



Universidad
Carlos III de Madrid

DEPARTAMENTO DE INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

TRABAJO FIN DE MÁSTER

EJECUCIÓN MIXTA DE EJERCICIOS DE VISIÓN ARTIFICIAL Y ROBÓTICA A TRAVÉS DE LA WEB

Autor: Carlos Kamal Awadallah Estévez

Tutor: Arturo de la Escalera Hueso

Co-Tutor: José María Cañas Plaza

MÁSTER OFICIAL EN
ROBÓTICA Y AUTOMATIZACIÓN

LEGANÉS, MADRID

FEBRERO 2020

UNIVERSIDAD CARLOS III DE MADRID
MÁSTER OFICIAL EN ROBÓTICA Y AUTOMATIZACIÓN

El tribunal aprueba la Tesis de Máster titulada “**Ejecución Mixta de Ejercicios de Visión Artificial y Robótica a través de la Web**” realizada por **Carlos Kamal Awadallah Estévez**.

Fecha: Febrero 2020

Tribunal:

Dr./Dra.

Dr./Dra.

Dr./Dra.

“The only thing that comes to a sleeping man is dreams”

Tupac Amaru Shakur a.k.a Makaveli

Índice general

Índice de Tablas	XI
Índice de Figuras	XIII
Agradecimientos	XV
Resumen	XVII
Abstract	XIX
1. Introducción	1
1.1. Robótica	2
1.2. Tecnologías Web	4
1.3. Educación en Robótica a través de las Tecnologías Web	6
1.4. Planteamiento del Problema	12
2. Objetivos y Metodología de Trabajo	15
2.1. Proyecto Planteado	15
2.2. Objetivos del Proyecto	17
2.3. Requisitos	18
2.4. Metodología Empleada	18

2.4.1. Metodología de Investigación	19
2.4.2. Metodología de Implementación	20
3. Infraestructura Utilizada	23
3.1. Proyecto Jupyter	23
3.2. Plataforma Docker	27
3.3. Entorno Django	29
3.4. ROS (Robot Operating System)	31
3.5. Simulador Gazebo	33
3.6. Biblioteca OpenCV	36
3.7. Lenguajes de Programación: Python, JavaScript	37
4. Plataforma Web para Enseñanza en Visión Artificial y Robótica	39
4.1. Servidor Web en Ejecución Remota	40
4.1.1. Diseño	40
4.1.2. Back-end del Servicio	41
4.1.3. Front-end del Servicio	45
4.2. Servidor Web en “Ejecución Mixta”	46
4.2.1. Diseño	47
4.2.2. Back-end Remoto	49
4.2.3. Back-end Local	50
4.2.3.1. Protocolo de Intercambio	50
4.2.3.2. Contendor Docker	54
4.2.3.3. API de “Ejecución Mixta”	64
4.2.4. Seguridad	66
4.2.5. Flujograma	68
5. Validación Experimental	73
5.1. Contenidos Académicos de Pruebas	73
5.1.1. Ejercicio del Filtro de Color	73
5.1.2. Ejercicio del Sigue Líneas con Fórmula 1	77

5.1.3. Ejercicio del Cuadrado con el dron Tello Real	79
5.2. Experimentos Realizados	82
5.2.1. Servidor en Producción	82
5.2.2. Grupos de Pruebas: <i>betatesters</i>	83
6. Conclusiones y Líneas Futuras	87
6.1. Conclusiones	87
6.2. Análisis de prestaciones	90
6.3. Casos de Uso	91
6.4. Trabajos Futuros y Líneas Futuras de Investigación	91
6.5. Vídeos Demostrativos	92
Referencias	95

Índice de Tablas

5.1. Tabla de Resultados de Parámetros de Red.	83
5.2. Tabla de Resultados de Sujetos de Pruebas.	84
6.1. Tabla de Análisis de Prestaciones.	90

Índice de Figuras

1.1.	Aplicaciones actuales de los robots.	2
1.2.	Evolución de las Tecnologías Web	5
1.3.	Entorno Robótico en la Web	7
1.4.	Conexión Local de Colaboratory	8
1.5.	Robots Simulados soportados en Robot Ignite Academy.	9
1.6.	Simulación con AWS RoboMaker.	10
1.7.	Simulación de aplicación robótica con JdeRobot.	11
2.1.	Modelo en Espiral del Proceso de Desarrollo Software.	21
3.1.	Cuadernillos de Jupyter	25
3.2.	Estructura de Docker	28
3.3.	Arquitectura de Django	30
3.4.	Motor de físicas de Gazebo	35
3.5.	Modelos de Simulación en Gazebo	35
3.6.	Herramientas de procesado con OpenCV: Segmentación de caras.	37
4.1.	Arquitectura del Servidor en Ejecución Remota.	40
4.2.	UI de la Plataforma Web en Django	42
4.3.	UI de Simulación de la Aplicación	45
4.4.	Arquitectura de la Ejecución Mixta para Aplicaciones de Robótica.	46

4.5.	Estructura de Capas de la “Ejecución Mixta”	48
4.6.	Capura de Paquetes del Mecanismo de Comunicación de Jupyter	52
4.7.	REST API de Jupyter	52
4.8.	Escenario de Simulación	59
4.9.	Mensaje de Entrega del Cuadernillo o Notebook	66
4.10.	Mensaje de Respuesta de Petición de Ejecución	66
4.11.	Volumen de Docker	68
4.12.	Flujograma	69
5.1.	Input: Fotograma de la Fuente de Vídeo	75
5.2.	Procesado de Imagen para el Filtro de Color	76
5.3.	UI de Jupyter para el Filtro de Color	77
5.4.	Procesado de la imagen del F1	78

Agradecimientos

Es curioso ver cómo se cierra ésta importante y larguísima etapa estudiantil sin sentir ya ningún cambio en especial y, sin embargo, darme cuenta de que siempre estuve preparado para todo lo que esté por venir. Esta seguridad es en gran parte debida a los aportes y el sustento, ya fuera anímico o económico, de todos aquellos a los que a continuación quiero dedicar algunas palabras.

Hace algunos años ya desde que no puedo evitar pensar en todo lo que mi madre dio, da y dará por mí durante mi vida, e incluso antes de ella. No sorprendo a nadie si confieso que sin ella jamás habría llegado aquí y, aún más importante, no sería quien soy. Su fuerza y su visión del camino es lo que me impulsó día a día para crecer, y es ahora cuando veo el valor de sus sacrificios. Ninguna palabra que escriba serviría para darte las gracias por lo que haces, M^aLuz, y aunque ya lo sabes quiero que quede escrito que te quiero con todo mi corazón.

Mi familia estuvo siempre al pie del cañón, tanto mi hermana Shadia, como mi abuela M^aRosa y mi tía Rosi siempre me desearon lo mejor y me agasajaron con sus ánimos de valor incalculable. Cómo no mencionar también a Antonio, sin él ni siquiera habría tenido la oportunidad que hoy estoy aprovechando. A todos ellos, y también al resto de mi familia presente, gracias por todo.

Continuo con mis tutores, quienes hicieron el terreno muy llano, acallaron

mis dudas y me orientaron siempre con sabiduría y razón. Gracias a José María por transmitirme el gusto por la Robótica y por tratarme siempre de la mejor manera, por saciar mi curiosidad y por tu dedicación, sin la que el proyecto no habría salido adelante. Muchas gracias a Arturo por hacerlo todo tan fácil, y en especial por haberme permitido desarrollar mi propia idea y por impulsarme a alcanzar mis objetivos.

En este punto de mi vida es también cuando he caído en la cuenta del precioso tesoro que tengo alrededor. Me refiero, por supuesto, a mis amigos de siempre. Reciben una mención especial Joel. C, Patri, Lidia y Joel. Y, y no necesito dar detalles del merecimiento de la misma. Crecemos y tomamos caminos distintos, pero siempre tenemos un hueco para tomarnos una cerveza y construir lazos mucho más fuertes aún con las escasas oportunidades que tenemos para ello. Es increíble, son increíbles. Sigan inspirándome como lo hacen.

Por todas aquellas personas que conocí en Madrid y que cobraron gran protagonismo en mi vida. Por mis amigos de la Universidad y por mis chicas del gimnasio, estoy agradecido. Han hecho esta dura etapa un camino limpio e iluminado. A todos, un fuerte abrazo.

Por último, pero quizás más importante, gracias a mí. Probablemente suene algo narcisista pero debo ser consciente de que si estoy aquí es porque yo me he traído. Que por encima de todo he sido yo el responsable de decidir, hacer y decidir hacer. Nunca cambies tu forma de trabajar y sobre todo de ver, porque has comprobado su valor único y su efectividad al llegar donde estás. Recuerda estas palabras para que nunca te falte fuerza de voluntad.

Resumen

Esta tesis busca dar solución a un problema latente en el joven campo de la Ingeniería Robótica: su difícil accesibilidad. Se ha llevado a cabo un exhaustivo proceso de investigación que ha desembocado en las causas de ésta limitación y se ha diseñado una herramienta, cuyo desarrollo también se recoge en este documento, de características potentes y novedosas, que puede suponer un paso adelante en la resolución del problema mencionado.

El proyecto comienza con el estudio del panorama robótico actual y de las tendencias últimas en lo relativo al desarrollo de aplicaciones robóticas. Tras este análisis de estado se plantea el problema y se propone una idea, la “Ejecución Mixta” para tareas de visión artificial y robótica, como primera aproximación existente a la solución.

Tras la descripción y planteamiento iniciales se establecen en el Capítulo 2 unos objetivos determinados y se explica cómo se abordarán para, después, comenzar en el Capítulo 3 con la etapa de investigación asociada al proyecto que culminó con la selección de las herramientas y tecnologías involucradas en el proyecto.

Se detalla ya en el Capítulo 4 la estructura diseñada para abordar el problema y su arquitectura. También se recoge el proceso de implementación de la herramienta junto con los problemas que surgieron y la solución adoptada,

además del funcionamiento de la misma.

Se expone en Capítulo 5 el conjunto de pruebas utilizado para validar la herramienta y medir sus prestaciones, también útil para la extracción de inferencias, las cuales se recogen en el último capítulo a modos de conclusión, junto con una hoja de ruta a corto plazo y las aplicaciones principales del desarrollo.

Abstract

This thesis looks forward to providing a solution to an underlying problem in the young field of Robotic Engineering: its difficult accessibility. An exhaustive research process has been carried out that has led to the causes of this limitation and a tool, whose development is also included in this document, with powerful and innovative features, has been designed and may represent a step forward in solving the aforementioned problem.

The project begins with the study of the current robotic outlook and the latest trends in the development of robotic applications. After this state analysis the problem is presented and an idea is proposed, the 'Mixed Execution' for artificial vision and robotic tasks, as a first existing approach to the solution.

After the initial description and consideration, in Chapter 2 some specific objectives are established and it is explained how they will be achieved. Then, in Chapter 3, the research stage associated to the project is started, culminating with the selection of the tools and technologies involved in the project.

The structure designed to address the problem and its architecture is detailed in Chapter 4. The process of implementing the tool is also described, along with the problems that arose and the solution adopted, as well as how the final tool operates.

The test suite used to validate the tool and measure its performance, also

useful for drawing inferences, is presented in Chapter 5. Those findings are set out in the last chapter in conclusion mode, together with a short-term roadmap and the main applications of the development.

Introducción

Este documento propone y recoge el desarrollo de un proyecto de tesis que surge de la evaluación del estado actual de la robótica. En un punto en el que el despegue del sector ya ha comenzado, empieza a ser necesario atender las necesidades que surgen a su paso. Se llevó a cabo un exhaustivo estudio que trató de precisar cuáles eran estas necesidades y, sobre todo, cuáles de ellas tenían mayor importancia y mayor valor contributivo. Se llegó a una conclusión que motivó el planteamiento de la idea de este Trabajo Fin de Máster: “hacer la robótica más accesible”.

Se buscará detallar de la manera más fiel posible el estudio y análisis de las herramientas asociadas al sector robótico y automático utilizadas, el método y los criterios de selección que llevaron a la elección de unos agentes frente a otros y su conveniente combinación para la construcción de una herramienta, que creemos novedosa, que puede dar solución a una de las lagunas más visibles en lo que a docencia, aprendizaje e investigación se refiere: **su accesibilidad**. En este primer capítulo se expone tanto el contexto en el cual se sitúa este proyecto como el motivo por el cual su desarrollo puede suponer un paso más hacia la satisfacción de una necesidad social.

1.1. Robótica

Aunque el origen de la robótica se remonta a los principios de la década de los 50, es en los últimos años cuando ha incrementado exponencialmente la presencia de estos sistemas electromecánicos en la vida cotidiana, además de en el ámbito laboral, para enfrentar problemas que resultan tediosos, repetitivos e incluso peligrosos para las personas, a la par que reducen en gran medida la carga de trabajo a la que están sometidas. Hoy en día no solamente nos rodean los robots industriales, como los presentes en líneas de producción o los involucrados en cadenas de envasado entre otros, sino que los robots adquieren cada vez mayor protagonismo y presencia en entornos alternativos, como el doméstico, militar, o el agrícola (Fig. 1.1), incluso dando pie a la creación de numerosas y nuevas aplicaciones para las que son idóneos como las tareas relacionadas con logística de almacenes y envíos de mercancía o exploración espacial.



Figura 1.1: Aplicaciones actuales de los robots.

Con el paso de los años se observa una clara tendencia hacia el diseño de sistemas robotizados para cubrir necesidades o desempeñar tareas que nunca

se imaginaron automáticas. Surge, con ello, una necesidad social que requiere la instrucción de nuevos profesionales y expertos en esta potente rama de la ciencia y la tecnología que puedan lidiar con el diseño y desarrollo de estas entidades mecánicas cuyo comportamiento se espera que sea cada vez más complejo y completo, ayudando a los seres humanos no sólo a reducir costos, sino también a simplificar tareas, optimizar recursos y contribuir en labores de investigación. Es por eso que aumentan considerablemente aún hoy las ramas de estudio asociadas a la Ingeniería Automática, Informática, Mecánica, Electrónica y de Telecomunicaciones principalmente, que persiguen precisamente cubrir esa exigencia de formación para abordar un futuro lleno de robots, a la par que contribuyen a la generación de investigadores y proyectos que propulsarán el desarrollo de este sector en búsqueda del beneficio social.

Las circunstancias idóneas para la instrucción e investigación necesarias para el estudio de la Robótica no resultan sencillas de conseguir. Como ya se ha dicho, nuestro entorno actual engloba robots en grandes cantidades, pero también es cierto que el acceso a los mismos está muy restringido a cierto sector social. No se trata de un problema económico en cuanto a su adquisición, pues poco a poco el precio de producción de sistemas robóticos se reduce gracias al abaratamiento de componentes electrónicos y materiales con los que se elaboran, además de la optimización de los procesos de construcción y ensamblado de los mismos, que hacen que el precio final de los robots sea, en cierta medida, asequible para el consumidor medio dependiendo siempre de la aplicación. El problema reside más bien en que los entornos de desarrollo, investigación, docencia e instrucción en la robótica son a la vez difícilmente accesibles, escasos, enormemente controlados y por último, pero quizá más importante, económicamente exigentes. Es por eso que resulta complicado el proceso de instrucción en robótica. Jamás pediríamos al escultor que aprendiera a esculpir sin cincel, o al biólogo que estudiase la naturaleza en su sótano. Sin embargo, debido al bajo grado de accesibilidad de los entornos docentes en el campo de la robótica,

en muchas ocasiones se priva de su principal herramienta de aprendizaje a los estudiantes e investigadores del campo, o se complica en gran medida su curva de entrada.

Destaca también la carencia o escasez de entornos de aprendizaje graduales como los existentes en otros ámbitos, donde se comienza por un nivel básico que poco a poco se va incrementando. La robótica requiere, por definición, de entornos de aprendizaje e investigación altamente especializados, donde el usuario debe conocer de antemano no sólo los fundamentos básicos de los sistemas robotizados, sino también las nociones necesarias de matemáticas, física, mecánica y electrónica como mínimo, pudiendo extenderse el repertorio según el área de especialización dentro del campo.

Quedan de manifiesto las dificultades a las que se enfrenta todo aquel que quiere tener relación con la robótica.

1.2. Tecnologías Web

El crecimiento exponencial al que está sujeta la Web en los últimos tiempos hace que se vaya constituyendo un entorno ideal para el desarrollo de nuevas aplicaciones en lo relacionado con el ocio, los servicios y las necesidades de las personas. De entre los nuevos productos web más influyentes podemos encontrar ejemplos como Netflix, un servicio de video bajo demanda (VoD) que ofrece películas y series en *streaming* a través de la web, o Spotify, de características similares en el caso del contenido musical, e incluso servicios de computación en la nube como Amazon Web Services sobre los que se puede construir potentes plataformas que sirven a las personas a través de Internet como Dropbox, HootSuite o Foursquare.

Algunos años atrás resultaba impensable plantear un desarrollo robótico empleando este tipo de tecnología dado que los motores web apenas tenían potencia de cómputo suficiente para reproducir un vídeo, y mucho menos podían siquiera superar los límites del navegador para acceder a dispositivos externos.

Las nuevas aplicaciones de las tecnologías web han demostrado que esta circunstancia ha cambiado hasta tal punto en que se pueden crear aplicaciones tan exigentes como juegos en realidad virtual, procesamiento de imágenes, *BigData* e incluso tareas de aprendizaje automático gracias, entre otras cosas, a la posibilidad de acceder al *hardware* de aceleración gráfica por parte del navegador y a la incorporación en el mismo de potentes núcleos o motores de ejecución, dando lugar a las aplicaciones dinámicas en la nube (Fig. 1.2).

