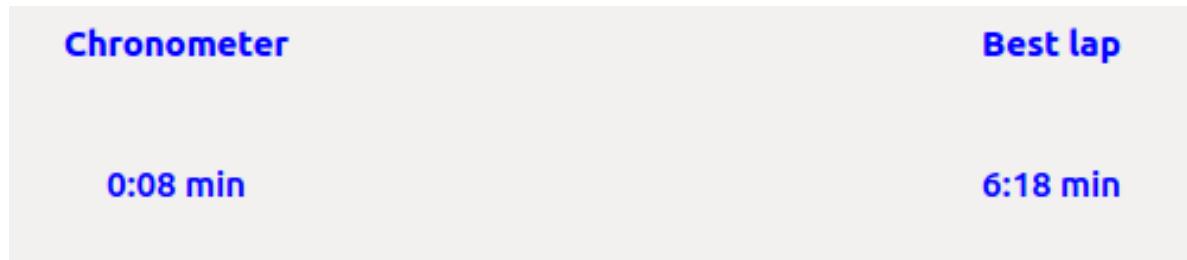


Figura 4.10: Lector de tiempos de Chrono

## 4.4. Soluciones de referencia

Para la práctica, ha sido necesario el desarrollo de dos soluciones de referencia distintas. Una de ellas ha sido adaptada de una solución previamente hecha y la otra solución ha sido programada por completo.

La primera solución ha sido adaptada de otra práctica de *JdeRobot Academy* llamada *Follow Line*. Aunque esta práctica es similar, la solución estaba desactualizada por lo que ha sido necesaria una actualización del código, en específico de la parte de comparación de píxeles en la imagen filtrada y de la parte de control de movimiento. La actualización de la parte del control de movimiento ha sido más completa debido a que la velocidad del coche en las curvas era demasiado elevada y no estaba sincronizada con el refresco y filtrado de imagen que realiza la cámara insertada en el coche de 20 imágenes por segundo.

La segunda solución desarrollada, ha sido creada por completo utilizando la librería *OpenCV*, con funciones *Built-in* proporcionadas por la misma que facilitan la comprensión del código programado. Para ello ha sido necesario un estudio más profundo de esta librería que ha dotado de la inteligencia necesaria para reescribir el filtrado de imagen en un algoritmo mucho más eficiente y que permite un procesamiento de las imágenes más elevado. Gracias a esto, la velocidad del robot es mayor y por consiguiente, completa la vuelta al circuito sin colisiones y en un tiempo menor.

### 4.4.1. Procesamiento de imagen

El procesamiento de la imagen captada por la cámara, comienza con la recogida de la imagen guardada en el búffer de la misma con la instrucción:

## CAPÍTULO 4. CHRONO

---

```
input_image = self.camera.getImage().data
```

Gracias a esta instrucción es posible ver la imagen captada en la interfaz gráfica del usuario.

Una vez obtenida la imagen, hay que transformarla a una imagen HSV<sup>1</sup> para poder seleccionar el color rojo de la línea central del circuito:

```
image_HSV = cv2.cvtColor(input_image, cv2.COLOR_RGB2HSV)
```

Tras obtener la imagen HSV, podemos seleccionar el rango de valores que componen el rojo de la línea como un array con la intensidad mínima y máxima del color.

```
value_min_HSV = np.array([0, 150, 0])
value_max_HSV = np.array([180, 255, 255])
```

Con esos valores realizamos un filtrado de la imagen para obtener la línea roja exclusivamente:

```
image_HSV_filtered = cv2.inRange(image_HSV, value_min_HSV, value_max_HSV)
```

Usamos el filtro obtenido como una máscara para filtrar la imagen y obtener de esta forma la imagen filtrada en blanco y negro. De esta manera el blanco serán los colores que pasen el filtro y en negro obtendremos el resto de colores. En este caso, obtendremos la línea roja del circuito en color blanco y el resto de la imagen en negro:

```
image_HSV_filtered_Mask = np.dstack((image_HSV_filtered, image_HSV_filtered,
                                         image_HSV_filtered))
```

Una vez filtrada la imagen, se procede a obtener los contornos de la zona filtrada. Para ello, es necesario convertir la imagen a escala de grises:

```
imgray = cv2.cvtColor(image_HSV_filtered_Mask, cv2.COLOR_BGR2GRAY)
```

De esta manera, se pueden conseguir los píxeles que tiene un contraste mayor con sus vecinos, obteniendo el contorno:

```
ret, thresh = cv2.threshold(imgray, 127, 255, 0)
_, contours, hierarchy = cv2.findContours(thresh, cv2.RETR_TREE, cv2.
                                         CHAIN_APPROX_SIMPLE)
```

<sup>1</sup>[https://es.wikipedia.org/wiki/Modelo\\_de\\_color\\_HSV](https://es.wikipedia.org/wiki/Modelo_de_color_HSV)

## CAPÍTULO 4. CHRONO

---

Una vez conseguido el contorno, se dibuja en la imagen filtrada:

```
cv2.drawContours(image_HSV_filtered_Mask, contours, -1, (0,255,0), 3)
```

El siguiente algoritmo, una vez realizado el filtro, es para robustecer el código en el caso de que, en el filtrado de la imagen, se detecten dos zonas con filtro. Esto se puede producir cuando hay una curva y, debido a la resolución de la cámara, recoge la línea roja pero incompleta. Esto produce lagunas de filtro en las que no se visualiza la línea por completo, sino cortada. Para evitar un fallo en el algoritmo, ha sido necesaria la inclusión del siguiente código que recoge todas las zonas filtradas y selecciona la de mayor área:

```
area = []
for pic, contour in enumerate(contours):
    area.append(cv2.contourArea(contour))
if len(area) > 1:
    if area[0] < area[1]:
        M = cv2.moments(contours[1])
    else:
        M = cv2.moments(contours[0])
else:
    M = cv2.moments(contours[0])
```

Tras esta comprobación, se obtienen los valores de los ejes x e y de la zona filtrada. Estos valores forman el centro del área que ha sido filtrada:

```
if int(M['m00']) != 0:
    self.cx = int(M['m10']/M['m00'])
    self.cy = int(M['m01']/M['m00'])
```

Gracias a estos valores, en concreto al valor del eje x, podemos saber la diferencia de posición que tiene el robot con el centro de la línea roja, es decir, la diferencia de posición del robot con el centro del circuito. Para facilitar este cálculo, se ha dibujado un punto en la imagen filtrada con los valores del eje y el eje y para que sea visualizado:

```
cv2.circle (image_HSV_filtered_Mask, (self.cx, self.cy), 7, np.array([255, 0, 0]), -1)
```

Para la visualización de la imagen procesada, basta con utilizar la instrucción proporcionada por la interfaz gráfica del usuario:

```
self.setImageFiltered(image_HSV_filtered_Mask)
```

## 4.4.2. Control de movimiento

En esta sección vamos a tratar el control del movimiento del robot. Es una parte compleja ya que supone un filtrado correcto. En caso contrario, el comportamiento del movimiento del robot será incorrecto. Esto es debido a que el movimiento del robot se realiza en consecuencia al procesamiento de la imagen.

### 4.4.2.1. Filtrado de imagen y control de movimiento basado en píxeles

Este tipo de control de movimiento se basa en la primera solución de la práctica y supone una transformación de la imagen a blanco y negro, para después comprobar los píxeles que cambian de tono para saber hacia qué lado hay que orientar al coche. Se trata de una solución eficaz aunque poco eficiente. Esto es debido a que es necesario comprobar los píxeles uno a uno para averiguar qué píxeles son los que han sufrido esta transformación de tono:

```
# Shape gives us the number of rows and columns of an image
size = image.shape
rows = size[0]
columns = size[1]

# Looking for pixels that change of tone
position_pixel_left = []
position_pixel_right = []

for i in range(0, columns-1):
    value = image_HSV_filtered[365, i] - image_HSV_filtered[365, i-1]
    if(value != 0):
        if (value == 255):
            position_pixel_left.append(i)
        else:
            position_pixel_right.append(i-1)

# Calculating the intermediate position of the road
if ((len(position_pixel_left) != 0) and (len(position_pixel_right) != 0)):
    position_middle = (position_pixel_left[0] + position_pixel_right[0]) / 2
elif ((len(position_pixel_left) != 0) and (len(position_pixel_right) == 0)):
    position_middle = (position_pixel_left[0] + columns) / 2
```

```

elif ((len(position_pixel_left) == 0) and (len(position_pixel_right) != 0)):
    position_middle = (0 + position_pixel_right[0]) / 2
else:
    position_pixel_right.append(1000)
    position_pixel_left.append(1000)
    position_middle = (position_pixel_left[0] + position_pixel_right[0])/ 2

# Calculating the desviation
desviation = position_middle - (columns/2)

```

Una vez obtenidos estos píxeles, el control de movimiento se ejecuta según el siguiente algoritmo, que comprueba estos píxeles y se mueve en consecuencia:

```

if (desviation == 0):
    self.motors.sendV(3)
elif (position_pixel_right[0] == 1000):
    self.motors.sendW(-0.0000035)
elif ((abs(desviation)) < 85):
    if ((abs(desviation)) < 31):
        self.motors.sendV(3)
    else:
        self.motors.sendV(1)
        self.motors.sendW(-0.000045 * desviation)
elif ((abs(desviation)) < 150):
    if ((abs(desviation)) < 120):
        self.motors.sendV(1)
    else:
        self.motors.sendV(1)
        self.motors.sendW(-0.00045 * desviation)
else:
    self.motors.sendV(1)
    self.motors.sendW(-0.0055 * desviation)

```

#### 4.4.2.2. Filtrado de imagen y control de movimiento basado en el centro del contorno

Este tipo de filtrado de imagen es el descrito en la sección 4.4.1 y es mucho más eficiente y con una eficacia mayor que el descrito en el apartado anterior 4.4.2.1. Esto es debido a que no necesita comprobar la imagen completa píxel a píxel, sino que filtra el

área de la línea y procesa su centro.

Una vez hecho este filtro, la control de movimiento es sencillo, pues basta con obtener el valor del eje x cuando el coche está alineado con la línea recta y tomarlo como el movimiento nulo. Tras esto, se puede definir el giro del robot en consonancia con este valor de movimiento nulo. Gracias a ello, el robot girará más o menos cuanto mayor sea la diferencia entre el valor de movimiento nulo (153 aproximadamente) y el valor actual del eje x:

```
if self.cx < 50:  
    self.motors.sendV(1.5)  
else:  
    self.motors.sendV(3.5)  
  
self.motors.sendW((153-int(self.cx))*0.01)
```

## 4.5. Experimentación

La optimización de los algoritmos anteriores ha sido posible gracias a la realización de diversos experimentos. Estos experimentos han hecho salir a la luz errores en el algoritmo desarrollado que han sido subsanados.

Además se han realizado experimentos globales donde se ha testeado la práctica en su totalidad, nodo académico, infraestructura de la práctica y solución desarrollada.

### 4.5.1. Ejecución típica

Se ha preparado un documento *README.md*, incluido en la infraestructura de la práctica, que sirve de guía al alumno a la hora de ejecutar la práctica. En él se incluye información acerca de su ejecución, la API de los sensores y actuadores de ROS e, incluso, un vídeo demostrativo con una ejecución.

Para ejecutar la práctica, es necesario lanzar en una terminal el fichero de configuración de ROS, llamado *f1-chrono.launch*, descrito en la sección 4.2.5. Para lanzar el fichero hay que ejecutar el siguiente comando:

```
roslaunch f1-chrono.launch
```

## CAPÍTULO 4. CHRONO

---

Una vez lanzado el comando en la terminal, se abrirá el simulador *Gazebo* con el escenario del circuito (Figura 4.3).

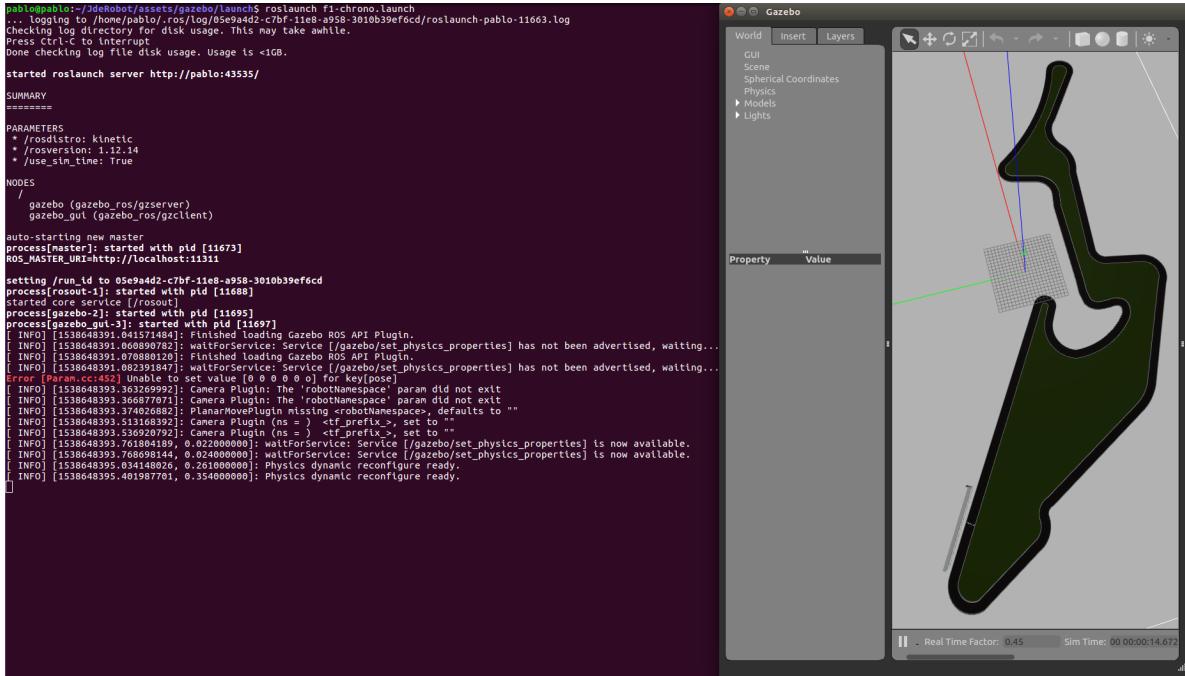


Figura 4.11: Inicialización ROS y Gazebo

Para iniciar el componente académico, será necesario ejecutar otro comando en una terminal distinta:

```
cd ~/Academy/exercises/chrono
python2 chrono.py
```

Una vez ejecutado el comando, el componente académico enlazará los sensores y actuadores proporcionados por *ROS-Kinetic* mediante el fichero de configuración lanzado previamente a las variables:

- self.camera
- self.pose3d
- self.motors

Con estas variables, el nodo académico se comunica con los *drivers* de *ROS-Kinetic*. Además de realizar la conexión con los sensores y actuadores, al ejecutar la instrucción, nos

aparecerá la interfaz gráfica de usuario (GUI) en la que se podrá visualizar las imágenes recogidas por la cámara, los botones de control, el mapa del circuito y el lector de tiempos (Figura 4.12).

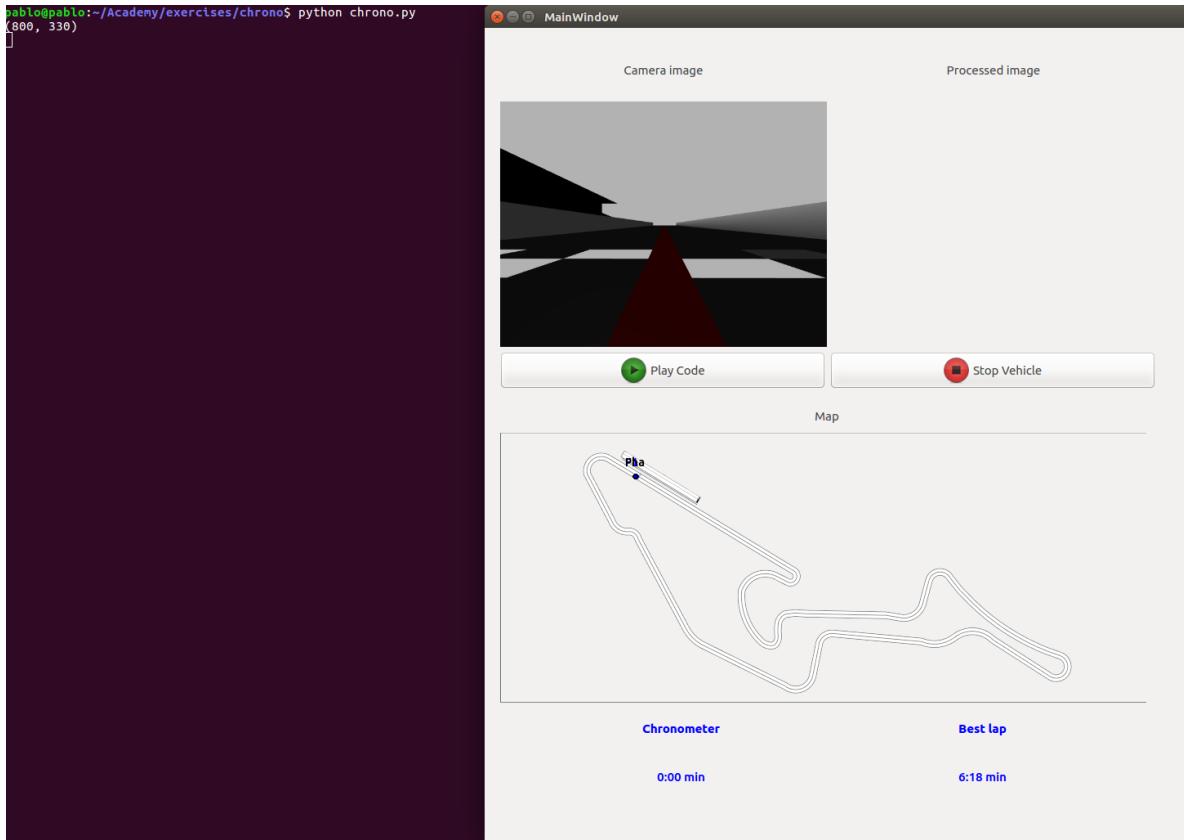


Figura 4.12: Inicialización del nodo académico y el GUI

Una vez inicializados los *drivers* de *ROS-Kinetic*, el escenario en el simulador y el nodo académico, con su GUI, se puede iniciar el algoritmo desarrollado por el alumno pulsando sobre el botón “Play Code”. De esta manera, la lógica programada podrá ser visualizada tanto en el GUI, como en el simulador.

### 4.5.2. Ejecución estática

En el caso en el que el alumno no programe el control de movimiento en su algoritmo, es posible ejecutar el código de igual manera. Esto es útil para depurar el algoritmo de procesamiento de imagen.

El alumno tiene dos opciones en este aspecto. La primera consiste en dejar el robot

## CAPÍTULO 4. CHRONO

---

inmóvil y ver el procesamiento realizado en la ventana para la imagen procesada del GUI.

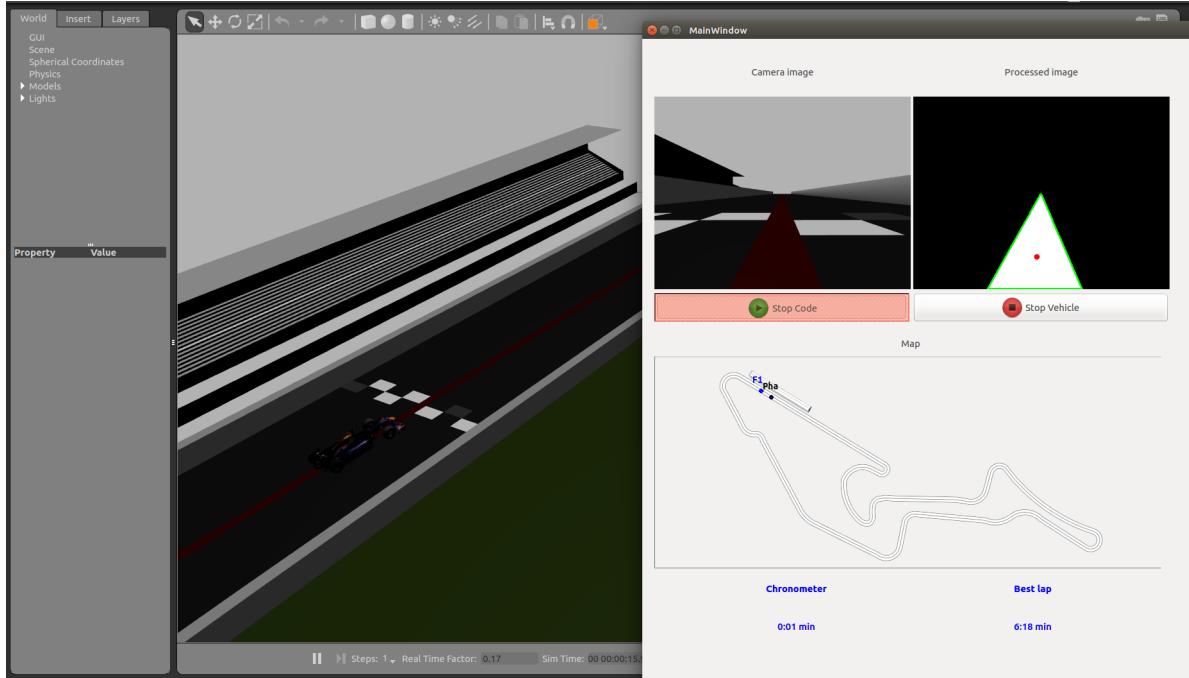


Figura 4.13: Ejecución estática de la práctica

Otra opción, una vez haya superado esa primera prueba, es seleccionar una velocidad fija para mover el robot y visualizar el filtrado en movimiento.

```
self.motors.sendV(1)
```

## CAPÍTULO 4. CHRONO

---

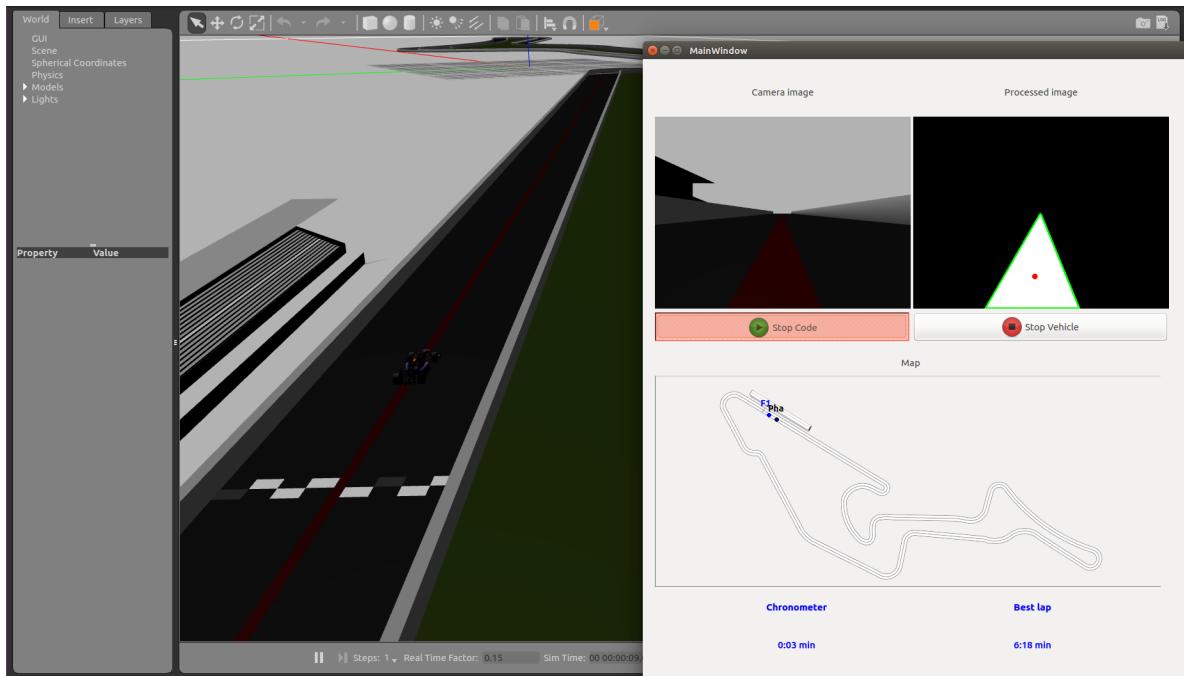


Figura 4.14: Ejecución con velocidad fija

Con esta ejecución semi-móvil de la práctica, el alumno también puede visualizar, no solo la odometría del fantasma, que se visualizará cada vez que se ejecute el algoritmo, sino la odometría del robot a programar. Con esto, el alumno se puede hacer una idea de la velocidad a la que tiene que programar el control de movimiento para ser más veloz que el récord del circuito.

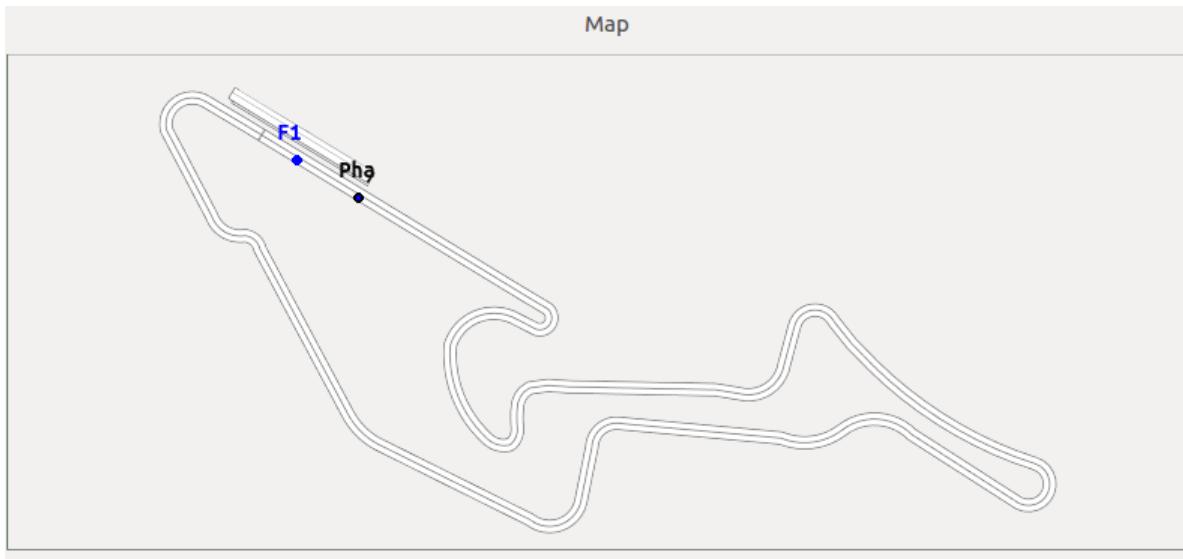


Figura 4.15: Odometría en ejecución con teleoperador

#### 4.5.3. Ejecución en movimiento

Una vez conseguido un algoritmo que procese las imágenes en movimiento con el teleoperador, el alumno puede pasar a la última fase del desarrollo de la solución, el control de movimiento del robot. Para ello deberá utilizar el procesamiento de la imagen realizado en la primera parte del desarrollo del algoritmo para poder dotar al robot de un movimiento en función del filtrado que realice de la línea roja del circuito. En este punto hay que tener especial cuidado con las curvas dado que la línea varía bruscamente y, con una velocidad elevada, la línea puede salirse del rango de captura de imagen de la cámara y, por lo tanto, el algoritmo producirá un error (a menos que se programe una solución para ese caso). Es por esto que el alumno debe ser consciente de la velocidad de refresco de la imagen de la cámara, 20 frames por segundo, y de la velocidad con la que el nodo recoge las imágenes de la cámara. Esta última función es la más limitante en cuanto a refresco en el procesamiento de la imagen, dado que, aunque utiliza una hebra para la interconexión entre los sensores y actuadores con el nodo académico. Debido a la odometría y a la sincronización, el hilo de ejecución no recoge las 20 imágenes por segundo que capta la cámara sino que dependerá de la potencia computacional de cada ordenador.

## CAPÍTULO 4. CHRONO

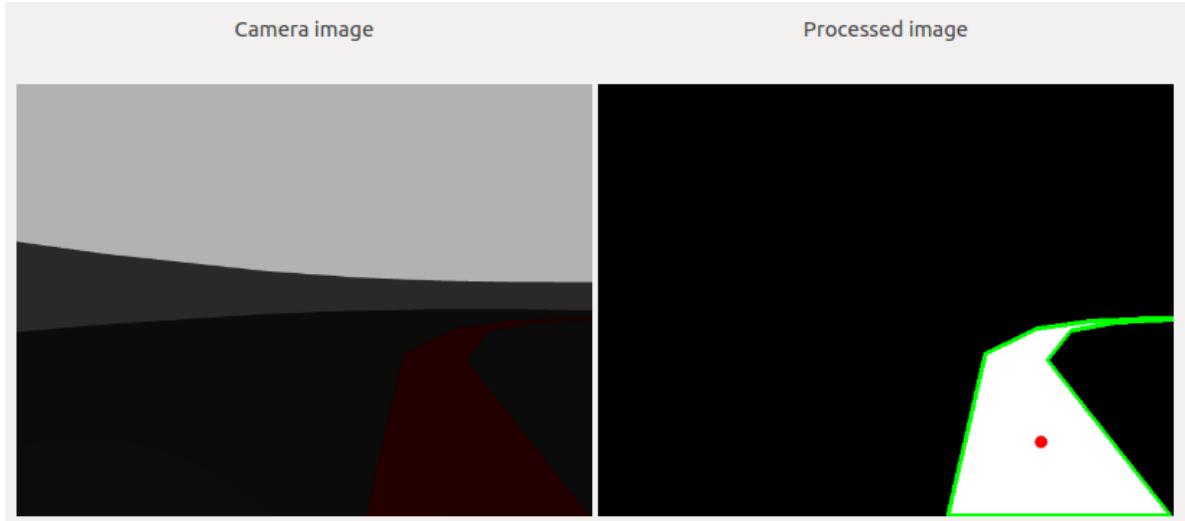


Figura 4.16: Ejemplo de procesamiento de la imagen en las curvas

Por último, una vez que se ha desarrollado el algoritmo completo, se puede ejecutar la práctica con el algoritmo.