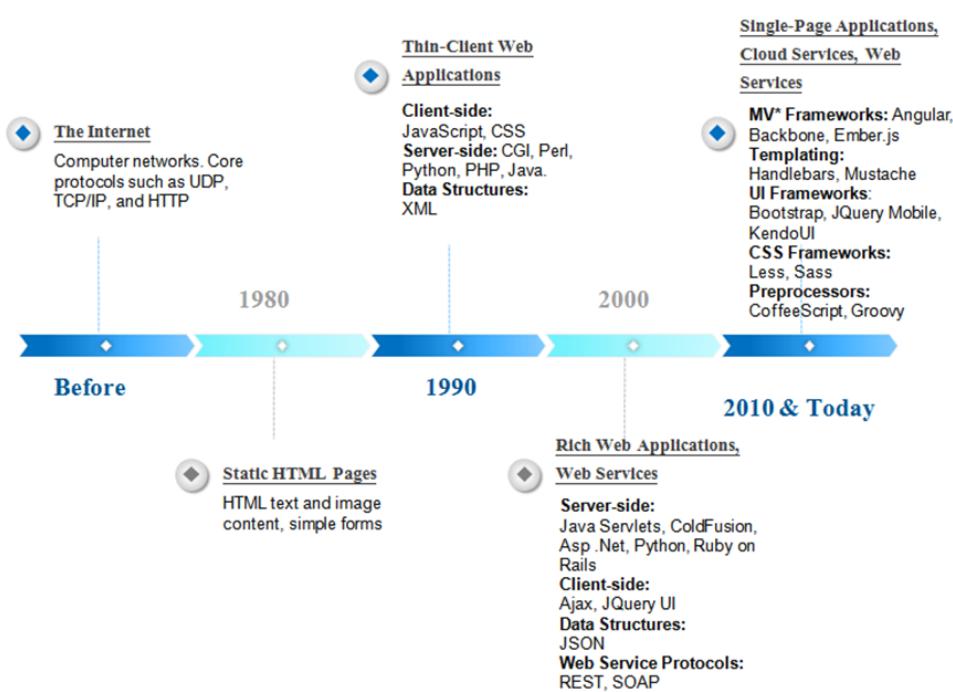


Figura 1.2: Evolución de las Tecnologías Web

En la actualidad, se puede construir aplicaciones para casi cualquier caso de uso, incluido el de la robótica, completamente formadas por herramientas web. Gran parte de sus ventajas residen en que el acceso a la web es totalmente público y sencillo, de tal manera que un servicio ofrecido a través de la web puede ser usado por cualquier persona del planeta, con independencia de su sistema operativo, dispositivo o entorno.

Dado que Internet no es más que un conjunto debidamente interconectado de sistemas y redes de sistemas que se comunican a través de protocolos conocidos (mirándose con un gran nivel de abstracción), se puede conseguir de manera bastante eficaz conectar herramientas de distinta naturaleza para que funcionen como un conjunto homogéneo a través de Internet. Entre las funciones interconectables que nos ofrece la web destacamos especialmente el acceso al *hardware* del equipo cliente, el soporte de almacenamiento en la nube, los mecanismos de seguridad, las características de sus canales de comunicación (baja latencia punto a punto, retransmisiones, baja tasa de error y de paquetes perdidos,...) y la capacidad de ejecutar el código de la aplicación.

Aún con todas las ventajas ya mencionadas, la característica responsable del éxito de las aplicaciones web es probablemente la facilidad de acceso, que se realiza a través de un navegador web. Este es el agente que tiene visibilidad tanto interna como externa, es decir, que puede comunicarse tanto con el PC o sistema en el que está instalado como con el resto del mundo, y funciona con independencia del sistema latente bajo él. Esto quiere decir que cualquier ente conectado a Internet puede acceder a las aplicaciones web, sin importar cuál sea su sistema.

1.3. Educación en Robótica a través de las Tecnologías Web

La intersección entre el campo de la robótica y las herramientas web conforma a su vez un campo con un gran potencial para el desarrollo de aplicaciones que permitan enfrentarse a tareas de investigación y aprendizaje con características muy atractivas.

Investigando el panorama docente y de experimentación moderno en cuanto a robótica se refiere, encontramos numerosas librerías que ya facilitan en gran medida el trabajo con robots, como pueden ser OpenCV para tareas de Visión Artificial u OMPL para planificación de rutas, y también entornos de desarrollo y *frameworks* que simplifican el proceso como ROS o YARP. Sin embargo, aún

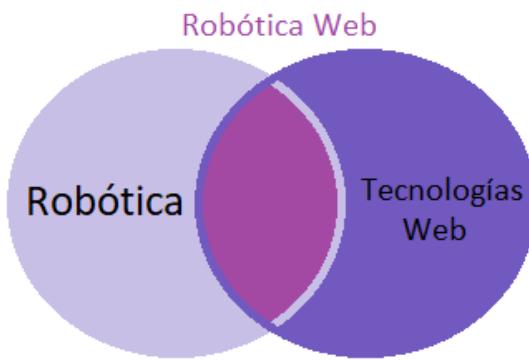


Figura 1.3: Entorno Robótico en la Web

no se han desarrollado herramientas maduras que engloben estas nuevas tecnologías en una sola para que el trabajo con sistemas robóticos sea casi tan fácil como pulsar un botón y ver el código en acción. Hasta el momento, es necesario disponer de numerosos módulos inteligentes cuidadosamente interconectados que sincronicen y supervisen la ejecución de diferentes aspectos para lograr controlar cualquier aspecto de un robot, todos ellos necesariamente ideados y construidos por el sujeto que trata de desarrollar algo.

Aún no existen plataformas educativas de uso extendido o estándar “*de facto*” que permitan una buena instrucción en este joven campo de la ingeniería, siendo en muchos casos debido a que resulta muy difícil acceder a *hardware* adecuado para el aprendizaje por temas económicos o restricciones externas, o llevar los conocimientos teóricos a un entorno práctico para asentar los conocimientos. Analizando el estado actual se ha encontrado algunas plataformas que ya buscan una primera aproximación hacia esa aplicación de propósito general en docencia e investigación, que tratan de disponer de todo lo necesario para que sentarse a programar un robot, o varios, resulte incluso sencillo, a la par que instructivo y alentador. Algunos ejemplos son:

- La herramienta *Colaboratory*¹ de Google es un entorno que permite escribir

¹<https://colab.research.google.com/notebooks/welcome.ipynb>

y ejecutar código, guardar el desarrollo y compartir su análisis por distintos medios, y que tiene funciones que permiten utilizar el *hardware* de aceleración gráfica del equipo que accede a la aplicación para realizar tareas de gran consumo de cómputo haciendo uso de recursos informáticos muy potentes, todo desde el navegador. Está construida sobre el proyecto *Jupyter*, que a su vez se ejecuta completamente en la nube, y se utiliza mayoritariamente para el desarrollo de aplicaciones que tienen que ver con el *Deep Learning*. Así, la gran ventaja que ofrece es que permite a los usuarios descargar su código sobre su propia tarjeta gráfica para acelerar los procesos (Fig. 1.4).

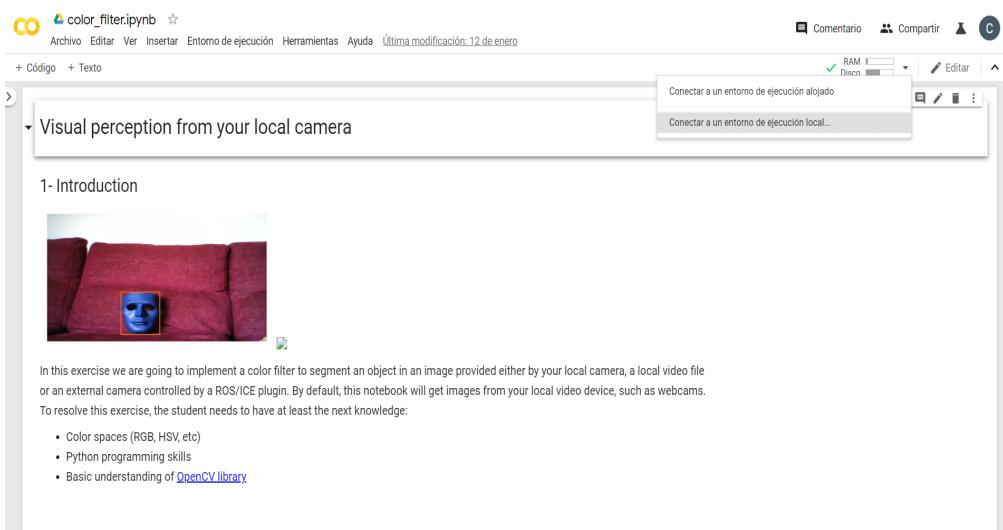


Figura 1.4: Conexión Local de Colaboratory

- Una plataforma con un gran potencial es “*Robot Ignite Academy*² de la empresa The construct³. Esta plataforma, orientada sobre todo al aprendizaje relacionado con ROS y a las buenas prácticas de trabajo con robots, es una plataforma *on-line* de pago cuyo uso objetivo es principalmente docente

²<https://www.robotigniteacademy.com/en/teachers/>

³<https://www.theconstructsim.com/>

para equipos de trabajo o desarrollo en empresas, o para estudiantes de todo el mundo. Busca instruir a los usuarios en entornos prácticos simulados de aprendizaje en forma de cursos de distintos niveles, todos ellos con gran cantidad de detalle para una buena formación. Entre los servicios que oferta se incluye la construcción desde el principio de un módulo robótico programable cuyo valor docente es importante. Entre sus características destaca que es completamente basado en tecnologías web, y por lo tanto accesible para cualquiera que disponga de un navegador. El rango de robots y escenarios que soporta es limitado (Fig. 1.5).



Figura 1.5: Robots Simulados soportados en Robot Ignite Academy.

→ Otro gran ejemplo muy reciente es el entorno *AWS RoboMaker*⁴ de Amazon. Se trata de un servicio web que integra herramientas de trabajo con robots, como ROS y entornos de simulación robóticos, que además aprovecha la conectividad en la nube para ofrecer otros servicios, todos ellos propiedad de la empresa, que de nuevo cobra por el servicio que ofrece. Este entorno tiene muchas ventajas en cuanto al proceso de desarrollo e implementación de aplicaciones robóticas inteligentes a gran escala, además de facilitar el proceso de test para probar las aplicaciones que se construyen. Ofrece simulaciones muy realistas y entornos complejos para los robots, de tal manera que se pueden desarrollar comportamientos muy potentes, todo ello en simulación (Fig. 1.6). No ofrece soporte para robots reales, aunque sí proporciona un servicio de administración de flotas que

⁴<https://aws.amazon.com/es/robomaker/>

simplifica en gran medida el paso de la simulación al sistema físico.

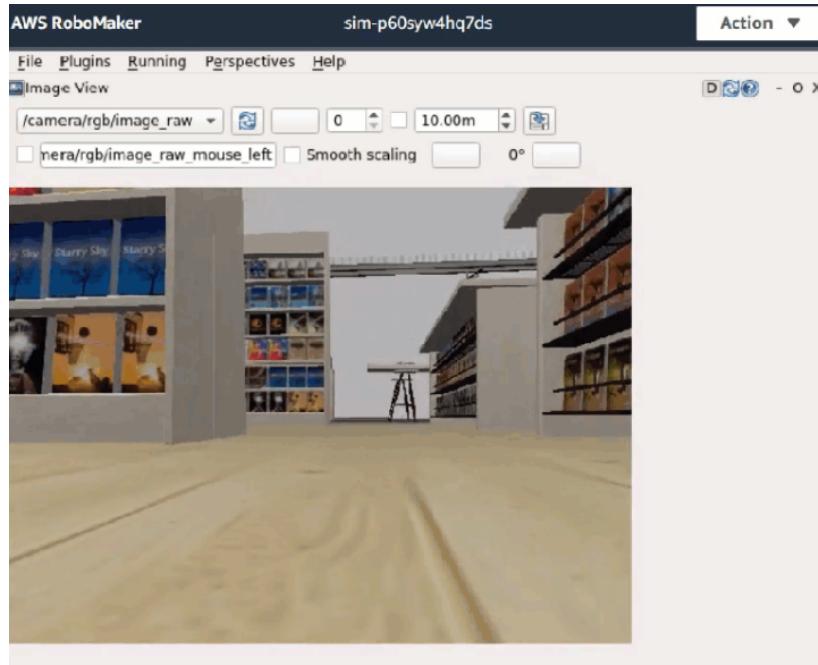


Figura 1.6: Simulación con AWS RoboMaker.

- Existen otro tipo de plataformas menos extendidas, en el ámbito nacional e internacional, como JdeRobot Academy⁵, un entorno basado en componentes que se ejecutan como procesos que requiere instalación, y que se especializa en la disposición de todos los elementos necesarios para la fácil construcción de aplicaciones con robots y visión artificial. Si bien soporta tanto robots simulados como reales, este soporte es también limitado (repertorio concreto) y opera sobre las distintas distribuciones del sistema operativo Ubuntu, siendo que en el resto de casos es necesario recurrir a herramientas de virtualización para poder utilizarlo. Este entorno es muy completo y de código abierto, por lo que cualquier usuario puede acceder a él y utilizar toda su funcionalidad. A pesar de su filosofía de *software*

⁵<https://jderobot.github.io/RoboticsAcademy/>

libre ofrece calidad profesional (Fig. 1.7) a través de la integración de herramientas clásicas de desarrollo robótico como ROS, el simulador Gazebo, las librerías OpenCV y OMPL, etc. Aunque la aplicación es gratuita, requiere amplios conocimiento de programación de robots avanzada para trasladar el código de la simulación al sistema físico.

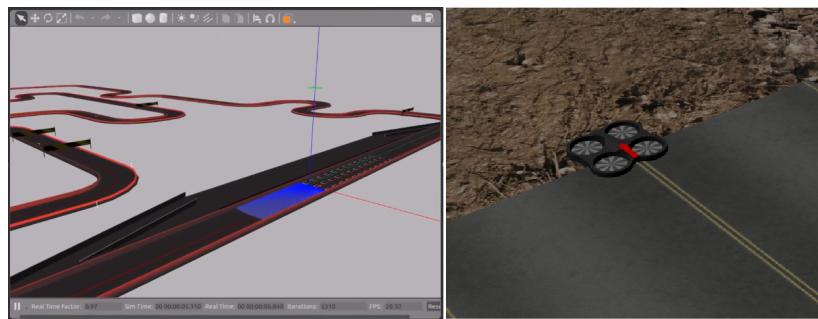


Figura 1.7: Simulación de aplicación robótica con JdeRobot.

Inferimos que hay algo que todos ellos tienen en común: tanto su accesibilidad como su uso son limitados. Las carencias de los entornos actuales radican principalmente en que la mayoría son de pago, muchos de ellos sólo permiten el trabajo en simulación y la totalidad de ellos son incompletos, especialmente en el sentido de que sólo permiten trabajar en un área de la robótica, sólo con un tipo de robot (o una clase) o sólo permiten resolver ciertos problemas para los cuales se prepara un escenario muy controlado. Se observa, aún así, una clara tendencia a ofrecer este tipo de servicio de construcción de aplicaciones como *middleware* a través de la nube para hacerlo más accesible, lo cual supone un punto a tener en cuenta a la hora de evaluar la situación actual y diseñar una nueva idea que colecte y compacte lo mejor de cada uno de los proyectos listados que existen a día de hoy. Es precisamente aquí donde se forma el lugar común entre robótica y tecnologías web, dando paso al entorno que alojará muchas de las aplicaciones robóticas del futuro próximo.

1.4. Planteamiento del Problema

El problema a tratar en esta tesis proviene del análisis de los factores que hacen distinto el desarrollo y el aprendizaje en este campo, y de cómo estos factores suponen un obstáculo, y en ocasiones incluso un impedimento, a la hora de realizar un proyecto específico. Cabe preguntarse por qué no existen infinidad de plataformas y herramientas en la nube dotadas de aquellas características deseables para el aprendizaje y la investigación en el ámbito de los robots.

Primero está el ya mencionado factor económico. Si nos paramos a pensar en el motivo por el cual las plataformas existentes se plantean como un servicio por el que hay que pagar caemos en la cuenta de que, aunque haya una parte debida al coste de desarrollo de dicha plataforma y al valor estratégico del producto que cubre una necesidad global en auge, principalmente se debe al consumo de recursos asociado a cualquier proyecto robótico en cualquier ámbito. Trabajar en robótica es sinónimo de disponer de recursos. Es bien sabido que el *hardware* es caro y frágil, que la implementación tanto del sistema físico como de la lógica programada es temporalmente costosa y que incluso con la ausencia de sistemas reales, el coste computacional es enormemente elevado debido a que están involucrados programas exigentes como los simuladores robóticos y también dada la necesidad de un control estricto y veloz de todos los submódulos que componen un proyecto robótico, cada uno de ellos encargado de una tarea que requiere de más y más capacidad para realizar operaciones por unidad temporal. El primer obstáculo encontrado durante el análisis fue precisamente que **el consumo de recursos de cómputo genera costes**, y que estos costes son elevados.

Segundo, en mi opinión, aprender robótica exige robots. Ya no sólo desde el punto de vista económico sino también desde el logístico, es complicado crear un servicio que disponga de tantos robots como usuarios y que pueda acarrear con los costes derivados de su uso por todo el que lo requiera. Aún pudiendo afrontarlos, construir un sistema que permitiese al usuario, sea estudiante, investigador o simple curioso, evaluar el código de su aplicación sobre un ele-

mento *hardware* remoto y acceder a los resultados en tiempo de ejecución sería una ardua tarea. **Desarrollar aplicaciones robóticas requiere disponer de realimentación, visual y paramétrica, en tiempo real**, sólo alcanzable a través de la observación de sistemas electro-mecánicos reales.

Y tercero, generalmente se desarrolla una gran cantidad de herramientas y plataformas que soportan sistemas específicos, es decir, de funcionalidad acotada. Esto supone un problema para el usuario medio ya que, normalmente, **c carece de hardware específico. Sin embargo, sí que tiene acceso a sistemas más básicos.**

Así las cosas, la conclusión a la que se llegó es que no existen actualmente herramientas que simplifiquen el uso docente de la robótica. Con todo lo anterior y sin perder de vista la idea de “Acercar la robótica a la gente”, el problema radica precisamente en sortear esos obstáculos.

Capítulo 2

Objetivos y Metodología de Trabajo

Este capítulo detallará los objetivos planteados tras el trabajo de análisis inicial que se pretenden alcanzar mediante los periodos de investigación e implementación de esta tesis de fin de Máster. Se describirá brevemente lo que se quiere abordar y cómo se pretende hacer, además de incluir sub-objetivos en base a los cuales hemos articulado el desarrollo. Así mismo, se expondrá la metodología bajo la que se organizó el proyecto.

2.1. Proyecto Planteado

Esta sección se reserva para exponer la idea objeto de esta tesis y la motivación o motivaciones principales que han llevado a su desarrollo, de vital importancia para comprender qué se pretende conseguir con ella y por qué se escogió esta idea y no otra.

La “**Ejecución Mixta**” de una plataforma web para proyectos de robótica y visión artificial es la respuesta que se propone en esta Tesis de Fin de Máster a la conjunción de las ideas planteadas en el capítulo anterior, persiguiendo incre-

mentar la accesibilidad de la robótica y su aprendizaje. Se busca elaborar una herramienta que pueda ser conectada de manera modular con cualquier aplicación de carácter robótico y con soporte para cualquier usuario, sean cuales sean sus circunstancias.

En tanto que se quiere facilitar el uso de la robótica, existirá un Interfaz de Programación o API que abstraiga a los usuarios de todo lo relacionado con la infraestructura de la herramienta y la aplicación, de los mecanismos de comunicación con los sistemas involucrados (tanto robots como sensores y PCs, servidores, *proxys*, etc.) y del control de flujo propio de un sistema complejo, permitiéndole centrarse en su tarea específica de desarrollar con robots. Además, dado que los usuarios potenciales no desean asumir los costes sobre los constructores de las aplicaciones que necesitan usar para su trabajo o estudio y que tampoco quieren que se les cobre la utilización de plataformas, la idea que se expone está orientada al cliente, siendo este el que acarree con toda la carga de cómputo posible y asuma el consumo de recursos, que utilizará a su conveniencia según la disponibilidad de la que goce.

La herramienta final objetivo constituirá una forma de lograr no sólo la ejecución de código específico accediendo a un único elemento *hardware* concreto de la máquina cliente, sino la ejecución de aplicaciones robóticas completas de cualquier ámbito en cualquier dispositivo *hardware* del que el cliente disponga, incluso si éste está conectado a través de un puerto USB o una red WiFi (ya sea un ordenador, sensores, un robot, etc.) mientras el código se almacena remotamente y se ofrece al usuario a través de la nube, por medio de las tecnologías web. Bajo esta filosofía el código o la aplicación desarrollada por el cliente a través de la web se almacena en un servidor remoto, pero éste sigue teniendo acceso a su edición, análisis y resultados al ejecutarse en sus propios equipos y sistemas robóticos locales. Con este diseño todas las tareas de bajo nivel involucradas en la aplicación robótica quedan “escondidas” a nivel de usuario y solventadas por la plataforma que se sirve, como puede ser la comunicación

entre servidor y cliente y el acceso a su *hardware*, el diálogo entre el código implementado y robot (simulado o real) y con lo relacionado con el soporte gráfico (UI) y el entorno de trabajo. De este modo que se dota al usuario de cierta abstracción de los temas que para él o ella son irrelevantes, permitiéndole centrar sus esfuerzos en idear y desarrollar la lógica que controle al robot y le permita abordar el problema que se propone.

La herramienta deberá encajar con aplicaciones docentes y de investigación, actuando más bien como una capa intermedia que hace las funciones de *middleware* entre el robot del usuario y un sistema remoto.

2.2. Objetivos del Proyecto

Este trabajo está compuesto por cuatro grandes objetivos planteados en primera instancia, pudiendo todos ellos descomponerse a su vez en sub-objetivos de menor complejidad que sirven para disponer un escenario de trabajo inicial:

1. Realizar un profundo trabajo de investigación acerca de las herramientas, plataformas, componentes y entornos más utilizados en los proyectos robóticos.
 - Se trata de analizar las características de los más importantes para poder elegir o descartar con criterio su incorporación al proyecto.
 - Probar y seleccionar aquellas herramientas que puedan adaptarse al problema que se pretende resolver.
2. Construir una plataforma web robótica preliminar que engloba un cliente y un sistema remoto que se quieren interconectar.
 - Este hito pasa por ofrecer un interfaz de programación de sencillo uso para la aplicación robótica.
3. Construir una nueva plataforma web que cumpla la “Ejecución Mixta” en forma de módulo que resuelva el problema de ubicación del cómputo, el

cual debe recaer principalmente en el lado cliente, sin más costes adicionales.

- Este proceso supone a su vez la implementación de prototipos,
- nuevas etapas de investigación para el enriquecimiento del sistema y
- continuos procesos de supervisión para asegurar que el sistema resultante contiene los requisitos deseados.

4. Llevar a cabo un exhaustivo programa de pruebas para evaluar la validez y la viabilidad de la idea planteada y su implementación.

2.3. Requisitos

El prototipo final debe incluir al menos las siguientes características:

- Accesible (Multiplataforma, Multilenguaje).
- Soporte para simulación y sistemas reales.
- Ejecución en el cliente.
- Fácil de usar.
- Versátil (Visión Artificial, Robots de cualquier clase).

Éstas constituyen los requisitos de la “Ejecución Mixta”.

2.4. Metodología Empleada

Este proyecto se dividirá, principalmente, en dos partes bien diferenciadas: etapa de investigación y etapa de implementación. Establecemos de antemano una serie de convenios que organizarán la manera en que se trabaja en ambas partes en busca de la mayor eficiencia y efectividad posible.

En cuanto al método de sincronización del trabajo se puede decir que se descompuso su elaboración en una serie de iteraciones formadas por varias fases en las que, por medio de una reunión periódica con los mentores, se pusieron en común los avances obtenidos en cada período para evaluar el estado del proyecto y corregir posibles fallos, además de establecer los sub-objetivos siguientes, discutir la mejor manera de abordarlos y determinar los obstáculos que podían surgir de cara a la siguiente iteración. Esto no sólo constituye un método de trabajo fluido y completo, sino que además permite asentar bien los conocimientos adquiridos y puestos en práctica para resolver cada problema. Gracias a esta rápida realimentación, los errores no se propagan y las dudas se despejan con gran eficacia.

Durante los intervalos de trabajo entre cada reunión, el seguimiento se hizo a través de una bitácora semanal¹ donde se iba actualizando asiduamente el estado del proyecto a través de material audiovisual, *snippets* de código y detalladas descripciones de lo hecho.

Tanto esta bitácora como el código del proyecto, de carácter abierto, estuvo en todo momento accesible para los tutores en un repositorio de *software*, con lo cual se podían preparar las reuniones de antemano e incluso intercambiar ideas y sugerencias de manera anticipada².

2.4.1. Metodología de Investigación

Dado que estamos tratando de abordar un problema para el que aún no existe una solución, gran parte del proyecto se dedicó a la investigación y exploración. Se hizo necesario reunir la documentación asociada a una gran cantidad de herramientas y proyectos en auge que reunían algunas de las características que buscábamos. Para abordar esta parte del trabajo, tratamos de realizar una primera pasada escogiendo todas aquellas opciones que contaran con al menos una

¹<https://github.com/cawadall/TFM-Carlos-Awadallah/blob/master/docs/README.md>

²<https://github.com/cawadall/TFM-Carlos-Awadallah>

de las características que habíamos establecido como objetivo. A partir de este punto, realizamos numerosos ciclos, cada uno de ellos con su respectivo proceso de testeo, para filtrar en primera instancia todas aquellas que fueran poco maduras, incompletas o que no encajasen con los requisitos de nuestro proyecto, y luego se trató de descartar los peores candidatos entre las herramientas finalistas en base a una evaluación del porcentaje de superposición entre el conjunto de prestaciones de la herramienta y las características deseables que habíamos establecido.

Esta filosofía se siguió hasta el comienzo de la implementación, punto a partir del cual cualquier tarea que requiriese volver a investigar se produjo simplemente con la idea de resolver un problema muy concreto o mejorar algún punto del trabajo.

2.4.2. Metodología de Implementación

Para la consecución de los tres últimos objetivos se optó por la filosofía de Barry Boehm del modelo de desarrollo en espiral (Boehm, 1986). Este método busca separar el comportamiento objetivo en diferentes sub-tareas de menor complejidad conservando la flexibilidad ante nuevos objetivos que surjan durante el desarrollo o sucesos no previstos y requisitos variantes, circunstancias que suelen estar presentes en la mayoría de proyectos de este calibre. Así, se consigue fijar la arquitectura y el flujo de trabajo en la fase inicial e ir construyendo la aplicación gradualmente mientras se verifica simultáneamente la calidad en cada paso (Fig. 2.1).

Gracias a la estructura cíclica se consigue tener un prototipo funcional que va creciendo en riqueza y complejidad en cada pasada, construyéndose cada vez sobre el prototipo de la iteración anterior para abordar los objetivos de ciclo, a través de los cuales se consigue mejorar el desarrollo y, en última instancia, pulirlo para cubrir todos los requisitos planteados y obtener la versión final. Cada fase del desarrollo incremental se divide en 4 fases bien definidas por las

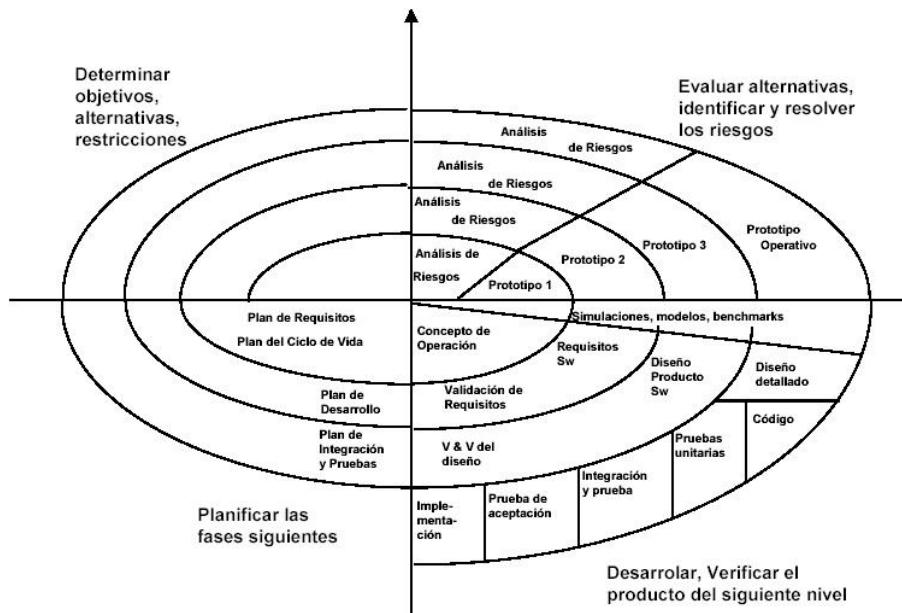


Figura 2.1: *Modelo en Espiral del Proceso de Desarrollo Software.*

que el proyecto debe pasar:

1. **Determinar los objetivos del ciclo.** Esta primera fase consiste en definir las metas que se pretende alcanzar en el ciclo. Estas metas son a su vez las que determinan cuándo se da por finalizada la iteración.
2. **Análisis del riesgo.** En segundo lugar se trata de analizar los objetivos propuestos en búsqueda de las partes más delicadas para determinar qué problemas u obstáculos pueden surgir durante el desarrollo y planificar cómo abordarlos.
3. **Desarrollar y probar.** Con un plan de desarrollo definido en base a los riesgos potenciales, y con los objetivos de ciclo presentes, se comienza con la implementación. También ha de pasarse la correspondiente batería de pruebas para garantizar un desarrollo funcional y completamente operativo.

4. **Planificación de la siguiente fase.** Para acabar, se evalúan los resultados obtenidos en la etapa actual del proyecto y se comienza con la planificación de las próximas fases del proyecto.

Capítulo 3

Infraestructura Utilizada

Este capítulo recoge el conjunto de herramientas seleccionadas para este proyecto, así como una detallada descripción de la relación entre unas y otras. Se introducirá cada una de las herramientas y se especificará tanto las razones de su elección para el proyecto como la función que cumplirá en él.

Lo que se recoge a continuación es el resultado de los meses de investigación, en los cuales estudiamos numerosas herramientas, entornos y plataformas aplicables a la robótica web que podían aportar algo a nuestros intereses. Finalmente se escogieron los que se listan en este tercer capítulo para formar parte de la solución de “Ejecución mixta” para aplicaciones robóticas que proponemos en este Trabajo Fin de Máster.

3.1. Proyecto Jupyter

La herramienta fundamental en la que se basa este desarrollo es el proyecto Jupyter¹, la evolución de *IPython 3.0*. Se trata de una aplicación web de código abierto que ofrece la capacidad de crear y compartir documentos, llamados cuadernillos o *Notebooks*, los cuales puede contener textos narrativos y elemen-

¹<https://jupyter.org/>

tos de texto enriquecido (párrafos, ecuaciones, figuras, enlaces, etc.), imágenes ilustrativas, códigos empotrados, ecuaciones, visualizaciones (figuras, gráficos, tablas, etc.) y gran cantidad de elementos gráficos inteligentes. La razón principal de su elección, además de por pertenecer al ámbito de las tecnologías web, es que además de empotrar y editar código el usuario puede ejecutarlo desde su navegador, gracias a la comunicación implícita de la aplicación con los núcleos o *kernels* de computación que incluye el sistema, incluyendo intérpretes para distintos lenguajes. Con ello se dispone de una herramienta a través de la cual se pueden llevar a cabo potentes tareas como la transformación de datos, el modelado estadístico de los mismos, simulaciones numéricas, visualización y análisis de los datos y aprendizaje automático, siendo todos ellos sus principales usos, aunque su funcionalidad es mucho más amplia.

La característica principal del entorno son sus *Notebooks*, documentos generados a través del interfaz de usuario que pueden contener todo lo anteriormente mencionado, completamente legibles por usuarios humanos pero con capacidad para ejecutar código de computadora en hasta 100 lenguajes diferentes para los que da soporte en la actualidad. Básicamente consisten en una secuencia lineal de celdas que pueden ser de diferentes tipos, dependiendo de su contenido:

- Celdillas de código: entrada y salida del código en el lenguaje seleccionado para el *Notebook* que se ejecuta en el *kernel*. Estas celdas son extensibles para poder visualizar el resultado de la ejecución del código especificado.
- Casillas de reducción: contienen texto narrativo con ecuaciones en formato LaTeX embebidas.
- Encabezado de celdas: texto de 6 niveles de organización jerárquica y formato. Útiles para establecer títulos y subtítulos y obtener un documento mucho más claro y ordenado.
- Celdas sin formato: texto sin formato específico para ser exportado a diferentes formatos mediante *nbconvert* sin sufrir cambios.

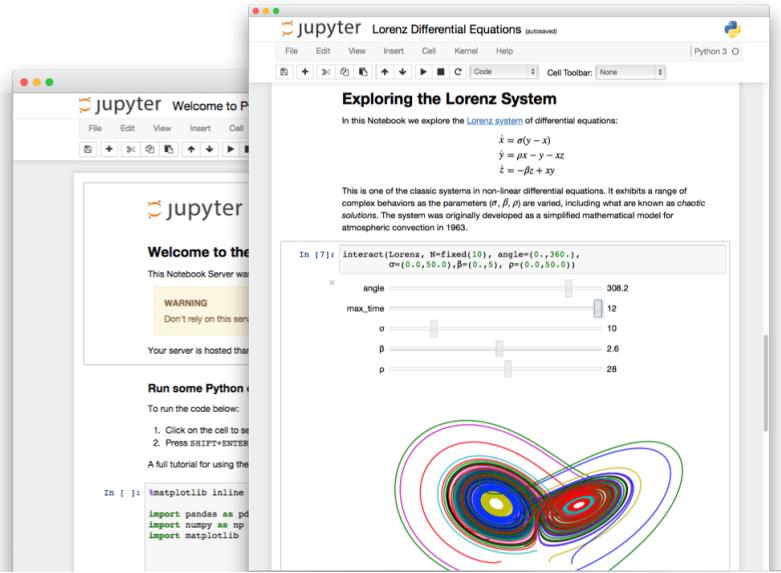


Figura 3.1: Cuadernillos de Jupyter

Se crean cuadernillos mediante la combinación de los elementos anteriores a través del navegador web, y la aplicación servidor-cliente permite editarlos y ejecutarlos dinámicamente. Para utilizar el servicio, se debe instalar su *software* en un escritorio local y lanzar la aplicación, supuesto en el que no se requiere conectividad de red, o bien puede instalarse en un servidor remoto y utilizar Internet para el acceso. Además de mostrar, editar y ejecutar *Notebooks*, el entorno dispone de un panel de control compuesto de una serie de instrumentos que ejercen alguna clase de acción sobre el cuadernillo, entre los que se encuentran las funciones de abrir y cerrar documentos, levantar, borrar o reiniciar un núcleo, cambiar el formato de una celda concreta o sus meta-datos, etc.

Quizá la parte más importante de esta aplicación son los *kernels* que se han mencionado. Se trata de “motores computacionales” que ejecutan el código que el usuario escribe en el cuadernillo. Corrigen la sintaxis del lenguaje y devuelven una serie de instrucciones en código máquina que pueden ser ejecutadas, evalúan su resultado y lo disponen para que el usuario tenga acceso a él a través

del interfaz, tal y como haría un intérprete en el caso de los lenguajes interpretados y la secuencia de un compilador y un ejecutor en el resto de casos. Cuando se inicia un *Notebook*, el *kernel* asociado (especificado automáticamente en los meta-datos del documento) se levanta automáticamente, de manera que al ejecutar cada celdilla con código éste realiza el cálculo y produce los resultados. Dependiendo del tipo de cálculos, el *kernel* puede consumir CPU y RAM significativas (siempre de la máquina en la que se está ejecutando la aplicación), pudiendo ejecutar procesos más exigentes y desarrollar aplicaciones de mayor complejidad. Finalmente, los cuadernillos pueden ser guardados por el usuario en su sistema de ficheros local, como un documento con extensión *.ipynb*.

Se han mencionado varias características muy interesantes de esta herramienta que encajan a la perfección con nuestros propósitos. En primer lugar, el *kernel* aprovecha toda la capacidad de cómputo del sistema del cliente en que se levanta la aplicación, es decir, puede utilizar su sistema a su conveniencia en función del código que ejecuta. Esto significa que tiene la capacidad de acceder al *hardware* del usuario y de actuar sobre él, justo lo que necesitamos para el proyecto que se presenta. Además, tal y como se especifica en la documentación, el interfaz de Jupyter puede ser accedido remotamente a través de un navegador si se conoce su origen. Eso supone que, aunque el código se está ejecutando en cliente, un agente remoto puede actuar sobre el documento y enviar la orden para que se ejecute de algún modo. Mediante la combinación de estas dos características obtenemos una herramienta que puede actuar de editor de texto para el código del usuario, y que proporciona herramientas para el acceso a su *hardware* de modo remoto, siempre y cuando sea el usuario quien escribe el código.

Los *Notebooks* que ofreceremos en este TFM se basarán en código Python, versiones 2.7 o 3.5 según las necesidades. Hemos escogido este lenguaje dadas sus amplias ventajas en accesibilidad (mismo intérprete en distintos SO), su versatilidad, su potencia y su fácil adaptación con los mecanismos robóticos, siendo

que existen numerosas librerías estándar en robótica con soporte para Python.