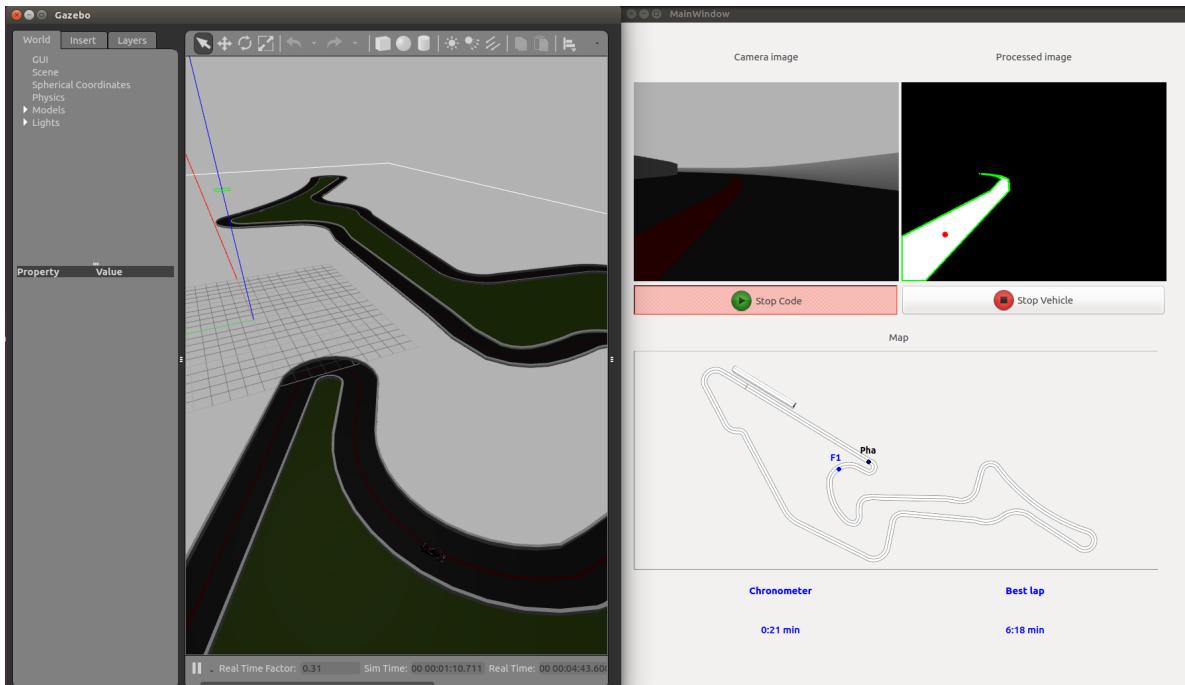


Figura 4.17: Ejecución con la solución

## 4.6. Cuadernillo académico Jupyter

Como parte paralela a la práctica incluida en *Robotics-Academy*, se ha desarrollado la misma práctica en cuadernillos de *Jupyter*. Gracias a esto, el alumno puede programar y ejecutar el código mediante la utilización del navegador web que prefiera.

Esto supone un paso importante hacia la multiplataforma del entorno docente *JdeRobot Academy*, dado que el alumno puede acceder a las prácticas desde el sistema operativo que prefiera, pues sólo necesita acceso a internet.

Para que sea posible este hecho, ha sido necesaria una reestructuración del nodo académico y de los ficheros que lo componen, además del método de desarrollo del algoritmo.

En primer lugar, el nodo académico tiene la siguiente estructura:



Figura 4.18: Estructura de la práctica en Jupyter

Como puede verse es diferente a la estructura presente en *Robotics-Academy*. Ahora se divide en los siguiente fichero y carpetas:

- images: en este directorio aparecen las imágenes contenidas en el cuadernillo.
- gazebo: en este directorio está el fichero de configuración con el escenario y con los *drivers* de *ROS-Kinetic*.
- interfaces: en este directorio se encuentran los *drivers* de *ROS-Kinetic*.
- chrono.ipynb: este fichero es el cuadernillo ejecutable en Jupyter.

Para acceder a la práctica, el alumno debe iniciar Jupyter introduciendo en la terminal el siguiente comando:

## CAPÍTULO 4. CHRONO

```
cd ~/Jupyter  
jupyter-notebook
```

Con esto se abrirá el navegador web por defecto en la carpeta local Jupyter. Una vez hecho esto, navegaremos hacia el directorio de la práctica de Jupyter “Chrono” y abriremos el fichero “Chrono.ipynb”. Tras esto se mostrará la siguiente imagen del cuadernillo:

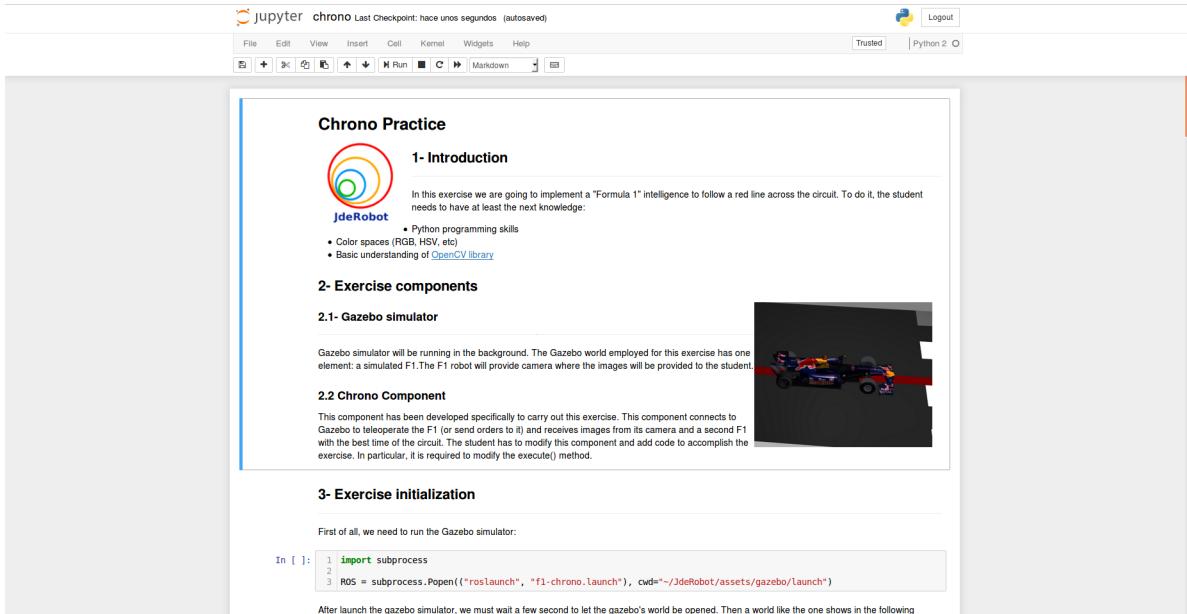


Figura 4.19: Fichero Chrono.ipynb

El alumno deberá ejecutar las celdas con código y seguir el guión mostrado. En primer lugar deberá ejecutar el fichero de configuración del mundo:

## CAPÍTULO 4. CHRONO

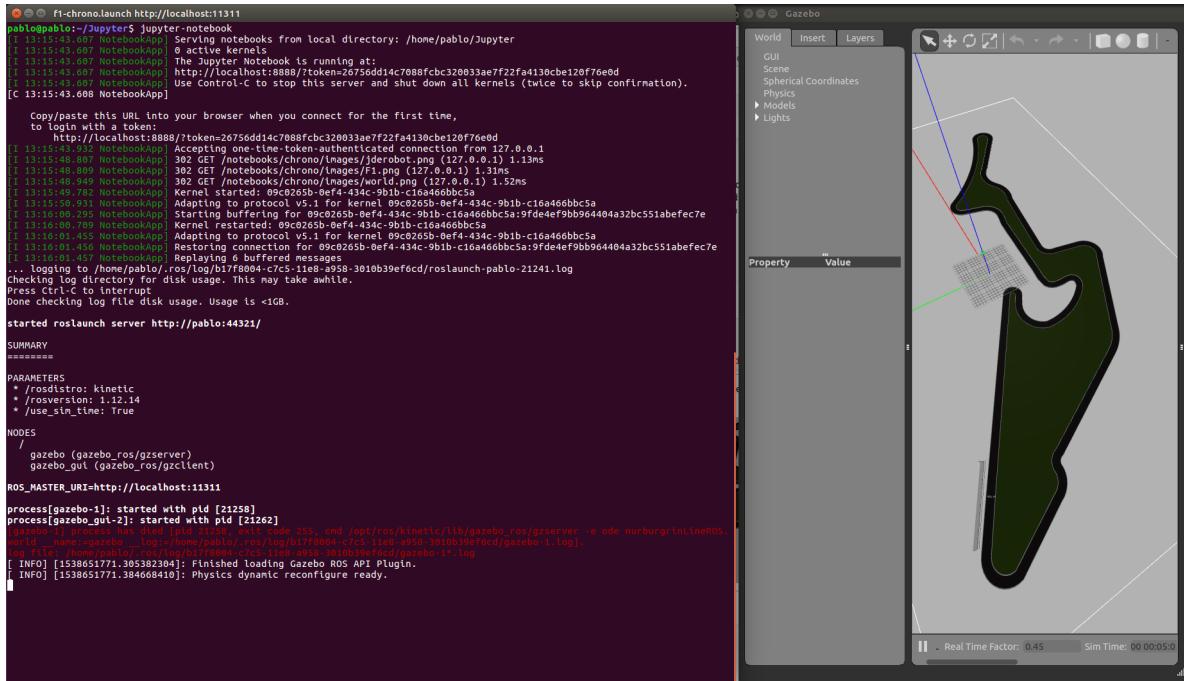


Figura 4.20: Celda con la inicialización del mundo y los drivers

Cuando se haya abierto el simulador, es necesario importar el módulo del paquete “MyAlgorithm.py” y “Chrono.py” para tener la funcionalidad proveída en el nodo académico. Para ello, se ejecutará la siguiente celda con el nodo académico de la práctica:

## CAPÍTULO 4. CHRONO

---

```
In [ 1]: #!/usr/bin/python
#-*- coding: utf-8 -*-

1 import numpy as np
2 import threading
3 import time
4 import robot
5 import cv2
6 from datetime import datetime
7 import subprocess
8 import types
9 import sys
10 import comm
11 import config
12 import os
13 import types
14 import types
15 import types
16 import types
17
18 from PyQt5.QtWidgets import QApplication
19 from Interfaces.camera import ListenerCamera
20 from Interfaces.pose3d import ListenerPose3d
21 from Interfaces.motors import PublisherMotors
22
23 time_cycle = 80
24
25 class MyAlgorithm(threading.Thread):
26
27     def __init__(self, camera, motors, pose3d):
28         self.camera = camera
29         self.motors = motors
30         self.pose3d = pose3d
31         self.imageNone = None
32         self.time = 0
33         self.cary = 0.0
34
35         self.threshold_image = np.zeros((640,360,3), np.uint8)
36         self.threshold_image_lock = threading.Lock()
37         self.stop_event = threading.Event()
38         self.kill_event = threading.Event()
39         self.algoritm()
40         self.color_image_lock = threading.Lock()
41         self.threshold_image_lock.acquire()
42
43     def getimage(self):
44         self.lock.acquire()
45         img = self.camera.getImage().data
46         self.lock.release()
47         return img
48
49     def set_color_image(self, image):
50         img = np.copy(image)
51         if len(img.shape) == 2:
52             img = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)
53         self.color_image_lock.acquire()
54         self.color_image = img
55         self.color_image_lock.release()
56
57     def get_color_image(self):
58         self.color_image_lock.acquire()
59         img = np.copy(self.color_image)
60         self.color_image_lock.release()
61
62         plt.axis('off')
63         a = plt.imshow(img)
64
65     def set_threshold_image(self, image):
66         img = np.copy(image)
67         if len(img.shape) == 2:
68             img = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)
69         self.threshold_image_lock.acquire()
70         self.threshold_image = img
71         self.threshold_image_lock.release()
72
73     def get_threshold_image(self):
74         self.threshold_image_lock.acquire()
75         img = np.copy(self.threshold_image)
76         self.threshold_image_lock.release()
77
78         plt.axis('off')
79         a = plt.imshow(img)
80
81     def getCarDirection(self):
82         self.cary = pose.getPosel().x
83         self.cary = pose.getPosel().y
84
85     def run(self):
86
87         while not self.kill_event.is_set():
88             start_time = datetime.now()
89
90             if not self.stop_event.is_set():
91                 self.algoritm()
92
93             finish_time = datetime.now()
94
95             dt = finish_time - start_time
96             ms = (dt.days * 24 * 60 * 60 + dt.seconds) * 1000 + dt.microseconds / 1000.0
97
98             if ms < time_cycle:
99                 time.sleep((time_cycle - ms) / 1000.0)
100
101         self.stop(self)
102
103     def stop(self):
104         self.stop_event.set()
105         self.motors.sendW(0)
106         self.motors.sendW(0)
107
108     def play(self):
109         if self.is_alive():
110             self.stop_event.clear()
111         else:
112             self.start()
113
114     def kill(self):
115         self.kill_event.set()
116
117     def algoritm(self):
118         # Add your code here
119         pass
120
121 class Chrono():
122
123     def __init__(self):
124         self.camera = ListenerCamera("/FIR05/camera/image_raw")
125         self.motors = PublisherMotors("/FIR05/cmd_vel", 4, 0.3)
126         self.pose3d = ListenerPose3d("/FIR05/odom")
127         self.pose3dphantom = ListenerPose3d("/FIR05_phantom/odom")
128
129         self.algoritm = types.MethodType(self.algoritm, self.motors, self.pose3d)
130
131         print "Chrono's Components initialized OK"
132
133     def play(self):
134         self.stop_event.unpause()
135         self.algoritm.play()
136         print "Chrono is running"
137
138     def pause(self):
139         self.algoritm.pause()
140         self.stop_event.pause()
141         print "Chrono has been paused"
142
143     def stop(self):
144         self.algoritm.stop()
145         print "Chrono stopped"
146
147     def setExecute(self, execute):
148         self.algoritm.algorithm = types.MethodType(execute, self.algoritm)
149         print "Code updated"
150
151     def move(self):
152         self.algoritm.move()
153
154     def get_threshold_image(self):
155         return self.algoritm.get_threshold_image()
156
157     def get_color_image(self):
158         return self.algoritm.get_color_image()
159
160     @matplotlib.inline
161     def _show(self):
162         ch = Chrono()
163         ch.play()
164
165         from IPython.display import HTML
166         # Script to hide the code
167         tag = HTML('<script>')
168         tag += 'function code_toggle() {'
169             if code_show:
170                 $(\'div.cell.code_cell.rendered.selected div.input\').hide();
171             } else {
172                 $(\'div.cell.code_cell.rendered.selected div.input\').show();
173             }
174         code_show = !code_show
175
176         $(document ).ready(code_toggle);
177         </script>'
178         To show/hide this cell's raw code input, click <a href="javascript:code_toggle()">here</a>.'')
179         display(tag)
180
181
182 Chrono's Components initialized OK
183 Chrono is running
184
185 To show/hide this cell's raw code input, click here.
```

Figura 4.21: Celda para importar el nodo académico

Cuando la ejecución imprima el mensaje “OK”. El alumno puede comenzar a programar su código en la celda especificada para ello.

```
In [ 1]: 1 def execute(self):
2     #GETTING THE IMAGES
3     image = self.getImage()
4
5     # Add your code here
6     print "Running"
7
8     #EXAMPLE OF HOW TO SEND INFORMATION TO THE ROBOT ACTUATORS
9     #self.sensor.sendV(10)
10    #self.sensor.sendW(5)
11
12    #SHOW THE FILTERED IMAGE ON THE GUI
13    #self.set_threshold_image(image)
14
15    ch.setExecute(execute)
```

Figura 4.22: Celda con el algoritmo del alumno

## CAPÍTULO 4. CHRONO

---

Para comprobar el código desarrollado, basta con ejecutar la celda donde ha programado el algoritmo. Si quiere parar la ejecución, es suficiente con pulsar el botón con el icono de “Stop” del cuadernillo.