3.2. Plataforma Docker

Docker² es una plataforma integral de código abierto para desarrollar, enviar y ejecutar aplicaciones en un entorno virtualizado ligeramente aislado, de manera que aprovecha todas las ventajas y funciones que pone a su disposición un sistema operativo sin requerir todas las dependencias que en primera instancia hacen falta para una virtualización completa como la que se puede encontrar en la tecnología de máquinas virtuales. Docker permite separar la aplicación final de la infraestructura de desarrollo, de tal manera que el *software* se puede compartir rápidamente. Incluso ofrece la capacidad de gestión de una infraestructura completa reduciendo significativamente el tiempo que transcurre entre la desarrollo de la aplicación y su puesta en marcha en un entorno de producción, eliminando los obstáculos que pueden surgir al llevar el código de una máquina a otra. Esto se consigue a través de unidades estándar de software capaces de empaquetar el código de una aplicación y todas sus dependencias para que se ejecute de forma rápida y fiable de un entorno informático a otro. Estas unidades reciben el nombre de contenedores Docker.

Los contenedores garantizan aislamiento y seguridad, lo que permite ejecutar varios contenedores simultáneamente en un *host* determinado sin que se produzcan colisiones y protege el sistema anfitrión sobre el que se monta el contenedor. Además, como ya se ha dicho, son ligeros porque no necesitan la carga extra de un hipervisor, sino que se ejecutan directamente en el *kernel* de la máquina anfitriona, lo que también supone que se superan los largos y tediosos procesos de instalación de la aplicación, sustituyéndose por una ligera descarga. La versatilidad de estas unidades de computación es muy elevada, permitiendo combinaciones complejas como construir una unidad de contenedor sobre un

²<https://www.docker.com/>

anfitrión que es una máquina virtual, sobre un centro de datos local, un proveedor de *cloud computing* o incluso un híbrido entre ambos (Fig. 3.2).

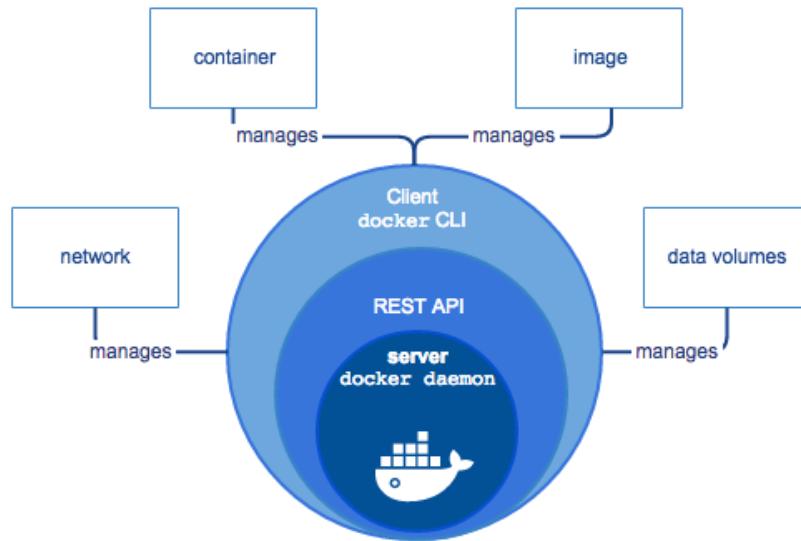


Figura 3.2: Estructura de Docker

Así, se puede decir que utilizar Docker multiplica exponencialmente la escalabilidad y portabilidad de una aplicación, ya que no hay que prestar atención al sistema operativo latente en el anfitrión, pues la infraestructura del contenedor configura el entorno que la aplicación necesita. Consideramos que esta característica puede resolver toda la parte de accesibilidad a la herramienta que queremos construir, en tanto que se pretende que ésta sea un cliente que se comunica con una aplicación robótica remota y que necesita disponer de ciertos recursos y ejecutar ciertas tareas en el sistema del usuario.

La necesidad de usar esta plataforma surgió a medida que avanzó el proyecto. Se usará en sustitución de la instalación en el ordenador del cliente de la parte de la plataforma web propuesta que es necesaria para que se ejecute en el ordenador del usuario el cómputo de las aplicaciones robóticas. Persiguen-

do las ventajas de un proyecto multiplataforma e independiente del sistema del cliente, investigamos Docker para mejorar la experiencia de usuario al evitar que éste tenga que llevar a cabo procesos de instalación, en tanto que podríamos disponer uno con todo lo necesario para que el cliente disfrute de las funciones para aplicaciones robóticas que pueda necesitar desde un entorno predispuesto y debidamente configurado a través de un contenedor.

3.3. Entorno Django

Django³ es un marco de trabajo clasificado como tecnología de servidor de alto nivel en Python que garantiza el desarrollo rápido del lado servidor de una aplicación y su diseño. Ofrece gran cantidad de funcionalidad resuelta que se encarga de los detalles del desarrollo Web. Con Django se puede construir una aplicación reutilizando módulos de código con la funcionalidad típica de una aplicación, de tal manera que el desarrollo puede centrarse en escribir la lógica específica del servicio que quiere ofrecer sin necesidad de empezar desde el principio programando cada módulo. Es gratuito y de filosofía abierta.

La arquitectura de Django resulta bastante simple. Como se puede ver en la Figura 3.3, organiza la aplicación de tal manera que resulta muy sencillo conocer en todo momento el estado del proceso de servicio.

Se controla muy fácilmente lo que el usuario ve a través de las vistas de Django, con las que además se dota a la aplicación web de dinamismo e inteligencia. También se controla cómo ve el usuario el contenido de la aplicación a través de las plantillas, que proporcionan herramientas de interacción entre el usuario y el servidor y le permiten realizar peticiones simples para potenciar la funcionalidad que se puede ofrecer. Por último facilita mecanismos muy simples de gestión y acceso a bases de datos a través de distintas tecnologías como MySQL o MongoDB, necesarias para las aplicaciones y que evita la especialización en lenguajes utilizados para peticiones a bases de datos por parte del desarrollador

³<https://www.djangoproject.com/>

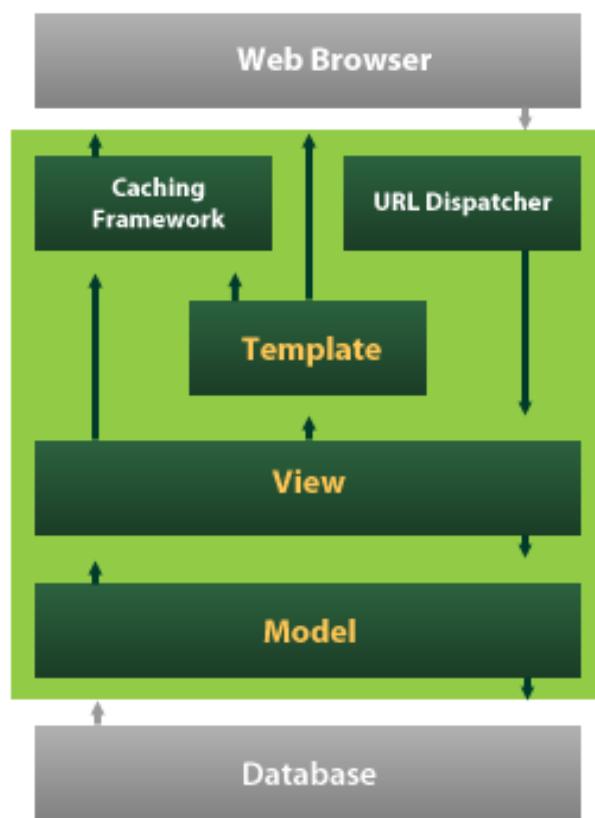


Figura 3.3: Arquitectura de Django

web. En resumen, facilita el proceso de construcción y diseño de aplicaciones web para que el desarrollador no tenga que preocuparse por el bajo y medio nivel.

Esta característica, junto con la escalabilidad del lado servidor construido con tecnologías web de servidor, es precisamente la que decantó la balanza hacia el uso de Django en nuestra plataforma. La aplicación web resultante será enriquecida con la herramienta de “Ejecución Mixta” para ofrecer cierto servicio robótica docente. La aplicación remota basada en Django ofrecerá la funcionalidad para la que ha sido diseñada sin consumir recursos en el sistema anfitrión, y llevará a cabo el proceso de “Ejecución mixta” que queremos conseguir para garantizar el acceso a cualquier cliente.

3.4. ROS (Robot Operating System)

ROS (Robot Operating System)⁴ es un *middleware* flexible ideado para desarrollar *software* robótico. Proporciona multitud de bibliotecas, paquetes de código y herramientas para ayudar a los desarrolladores de aplicaciones relacionadas con la robótica en todos sus ámbitos a crear aplicaciones de robots. Su uso está muy extendido por todo el mundo, desde caseros a profesionales, dadas sus ventajas frente a otros soportes similares. Su extensión se debe en parte a que surgió en una etapa en la que no existía un entorno abierto que unificara todos los módulos necesarios para construir una aplicación robótica completa, y en la que tampoco existía *software* estándar para la programación de robots. En esos años el desarrollo de un proyecto en muchos casos se tornaba arduo, además de la complejidad presente a la hora de extenderlo, incluir herramientas o funcionalidad externa o incluso al agrandar el grupo de desarrollo, en el que cada uno podía tener su método de trabajo. ROS alivia todos estos problemas al ofrecer los módulos necesarios para implementar el código y llevárselo al robot, de principio a fin, todo en un mismo sitio distribuido bajo una licencia BSD de

⁴<https://www.ros.org/>

código abierto. Permite la abstracción de la capa *hardware*, proporciona controladores de dispositivo (*drivers* o *plugins*), bibliotecas, visualizadores, resuelve el intercambio de mensajes y facilita la administración de paquetes entre muchas otras cosas.

Resulta bastante lógica su elección para este TFM, en el que precisamente queremos lograr esa abstracción del bajo nivel para ofrecer una plataforma de desarrollo de aplicaciones de alto nivel para que el usuario se sienta cerca de la robótica. Cabe destacar también que ROS ofrece soporte para distintos lenguajes, entre ellos Python, y que dispone de canales de comunicación automáticos configurables y parametrizables que simplifican en gran medida la comunicación con el *hardware* y con aplicaciones externas.

En esta línea destaca otra de las fortalezas de ROS, que es su integración con el simulador Gazebo mediante la serie de paquetes *gazebo_ros_pkgs*⁵ que utilizan estructuras de tipo *ROS Messages* para permitir la simulación de robots y ofrecer servicios sobre la simulación como la reconfiguración dinámica. Esta ventaja puede ser muy útil para el soporte de simulación que queremos incluir en la solución de “Ejecución Mixta”, para aquellos usuarios que no disponen de sistemas electro-mecánicos reales. ROS provee de los módulos *software* que permiten controlar las interfaces de los robots simulados pero también reales, proporcionándonos acceso a sus sensores y control sobre sus actuadores.

ROS propone un desarrollo basado en la estructura de la aplicación organizada como una colección de nodos (procesos que conllevan computación) que se jerarquizan y se combinan en un grafo comunicándose entre ellos mediante *topics* o canales de transmisión, servicios RPC y el Servidor de Parámetros. Con esto, podemos ejercer un control específico y restrictivo sobre cada módulo implicado en el sistema de control de un robot, que generalmente comprenderá muchos, y ofrecer al usuario sólo aquellos que necesita en cada ocasión para facilitarle el desarrollo. El empleo de nodos en ROS proporciona beneficios como

⁵https://wiki.ros.org/gazebo_ros_pkgs

la tolerancia adicional a errores (quedan contenidos en nodos individuales) y la disminución de la complejidad del código.

En cuanto a los *topics* como medio de comunicación, se definen como buses que los nodos utilizan para intercambiar mensajes con formatos específicos. Tienen una semántica basada en la publicación de mensajes y/o suscripción anónima a un canal concreto o varios, que desacopla el cómo o quién genera cierta información del quién o qué la consume. Con ello, los nodos no tienen por qué saber la fuente o el receptor de sus datos, sino que simplemente adquieren aquello que necesitan y divulgan lo que puede resultar útil para cumplir otras funciones según lo que se necesite en cada momento. De esta manera puede haber muchos consumidores de una misma fuente, e incluso se puede obtener y difundir datos desde el mismo nodo.

Usaremos ROS para resolver todo el bajo nivel robótico de nuestra plataforma, así como de *middleware* de comunicaciones entre el robot (real o simulado) y el código del usuario, que además dispondrá de un API público programado en Python que le proporcione la abstracción mencionada y le facilite el envío de órdenes y la recepción de resultados del sistema robótico.

3.5. Simulador Gazebo

Gazebo⁶ es un simulador cuyo uso está muy extendido en el campo de la robótica. Para poder evaluar el código destinado a un robot o sistema en un paso previo al de cargar dicho código en él, este simulador ofrece la capacidad de emular diversos escenarios tridimensionales customizables para robots autónomos. Está diseñado por ejemplo para probar lógica de elusión de objetos y algoritmos de visión artificial, aunque su uso es extensible a cualquier tipo de implementación.

Para nuestro proyecto era necesario disponer de soporte de simulación, dado que la intención es ofrecer un interfaz que funcione con robots reales y simula-

⁶<http://gazebosim.org/>

dos para ofrecer la robótica a usuarios con y sin recursos. Además, especialmente en un proceso de aprendizaje o investigación en el que no se es experto en la materia tratada, se hace necesario probar el *software* desarrollado antes de correr el riesgo de utilizarlo en un sistema real, que suele ser frágil y costoso en este campo, e incluso peligroso para los usuarios según el diseño mecánico del sistema y la tarea a realizar. El uso de simuladores en robótica evita la implementación de costosas y poco útiles pruebas de código en *hardware* real en las primeras fases del desarrollo. Ofrecer este “*hardware virtual*” permite evaluar y supervisar el desempeño de la lógica en todo momento, agilizando el desarrollo y abaratando costes, y abre el abanico de usuarios que pueden acceder a la robótica. Estas son las razones principales de su elección como simulador para este proyecto, además del hecho de que es de código abierto.

Existen muchos otros simuladores con características similares como Webots, V-Rep, Morse, etc. que hacen que Gazebo esté lejos del estándar de simulación robótica. A continuación veremos otras características que hemos analizado para elegir Gazebo y no otro en relación con el uso final que se va a hacer de él: docente, investigativo o formativo.

La característica más importante es su versatilidad, ya que puede simular robots, objetos y los sensores actuales más utilizados en entornos complejos de interior y exterior con multitud de elementos realistas para los robots. Es de los pocos simuladores que incluyen elementos que pueden emular problemas reales como ruido sensorial producido por superficies reflectantes, o la inclusión de texturas que el robot puede detectar, entre otras cosas. Además, posee un interfaz de gran calidad.

Por otro lado, cabe mencionar su robusto motor de físicas⁷ (Fig. 3.4), creado por Russel Smith, que permite caracterizar elementos del robot como su masa, el rozamiento al que se somete, su inercia, amortiguamiento, etc. Con ello y los mecanismos que proporciona para conectar los cuerpos entre sí formando

⁷<http://opende.sourceforge.net>

relaciones cinemáticas y dinámicas, y las propiedades de color, textura y transparencia que se pueden incluir, se consigue diseñar modelos de simulación con visualizaciones muy realistas a través de OpenGL⁸ que se adaptan bien a situaciones reales (Fig. 3.5).

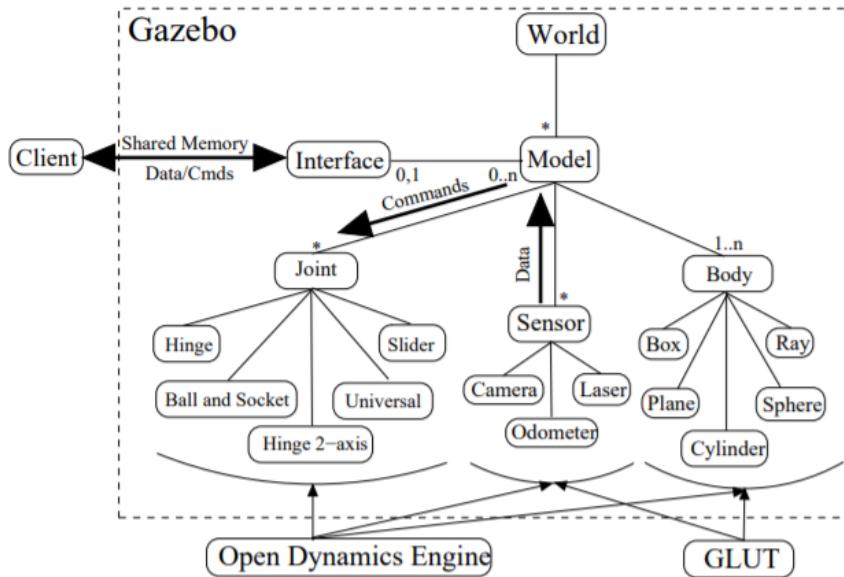


Figura 3.4: Motor de físicas de Gazebo



Figura 3.5: Modelos de Simulación en Gazebo

La configuración de mundos 3D con Gazebo se describe en ficheros con extensión “.world”, que son documentos de texto en formato de marcado XML (Ex-

⁸<https://www.opengl.org/>

tensible Markup Language) de documentos, con etiquetas definidas en el lenguaje *Simulation Description Format* (SDF), donde se recogen todos los elementos del escenario (luz ambiente, propiedades del cielo, sombras, conjunto de modelos, *plugins*, propiedades físicas y temporales etc.).

Aunque la versión 7 de Gazebo incluye un editor de modelos muy básico con el que se pueden crear robots y mundos simples, el simulador acepta la importación de modelos complejos creados con programas de modelado como Blender o Sketchup. A partir del modelo, se especifica un *plugin* asociado al mismo para recoger y publicar la información de los sensores, y para enviar órdenes a los actuadores y dotar al robot simulado de inteligencia e interacción. Es aquí donde se materializa la relación entre el simulador y el sistema de comunicaciones de ROS, que hará las funciones de mediador entre la aplicación y el robot simulado en este caso.

Por último, es importante destacar que cuenta con un cliente web, llamado GzWeb que permite acceder al interfaz de simulación desde un navegador web. Se trata de un cliente con tecnología WebGL que se comunica con un proceso de Gazebo para conseguir que la simulación sea multiplataforma y su acceso no requiera instalación, características ideales para el proyecto.

3.6. Biblioteca OpenCV

En primera instancia, la plataforma que planeábamos diseñar perseguía la ampliación del acceso al campo de la Visión Artificial de la robótica, en tanto que los sensores necesarios (cámaras, ToF, sensores de profundidad) son los más extendidos entre los usuarios de cualquier clase. Es por eso que, aunque la funcionalidad finalmente se extendió para que la plataforma soportase cualquier ámbito de la disciplina robótica, merece una mención especial la tecnología utilizada para el control de los sensores de imagen: las bibliotecas OpenCV⁹.

⁹<https://opencv.org/>

OpenCV es una librería de código abierto desarrollada inicialmente por Intel y publicada bajo licencia BSD. Incluye gran variedad de herramientas para el tratamiento digital de la imagen y el aprendizaje máquina. Su propósito principal es facilitar la programación de aplicaciones en tiempo real de visión por computador mediante la disposición de módulos de lógica resuelta que implementan alguna técnica de procesamiento o análisis de la imagen (Fig. 3.6).



Figura 3.6: Herramientas de procesado con OpenCV: Segmentación de caras.

Se escogió esta librería por ser el estándar en lo relativo a la visión artificial, y se usará tanto para obtener acceso al *hardware* que proporcione la fuente de fotogramas como para utilizarse en la aplicación robótica que se creará para tratar las imágenes.

3.7. Lenguajes de Programación: Python, JavaScript

En cuanto a los lenguajes de programación escogidos para la implementación, se hace necesaria una sub-división en dos ámbitos:

- Lenguaje de Aplicación y de Usuario. Es necesario escoger un lenguaje para programar toda la lógica de la aplicación a ofrecer. También se hizo necesario ofrecer un lenguaje al usuario para que aborde su desarrollo robótico.
- Lenguaje de Infraestructura. El contexto de tecnologías web en el que se encuentra la “Ejecución Mixta” requiere de la elección de un lenguaje que

pueda hacer que los distintos agentes colaboren entre sí, principalmente mediante protocolos.

Así, el primer lenguaje escogido ha sido Python¹⁰, como lenguaje de aplicación y usuario. En primer lugar, la elección se debe a la compatibilidad del lenguaje con la infraestructura detallada en este capítulo. Tanto Gazebo como ROS, OpenCV, Jupyter y Django ofrecen interfaces Python. Por otro lado están las atractivas características del lenguaje, como sus estructuras de datos integradas de alto nivel, su apariencia intuitiva, y la combinación del tipado y el enlace dinámicos, que hacen al lenguaje idóneo para desarrollar aplicaciones rápidamente. Además, su sintaxis resulta simple y fácil de aprender y enfatiza la legibilidad, lo que a grandes rasgos reduce el costo del mantenimiento del programa. Además fomenta la modularidad del programa y la reutilización de código, lo que lo convierte en un lenguaje de programación muy adecuado para el aprendizaje. Eliminando las peculiaridades que pueden hacer a un lenguaje difícil de tratar, conseguimos que el usuario se centre en su objetivo de carácter robótico y evite los dilemas y contratiempos relacionados con la programación.

En cuanto al lenguaje a usar para la infraestructura de la herramienta, el entorno web hace que la elección de JavaScript¹¹ resulte prácticamente obvia. Desde el 2012, todos los navegadores modernos soportan completamente ECMAScript 5.1¹², el estándar de JavaScript. Por tanto, este lenguaje va a permitir realizar actividades complejas en páginas web, entorno en el que deberá operar la “Ejecución Mixta” para aplicaciones robóticas. Este entorno permitirá aprovechar los mecanismos y protocolos de comunicación de Internet para resolver el intercambio de mensajes entre nuestra herramienta, la aplicación robótica, el navegador y el cliente.

¹⁰<https://www.python.org/>

¹¹<https://developer.mozilla.org/es/docs/Web/JavaScript>

¹²https://developer.mozilla.org/en-US/docs/Web/JavaScript/Language_Resources

Capítulo **4**

Plataforma Web para Enseñanza en Visión Artificial y Robótica

En este capítulo se explican el diseño y la implementación de la plataforma web educativa que se propone como solución a los objetivos planteados en este TFM, detallando minuciosamente cómo se alcanzó cada uno de los hitos mencionados y aportando la explicación técnica de los problemas encontrados y las soluciones propuestas.

Se comenzará por describir el diseño de la plataforma robótica a través de la web en cuestión que simplificará el acceso de los usuarios al mundo de la robótica. Se especificarán los puntos desfavorables que desembocan en la necesidad de un mecanismo que resuelva, además de lo anterior, tanto los posibles defectos como la ubicación del peso del cómputo. Para el primer paso se especificará la arquitectura de cada uno de los módulos que componen el servidor remoto y su interfaz de abstracción, para luego desembocar en el diseño y arquitectura finales de la “Ejecución Mixta” dentro de la herramienta que aportan el mencionado cómputo en el cliente y la escalabilidad de la aplicación web.

4.1. Servidor Web en Ejecución Remota

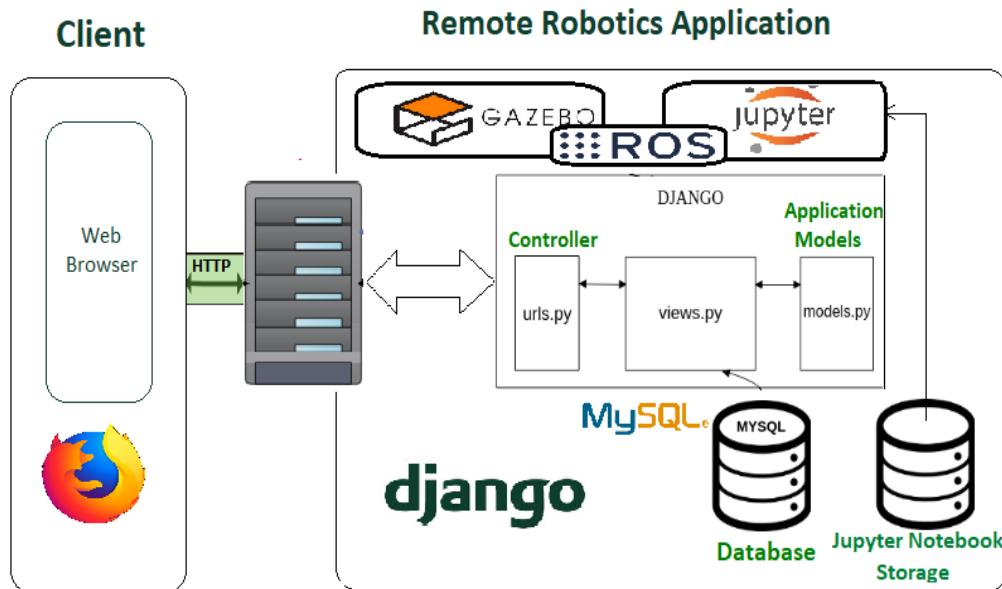


Figura 4.1: Arquitectura del Servidor en Ejecución Remota.

Del mismo modo que no se puede comenzar la casa por el tejado, no podemos empezar con la implementación del mecanismo de la “Ejecución Mixta” dentro de la plataforma sin un servidor al que conectarla, lo que también servirá para sacar a la luz las carencias o puntos desfavorables relacionados con el servicio web aplicado a la robótica.

4.1.1. Diseño

El diseño del contenido prototípico que se quería servir a través de la plataforma web robótica utiliza Django como tecnología de servidor, Jupyter y Gazebo como herramientas y sus versiones web para formar el interfaz de usuario y Apache como envoltura para entornos de producción. A fin de ilustrar el funcionamiento y la orientación de la herramienta, decidimos organizar el servicio como una aplicación web para el aprendizaje de distintos ámbitos de la robótica

por medio de ejercicios con contenido académico.

Como todo servicio web, está ubicado en un punto físico que resulta siempre remoto al usuario que accede a él, es decir, que se ubica en una red distinta a la del usuario que accede a Internet. Por tanto, este servicio web se basa en el clásico entorno remoto en el que es el lado servidor el que carga con el peso computacional del servicio que ofrece, y el cliente accede a él de manera muy sencilla a través de su navegador web.

Para el diseño de la funcionalidad hemos recurrido a los referentes en aplicaciones web de este tipo, concretamente a la plataforma JdeRobot-Academy, para partir de una organización inicial.

4.1.2. Back-end del Servicio

Como se puede ver en el esquema de arquitectura del servicio web (Fig. 4.1), se ha utilizado Django y su gran cantidad de librerías relacionadas con el servicio web y el acceso y gestión de bases de datos para Python para construir el *back-end* o infraestructura del servidor remoto.

Como parte de esta estructura de bajo nivel se diseñaron una serie de plantillas HTML que actúan como interfaz de usuario que están alimentadas por variables Python que provienen de las vistas o *views* de Django, las cuales se encargan de la gestión y el procesado de todas las peticiones que los usuarios de la aplicación generan a través del interfaz. La parte más importante de estas vistas es la transformación de las peticiones recibidas en este caso en ejercicios concretos de visión artificial y robótica. Para ello es necesario proveer al servidor con ficheros estáticos que harán las funciones de infraestructura de cada ejercicio, con el fin de actuar como capa de abstracción para que el usuario tenga un API sencillo de utilización del ejercicio y las herramientas utilizadas en el *front-end* puedan transformar sus interacciones en acciones concretas sobre el ejercicio y sus agentes simulados o reales.

Cabe destacar que el servicio implementado cuenta con muchos módulos

internos encargados de las herramientas básicas de usabilidad y experiencia de usuario, como el inicio de sesión y la gestión de sesiones activas, en las cuales no se va a profundizar al no ser objeto específico de este proyecto.

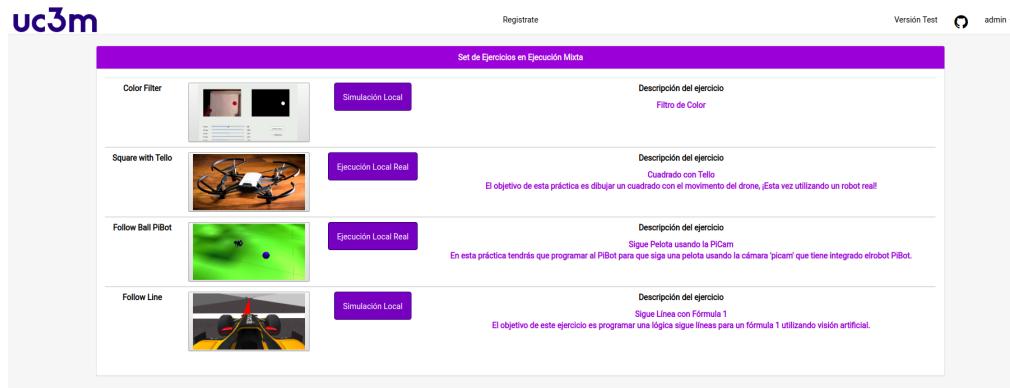


Figura 4.2: UI de la Plataforma Web en Django

Como parte del *back-end* encargado del soporte de servicio, se contará tanto con un servidor de Jupyter como con un servidor de Gazebo (*GzWeb*) que estarán en constante ejecución durante la solicitud y desarrollo de un ejercicio académico.

Recordemos que Jupyter es una plataforma de escritorio (requiere instalación) con capacidad de visualización a través de la web. El motor computacional de Jupyter requiere del uso de un *kernel* físico del equipo para llevar a cabo sus funciones, y luego comunica los resultados al interfaz web. Por tanto, el procesamiento de peticiones cuenta con señales de control que disparan el levantamiento de un servidor remoto de Jupyter (local al servidor web) que será el encargado de ejecutar el código del ejercicio del usuario. Como se puede ver, el contexto inicial es puramente local y supone asumir que el código a ejecutar para el ejercicio robótico proviene de la misma máquina que lo sirve, y por tanto se va a encargar de todo el cómputo.

Por otro lado, la simulación a través de Gazebo sucede en un escenario análogo al de Jupyter, ya que también se trata de una aplicación de escritorio con

interfaz web. Por tanto, también ha de levantarse un servidor de Gazebo que atienda a las peticiones de simulación. Todo lo anterior supone un grave problema atendiendo a la escalabilidad y la eficiencia computacional, dado que la conjunción de varios usuarios simultáneos generaría una carga de cómputo no asumible para el servidor, que para dar soporte a cada uno debería levantar una instancia de ambas aplicaciones.

En cuanto a la conexión entre ambas partes, simulador y código del usuario, utilizamos ROS como *middleware* de comunicaciones, dada su fácil integración con Gazebo y el lenguaje Python. Esto permite construir una aplicación completa y autocontenido asociada al código del ejercicio que creará el mecanismo de inter-comunicación a través de mensajes de ROS (*ROS Messages*) que se intercambian por canales o *topics* de publicación y suscripción de información (Listing. 4.1). De esta manera los resultados de las instrucciones que el usuario coloque en su código se reflejan en la simulación.

Listing 4.1: Topics de ROS asociados a los Canales de Comunicación Internos

```
~$ rostopic list
/F1ROS/cameraL/camera_info
/F1ROS/cameraL/image_raw
/F1ROS/cameraL/parameter_descriptions
/F1ROS/cameraL/parameter_updates
/F1ROS/cmd_vel
/F1ROS/odom
/clock
/gazebo/link_states
/gazebo/model_states
/gazebo/parameter_descriptions
/gazebo/parameter_updates
/gazebo/set_link_state
```

```
/gazebo/set_model_state  
/rosout  
/rosout_agg  
/tf
```

De nuevo, todo este mecanismo añade demasiada carga al lado servidor, que empieza a requerir soporte computacional adicional para dar servicio, como granjas de ordenadores de soporte o procesadores adicionales. Además, cabe destacar también que el uso de un servicio web remoto como el descrito está limitado a utilizar robots simulados o bien reales conectados al servidor. Por ejemplo en el uso de una cámara, el ejercicio que se construya estará supeditado al tipo de *hardware* del que se disponga en el lado servidor, al entorno de servicio en el que la cámara será estática (lo que limita la riqueza de ejercicios que se pueden construir al estar siempre enfocando al mismo entorno), y en todo caso imposibilita el empleo de los robots de los que los usuarios puedan disponer.

Para poner este servicio en producción y que los usuarios puedan acceder a él es necesario el empaquetamiento de toda la estructura en una envoltura de servidor HTTP en producción dado que Django no ofrece por sí mismo este soporte. La tecnología Apache interviene para cumplir esta función. Se trata simplemente de una capa de servidor HTTP orientado a dar un formato correcto a todas las peticiones generadas desde el servicio web y destinadas a él para que puedan viajar por Internet sin percances. Aporta también mecanismos extra de seguridad para proteger el servicio frente a las intrusiones y permite configurar una capa de TLS a través del protocolo SSL y el uso de certificados de encriptación de información.

Todo ello se refleja en los ficheros de configuración de servicio de Apache, compuestos por directivas que indican el tipo de cabeceras HTTP que deben llevar los mensajes, los *endpoints* donde la aplicación web espera a recibir peticiones, la ubicación de los ficheros estáticos y la configuración básica de seguridad, permisos de actuación y servicio.

4.1.3. Front-end del Servicio

Una vez preparada la estructura básica de gestión de peticiones, se construye encima de la misma la capa de usuario o *front-end* de la aplicación para los ejercicios. El agente encargado de disponer el interfaz de usuario será el navegador web del cliente. Este interfaz se basa en un cuadernillo de Jupyter que permita al usuario, desde el navegador, editar su código fuente y visualizar el resultado de una ejecución de su código Python. Dado que los ejercicios emplearán algún tipo de robot o sensor, también se incluye en la interfaz una sección de simulación con GzWeb o divisiones auxiliares que contendrán la información necesaria para que el usuario pueda depurar y resolver el ejercicio, como visualizadores de imágenes de una cámara.

Así, la interacción del usuario se realiza sobre un *Notebook* de Jupyter almacenado en el servidor cuya función es actuar como interfaz de edición de la aplicación (IDE), que envía mensajes al *kernel* del servidor para que compute.

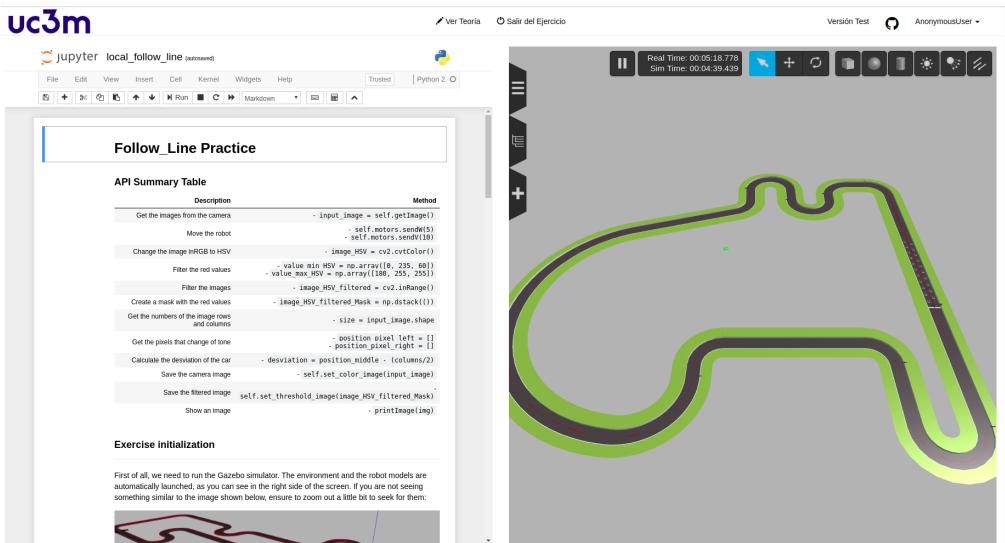


Figura 4.3: UI de Simulación de la Aplicación

Se puede ver que el cliente web no asume ninguna carga de procesamiento, sino que sólo cuenta con un navegador que en todo momento solicita la visua-

lización de varias aplicaciones que están corriendo coordinadas en el lado del servidor web.

4.2. Servidor Web en “Ejecución Mixta”

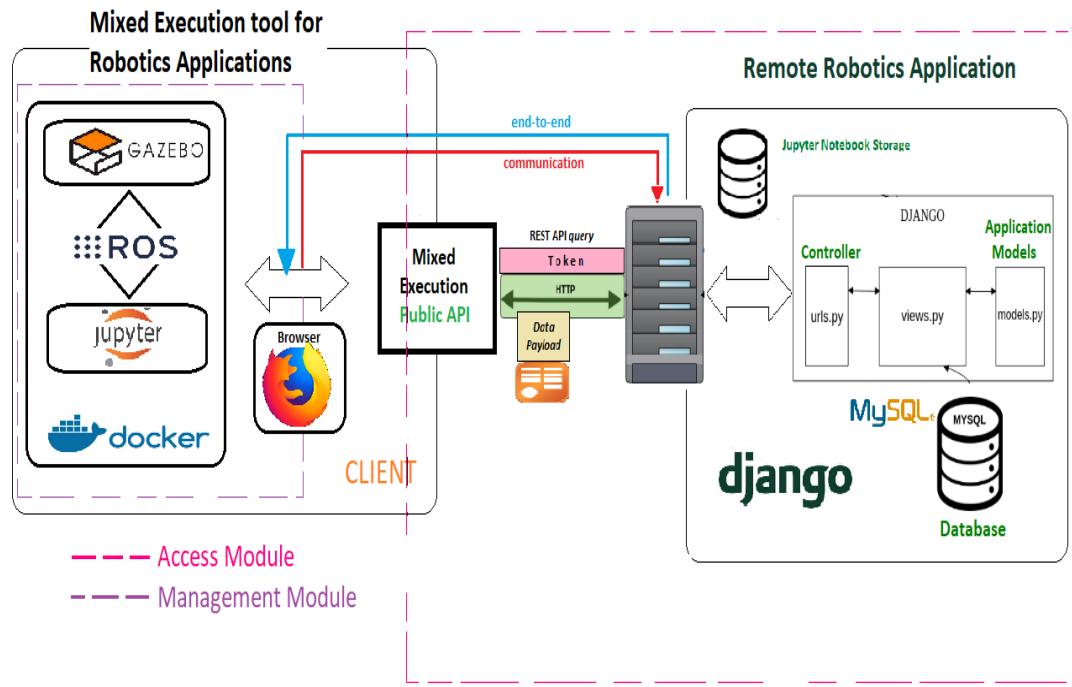


Figura 4.4: Arquitectura de la Ejecución Mixta para Aplicaciones de Robótica.