Figura 4.23: Ejecución de la celda del algoritmo

Figura 4.24: Pausa de la ejecución del algoritmo

Puedes ver un vídeo de la ejecución de Jupyter con esta práctica.<sup>2</sup>

<sup>2</sup><https://www.youtube.com/watch?v=vTAekepTRkQ>

# Capítulo 5

## Ejercicio de seguimiento automático de carreteras con drones

En este capítulo se va a describir la segunda práctica actualizada para el elenco de prácticas de robótica de *Robotics-Academy*. Esta práctica se llama *Follow Road* y se explicar todo lo relacionado con su infraestructura, las dos plantillas *software* (nodos ROS y cuadernillos *Jupyter*), la solución desarrollada y su experimentación.

### 5.1. Enunciado

El objetivo de esta práctica, enfocado al estudiante, es el desarrollo de un algoritmo que dote de la inteligencia necesaria a un dron para que sea capaz de seguir una carretera. En este caso, se deben controlar la altitud, la velocidad y el giro del dron para poder realizar un correcto seguimiento de la carretera, por parte del dron. Esto supone un desafío para el alumno que deberá tener un control absoluto del robot en todo momento.

Para desarrollar la solución de esta práctica, el alumno debe enfrentarse a diversos problemas relacionados con la robótica. El primero de ellos es la visión, el alumno deberá realizar un procesado de imagen para filtrar los componentes que le interesen, en este caso, deberá filtrar la carretera. El segundo problema a abordar, es el control de movimiento del dron, en este caso, debe ser muy preciso porque se manejan alturas, además de giros y velocidades. El último problema viene derivado de los dos anteriores y supone realizar un movimiento controlado y dinámico del dron, para ello el alumno debe procesar las

imágenes y realizar un movimiento controlado del dron en función de la imagen procesada.

El desafío de esta práctica reside en la dificultad de mantener el dron estable en todo momento y adecuar sus movimientos al procesado de imagen que se realice. Esto supone solucionar los problemas anteriores de manera conjunta. Además, es necesario el desarrollo de un algoritmo robusto para evitar que el dron se estrelle o tenga un comportamiento anormal en casos extremos o en situaciones no previstas.

## 5.2. Infraestructura

Este ejercicio funciona en un determinado escenario de *Gazebo*, que ha habido que desarrollado y con un dron importado que ha tenido que ser modificado. El dron tiene dos cámaras y un controlador de velocidad y altitud que provee al dron de movimiento (Figura 5.1).



Figura 5.1: Infraestructura del ejercicio Follow Road

### 5.2.1. Arquitectura

La arquitectura de esta práctica es distinta a las existentes hasta ahora. Esto se debe a la utilización de paquetes ROS en el dron. Para el funcionamiento del dron con

ROS, es necesario un soporte en *C++* llamado *MAVLink*. *MAVLink* es un protocolo de comunicación para vehículos autómatas. Este protocolo proporciona los mensajes para comunicarse con el dron. Apoyado en *MAVLink*, se encuentra *MavROS*. Se trata de un nodo de *ROS* que realiza un conversión entre los *topics* que se utilizan en *ROS* para comunicarse con los robots y los mensajes de *MAVLink* permitiendo, de esta manera, la comunicación de los vehículos autónomos con *ROS*. Además, proporciona los *plugins* y *topics* finales del dron. Por encima de *MavROS*, se han desarrollado unos drivers específicos para permitir la comunicación de la infraestructura *JdeRobot* con *ROS*, de esta manera, la plataforma *JdeRobot* es integrable con *ROS Kinetic*. En cuanto al control de robot o vehículo autónomo, se ha utilizado el software libre *Px4*. Se trata de un sistema de pilotaje automático de drones especialmente orientado a pilotaje fuera de la línea de visión del usuario. Este *software* aporta la infraestructura básica del dron.

#### 5.2.1.1. Modelo dron “Iris”

Este modelo de dron ha sido escogido para la realización de esta práctica. Consta de un cuerpo principal, en el que va instalado el *hardware*, cuatro rotores, una antena y dos cámaras. La inteligencia del dron viene dada por el sistema de pilotaje automático *Px4*.



Figura 5.2: Ilustraciones del dron “Iris”

#### 5.2.1.2. Px4

El piloto automático *PX4*<sup>1</sup> es un sistema de piloto automático de código abierto orientado a aeronaves autónomas de bajo coste. Tanto el *hardware* como el *software* es

---

<sup>1</sup><https://px4.io/>

*open-source* y accesible gratuitamente bajo una licencia *BSD*. Esta plataforma, derivó del proyecto *PIXHWAK*<sup>2</sup> que se centró, específicamente, en control de vuelo basado en visión. El *software* incluido bajo *Px4* incluye:

- QGroundControl<sup>3</sup> y MAVLink para las comunicaciones con el dron.
- Mapas aéreos en 2D y 3D con soporte de Google Earth<sup>4</sup>.
- Puntos de control *Drag-and-Drop*.

Esta infraestructura ha sido escogida por su compatibilidad de comunicaciones mediante el protocolo *MAVLink*. Además, este sistema de auto pilotaje tiene distintas características que lo convierten en la opción escogida para soporte de *hardware* y *software* de *JdeRobot*:

- Arquitectura modular y extensible.
- Código base simple para todos los vehículos.
- Desarrollado para conducción autónoma.
- Gran conjunto de periféricos compatibles.
- Modos de vuelo flexibles y potentes.
- Dispone de herramientas complementarias de desarrollo.
- Probado en modelos reales.
- Código *open-source*.

Para esta práctica se ha utilizado la comunicación con el dron bajo los mensajes *MAVLink*. De esta manera, *Px4*, realiza la conversión de mensajes *MAVLink* a comandos para el dron y viceversa.

---

<sup>2</sup><http://pixhawk.org/>

<sup>3</sup><http://qgroundcontrol.com/>

<sup>4</sup><https://www.google.com/earth/>

### 5.2.1.3. MAVLink

El *software* libre *MAVLink*<sup>5</sup> (Micro Air Vehicle Link), es un protocolo de mensajes de comunicación en vehículos autónomos. Se ha diseñado como una librería de cálculo de referencia de encabezado simple. Fue lanzado a principios del año 2009 por Lorenz Meier bajo una licencia *LGPL*<sup>6</sup>.

Los mensajes están definidos con ficheros *XML*. Cada fichero *XML* define un conjunto de mensajes admitidos por un sistema *MAVLink* particular o dialecto.

### 5.2.1.4. MavROS

El nodo de *ROS*, *MavROS*<sup>7</sup>, es un paquete que proporciona un controlador de comunicación para varios pilotos automáticos con el protocolo de comunicación *MAVLink* y un puente *UDP MAVLink* para estaciones de control terrestre.

*MavROS* esta formado por cuatro componentes:

- Nodos *MavROS*. Estos nodos están formados por *topics* de suscripción, *topics* de publicación y parámetros de configuración.
- Puente *GCS*. Es la conexión de *MavROS* con el protocolo de mensajes *MAVLink*.
- Lanzador de eventos. Comprueba el estado de los eventos producidos por los *topics*.
- *Plugins*. *MavROS* proporciona un gran número de *topics* a los que poder suscribirte para recibir información del estado del dron o para publicar comandos de control del dron. A este conjunto de *plugins* se accede gracias a los *topics*. Cada *plugin* contiene distintos *topics* o tipos de mensajes distintos con los que enviar o recibir información del *plugin*. Una vez recibida esta información, el *plugin* realiza una transformación de mensajes *MAVLink* a *topics* de *ROS* y viceversa. Los *topics* ofrecidos por los *plugins* de *MavROS* son más de cien, pero para el control del dron en *JdeRobot*, se han utilizado los siguientes:
  - /mavros/cmd/arm: para armar el dron.

---

<sup>5</sup><https://mavlink.io/en/>

<sup>6</sup>[https://en.wikipedia.org/wiki/GNU\\_Lesser\\_General\\_Public\\_License](https://en.wikipedia.org/wiki/GNU_Lesser_General_Public_License)

<sup>7</sup><http://wiki.ros.org/mavros>

<sup>8</sup><http://ardupilot.org/dev/docs/ros.html>

- /mavros/global\_position/global: para conocer la posición GPS del dron y realizar el despegue.
- /mavros/cmd/takeoff: para despegar el dron.
- /mavros/cmd/land: para aterrizar el dron.
- /mavros/set/mode: para cambiar el dron entre sus distintos modos.
- Auto\_RTL: Modo automático. Se utiliza para despegar y aterrizar.
- Offboard: Modo manual. Se utiliza para controlar el dron.
- /mavros/setpoint\_raw/local: para controlar el movimiento del dron. Permite enviar información de posición, altura, velocidad, aceleración y giro.
- /mavros/local\_position/odom: para recibir la odometría del dron.
- /iris\_fpv\_cam/cam\_XXXXX/image\_raw: aunque no se trata de un *topic* proporcionado por *MavROS*, utiliza *ROS* para recibir información de las cámaras, por lo que es integrable con *MavROS*.

La relación entre estos tres componentes puede verse en la figura 5.3.

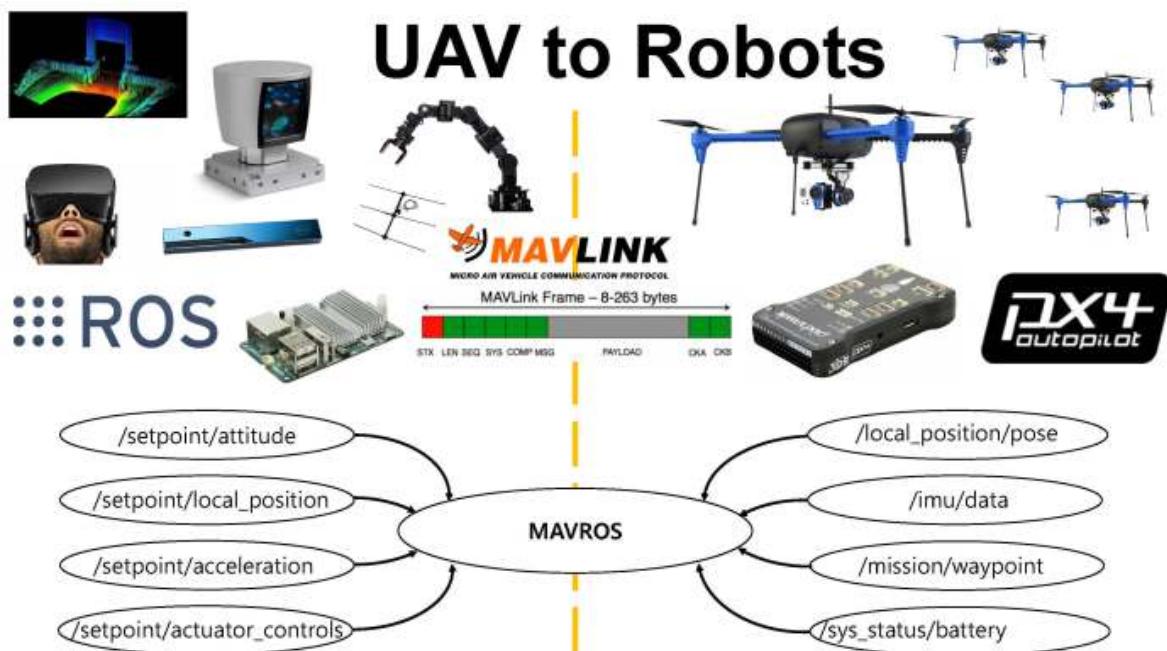


Figura 5.3: Relación entre MavROS, MAVLink y Px4

### 5.2.1.5. Drivers

Para realizar las conexiones a los *topics* ofrecidos por *MavROS*, se han desarrollado unos drivers, específicos para la infraestructura de *JdeRobot*. Para ello se han desarrollado un total de seis ficheros con código en *Python* que dotan de la infraestructura necesaria, al nodo académico, para comunicarse con el dron.

- En el fichero `__init__.py`, se establecen las cabeceras de las funciones principales de interconexión y las conexiones de los *topics*.
- En el fichero `camera.py`, se han desarrollado las funciones necesarias para recoger las imágenes captadas por las cámaras integradas en el dron.
- En el fichero `cmdvel.py`, se incorporan las funciones necesarias para enviar los comandos de velocidad al plugin de *MavROS*.
- En el fichero `extra.py`, se han desarrollado las funciones para controlar el despegue y aterrizaje del dron.
- En el fichero `pose3d.py`, están las funciones para recoger la información sobre la odometría del dron.
- En el fichero `threadPublisher.py`, está la función de creación de la hebra para las comunicaciones de las hebras de publicación de *topics*.

### 5.2.1.6. Sensores y actuadores

Una vez descritos los componentes de comunicación con el dron, se va a explicar los componentes que forman el modelo del dron. Estos componentes dotan al dron de la estructura y movilidad necesaria para su funcionamiento.

#### 5.2.1.6.1. Cámara

La cámara proporciona al dron visión del escenario en el que se encuentra. Es necesaria para dotar al dron de una interfaz sobre la que basar su inteligencia. En esta práctica son necesarias dos cámaras, una frontal y otra ventral para tener una visión del entorno tanto en la parte delantera del dron, para evitar obstáculos, como inferior para poder visualizar correctamente la carretera.

El *plugin* de la cámara es proporcionado por *ROS*. Este *plugin* proporciona el código necesario para conectar una cámara mediante USB con una velocidad de refresco de 30 imágenes por segundo con una longitud focal de 277 milímetros. El tamaño de las imágenes obtenidas periódicamente es de 360x240 píxeles y están en formato crudo RGB. Con estas características, el dron puede recoger imágenes a una velocidad lo suficientemente rápida y a una distancia apropiada, para procesar las imágenes y conseguir evitar los obstáculos con suficiente antelación.

Las imágenes captadas por la cámara son recogidas por el nodo académico gracias a su interfaz gráfica y mediante un API simple. De esta manera se pueden visualizar las imágenes de las cámaras en la interfaz gráfica del nodo académico, así como su procesado.

#### 5.2.1.6.2. Odometría

Con este sensor, el dron es capaz de conocer su posición. Este sistema se implementa gracias al GPS que incorpora el dron y le permite conocer su latitud y longitud en el escenario en el que se encuentre. *MavROS* incorpora *topics* para conocer esta posición. El *topic* utilizado para estimar esta posición ofrece la posición del dron en coordenadas relativas al lugar de despegue del dron. Esta funcionalidad viene soportada en el *plugin setpoint\_position*.

#### 5.2.1.6.3. Rotores

El dron está formado por cuatro rotores que permite su movimiento. Para el control de estos rotores, se incorpora un *plugin* de control de rotor por cada uno de los rotores del dron. Este *plugin* es proporcionado por *Px4* y es de *ROS*. Sin embargo, el *plugin* que se utiliza para mover el dron es proporcionado por *MavROS* bajo el nombre *setpoint\_raw* al que se accede con el *topic /mavros/setpoint\_raw/local*.

#### 5.2.1.7. Escenario de Gazebo

Para esta práctica se ha actualizado el escenario inicial del que disponía la práctica y se han arreglado problemas que presentaba relacionados con la visualización del suelo, un nuevo modelo de casa y la incorporación del modelo del dron nuevo. El nombre de este escenario es “*road\_drone\_textures\_ROS.world*”. El aspecto del escenario puede verse en la figura 5.4.

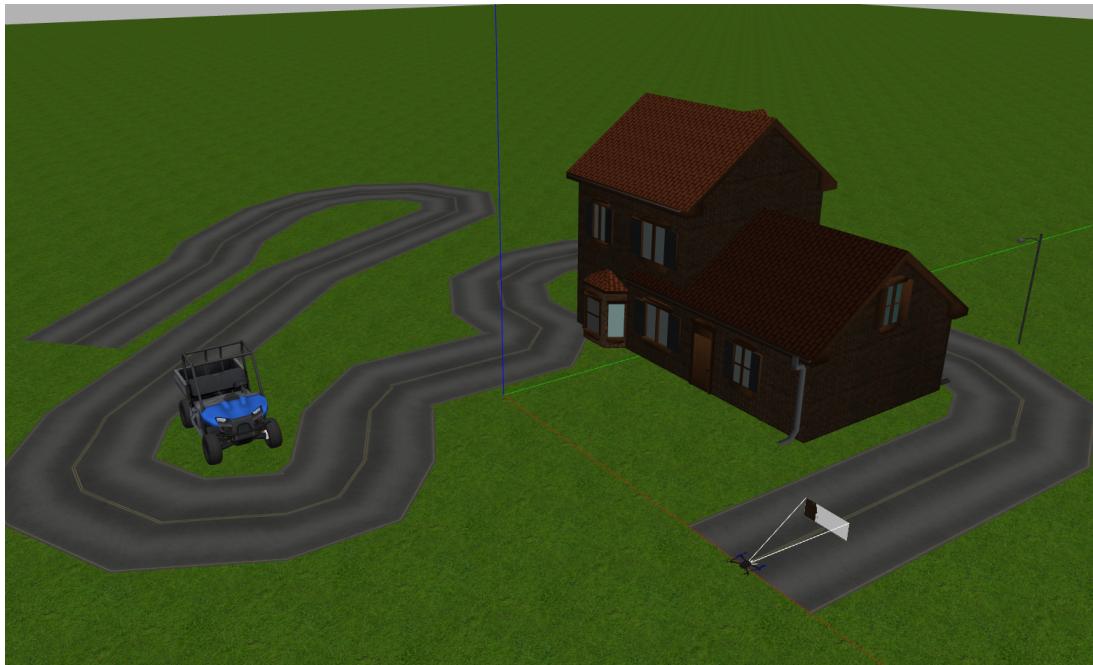


Figura 5.4: Escenario de Follow Road

### 5.2.1.8.

Ficheros de configuración Para la incorporación del modelo del circuito y del robot, es necesario la creación de un modelo de configuración que importe en *Gazebo* los elementos de los que consta el escenario y su localización. Este fichero tiene la extensión *.world* y *Gazebo* es capaz de leerlo y mostrar el escenario al iniciarse. El código del fichero es el siguiente:

```
<?xml version="1.0" ?>
<sdf version="1.5">
  <world name="default">
    <scene>
      <grid>false</grid>
    </scene>

    <include>
      <uri>model://sun</uri>
    </include>
    <include>
      <uri>model://grass_plane</uri>
    </include>
```

```
<include>
<uri>model://house_4</uri>
<pose>1 6.43 0 0 0 0</pose>
</include>
<include>
<uri>model://polaris_ranger_ev</uri>
<pose>-1.48 -6 0.1 0 0 0</pose>
<static>true</static>
</include>
<include>
<uri>model://lamp_post</uri>
<pose>5 13 0 0 0 0</pose>
</include>
<include>
<uri>model://lamp_post</uri>
<pose>-4 13 0 0 0 0</pose>
</include>
<include>
<uri>model://iris_fpvcam</uri>
<pose>8 0 0.3 0 0 1.57</pose>
</include>
<road name="my_road">
<width>3</width>

<point>8 0 0.01</point>
<point>8 3 0.01</point>
<point>8 8 0.01</point>
<point>7 10 0.01</point>
<point>5 11 0.01</point>
<point>-5 11 0.01</point>
<point>-7 10 0.01</point>
<point>-8 8 0.01</point>
<point>-8 6 0.01</point>
<point>-7 4 0.01</point>
<point>-5 3 0.01</point>
<point>-3 3 0.01</point>
<point>-2 2 0.01</point>
<point>-2 -2 0.01</point>
<point>-1 -3 0.01</point>
```

```
<point>1 -4 0.01</point>
<point>2 -5 0.01</point>

<point>2 -6 0.01</point>
<point>1 -8 0.01</point>
<point>-1 -9 0.01</point>
<point>-2 -9 0.01</point>
<point>-4 -8 0.01</point>

<point>-11 -1 0.01</point>
<point>-14 5 0.01</point>

<point>-15 7 0.01</point>
<point>-17 8 0.01</point>
<point>-19 7 0.01</point>

<point>-20 5 0.01</point>
<point>-20 3 0.01</point>
<point>-19 1 0.01</point>
<point>-17 -1 0.01</point>

<point>-16 -2 0.01</point>
<point>-14 -3 0.01</point>
<point>-9 -8 0.01</point>
</road>
</world>
</sdf>
```

Además de este fichero de configuración, es necesario un fichero complementario que importe los *plugins* y *drivers* de ROS-Kinetic. Este tipo de fichero tienen la extensión *.launch*. En este fichero se pasan a *Gazebo* argumentos como el nombre del fichero de configuración con el escenario, establecer el tiempo que se va a utilizar en el escenario, la posible implementación de un GUI y otras opciones de depuración. El fichero es el siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<launch>
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="road_drone_textures_ROS.world"/>
```

```
<arg name="paused" value="false"/>
<arg name="use_sim_time" value="true"/>
<arg name="gui" value="true"/>
<arg name="headless" value="false"/>
<arg name="debug" value="false"/>
<arg name="verbose" default="true"/>
</include>

<arg name="fcu_url" default="udp://:14540@127.0.0.1:14540" />
<arg name="gcs_url" default="" />
<arg name="tgt_system" default="1" />
<arg name="tgt_component" default="1" />
<arg name="log_output" default="screen" />
<arg name="fcu_protocol" default="v2.0" />
<arg name="respawn_mavros" default="false" />

<include file="$(find mavros)/launch/node.launch">
<arg name="pluginlists_yaml" value="$(find mavros)/launch/px4_pluginlists.yaml" />
<arg name="config_yaml" value="$(find mavros)/launch/px4_config.yaml" />

<arg name="fcu_url" value="$(arg fcu_url)" />
<arg name="gcs_url" value="$(arg gcs_url)" />
<arg name="tgt_system" value="$(arg tgt_system)" />
<arg name="tgt_component" value="$(arg tgt_component)" />
<arg name="log_output" value="$(arg log_output)" />
<arg name="fcu_protocol" value="$(arg fcu_protocol)" />
<arg name="respawn_mavros" default="$(arg respawn_mavros)" />
</include>

<node pkg="mavros" type="px4.sh" name="px4" output="screen"/>
</launch>
```

### 5.3. Plantilla de nodo ROS

Una vez explicada la infraestructura de la práctica, en esta sección se va a explicar la plantilla de nodo ROS empleada. Esta plantilla supone una manera de ejecutar las prácticas de *Robotics-Academy*. La arquitectura de la plantilla puede observarse en la

figura 5.5.



Figura 5.5: Plantilla de nodo ROS del ejercicio Follow\_Road

### 5.3.1. Arquitectura software

Esta práctica tiene tres hilos de ejecución para aliviar la carga computacional de la práctica. De esta manera se aumenta la velocidad a la que puede trabajar el simulador *Gazebo*.

- Hilo de percepción: este hilo se encarga de la actualización de los datos de los sensores del robot. Este hilo se comunica con *Gazebo* para recoger datos de la odometría y la cámara y entregárselos al nodo ROS.
- Hilo de la interfaz gráfica del usuario (GUI): este hilo se encarga del refresco del GUI de la práctica. En esta práctica tiene una carga computacional elevada dado que se encarga de refrescar las imágenes obtenidas por las cámaras y el procesamiento visual realizado el alumno.
- Hilo de control: este hilo se encarga de enviar la información del nodo ROS al dron. Se conecta con los *topics*, proporcionados por *MavROS*, para enviar las órdenes de movimiento.

### 5.3.2. Interfaz de sensores y actuadores

Tanto para recoger los datos de los sensores como para enviar los datos de los actuadores, el nodo ROS proporciona un HAL-API de interconexión con los *topics*. De esta manera, es sobre esta interfaz sobre la que el alumno debe apoyarse para desarrollar su algoritmo, dejando los detalles de más bajo nivel transparentes para el mismo. EL HAL-API proporcionado por este nodo ROS es el siguiente:

- self.drone.getPose3d(): con esta función se obtiene la odometría del dron.
- self.drone.getImage(): con esta función se obtienen las imágenes captadas por la cámara.
- self.drone.sendCMDVel(): con esta función se envía las velocidades y el giro del dron.

### 5.3.3. Interfaz gráfica

La interfaz gráfica del usuario (GUI), se utiliza para representar información relacionada con los sensores del robot. Además, permite teleoperar el robot y lanzar/detener la ejecución del algoritmo programado. Esto es muy útil para la depuración del algoritmo del estudiante ya que permite la visualización de las publicaciones que realiza el algoritmo al robot y comprobar su movimiento.

Esta GUI (Figura 5.6) está formada por tres conjuntos de *widgets*, uno para el visionado de las imágenes, otro para el control del dron y otro para el teleoperador.

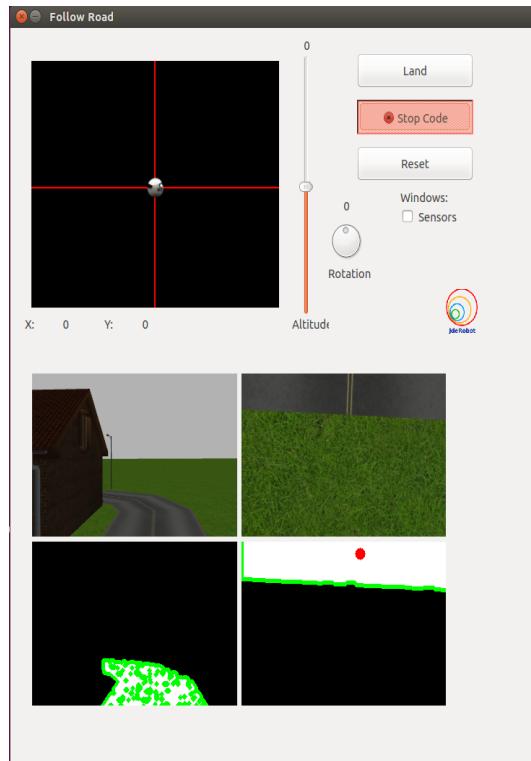


Figura 5.6: Interfaz Gráfica Folow\_Road

El *widget* del teleoperador se utiliza para controlar el dron una vez ha despegado. De esta manera se puede comprobar que las conexiones del dron están correctamente realizadas.

El conjunto *widgets* destinados al control del dron se dividen en tres subconjuntos:

- Control de altitud y rotación.
- widgets de estabilización.
- Botón de despegue/atterrizaje, botón de ejecución y parada del algoritmo de solución y botón de reinicio del puntero del teleoperador.

Con estos *widgets* se puede controlar el dron de manera manual o mediante la programación de la inteligencia del dron en el fichero de solución.

El *widget* de visionado del dron incorpora cuatro pantallas. Las pantallas superiores reflejan las imágenes recogidas por las cámaras del dron y las pantallas inferiores muestran el procesado de las imágenes realizado en el algoritmo de solución del alumno. Este *widget*

es muy útil para depurar el código del alumno ya que puede verse, fácilmente, el procesado que se ha realizado de la imagen.

El último conjunto de *widgets* lo forman los botones de control del algoritmo. Con ellos puedes ejecutar y detener el algoritmo desarrollado, despegar y aterrizar el dron y reiniciar la posición del teleoperador.

## 5.4. Configuración de los ficheros

Una vez explicado toda la infraestructura circundante al nodo académico de la práctica, se va a explicar la infraestructura del propio nodo (Figura 5.5). Está formado por:

- Fichero principal.
- Fichero de solución
- GUI. Directorio con todos los ficheros para el funcionamiento del GUI.

El fichero principal es el código que va a ejecutarse para lanzar el nodo académico de la práctica. En él se especifican las conexiones con los *topics* que se van a establecer, así como la inicialización del GUI.

En el fichero de solución, hay un conjunto de funciones que aportan la funcionalidad de la práctica, como comenzar, parar, recoger imagen, así como las que desarrolle el alumno y el propio código con la solución de referencia.

## 5.5. Solución de referencia

Para esta práctica se han desarrollado dos soluciones de referencia. La primera solución se ha desarrollado para la práctica inicial que no contaba con soporte para *ROS* y se realizaba con *drivers ICE*. La segunda solución desarrollada es compatible con *ROS*. Ambas soluciones han sido desarrolladas íntegramente.

### 5.5.1. Procesamiento de imagen

El procesamiento de las imágenes programado se ha realizado en función del contorno del filtrado. Para ello se comienza con la recogida de las imágenes captadas por la cámara:

```
input_imageV = self.drone.getImageVentral().data  
input_imageF = self.drone.getImageFrontal().data
```

Con esta instrucción, es posible visualizar las imágenes en el GUI.

Tras obtener las imágenes frontal y ventral de la cámara, hay que transformarla a una imagen HSV<sup>9</sup> para poder seleccionar el color de la carretera:

```
image_HSV_V = cv2.cvtColor(input_imageV, cv2.COLOR_RGB2HSV)  
image_HSV_F = cv2.cvtColor(input_imageF, cv2.COLOR_RGB2HSV)
```

Tras obtener la imagen HSV, podemos seleccionar el rango de valores que componen color de la carretera como un array con la intensidad mínima y máxima del color.