En este punto, disponemos de una versión completamente remota del servicio de aprendizaje y docencia que se quiere construir. Se han relatado los problemas derivados de este tipo de servicio, y se puede ver claramente cómo no encajan con las características deseadas en lo relativo al cómputo, la escalabilidad y la versatilidad de uso para la herramienta robótica objeto de esta tesis. Por tanto, y tras la investigación correspondiente, se diseñó e implementó un *local runtime*, es decir, un entorno de ejecución utilizando un *kernel* local al cliente, en comunicación constante con la aplicación web remota. En este nuevo plantea-

miento el servidor remoto contiene tanto el modelo de ejercicio creado como el código del mismo y, ante una nueva petición del ejercicio, establece una comunicación fluida con el *kernel* del cliente para coordinar la ejecución en su *hardware*, por medio del lanzamiento del servidor de “Ejecución Mixta” en su máquina.

4.2.1. Diseño

Partiendo del básico servidor web en Django (Fig. 4.1) y de su interfaz web descritos anteriormente, con capacidad para atender a un cliente que solicitase un ejercicio por medio de un click en dicho interfaz, se diseñó y colocó la arquitectura de “Ejecución Mixta” debajo del botón sobre el que el usuario hace click para entrar al ejercicio. Este click dispara el mecanismo de enlazado y comunicación para disponer el entorno mixto. Se puede ver en el infograma (Fig. 4.4) que la solución ideada para la “Ejecución Mixta” se divide claramente en dos módulos, cada uno sub-dividido a su vez en una serie de capas encargadas de una parte específica del proceso:

- El primero de ellos es el **módulo de acceso**. Este módulo será el encargado de dirigir la comunicación entre el lado servidor y el lado cliente de la aplicación para que se produzca la “Ejecución Mixta”. Organizará ambas partes y actuará como un secuenciador que decidirá en cada momento quién lleva el peso de la comunicación. Su propósito principal será el de garantizar el acceso a la herramienta.
- El otro es el **módulo de gestión**. Estará recubierto por un API público que permitirá a las aplicaciones web externas realizar la conexión y aprovechar las funciones de “Ejecución Mixta” para ofrecer un servicio. Será el encargado de redirigir cada petición y respuesta de la aplicación al proceso que corresponda de forma segura, dentro de la ejecución del cliente.

Dada la clara distinción entre las distintas sub-tareas que realiza cada módulo, consideramos adecuado bajar otro escalón en cuanto a la arquitectura, y di-

vidir cada módulo en capas encargadas de tareas sencillas, sobre las que se pudiese construir de manera modular.

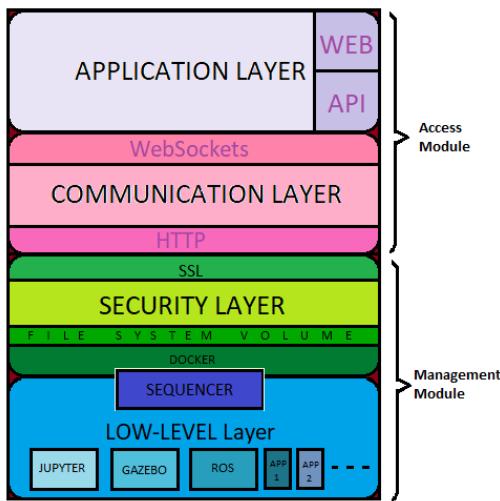


Figura 4.5: Estructura de Capas de la “Ejecución Mixta”

La primera capa del Módulo de Acceso es la **capa de aplicación**, cuya tarea es la de emparejar la funcionalidad que ofrece la plataforma remota como servicio con la información necesaria de la ejecución en el lado cliente para dar soporte a dicha funcionalidad, utilizando uno o varios *endpoints* en función de los canales de comunicación que se quiera establecer para recopilar dicha información valiéndose del API de “Ejecución Mixta”.

Debajo de esta capa se encuentra, en el módulo de acceso, la **capa de comunicación**. Se encarga del establecimiento de los canales de comunicación entre la capa de aplicación situada en la plataforma robótica externa y la herramienta de “Ejecución Mixta” una vez iniciada la conexión. La naturaleza de los canales depende del tipo de información a intercambiar, destacando los protocolos HTTP y WebSockets como esqueletos o plantillas de dichas interacciones.

Ya dentro del Módulo de Gestión, encontramos la **capa de seguridad**, que tiene la tarea de garantizar tanto la integridad y fiabilidad de la información que viaja por Internet en todo momento como la seguridad del sistema cliente.

La última capa que compone el Módulo de Gestión es la **capa de bajo nivel**, cuyo objetivo es resolver la interconexión de todas las aplicaciones y herramientas que tomarán parte en la ejecución en el sistema cliente, así como el de orquestar la redirección de mensajes a cada agente involucrado.

4.2.2. Back-end Remoto

Mover todo el peso computacional de las herramientas que se usan para dar el servicio de ejercicios de robótica al lado del cliente web resulta en una infraestructura muy sencilla para el servidor web remoto.

El *back-end* remoto queda liberado del trabajo asociado a Jupyter y Gazebo, además de las comunicaciones entre ambos a través de ROS, y sólo se encarga del procesamiento de las peticiones entrantes para generar una serie de instrucciones que pongan en marcha el mecanismo de “Ejecución Mixta”. El servidor sigue almacenando los cuadernillos o *Notebooks* de Jupyter y el código Python del usuario, pero ya no es el encargado de ejecutarlo.

La petición de un nuevo ejercicio supone ahora para el servidor remoto el establecimiento de la conexión con el servidor de “Ejecución Mixta” del lado cliente a través de la capa de aplicación, utilizando un API que comentaremos más adelante. El procesamiento en el servidor genera una sesión de ejecución y continua con el envío al cliente del código del cuadernillo y de cada fichero de código de aplicación auxiliar que necesite el ejercicio, para acabar con una orden de inicio del *kernel* del lenguaje adecuado encargado de la ejecución, que se montará sobre un núcleo latente en el sistema del usuario web. La utilización de direcciones IP públicas, el protocolo HTTP (estándar de Internet) que utiliza por defecto el puerto 8080 (puerto de Internet, abierto a mensajes transportados sobre TCP o UDP por la red, con cualquier origen) que puede atravesar *firewalls* y resolver conexiones sea cual sea el dominio público, y el *token* de autorización embebido en los mensajes garantizan la conectividad y la integridad de la sesión de Jupyter, así como el funcionamiento entre sistemas bajo distintas redes o sub-

redes de la misma red.

4.2.3. Back-end Local

Se infiere la necesidad de una especie de servidor en el lado cliente web que difunda la información de una ejecución local a quien la solicite. Este es precisamente el servidor de “Ejecución Mixta”, que replica el conjunto de aplicaciones y herramientas que disponía el servidor web en la sección anterior (4.1) en un entorno de ejecución local al cliente que debe estar en constante comunicación con el lado servidor web.

Las interacciones del usuario con el ejercicio académico se producen en el interfaz web y son captadas por el servidor remoto, y éstas deben pasar ahora por el API de “Ejecución Mixta” para llegar a la capa de bajo nivel y ser recibidas por cada herramienta que se ejecuta en el sistema del cliente. Para lograr la ejecución del código del cliente localmente a través de órdenes generadas en el interfaz ofrecido por un servidor remoto, se hace indispensable proveer al servidor de “Ejecución Mixta”, concretamente al *kernel* de Jupyter, del código asociado al ejercicio para que pueda ejecutarlo, y también de un mecanismo de retorno de los resultados para que el servicio web pueda reflejarlos en el interfaz de visualización de usuario.

Por tanto, la infraestructura del lado cliente cuenta con un contenedor Docker con el conjunto de herramientas necesarias para el servicio y con un API de acceso y gestión de la ejecución que dispara el mecanismo de comunicación entre el lado servidor y el cliente web.

4.2.3.1. Protocolo de Intercambio

El primer mecanismo con el que debe contar la infraestructura local de la “Ejecución Mixta” es un protocolo de intercambio de información que permita poner en contacto el *back-end* remoto con el local. Es necesario establecer un método de acceso que pueda usar el lado servidor web para conectarse a la eje-

cución que tiene lugar en el lado cliente. Al conservar el interfaz que ofrece el servidor web a su cliente y el funcionamiento de los ejercicios a través de Jupyter es necesario para el correcto desempeño del proceso que el receptor principal de los mensajes de “Ejecución Mixta” sea precisamente éste, por lo que el protocolo de comunicación debe basarse en los mensajes que el servidor de Jupyter intercambia con el interfaz web en su funcionamiento natural.

Se resuelve el establecimiento de la comunicación con el servidor de Jupyter situado en el sistema del cliente mediante el envío de un simple mensaje bajo el método OPTIONS, al que dicho servidor responde con toda la información que necesitamos para el establecimiento a través de cabeceras HTTP, entre ellas el *token* de autenticación de Jupyter, los métodos y tipos de contenido que se aceptan, y el código CSRF¹ necesario para enviar información al servidor de Jupyter desde otros puntos. Esto requiere hacer unos cambios sobre el servidor provisto por el proyecto Jupyter con el fin de abrir el acceso a él desde el exterior, no sin antes establecer los correspondientes métodos de seguridad. Se profundizó en la arquitectura de servicio de Jupyter para poder adaptar su funcionamiento a nuestras necesidades.

Al obtenerse esta información se ha de disponer un canal entre el servicio web remoto y el servicio de ejecución local, para lo cual no existen soluciones. Estudiamos el mecanismo de comunicación de la plataforma Jupyter, que emplea principalmente el protocolo ZeroMQ² o 0MQ para el intercambio asíncrono de mensajes *N-a-N* a través de la web. Esto nos proporciona, además de una plantilla para los tipos de mensajes que debemos enviar al *kernel* de Jupyter, una idea bastante fiel del proceso que desencadena cada orden o petición que se le hace al servidor de Jupyter. También se utilizó *sniffers*³ para monitorizar el funcionamiento local de Jupyter (Fig. 4.6). Este paso fue vital para comprender el procedimiento que debíamos seguir para iniciar una sesión de Jupyter completa

¹What is CSRF and how to prevent it?

²<https://zeromq.org/>

³Wireshark Sniffer

y válida.

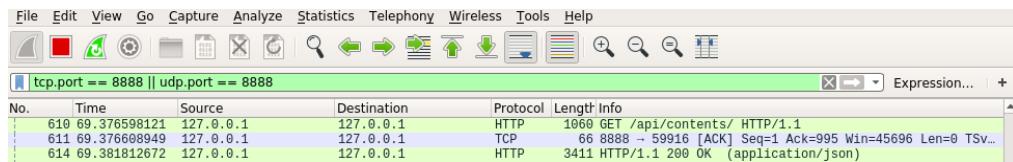


Figura 4.6: Captura de Paquetes del Mecanismo de Comunicación de Jupyter

Por tanto tras el establecimiento, se puede comenzar con el intercambio de mensajes 0MQ. Aprovecharemos el REST API que ofrece Jupyter para la comunicación con el *kernel* en tanto que este *RESTfull Service*⁴ ofrece todo lo necesario para enviarle ficheros de código, órdenes de ejecución, órdenes de control (reinicio, apagado, encendido, pausa, cambio de lenguaje, limpieza de salidas, etc.) (Fig. 4.7) y, en general, lo necesario para facilitar el uso de la plataforma a través de la web. Por tanto, por el canal de “Ejecución Mixta” estaríamos enviando, principalmente, mensajes HTTP sobre una capa de TLS con datos de tipo 0MQ en el cuerpo, destinados a Jupyter que viajan por Internet.

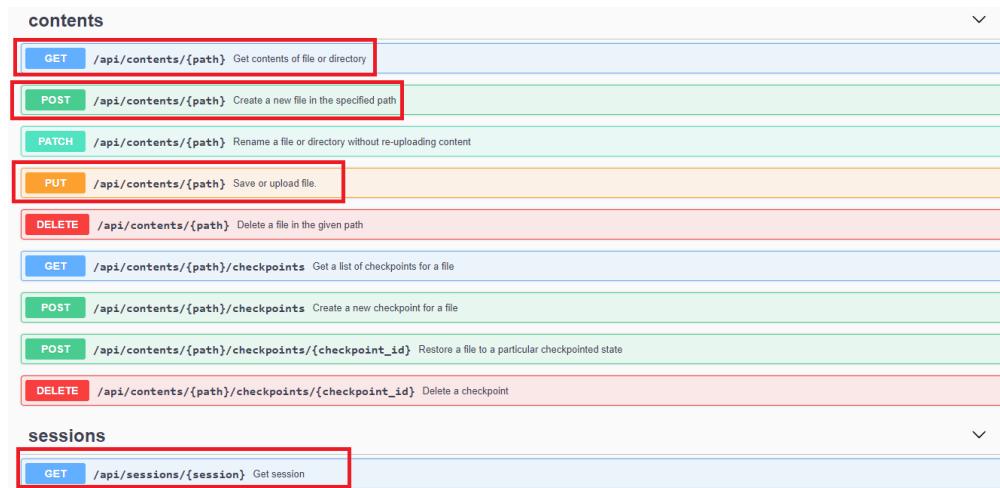


Figura 4.7: REST API de Jupyter

En base al conocimiento obtenido del motor de comunicación de Jupyter,

⁴What is a RESTfull Service?

seleccionamos aquellos métodos del REST API que necesitaríamos, marcados en la imagen superior. Se muestra a continuación un ejemplo del mensaje que se enviaría al servidor de Jupyter para hacer llegar el código al *kernel* encargado de un cuadernillo concreto:

```
1 const url = 'http://'+_ip+':'+_port+'/api/contents/' + nbname;
2 const data_put = ' {"type": "notebook",
3                     "format": "json",
4                     "content": ' + nbcontent + '} ';
5 const message = {
6     headers:{'Content-Type':'application/json',
7               //'Authorization': 'token ' + _token,
8               },
9     body:data_put,
10    method:"PUT"
11 };
```

Listing 4.2: Formato de Mensajes de “Ejecución Mixta”

donde cada variable que aparece en el mensaje (*_ip*, *_port*, *nbname*, *_token*, *nbcontent*) sería establecida por el servidor web y contendría la dirección del servidor Jupyter y el contenido y nombre del cuadernillo que queremos enviar, almacenado en el servidor remoto y solicitado por el usuario web. El formato de todos los mensajes es similar, en tanto que se trata de peticiones HTTP como se puede ver en la constante *message* con el formato de cuerpo 0MQ que se refleja en la constante *data_put*. Este tipo de mensajes, con su respectiva codificación, constituye el tipo de mensajes a intercambiar por la herramienta “Ejecución Mixta” para el establecimiento y envío iniciales.

En este punto del desarrollo, hemos superado el principal obstáculo de la “Ejecución Mixta” al organizar una ejecución compartida capaz de ejecutar código entre redes mediante el intercambio de mensajes de ejecución, con actualización de los resultados en tiempo real en ambos lados servidor y cliente web. Esto sortea tanto el obstáculo de desbordamiento de cómputo en el lado servidor, pues es ahora el cliente el que carga con la mayor parte del peso computacio-

nal, como la limitación de *hardware* conectable a la aplicación web, ya que ahora el cliente puede emplear su propia cámara o incluso sus propios robots en los ejercicios, accesibles desde su máquina y su red, pero no desde el exterior.

Sin embargo, la dependencia de este servidor de ejecución local supone para el cliente la instalación de todas las herramientas, tanto Jupyter como Gazebo y ROS, para mantener el soporte del servicio web de manera local. Las ventajas obtenidas se ven empañadas por una serie de desventajas claras entre las que se encuentran principalmente el proceso de instalación (tedioso, requiere espacio de disco del cliente) y la compleja organización inter-plataforma necesaria para comunicar las herramientas locales. El usuario necesitaría un programa encargado de cumplir estas funciones, lo cual desmontaría el servicio web.

4.2.3.2. Contendor Docker

Ya se ha descrito la necesidad de servidores físicos para las herramientas que se usan, especialmente Jupyter y Gazebo. Ambas deben ser reubicadas al lado cliente para liberar al lado servidor, pero al mismo tiempo se debe evitar su instalación para garantizar un fácil acceso a la herramienta. A estos efectos la “Ejecución Mixta” se basa en un contenedor Docker en el cual se ejecutan las herramientas necesarias y las aplicaciones auxiliares que reemplaza el complejo proceso de instalación por una sencilla descarga de la imagen Docker.