```
value_min_HSV = np.array([20, 0, 0])  
value_max_HSV = np.array([100, 130, 130])
```

Con esos valores realizamos un filtrado de la imagen para obtener la carretera, exclusivamente:

```
image_HSV_filtered_V = cv2.inRange(image_HSV_V, value_min_HSV, value_max_HSV)  
image_HSV_filtered_F = cv2.inRange(image_HSV_F, value_min_HSV, value_max_HSV)
```

Para reducir el ruido que contiene la imagen sobre los colores filtrados es necesario aplicar técnicas de reducción de ruido. Para ello hemos realizado una apertura y un cierre de la imagen:

```
opening_V = cv2.morphologyEx(image_HSV_filtered_V, cv2.MORPH_OPEN, np.ones((5,5),np.uint8))  
closing_V = cv2.morphologyEx(opening_V, cv2.MORPH_CLOSE, np.ones((10,10),np.uint8))
```

Para el caso de la imagen frontal, hemos realizado una técnica de apertura, solamente, con lo que se obtendrá más ruido que con la combinación anterior:

```
opening_F = cv2.morphologyEx(image_HSV_filtered_F, cv2.MORPH_OPEN, np.ones((5,5),np.uint8))
```

<sup>9</sup>[https://es.wikipedia.org/wiki/Modelo\\_de\\_color\\_HSV](https://es.wikipedia.org/wiki/Modelo_de_color_HSV)

Una vez tenemos el filtro, lo utilizamos como una máscara para realizar el filtrado de la imagen y obtener la imagen filtrada en formato binario, con el rango de colores que pasan el filtro en blanco:

```
image_HSV_filtered_Mask_V = np.dstack((closing_V, closing_V, closing_V))
image_HSV_filtered_Mask_F = np.dstack((opening_F, opening_F, opening_F))
```

Una vez tenemos el filtro, lo utilizamos como una máscara para realizar el filtrado de la imagen y obtener la imagen filtrada en formato binario, con el rango de colores que pasan el filtro en blanco:

```
imgray_V = cv2.cvtColor(image_HSV_filtered_Mask_V, cv2.COLOR_BGR2GRAY)
ret_V, thresh_V = cv2.threshold(imgray_V, 127, 255, 0)
_, contours_V, hierarchy_V = cv2.findContours(thresh_V, cv2.RETR_TREE, cv2.
    CHAIN_APPROX_SIMPLE)
cv2.drawContours(image_HSV_filtered_Mask_V, contours_V, -1, (0,255,0), 3)

imgray_F = cv2.cvtColor(image_HSV_filtered_Mask_F, cv2.COLOR_BGR2GRAY)
ret_F, thresh_F = cv2.threshold(imgray_F, 127, 255, 0)
_, contours_F, hierarchy_F = cv2.findContours(thresh_F, cv2.RETR_TREE, cv2.
    CHAIN_APPROX_SIMPLE)
cv2.drawContours(image_HSV_filtered_Mask_F, contours_F, -1, (0,255,0), 3)
```

Tras dibujar el contorno de la sección, es necesaria una comprobación del filtrado de la imagen para evitar errores en la ejecución. El siguiente algoritmo comprueba si se ha recogido alguna imagen, si hay alguna zona que ha pasado el filtro y se hay más de una sección filtrada (para el caso en que se detecten varias carreteras):

```
area = []
for pic, contour in enumerate(contours_V):
    area.append(cv2.contourArea(contour))

if len(area) > 1:
    if area[0] < area[1]:
        M = cv2.moments(contours_V[1])
    else:
        M = cv2.moments(contours_V[0])

else:
    try:
        M = cv2.moments(contours_V[0])
```

```
except IndexError:  
    self.drone.sendCMDVel(0,0.3,0,0)  
    M = cv2.moments(0)
```

Si la comprobación es exitosa, se extraen las coordenadas del centro de la sección filtrada:

```
if int(M['m00']) != 0:  
    cx = int(M['m10']/M['m00'])  
    cy = int(M['m01']/M['m00'])
```

Se dibuja un punto rojo para saber el centro de la sección filtrada, punto el cual seguirá el dron:

```
cv2.circle (image_HSV_filtered_Mask_V, (cx, cy), 7, np.array([255, 0, 0]), -1)
```

Para visualizar en el GUI el procesado de las imágenes se utilizan las siguientes instrucciones:

```
self.setImageFilteredVentral(image_HSV_filtered_Mask_V)  
self.setImageFilteredFrontal(image_HSV_filtered_Mask_F)
```

### 5.5.2. Control de movimiento

Una vez procesada la imagen, es posible controlar el dron con el punto del centro de la sección obtenido. De esta manera, el dron se limitará a seguir dicho punto, adecuando sus movimientos hacia el mismo.

```
if cy > 120:  
    self.drone.sendCMDVel(0,0.3,0,0.2)  
    print("Turning")  
elif cx < 20:  
    print("Detected two roads")  
    self.drone.sendCMDVel(0,0.3,0.1,0.0)  
else:  
    self.drone.sendCMDVel(0,0.3,0,0.0)  
  
print("cx: " + str(cx) + " cy: " + str(cy))  
self.yaw = int(cx)
```

## 5.6. Experimentación

La optimización de los algoritmos anteriores ha sido posible gracias a la realización de diversos experimentos. Estos experimentos han hecho salir a la luz errores en el algoritmo desarrollado que han sido subsanados. Además, se han realizado experimentos globales donde se ha testeado la práctica en su totalidad, nodo académico, infraestructura de la práctica y solución desarrollada.

### 5.6.1. Ejecución de la práctica

Se ha preparado un documento *README.md*, incluido en la infraestructura de la práctica, que sirve de guía al alumno a la hora de ejecutar la práctica. En él se incluye información acerca de su ejecución, la API de los sensores y actuadores de ROS e, incluso, un vídeo demostrativo con una ejecución<sup>10</sup>.

Para ejecutar la práctica, es necesario lanzar en una terminal el fichero de configuración de ROS, llamado *follow\_road.launch*, descrito en la sección 5.2.1.8. Para lanzar el fichero hay que ejecutar el siguiente comando:

```
roslaunch follow_road.launch
```

Una vez lanzado el comando en la terminal, se abrirá el simulador *Gazebo* con el escenario del circuito (Figura 4.3).

---

<sup>10</sup><https://www.youtube.com/watch?v=76QNSUGXFT8>

## CAPÍTULO 5. FOLLOW ROAD

---

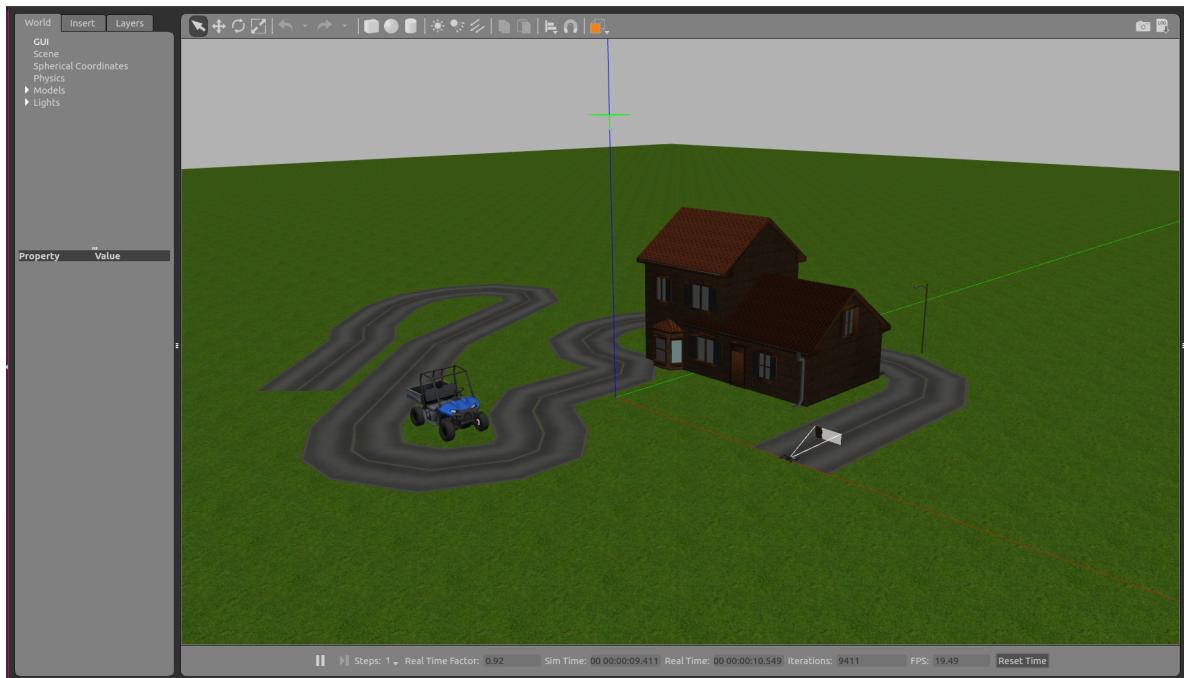


Figura 5.7: Inicialización ROS y Gazebo

Para iniciar el componente académico, será necesario ejecutar otro comando en una terminal distinta:

```
cd ~/Academy/exercises/follow\_road  
python2 follow\_road.py
```

Una vez ejecutado el comando, el componente académico enlazará los sensores y actuadores proporcionados por *ROS-Kinetic* mediante el fichero de configuración lanzado previamente a la variable *self.drone* que contiene las variables:

- *self.cameraVentral*
- *self.cameraFrontal*
- *self.pose3d*
- *self.cmdvel*
- *self.extra*

Con la variable *self.drone*, el nodo ROS se comunica con los *drivers* de *ROS-Kinetic*. Además de realizar la conexión con los sensores y actuadores, al ejecutar la instrucción, nos

## CAPÍTULO 5. FOLLOW ROAD

---

aparecerá la interfaz gráfica de usuario (GUI) en la que se podrá visualizar las imágenes recogidas por la cámara, los botones de control y el teleoperador (Figura 5.8).

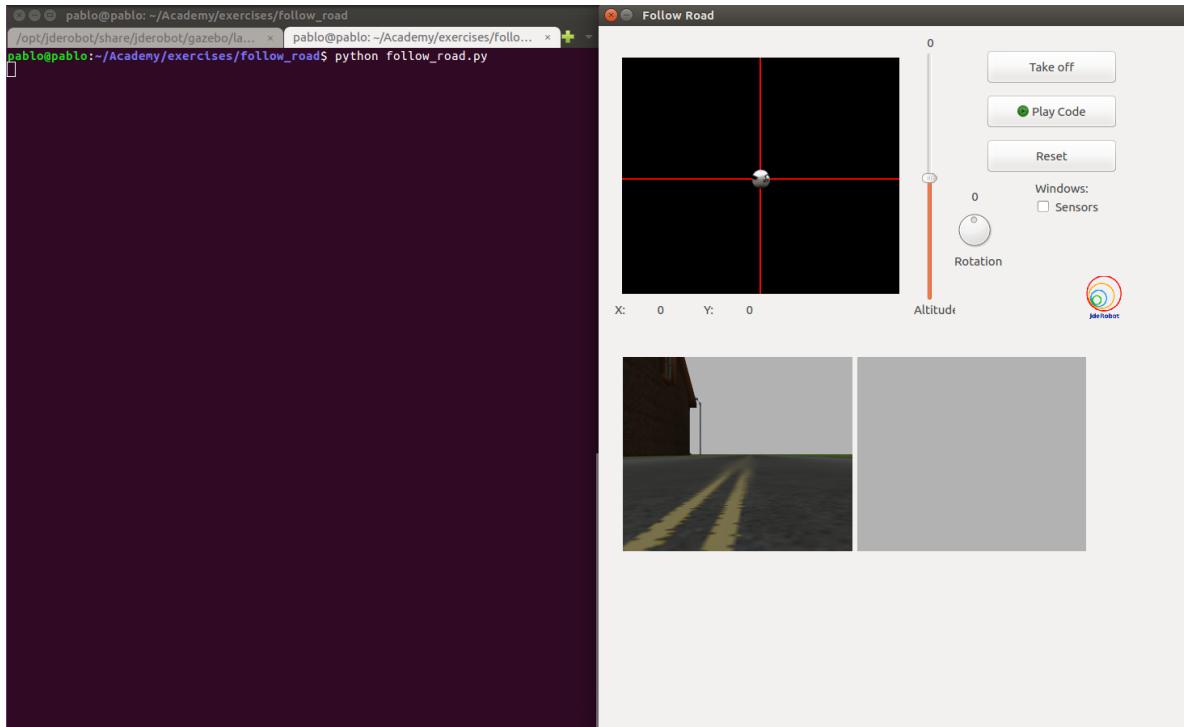


Figura 5.8: Inicialización del escenario y el GUI

Una vez inicializados los *drivers de ROS-Kinetic*, el escenario en el simulador y el nodo académico, con su GUI, se puede iniciar el algoritmo desarrollado por el alumno. Para ello, es necesario despegar el dron con el botón “TakeOff” (Figura 5.9).

## CAPÍTULO 5. FOLLOW ROAD

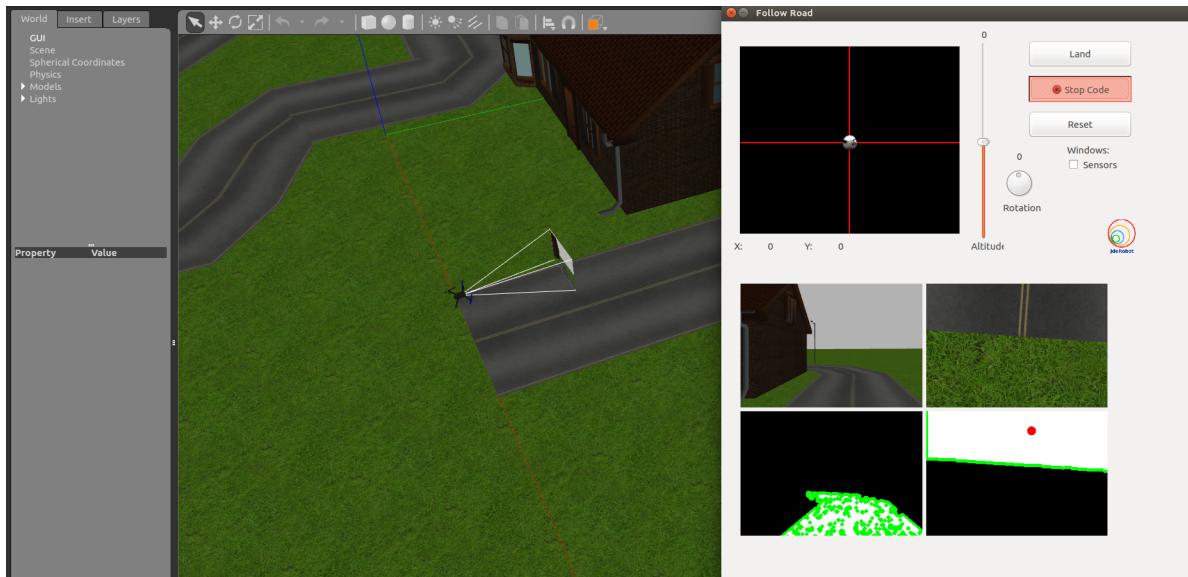


Figura 5.9: Ejecución de la práctica

Tras el despegue del dron, se puede lanzar el algoritmo programado con el botón “Play Code” (Figuras 5.10 y 5.11).

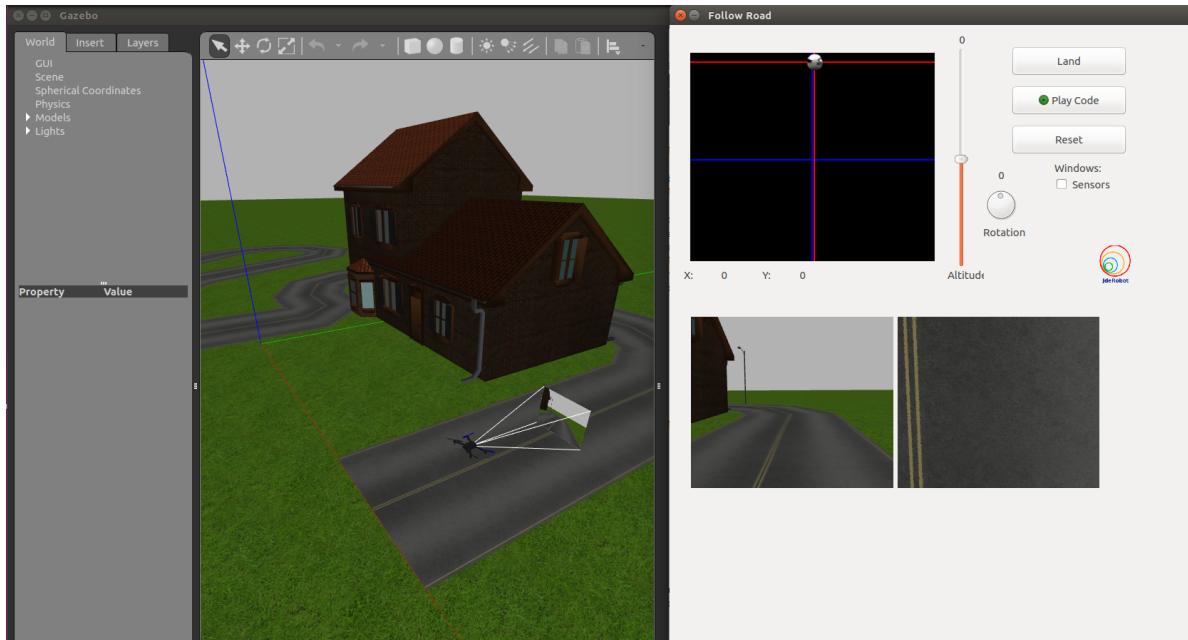


Figura 5.10: Ejecución con teleoperador

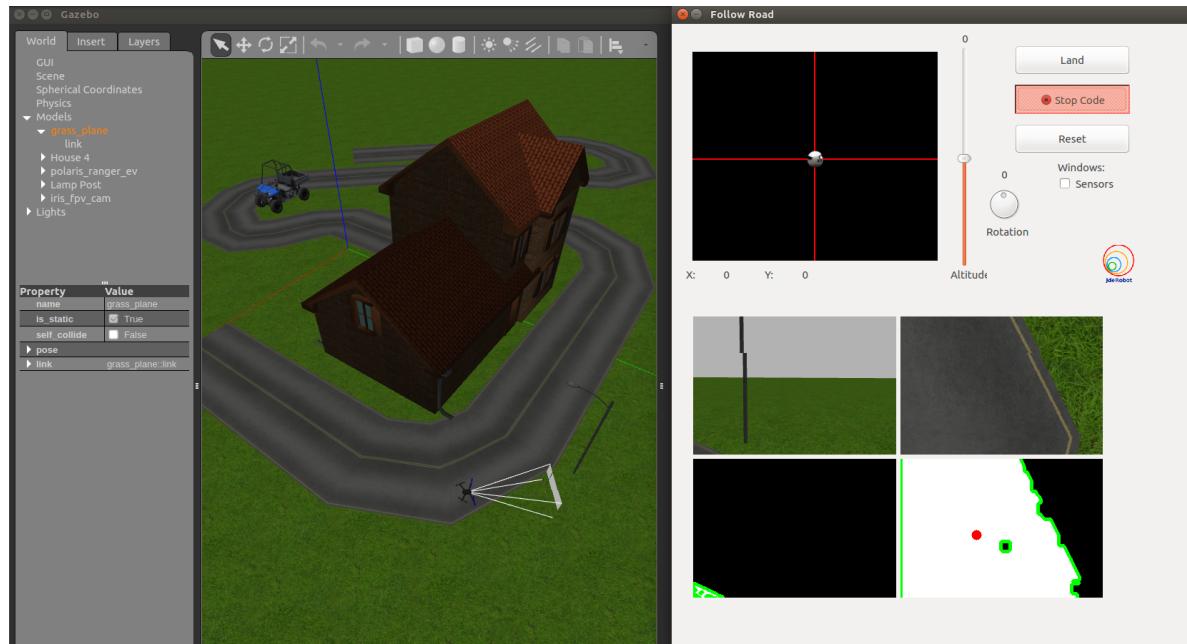


Figura 5.11: Ejecución con la solución

## 5.7. Plantilla del cuadernillo Jupyter

Como parte paralela a la práctica incluida en *Robotics-Academy*, se ha desarrollado otra plantilla diferente para la misma práctica con cuadernillos de *Jupyter*, en lugar de nodo ROS. Gracias a esto el alumno puede programar y ejecutar el código mediante la utilización del navegador web que prefiera.

Esto supone un paso importante hacia la multiplataforma del entorno docente *Robotics-Academy*, dado que el alumno puede acceder a las prácticas desde el sistema operativo que prefiera, pues sólo necesita acceso a internet.

Para que sea posible este hecho, ha sido necesaria una reestructuración del nodo ROS y de los ficheros que lo componen, además del método de desarrollo del algoritmo. Para ello se ha eliminado el GUI del nodo ROS y se han introducido los hilos de percepción y control en el cuadernillo de Jupyter. De esta manera, el alumno sólo debe llenar una celda del cuadernillo con el algoritmo desarrollado.

## CAPÍTULO 5. FOLLOW ROAD

---



Figura 5.12: Estructura de la plantilla de Jupyter de la práctica Follow\_Road

El cuadernillo tiene la estructura de la Figura 5.12. Como puede verse, es diferente a la estructura presente en el nodo ROS de Robotics-Academy. Ahora se divide en los siguientes ficheros y carpetas:

- images: en este directorio aparecen las imágenes contenidas en el cuadernillo.
- gazebo: en este directorio está el fichero de configuración con el escenario y con los *drivers* de *ROS-Kinetic*.
- interfaces: en este directorio se encuentran los *drivers* de *ROS-Kinetic*.
- chrono.ipynb: este fichero es el cuadernillo ejecutable en Jupyter.

Para acceder a la práctica, el alumno debe iniciar Jupyter introduciendo en la terminal el siguiente comando:

```
cd ~/Jupyter  
jupyter-notebook
```

Con esto se abrirá el navegador web por defecto en la carpeta local Jupyter. Una vez hecho esto, navegaremos hacia el directorio de la práctica de Jupyter “Follow\_Road” y abriremos el fichero “follow\_road.ipynb”. Tras esto se mostrará la siguiente imagen del cuadernillo:

### Follow Road Practice



#### 1- Introduction

In this exercise we are going to implement a drone intelligence to follow a road. To do it, the student needs to have at least the next knowledge:

- Python programming skills
- Color spaces (RGB, HSV, etc)
- Basic understanding of [OpenCV library](#)

#### 2- Exercise components

##### 2.1- Gazebo simulator

Gazebo simulator will be running in the background. The Gazebo world employed for this exercise has one element: a simulated drone. The drone robot will provide camera where the images will be provided to the student.



##### 2.2 Follow Road Component

This component has been developed specifically to carry out this exercise. This component connects to Gazebo to teleoperate the drone (or send orders to it) and receives images from its camera. The student has to modify this component and add code to accomplish the exercise. In particular, it is required to modify the execute() method.

#### 3- Exercise initialization

First of all, we need to run the Gazebo simulator:

```
In [ ]: 1 import subprocess  
2 p = subprocess.Popen(['rosrun', '/opt/jderobot/share/jderobot/gazebo/launch/follow_road.launch'])
```

Figura 5.13: Fichero follow\_road.ipynb

El alumno deberá ejecutar las celdas con código y seguir el guión mostrado. En primer lugar, deberá ejecutar el fichero de configuración del mundo:

## CAPÍTULO 5. FOLLOW ROAD

---

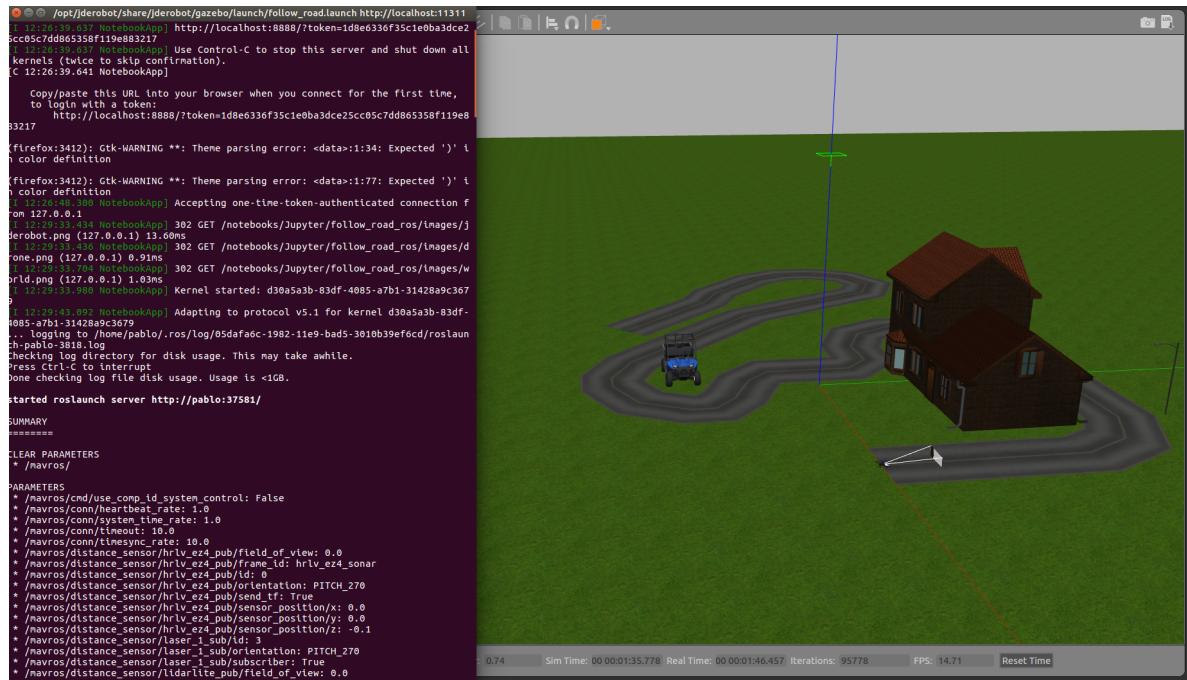


Figura 5.14: Celda con la inicialización del mundo y los drivers

Cuando se haya abierto el simulador es necesario importar el módulo del paquete “MyAlgorithm.py” y “follow\_road.py” para tener la funcionalidad provista en el nodo ROS. Para ello, se ejecutará la celda correspondiente con las clases “MyAlgorithm” y “FollowRoad”. Una vez importadas, aparece un mensaje de confirmación (Figura 5.15).

```
138 fr = FollowRoad()
139 fr.play()

Initializing
Follow_Road Components initialized OK
Follow_Road is running
```

Figura 5.15: Importación de las clases “MyAlgorithm” y “FollowRoad”

Cuando la ejecución imprima el mensaje “OK”, es necesario despegar el dron (Figura 5.16).

```
In [3]: 1 fr.takeoff()

Arming...
Arming Done
Taking Off...
TakeOff Done
Mode changed to: OFFBOARD
```

Figura 5.16: Despegue del dron

## CAPÍTULO 5. FOLLOW ROAD

---

Tras esto, el alumno puede comenzar a programar su código en la celda especificada para ello (Figura 5.17).

```
In [ ]: 1 # Implement execute method
2 def execute(self):
3     print "Running execute iteration"
4     #Add your code here.
5
6 fr.setExecute(execute)
```

Figura 5.17: Celda de importación del algoritmo

Una vez ejecutada la celda con el código, aparecerá mensaje iterativo “Running execute iteration” (Figura 5.18).