Se estudió el motor de Docker y su funcionamiento para crear un *DockerFile* con instrucciones en lenguaje SHELL para virtualizar una distribución Ubuntu con todas las herramientas y aplicaciones auxiliares involucradas en la “Ejecución Mixta”, para que utilizar la herramienta resultase tan fácil como descargar la imagen Docker y levantar el contenedor. Utilizar un contenedor Docker para empaquetar los procesos necesarios e incluir seguridad supone añadir complejidad al mecanismo de comunicación, en tanto que Docker levanta su propia sub-red dentro de la red del cliente, lo que nos deja una intratable sub-red dentro de una red distinta desde el punto de vista del servidor web. Se replanteó el

mecanismo de establecimiento de la comunicación para que utilízase el navegador del cliente web como intermediario entre el usuario y la aplicación robótica en lugar de la previa comunicación punto a punto, el cual sí que tiene conectividad directa con una sub-red de su propia red. Por tanto, el servidor web abre ahora una comunicación con el *browser* del cliente en lugar de directamente con el servidor de Jupyter, y éste actúa como *proxy* reenviando cada mensaje donde corresponde a través de código JavaScript, de manera que tanto de cara al cliente como al servidor web, la comunicación sigue siendo la misma que en el paso anterior, esta vez con un origen o destinatario distinto. Simplemente se trata de añadir la lógica del *proxy* para que viaje con la aplicación que se sirve al usuario web. El navegador ya utiliza protocolos como STUN o ICE que permiten descubrir e interaccionar con ambas partes del mecanismo. Así, los mensajes reenviados por el navegador serán recibidos por un módulo secuenciador de “Ejecución Mixta” situado en el contenedor Docker, que procesará la información que llega para orquestar el correcto inicio del entorno y los agentes, traduciendo las órdenes JavaScript iniciales a comandos de lanzamiento (Listing. 4.3) y entregando las órdenes de ejecución a quien corresponda (Listing. 4.4).

```
1 #!/bin/bash
2
3 rm -rf /tmp/.X0-lock
4
5 Xvfb -shmem -screen 0 1280x1024x24 &
6
7 source /opt/jderobot/setup.bash
8 source /opt/ros/kinetic/setup.bash
9 source /opt/jderobot/share/jderobot/gazebo/gazebo-assets-setup.sh
10 export PYTHONPATH=$PYTHONPATH:/home/jderobot/.exercises
11
12 cd ~/gzweb
13 npm start &
14
```

```

15 cd ~/volume/user/exercise
16
17 jupyter nbextension enable hide_input/main --user
18 jupyter nbextension enable init_cell/main --user
19 jupyter notebook --ip=0.0.0.0 --allow-root &
20
21 cd ~
22
23 EXTENSION=`echo "$1" | cut -d'.' -f2`
24 if [ $EXTENSION = "world" ]
25 then
26   roscore &
27 fi
28
29 if ! [ -z "$2" ]
30 then
31   python ~/referees/$2 &
32 fi
33
34 if ! [ -z "$1" ]
35 then
36   EXTENSION=`echo "$1" | cut -d'.' -f2`
37   if [ $EXTENSION = "launch" ]
38   then
39     roslaunch /opt/jderobot/share/jderobot/gazebo/launch/$1
40   else
41     rosrun gazebo_ros gazebo /opt/jderobot/share/jderobot/gazebo/
42       worlds/$1
43   fi
44 else
45   tail -f /dev/null
46 fi

```

Listing 4.3: Código de Inicio del Secuenciador

```

1 [I 11:51:09.187 NotebookApp] Saving file at /thumbnail_follow_line.png
2 [Gazebo] Sat Jan 11 2020 11:51:09 GMT+0000 (Coordinated Universal Time)

```

```
Received Message: {"op":"advertise","id":"advertise:~/heartbeat:14"
  , "type":"heartbeat","topic":"~/heartbeat"} from http
  ://127.0.0.1:8080 ::ffff:172.17.0.1
3 [Python Process] Sat Jan 11 2020 11:51:09 GMT+0000 (Coordinated
  Universal Time) Received Message: {"op":"publish","id":"publish:~/
  heartbeat:15","topic":"~/heartbeat","msg":{"alive":1}} from http
  ://127.0.0.1:8080 ::ffff:172.17.0.1
4 [I 11:51:11.758 NotebookApp] 302 GET /notebooks/world.png (172.17.0.1)
  0.95ms
5 [I 11:51:11.847 NotebookApp] Adapting to protocol v5.1 for kernel 368
  ale46-acc2-4976-acf3-25528ae77d77
```

Listing 4.4: Reenvío de Mensajes

El contenedor de “Ejecución Mixta” ejecuta tanto la lógica programada por el usuario como la simulación y las aplicaciones auxiliares como *plugins* o *drivers* robóticos que actúen como controlador del dispositivo *hardware* real, en caso de usarse. Por tanto, necesita de la intervención de un módulo que organice las diferentes partes para evitar colisiones y solapamientos. Interviene aquí el secuenciador del Módulo de Gestión del contenedor, que establece el mecanismo interno de comunicaciones a través de ROS (Listing. 4.5), y actúa de intermediario reenviando internamente los mensajes a quien corresponde y elaborando una respuesta destinada al servicio web remoto a través de canales WebSockets o HTTPS para que éste último lo procese. Según las órdenes que recibe de la plataforma web remota, el secuenciador activa en cada momento la aplicación, herramienta o plataforma auxiliar que se necesite para satisfacer la petición, y organiza el canal de bajo nivel abriendo nuevos sub-canales de comunicación de tipo p2p, ya en un entorno puramente local o *localhost*, sin pasar por todo el mecanismo de transmisión, recepción y procesado descrito anteriormente como parte de la arquitectura. Estos canales punto a punto evitan problemas de tipo *jitter*, grandes retardos de extremo a extremo o largas esperas de reenvíos que caracterizan a muchos enlaces de Internet basados en TCP/IP. En el mecanismo de comunicación interna el secuenciador tendrá el rol de *MASTER* de ROS y

las diferentes plataformas y aplicaciones en ejecución actuarán como suscriptores y publicadores de información a través de *topics* conocidos por todos. Así, se dispondrá por ejemplo un HAL API (*Hardware Abstraction Layer*) que permitirá a todos los programas que lo requieran el acceso a las interfaces de sensado y actuación del robot, ya sea real (con *plugins* y *drivers* reales involucrados) o simulado (con *drivers* virtuales). Este método de intercomunicación facilita el crecimiento de la red de agentes (que simplemente “escuchan” la información del canal que les conviene y publican los datos que generan que pueden resultar útiles para otros procesos) que se puede lanzar para proporcionar mayor volumen de información a la aplicación robótica remota, y por tanto a la construcción de aplicaciones remotas más ricas que no produzcan carga computacional en el lado servidor.

```

1 Camera:                                     # follow_line.yml
2   Topic: "/F1ROS/cameraL/image_raw"
3   Name: follow_line_camera
4 Motors:
5   Topic: "/F1ROS/cmd_vel"
6   Name: follow_line_motors
7   MaxV: 40
8   MaxW: 2
9 Websockets:
10  Host: 0.0.0.0
11  Port: 9002
12  SSL: False
13  Cert: '/etc/certs/fullchain1.pem'
14  Key: '/etc/certs/privkey1.pem'
```

Listing 4.5: Configuración de Canales Internos del Secuenciador en formato YAML

Para implementar el soporte de simulación, se diseñó un sistema de configuración que permite indicar los elementos, robots y características que debe tener la escena simulada a través de Gazebo (Fig. 4.8), haciendo uso de ficheros de extensión *.world* o *.launch*, siendo los primeros un subconjunto del lenguaje de marcado XML que indica los agentes involucrados en la simulación (Listing. 4.7), y el segundo un configurador inteligente que permite lanzar, además del mundo de simulación, una serie de nodos que se puedan necesitar para controlar por ejemplo las interfaces de un determinado robot (Listing. 4.6).

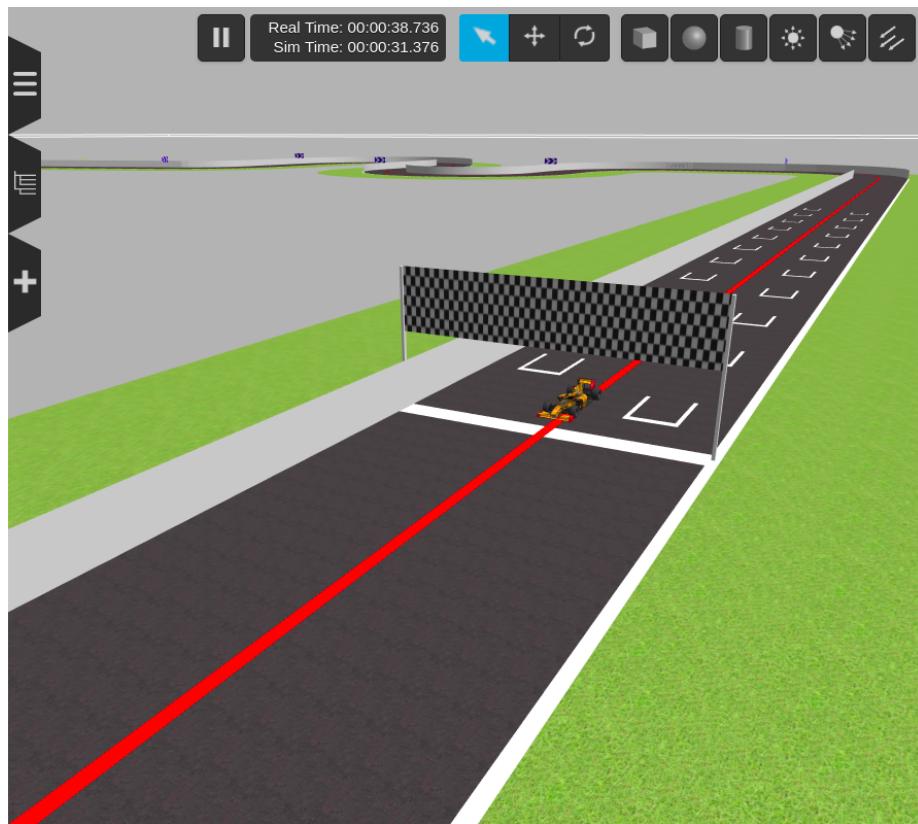


Figura 4.8: Escenario de Simulación

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <launch>
```

```

3   <!-- We resume the logic in empty_world.launch, changing only the
4       name of the world to be launched -->
5   <include file="$(find gazebo_ros)/launch/empty_world.launch">
6     <arg name="world_name" value="f1_1_simplecircuit.world"/> <!-- Note:
7         the world_name is with respect to GAZEBO_RESOURCE_PATH
8         environmental variable -->
9     <arg name="paused" value="false"/>
10    <arg name="use_sim_time" value="true"/>
11    <arg name="gui" value="true"/>
12    <arg name="headless" value="false"/>
13    <arg name="debug" value="false"/>
14    <arg name="verbose" default="false"/>
15  </include>
16 </launch>

```

Listing 4.6: Configuración de Lanzamiento de Simulaciones

```

1 <?xml version="1.0" ?>
2 <sdf version="1.5">
3   <world name="default">
4     <scene>
5       <grid>false</grid>
6     </scene>
7     <!-- A global light source -->
8     <include>
9       <uri>model://sun</uri>
10      </include>
11      <include>
12        <uri>model://pista_simple</uri>
13        <pose>0 0 0 0 0 0</pose>
14      </include>
15      <include>
16        <uri>model://f1_renault</uri>
17        <pose>53.462 -10.734 0.004 0 0 -1.57</pose>
18      </include>
19      <scene>

```

```
20      <sky>
21          <clouds>
22              <speed>12</speed>
23          </clouds>
24      </sky>
25  </scene>
26 </world>
27 </sdf>
```

Listing 4.7: Configuración de Lanzamiento de Simulaciones

Con ello, al lanzar la herramienta se indicará qué fichero de configuración se quiere usar a través de instrucciones del API de “Ejecución Mixta”, que se ocupará de levantar tanto la simulación como la red de comunicación interna basada en ROS como se puede ver en el *snippet* de código inferior (Listing. 4.8), y la capa de abstracción que permite al usuario programar su robot y acceder a toda la funcionalidad mientras se materializan los cambios en el simulador.

```
1  class ListenerCamera:
2      def __init__(self, topic):
3
4          self.topic = topic
5          self.data = Image()
6
7      # [...]
8
9      def start(self):
10
11          self.sub = rospy.Subscriber(self.topic, ImageROS,
12                                      self.__callback)
13
14 class PublisherMotors:
15
```

```
16     def __init__(self, topic, maxV, maxW):
17
18         self.maxW = maxW
19         self.maxV = maxV
20
21         self.topic = topic
22         self.data = CMDVel()
23         self.pub = rospy.Publisher(self.topic,
24                                     Twist,
25                                     queue_size=1)
26
27         rospy.init_node("FollowLineF1")
28
29
30     def publish(self):
31
32         self.lock.acquire()
33         tw = cmdvel2Twist(self.data)
34         self.lock.release()
35         self.pub.publish(tw)
36
37 class FollowLine():
38
39     def __init__(self):
40         cfg = readConfig()
41
42         cameraTopic = cfg["Camera"]["Topic"]
43         motorsTopic = cfg["Motors"]["Topic"]
44         maxv = cfg["Motors"]["MaxV"]
```

```
45     maxw = cfg[ "Motors" ][ "MaxW" ]  
46  
47     self.camera = ListenerCamera(cameraTopic)  
48     self.motors = PublisherMotors(motorsTopic, maxv,  
49                                     maxw)  
50  
51     # [...]
```

Listing 4.8: Creación de la Red Interna de Comunicación

Dada la filosofía modular interconectada que se había diseñado hasta este momento, el enfoque para el caso de querer controlar un robot real desde el lado cliente pasaba sencillamente por crear su controlador, e incluirlo como aplicación auxiliar lanzada dentro de la herramienta y orquestada como un módulo más por el secuenciador.

Se tiene una herramienta que actúa como *middleware* entre un servicio web remoto y una serie de aplicaciones locales al cliente capaz de lanzar un proceso de ejecución compartida con una serie de ventajas claras, entre las que se encuentran el uso de robots reales del cliente, el bajo cómputo en el lado servidor web, la versatilidad de contenidos que se puede crear para la aplicación y, sobre todo, la accesibilidad a las aplicaciones robóticas. La inclusión de la “Ejecución Mixta” ya no supone ninguna carga de cómputo para el lado servidor de la aplicación web y queda eliminado el proceso de instalación, dado que todas las dependencias están agrupadas en el contenedor. Lanzar la “Ejecución Mixta” pasa ahora por utilizar el API de Docker de control de contenedores, que pondrá en marcha todos los sub-sistemas necesarios para el funcionamiento de la aplicación robótica y los dejará disponibles para su consulta desde el exterior. Así, la aplicación puede enviar cualquier tipo de orden de ejecución en formato Python al servidor de Jupyter, que materializará esa orden en el simulador Gazebo a través de los canales ROS o, incluso, en el robot o sensor real conectado

al sistema del cliente.

4.2.3.3. API de “Ejecución Mixta”

El servicio web remoto tiene conectividad con el entorno de ejecución local al cliente, pero necesita una serie de métodos de acceso que le permitan materializar esa comunicación y ejercer algún tipo de control sobre el contenedor de “Ejecución Mixta” para indicar qué aplicaciones y herramientas se deben poner en marcha dentro de él y poder enviar mensajes de ejecución y control durante el servicio, así como recibir realimentación de la ejecución. Para esto se diseñó un API a nivel de la capa de aplicación que permite al servidor web engancharse a la “Ejecución Mixta”.

Será necesario que la aplicación robótica que se ofrece a través de la web que quiere beneficiarse de la “Ejecución Mixta” inicie la conexión con el cliente remoto, en calidad de cliente de “Ejecución Mixta”. Esto es así dada la naturaleza de la información a la que más tarde pedirá acceso, la cual se genera completamente en el cliente durante la ejecución y pondrá a disposición de quien la solicite. Con la conexión iniciada, esta capa debe solicitar la información que necesite al servidor de “Ejecución Mixta”, en tiempo de ejecución, para que la aplicación funcione como se espera. Esta solicitud se realizará a través del API de “Ejecución Mixta”, mediante un comando específico de inicialización que el servidor genera para configurar los parámetros de la ejecución que permiten crear un puente seguro entre un elemento virtualizado y su homólogo en la máquina anfitrión, como por ejemplo el mecanismo de acceso a la cámara integrada en el sistema cliente desde el interior de un contenedor Docker, resuelto con el mapeo `-v /dev/video0:/dev/video0`, o la redirección de interfaces de escucha del sistema a sus correspondientes dentro del contenedor (`-p 8888:8888 -p 8889:8889 -p 8080:8080`). También se indica el tipo de enlace que se requiere y el conjunto de herramientas que se necesita para la ejecución.

```
1 mkdir -p /tmp/siguelineaIR/ && docker run --rm -e DISPLAY=:0
2 -e JDEROBOT_SIMULATION_TYPE=REMOTE --entrypoint
3 /entrypoint_mixed_execution.sh -v
4 /tmp/siguelineaIR:/home/jderobot/volume/user/exercise:rw
5 -p 8888:8888 -p 8080:8080 -p 9001:9001 -p 9002:9002
6 -it tfmdocker/local:dev f1_1_simplecircuit.launch
```

Listing 4.9: Comando de configuración de Ejecución Mixta

Tras el establecimiento de conexión y configuración inicial, este API estará basado principalmente en el protocolo HTTP para la comunicación que tiene lugar a través de Internet y que pretende conectar la herramienta y la capa de aplicación. El formato de los mensajes intercambiados será similar al de un REST API basado en solicitudes y respuestas HTTP (Listing. 4.10) utilizando los métodos clásicos: GET, POST, PUT, DELETE y OPTIONS. Dicho API formatea las peticiones del servidor web incluyendo en cada paquete de información su fuente, su destinatario en la ejecución y su prioridad, a través de cabeceras y estructuras de datos JSON fácilmente procesables (Figs. 4.9 y 4.10).