```
Code updated
Running execute iteration
```

Figura 5.18: Ejecución del código

A continuación, se muestran algunas fotos de la ejecución:

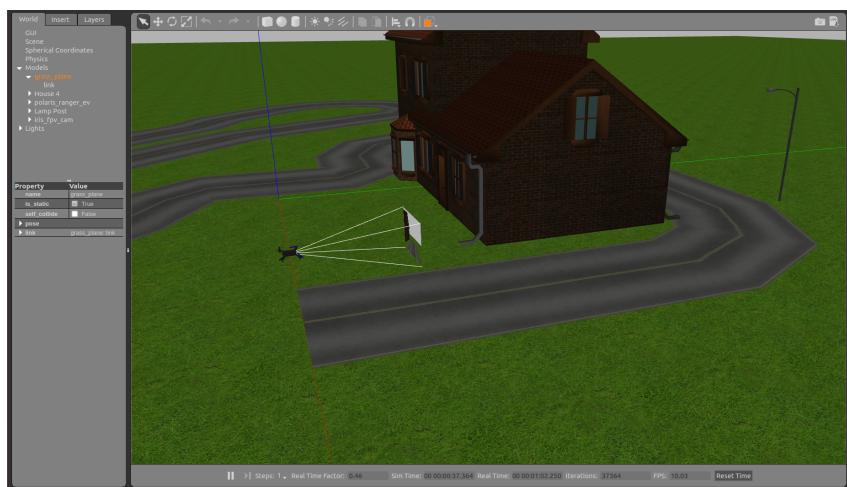


Figura 5.19: Ejecución del código

## CAPÍTULO 5. FOLLOW ROAD

---

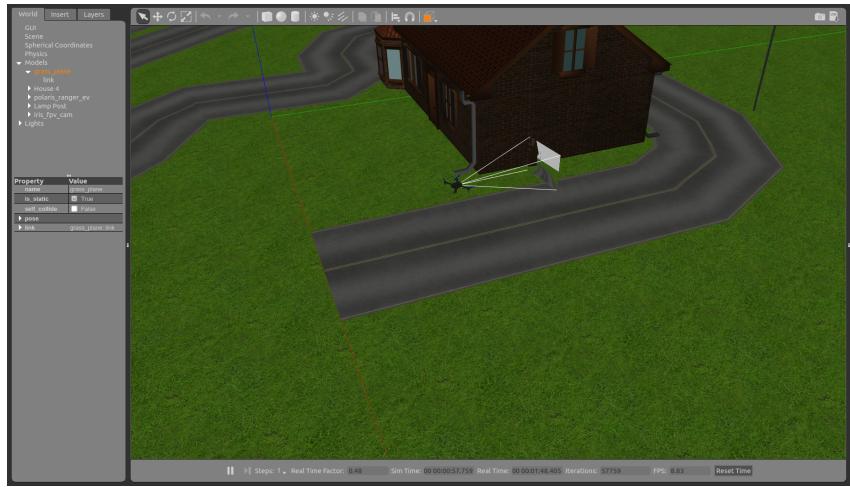


Figura 5.20: Ejecución del código

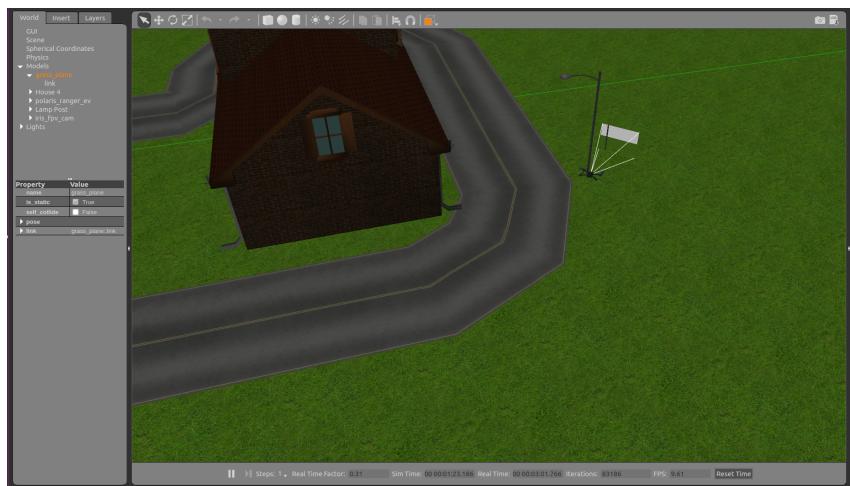


Figura 5.21: Ejecución del código

# **Capítulo 6**

## **Conclusiones**

Una vez documentadas en profundidad las dos prácticas desarrolladas en este Trabajo de Fin de Grado, se dedicará el siguiente capítulo a la comprobación de la consecución de los objetivos alcanzados, así como a la explicación de los conocimientos adquiridos y una breve exposición de posibles mejoras de las prácticas y de las líneas de actuación futuras.

### **6.1. Conclusiones**

El objetivo principal de desarrollar una nueva práctica y la consecución de una mejora notoria en otra de las prácticas del entorno JdeRbot-Academy ha sido alcanzado con éxito. Prueba de ello son, tanto su inclusión en el programa de prácticas, como el correcto funcionamiento del nodo académico y de la solución de referencia desarrollada. Este objetivo principal estaba subdividido en objetivos secundarios de los cuales se detallará a continuación si han sido alcanzado y la manera para ello.

El primer objetivo secundario establecido fue la mejora de una práctica existente del entorno de Robotics-Academy. La práctica escogida fue el sigue carreteras o *Follow Road*. Este objetivo se logró alcanzar, dado que la práctica fue completamente optimizada y mejorada. Se desarrollaron mejoras en la interfaz gráfica y el nodo académico, la introducción de un visor de imágenes filtradas y el desarrollo de una pausa académica. Además, se realizó una optimización de la práctica incluyendo *drivers* de ROS para dar un mejor soporte a su infraestructura, integrando *Px4*, *MAVLink* y *MavROS* para drones. De esta manera se tuvo que hacer una readaptación de las conexiones del nodo académico con

## CAPÍTULO 6. CONCLUSIONES

---

sensores y actuadores del robot. Después de todas estas modificaciones se consiguió una práctica totalmente operativa y actualizada. Tanto es así, que la renovación de la interfaz gráfica y la adaptación de *ROS* para drones se incluyeron en las prácticas similares del entorno Robotics-Academy.

El siguiente objetivo fue el desarrollo de una práctica completamente nueva llamada *Chrono*. Este objetivo fue bastante complejo debido a la implementación de una reproducción sincronizada que se adapte al rendimiento de cada ordenador. Además, la práctica debe grabar la simulación actual para su posterior reproducción en el caso de que el algoritmo desarrollado sea más rápido que la grabación. Adicionalmente, otro punto de gran complejidad era la visión de la posición del F1 con el récord del circuito en la interfaz gráfica del nodo académico, punto que también se consiguió solventar. A parte de estas tareas especialmente complejas, se tuvo que desarrollar la práctica desde cero, incluyendo el nodo académico, la conexión con los sensores actuadores del robot y el fichero para albergar la solución del estudiante.

En relación con los dos objetivos anteriores, se fijaron dos objetivos nuevos, los cuales establecían el desarrollo de una solución para cada práctica. Este punto incluía el estudio de técnicas de captación, procesado y control de imágenes y control del movimiento del robot. Además, sirve de punto de referencia para el desarrollo de la solución por los estudiantes. Ambos objetivos fueron alcanzados satisfactoriamente.

Para la solución de la primera práctica fue necesario el estudio de técnicas de procesado de imágenes para realizar un filtro de partículas para filtrar la carretera del resto del escenario. Una vez realizado el primer filtro fue necesario aplicar técnicas del postprocesado de imágenes para adaptar la imagen filtrada y localizar el centro de la carretera. Además, fue necesario el estudio de movimiento del dron para poder realizar un movimiento controlado. Este movimiento es muy sensible debido al control de la actitud y la inclinación del dron que pueden afectar a la visualización de las imágenes captadas por la cámara.

Para la solución de la segunda práctica se tuvo que realizar un estudio paralelo de filtrado y postprocesado de imagen para captar y procesar las imágenes grabadas por la cámara del coche. Además de adquirir los conocimientos de movimiento para el robot F1 utilizando los *drivers* de ROS. Es importante destacar que en este caso se tuvo que realizar una solución más rápida que el algoritmo con el récord del circuito optimizado a

partir de una solución previa. Es decir, en este caso se tuvieron que realizar dos soluciones distintas, una optimizada y otra desde cero.

Tras la consecución de los objetivos anteriores, se realizó una readaptación de la práctica *Crhono* para su inclusión en la plataforma web Jupyter. De esta manera el estudiante dispondrá de las mismas prácticas que las disponibles en el entorno JdeRobot pero con cuadernillos en lugar de nodo académico. De esta manera, el nodo académico es transparente al alumno, que dispone de una celda para desarrollar su algoritmo y solo tendrá que ejecutar dicha celda para ver su estado. Para lograr dicho objetivo fue necesario adquirir conocimientos de Jupyter así como un estudio de nuevos procesos de Python.

El siguiente objetivo respecto a la primera práctica fue la inclusión de la práctica inicial en el entorno docente Robotics-Academy-Web. De esta manera se da un paso más hacia un soporte multiplataforma. Esto es debido a que Academy-Web utiliza Dockers para ejecutar las prácticas en el navegador. Para poder alcanzar este objetivo hubo que estudiar el entorno Robotics-Academy-Web, y el estudio de Dockers, campos totalmente desconocidos.

Por último, gracias a una motivación personal, se ha conseguido adquirir conocimientos para afrontar problemas reales de ingeniería que comprenden tanto software como hardware. Debido a esto se ha adquirido experiencia para integrarse en proyectos de robótica, desde su infraestructura, las conexiones hardware-software, las interfaces, componentes, funcionalidad, así como la parte visible al usuario. También se han adquiridos conocimientos de simulación, y diseño gráfico, así como herramientas de tratamiento de imagen y técnicas de programación.

## 6.2. Trabajos futuros

El presente Trabajo de Fin de Grado ha expuesto una nueva vía para la realización de proyectos en el futuro.

En la práctica del dron que ha de seguir una carretera, se expone la posibilidad de realizar un *Delivery-Drone*, es decir, un dron con inteligencia parecida a la de esa misma práctica. De esta manera podría realizarse una práctica en la que, con un mapa previo, se le indique al dron al punto al que debe dirigirse en un plano de ciudad. De esta manera se podría dotar al dron de la inteligencia necesaria para ser un "Dron repartidor".

## CAPÍTULO 6. CONCLUSIONES

---

Respecto a la práctica de *Chrono*, sería interesante el desarrollo de una competición en distintos circuitos y la inclusión de distintos coches, cada uno con su propio algoritmo, que compitan entre ellos para conocer el algoritmo más eficiente en cada circuito. De esta manera se estaría desarrollando un Grand Pix de robots F1.

Otro punto de desarrollo futuro sería la realización de nuevos algoritmos que permitan un rendimiento más eficiente y rápido a los propuestos en este Trabajo de Fin de Grado.

# Bibliografía

- [1] Biblioteca types de Python. <https://docs.python.org/2/library/types.html>.
- [2] Contours in OpenCV. [https://opencv-python-tutroals.readthedocs.io/en/latest/py\\_tutorials/py\\_imgproc/py\\_contours/py\\_table\\_of\\_contents\\_contours/py\\_table\\_of\\_contents\\_contours.html#table-of-content-contours](https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_contours/py_table_of_contents_contours/py_table_of_contents_contours.html#table-of-content-contours).
- [3] Definición de Robot. <http://conceptodefinicion.de/robot/>.
- [4] Definición de Robótica. <https://www.definicionabc.com/tecnologia/robotica.php>.
- [5] Definición del formato SDF. <http://sdformat.org/>.
- [6] Descripción de OpenCV. <https://opencv.org/about.html>.
- [7] Guía de Jupyter Notebook. <https://www.datacamp.com/community/tutorials/tutorial-jupyter-notebook>.
- [8] Historia de la robótica. <http://wiki.robotica.webs.upv.es/wiki-de-robotica/introduccion/historia/>.
- [9] Historia de Python. <https://www.codejobs.biz/es/blog/2013/03/03/historia-de-python>.
- [10] Image Processing. <https://www.sciencedirect.com/topics/neuroscience/image-processing>.
- [11] Image Processing in OpenCV. [https://opencv-python-tutroals.readthedocs.io/en/latest/py\\_tutorials/py\\_imgproc/py\\_table\\_of\\_contents\\_imgproc/py\\_table\\_of\\_contents\\_imgproc.html](https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_table_of_contents_imgproc/py_table_of_contents_imgproc.html).
- [12] Introduction to Image Processing. <https://www.engineersgarage.com/articles/image-processing-tutorial-applications>.
- [13] IPython3. <https://ipython.org/ipython-doc/3/whatsnew/version3.html>.

## BIBLIOGRAFÍA

---

- [14] Jupyter Notebook, descripción del proyecto. <http://nbviewer.jupyter.org/github/jupyter/notebook/blob/master/docs/source/examples/Notebook/What%20is%20the%20Jupyter%20Notebook.ipynb#>.
- [15] Librería Matplotlib. [https://matplotlib.org/users/pyplot\\_tutorial.html](https://matplotlib.org/users/pyplot_tutorial.html).
- [16] MAVLink. <https://en.wikipedia.org/wiki/MAVLink>.
- [17] MAVLink Developer Guide. <https://mavlink.io/en/>.
- [18] MAVLink Interface. <http://ardupilot.org/dev/docs/mavlink-commands.html>.
- [19] MAVLink: protocolo de comunicación para drones. <https://www.xdrones.es/mavlink/>.
- [20] mavros. <http://wiki.ros.org/mavros>.
- [21] MAVROS. [https://dev.px4.io/en/ros/mavros\\_installation.html](https://dev.px4.io/en/ros/mavros_installation.html).
- [22] mavros/Plugins. <http://wiki.ros.org/mavros/Plugins>.
- [23] Modelo de desarrollo en Espiral. <https://pdfs.semanticscholar.org/presentation/2403/7678f5cf0d23d1a7d59b9b9ff2bab31d99.pdf>.
- [24] nav\_msgs/Odometry Message. [http://docs.ros.org/melodic/api/nav\\_msgs/html/msg/Odometry.html](http://docs.ros.org/melodic/api/nav_msgs/html/msg/Odometry.html).
- [25] Odometry. <https://es.coursera.org/lecture/mobile-robot/odometry-L4gPH>.
- [26] Plataforma Jupyter, página oficial. <http://jupyter.org/>.
- [27] Publishing Odometry Information over ROS. <http://wiki.ros.org/navigation/Tutorials/RobotSetup/Odom>.
- [28] PX4 Autopilot: DOCUMENTATION. <https://px4.io/documentation/>.
- [29] PX4 Drone Autopilot. <https://github.com/PX4/Firmware/blob/master/README.md>.
- [30] PyQt5. <https://www.riverbankcomputing.com/software/pyqt/intro>.
- [31] Qué es y para qué se usa la biblioteca PyQt5. <https://pypi.org/project/PyQt5/>.

## BIBLIOGRAFÍA

---

- [32] ¿Qué es Python? <https://www.python.org/doc/essays/blurb/>.
- [33] ROS. <http://ardupilot.org/dev/docs/ros.html>.
- [34] ROSbag: Package Summary. <http://wiki.ros.org/rosbag>.
- [35] Simulación en Gazebo. <https://robologs.net/2016/06/25/gazebo-simulator-simular-un-robot-nunca-fue-tan-facil/>.
- [36] Simulador Gazebo. <https://dev.px4.io/en/simulation/gazebo.html>.
- [37] Topics de ROS. <http://wiki.ros.org/Topics>.
- [38] What is odometry? <https://groups.csail.mit.edu/drl/courses/cs54-2001s/odometry.html>.
- [39] AForge.NET. AForge.NET Framework. <http://www.aforgenet.com/framework/>.
- [40] BAILLIE, J., DEMAILLE, A., HOCQUET, Q., NOTTALE, M., AND TARDIEU, S. *The Urbi Universal Platform for Robotics. Simulation, Modeling and Programming for Autonomous Robots. 4th International Conference, SIMPAR 2014, Bergamo, Italia, 2014.*
- [41] BEN-ARI, M., AND MONDADA, F. Elements of Robotics, 2017.
- [42] BRUYNINCKX, H., AND SOETENS, P. The OROCOS Project. <https://people.mech.kuleuven.be/~orocos/pub/stable/documentation/rtt/v1.10.x/doc-xml/orocos-overview.html>, 2007.
- [43] CAÑAS, J. Programación de robots con la plataforma JdeRobot. Universidad de Málaga., 2009.
- [44] CAÑAS, J., MARTÍ, A., PERDICES, E., RIVAS, F., AND CALVO., R. Entorno docente para la programación de la inteligencia de los robots. Revista Iberoamericana de Automática e Informática Industrial, 2016.
- [45] CAÑAS, J. M., FERNÁNDEZ, J. A., JIMÉNEZ, D., AND VELA, J. Programación de aplicaciones para drones con el entorno software JdeRobot. Congreso CivilDRON 2018, 2018.

## BIBLIOGRAFÍA

---

- [46] CERIANI, S., AND MIGLIAVACCA, M. *Middleware in robotics. Advanced Methods of Information Technology for Autonomous Robotics. Internal Report For Advanced Methods of Information Technology for Authonomous Robotics, Politecnico di Milano*, 2012.
- [47] CLARK, C. ARW – Lecture 01 Odometry Kinematics, 2016.
- [48] CRAIG, J. J. *Introduction to Robotics: Mechanics and Control*. 1986.
- [49] CYBERBOTICS. Webots. <https://www.cyberbotics.com/webots.php>, 2015.
- [50] EDUCATRONICS. Importancia de la Robótica en la Educación. Ciencia, arte y arquitectura. <http://educatronics.com/publicaciones/importancia-de-la-rob%C3%A9tica-en-la-educacion>.
- [51] ESTÉVEZ., C. A. Nuevas Prácticas Docentes de Robótica en el Entorno JdeRobot-Academy, 2018.
- [52] MARTÍNEZ., V. F. Nuevas Prácticas en el Entorno Docente de Robótica JdeRobot-Academy, 2017.
- [53] R., G. F. Modelo Espiral de un proyecto de desarrollo de software, Administración y Evaluación de Proyectos. <http://www.ojovisual.net/galofarino/modeloesprial.pdf>.
- [54] RODRÍGUEZ., I. L. Nuevas Prácticas en el Entorno Docente de Robótica JdeRobot-Academy, 2018.