```
1 PUT /api/contents/descarga_bundle.png HTTP/1.1
2 Host: 127.0.0.1:8888
3 Connection: keep-alive
4 Content-Length: 25899
5 Sec-Fetch-Mode: cors
6 Origin: http://localhost:8000
7 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36
8 (KHTML, like Gecko) Chrome/77.0.3865.120 Safari/537.36
9 Content-Type: application/json
10 Accept: */*
11 Sec-Fetch-Site: cross-site
12 Referer: http://localhost:8000/local
13 Accept-Encoding: gzip, deflate, br
14 Accept-Language: es-ES,es;q=0.9
```

Listing 4.10: Ejemplo de la parte HTTP de los mensajes

The screenshot shows a NetworkMiner capture of an HTTP PUT request. The payload is a JSON object representing a notebook submission:

```

{
  "type": "notebook",
  "name": "notebook_0001",
  "format": "json",
  "content": "El notebook contiene el plan de actuador es...\\n\\n## Tabla de control de los motores\\n\\n# Contr...\\n\\nFuerza de arrastre de los motores\\n\\n| De scripti...\\n"
}
  
```

A blue box highlights the content field, and an arrow points from it to the label "Entrega del Notebook".

Figura 4.9: Mensaje de Entrega del Cuadernillo o Notebook

The screenshot shows a NetworkMiner capture of a WebSocket message. The payload is a JSON object containing a session ID and a header:

```

{
  "msg_id": "b62cbd51162e488180b066f8c9f3b189",
  "session": "6f49a580e0414e86a9fb7e20cff3d78",
  "parent_header": {
    "username": "username",
    "version": "5.2",
    "msg_type": "execute_request"
  }
}
  
```

An arrow points from the payload area to the label "Ejecución de Petición de Ejecución".

Figura 4.10: Mensaje de Respuesta de Petición de Ejecución

4.2.4. Seguridad

Para que el intercambio de código y la acción de la ejecución del mismo sucedan de manera segura, si dispuso una capa de seguridad en el mecanismo de "Ejecución Mixta". Debemos distinguir dos niveles de seguridad en el proceso:

- En primer lugar, hay que garantizar la fiabilidad e integridad de los mensajes intercambiados con la herramienta. Al ser tanto HTTP como Web-Sockets dos protocolos en los que la información viaja en claro, como texto plano, por el canal de comunicación, decidimos incluir una capa que garantiza seguridad a través del protocolo SSL, que también asegurará que los mensajes no se corrompan. Este nivel tiene un grado bajo de criticidad.

- Por otro lado, existe un alto riesgo cuando se trata de permitir que un código cuya fuente es externa se ejecute en nuestro sistema. Esto es precisamente lo que se pretende hacer con esta herramienta, de manera que se debe asegurar que, sea cual sea la fuente del código y su contenido, los problemas de seguridad se pueden contener, evitando que actúen sobre el sistema del cliente, el cual lleva a cabo la ejecución. Esta es una de las funciones del contenedor Docker.

Para mantener segura la conexión entre cliente y servidor de “Ejecución Mixta” y entre cliente y servidor web a través de Internet utilizamos el protocolo TLS a Nivel de Transporte según la clasificación OSI, que asigna una serie de certificados SSL basados en criptografía asimétrica a cada agente de la comunicación para ser autenticados a la hora de intercambiarse la clave simétrica con la cual se pueden decodificar los datos intercambiados, que viajan encriptados en todo momento. Así funcionan la mayoría de sitios web en la actualidad.

El mayor peso en cuanto a seguridad recae sobre la tarea para la que ha sido diseñada la herramienta. El código que proviene del servidor web puede no ser de confianza o de dudosas intenciones, por lo que no se puede permitir a la ligera su ejecución en la máquina del cliente, dado que esto podría conllevar graves riesgos de pérdida de datos (mediante instrucciones de borrado de disco que viajen en el código), malfuncionamientos y ataques de toda clase. Por ello, se ha montado lo que se denomina “volumen” (Fig. 4.11) sobre el sistema de ficheros local del servidor de “Ejecución Mixta” (cliente web), y se ha restringido el acceso del contenedor Docker y de las instrucciones que en él se ejecutan a este volumen, impidiendo en todo caso que cualquier suceso tenga consecuencias o repercusiones fuera de él.

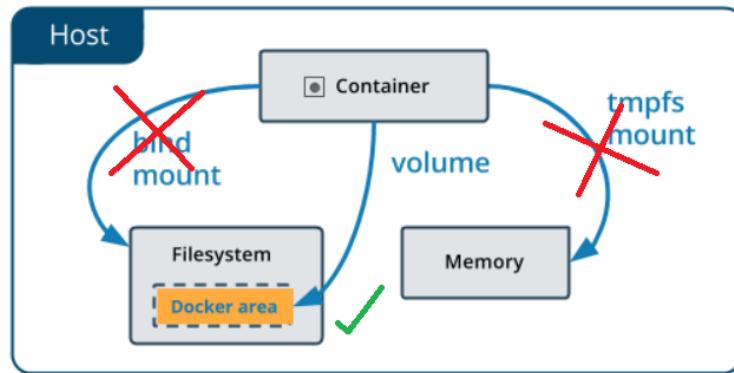


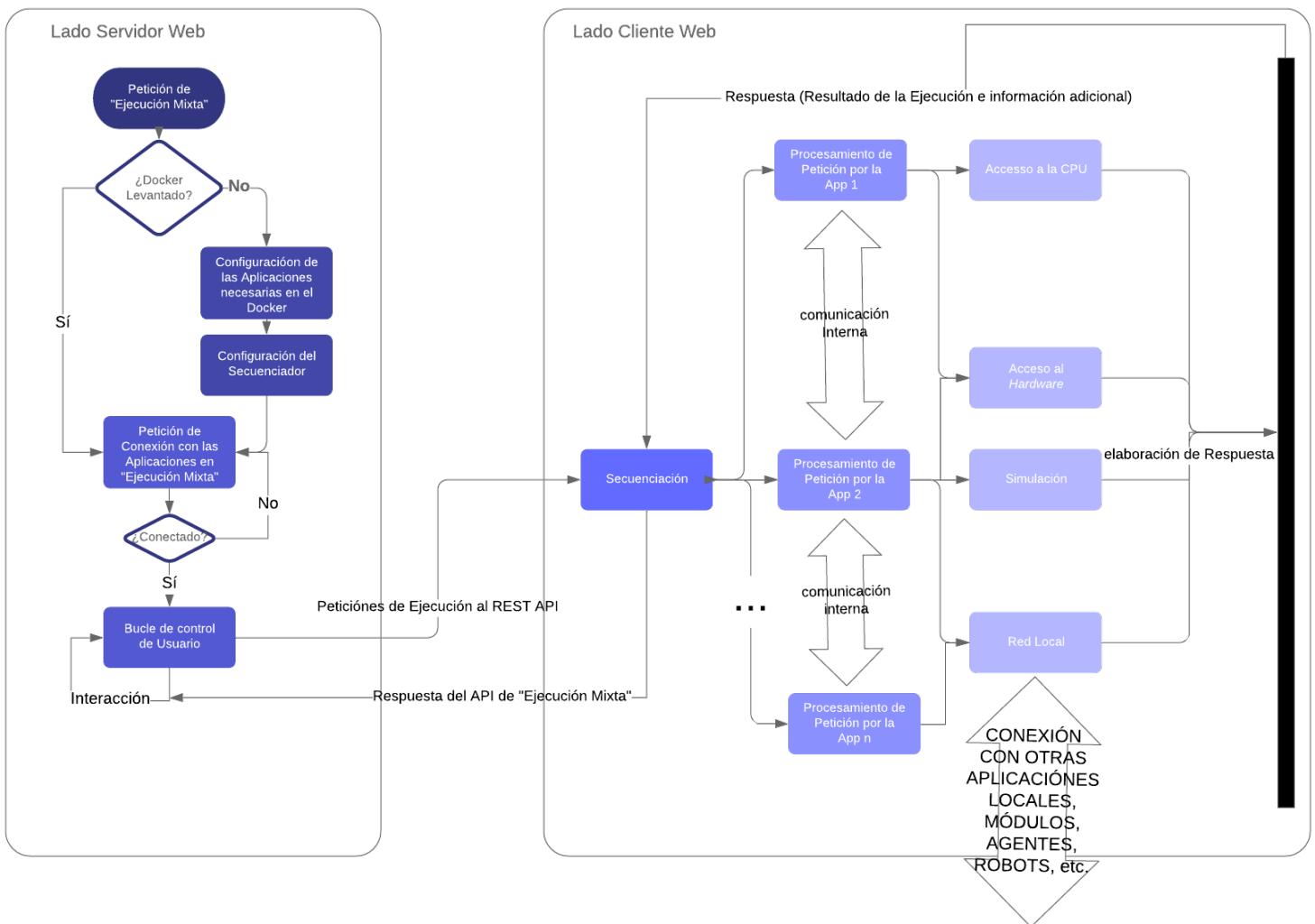
Figura 4.11: Volumen de Docker

Se puede ver un volumen como una forma de almacenar datos persistentes en el sistema de ficheros local del cliente, necesario para gestionar el acceso al *hardware* entre otras cosas para esta herramienta, que permanece completamente aislado de la funcionalidad central y el *core* de la máquina anfitriona. Esto significa que el contenedor sólo podrá actuar sobre el contenido del volumen y no sobre el contenido externo, pero se podrá beneficiar de todas las ventajas que ofrece el administrador del sistema de archivos del sistema operativo del usuario. Así, aún en el supuesto de que se tratase de utilizar la “Ejecución Mixta” en conjunto con una aplicación robótica maliciosa (supuesto muy improbable), el mayor daño posible que se podría causar se reduciría a parar y reiniciar el contenedor, eliminándose todo indicio de código maligno o sospechoso.

4.2.5. Fluograma

1

MECANISMO SUBYACENTE EN LA
"EJECUCIÓN mIXTA"



Para explicar el mecanismo que desata el uso de la “Ejecución Mixta” por parte de una aplicación, se utilizará el caso hipotético de un usuario que dispone de un robot móvil con conectividad inalámbrica a través de una red propia con visibilidad a la red local del usuario, al ser la del robot una subred de ésta.

Apoyándonos en el diagrama de flujo anterior (Fig. 4.12), el proceso comienza con la realización de la petición inicial de “Ejecución Mixta”, que se corresponde con la solicitud hecha por el usuario a la aplicación remota de un servicio que utiliza senda herramienta de ejecución. Dado que la aplicación web remota es la que ofrece el servicio solicitado, es también la aplicación la que conoce la configuración de “Ejecución Mixta” necesaria para el proceso, así como la combinación de aplicaciones de las que va a requerir información durante la ejecución que deben iniciarse en el lado cliente web. Es entonces cuando se envía esta configuración al cliente, quien debe iniciar el contenedor Docker que aloja la herramienta. Una vez iniciado, se ha de conectar con cada una de las aplicaciones contenidas en la herramienta de ejecución desde la aplicación web. Como se mencionó con anterioridad, esto desencadena en el servidor web una petición inicial al API de “Ejecución Mixta”, que pasa a desempeñar el rol de cliente de “Ejecución Mixta”, y que hace una solicitud de conexión. Si los mecanismos de seguridad se resuelven con éxito, el secuenciador del lado cliente web, que actúa como servidor de “Ejecución Mixta”, garantiza el acceso de la aplicación remota a la ejecución. Es entonces cuando se inicia el servicio que el cliente web solicitó en primera instancia, ingresando en un bucle de control y eventos de la misma forma que sucedería en cualquier otro tipo de aplicación web. En este bucle, la aplicación web atiende permanentemente las interacciones del usuario web, quien generará eventos en el contexto de la aplicación web, y que podrá generar peticiones concretas de ejecución local. Cuando se produce este tipo de petición es cuando la aplicación hace uso de la “Ejecución Mixta” para lanzar el código del usuario web, escrito y almacenado remotamente desde su punto de vista, sobre el *hardware* local al usuario. Como se comentó, estas

peticiones viajan con un formato concreto a modo de protocolo con destino al secuenciador de “Ejecución Mixta”, quien ya en el entorno local es capaz de analizar la petición y redirigirla convenientemente a su destinatario o destinatarios. Los receptores finales de estas solicitudes serán las diferentes aplicaciones lanzadas para soportar el servicio web, que se especificaban en el *entrypoint* de configuración inicial del secuenciador. Es entonces cuando los receptores pueden procesar la petición, siempre y cuando el método solicitado esté soportado por el REST API de “Ejecución Mixta”. El receptor principal será siempre Jupyter, pues es quién tendrá la capacidad de ejecutar código sobre la CPU local o el robot. También habrá un volumen considerable de peticiones de simulación, con el fin de mantener siempre actualizado el estado de la simulación, si existe, en la aplicación web y su interfaz gráfico, que es el que ve el usuario. Para el caso planteado, incluso habrá peticiones concretas o resultados de la ejecución del código que desembocarán en el establecimiento de un nuevo canal de comunicación a través de la red del cliente web. Como se puede ver en el diagrama anterior, este canal se puede utilizar para conectar el resultado de la “Ejecución Mixta” a cualquier otra aplicación externa del lado servidor de ejecución preparada para recibir los mensajes de respuesta que se generan. Esto hace crecer la potencia y el alcance de la ejecución, y las posibilidades de servicio web que se puede ofrecer. En el caso del ejemplo, la ejecución generaría mensajes que deben ser enviados al receptor del robot móvil a través de la red inalámbrica, que se materializarían en la actualización de sus actuadores y sensores. Según sea el mensaje, el robot devolverá cierta información. Con esta información, y la proveniente del resultado del procesamiento de la petición entrante por cada aplicación receptora, se compone un único mensaje de respuesta que se reenvía a través del API al servidor web, con toda la información que el servicio espera para su correcto funcionamiento. El ejemplo para este caso pueden ser mensajes de éxito o fracaso de acceso a las interfaces del robot y el resultado concreto de la ejecución solicitada. La información se usa en última instancia para actualizar

el interfaz y el estado del servicio para que el cliente web pueda ver el resultado de su interacción. Se finalizaría así la iteración del proceso de “Ejecución Mixta” y quedaría a la espera de nuevas peticiones hasta la solicitud de finalización de ejecución, con la cual se liberaría la memoria asignada a todos los procesos en el lado servidor de ejecución y se detendría el contenedor de forma segura, pudiéndose dar por terminado el servicio web.

Capítulo 5

Validación Experimental

En este capítulo se detallan tanto los contenidos académicos construidos sobre la herramienta como las diferentes pruebas y experimentos que se utilizaron para testar y verificar la “Ejecución Mixta” y sus capacidades. Además de los experimentos, se describirá también su contexto y la relación de éste con lo que se quería probar, además de los resultados obtenidos.

5.1. Contenidos Académicos de Pruebas

Para testar y validar toda la implementación, se construyeron una serie de ejercicios con contenido académico que se sirven desde la aplicación remota y se apoyan en el proceso de “Ejecución Mixta” para los 3 supuestos de su diseño: *hardware* integrado en el equipo del cliente (cámara), simulación local y *hardware* externo (robots reales).

5.1.1. Ejercicio del Filtro de Color

Dado el origen de la idea, resulta lógico que la temática del primer ejercicio tuviese que ver con la visión artificial. El cuadernillo define un ejercicio consistente en un sencillo filtro de color, una de las tareas más simples de la visión

artificial que suele aparecer en todo proyecto de esta rama de la robótica. El objetivo del mismo es hacer el *tracking* de un objeto de la imagen en base a cierta propiedad, en este caso su color, para obtener su posición en cada fotograma proporcionado por la fuente de vídeo. Se desarrolló por tanto un *back-end* de ejercicio basado en un bucle de iteraciones con intervalo variable (en función de la duración de ejecución de un ciclo) que actúa como “plantilla” que permite al usuario colocar su código y que éste se comporte de manera cíclica, con una serie de métodos a su disposición para facilitar el proceso de desarrollo de su código como parar, restablecer o ejecutar la lógica. El primer reto a resolver es el acceso a la fuente de imágenes, también como parte de esta infraestructura del ejercicio. Sea cual sea la naturaleza del servidor de vídeo (un fichero estático almacenado en el sistema de archivos, un dispositivo de vídeo integrado o un sensor de vídeo conectado al equipo a través de USB) es necesario solventar el acceso al *hardware* del cliente web. Para el caso concreto de los sensores de imagen existe mucha funcionalidad resuelta que ya permite acceder a los dispositivos de vídeo detectados en el sistema anfitrión, que se puede obtener a través de *plugins* de ROS o incluso haciendo uso de funciones empaquetadas en OpenCV. Cabe mencionar que éstos métodos solucionan el acceso nativo y que el puente Docker se encarga hacer que esa información esté disponible para el *kernel* de Jupyter. Se optó por ambas herramientas para implementar una fuente de vídeo configurable y así abrir la posibilidad de seleccionarla de entre las opciones antes propuestas. Una vez conectado el código al flujo de vídeo, se continuó con la infraestructura de soporte del filtro de color que incluye funcionalidad gráfica de visualización mediante el envío de las imágenes crudas y procesadas por canales WebSockets, módulos de control de flujo para controlar la ejecución iterativa del código y poder implementar algoritmos reactivos y métodos con funcionalidad específica resuelta para poder trabajar en la solución al ejercicio que se ofrecen a través de un API de ejercicio al usuario. Todo ello se ubica en la “Ejecución Mixta” como una aplicación auxiliar que se ejecutaría como un agen-

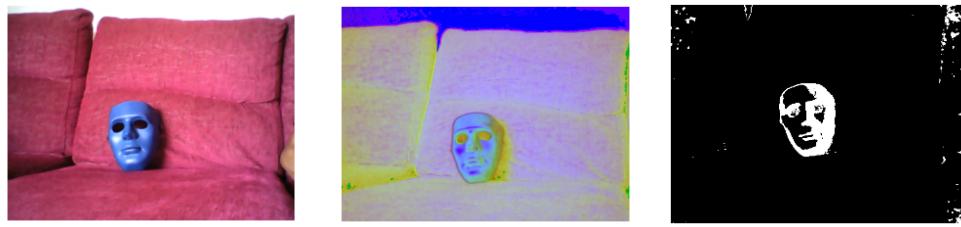
te de la capa de Bajo Nivel dentro del contenedor, y que aportaría la opacidad que se perseguía de cara a que el usuario no lidiase con problemas asociados al *hardware* y al bajo nivel, sino que dispusiese de las imágenes de su fuente a través de una sencilla instrucción en su código, y pudiese trabajar con ellas y visualizarlas con la misma facilidad. Para terminar con la construcción del primer ejercicio, se implementó una solución de referencia al mismo con una posible vía para obtener el resultado esperado, principalmente haciendo uso de las librerías OpenCV para el tratamiento digital. Aunque los algoritmos planteados para los ejercicios no son objeto de esta tesis, se describe brevemente a continuación el funcionamiento del filtro propuesto:

Ante la imagen de entrada



Figura 5.1: Input: Fotograma de la Fuente de Vídeo

Se aplicaría, en primer lugar, un suavizado a la imagen de entrada con un filtro Gaussiano para eliminar o reducir el posible ruido y, con él, los falsos positivos en el filtro. Luego convertiríamos el espacio de color de la imagen de entrada, típicamente RGB o BGR, al espacio HSV, donde resulta mucho más sencillo establecer límites para el filtrado de determinado color dado que esta



(a) Suavizado de la Imagen

(b) Conversión a HSV

(c) Imagen Umbralizada

Figura 5.2: Procesado de Imagen para el Filtro de Color

característica se encuentra completamente representada en la componente H, además de ser un espacio mucho más robusto a los cambios en intensidad de luz. Por último, se aplican los límites para obtener una imagen umbral binaria en la que los píxeles de la imagen de entrada cuyo valor de la componente H está entre las cotas establecidas quedan reflejados en la imagen B/N como píxeles de primer plano (de valor 255), y el resto de píxeles se etiquetan como píxeles de fondo (de valor 0). Para señalar el objeto que supera el filtro, bastaría con hacer una aproximación rectangular a los contornos del objeto blanco de la imagen umbral (detectados en aquellas regiones en las que la derivada es muy grande, cambios rápidos de píxeles blancos a negros). Se escoge el contorno más grande localizado para asegurar la identificación de todo el conjunto de píxeles que forman el objeto. Se pueden aplicar otras técnicas para mejorar el filtrado y eliminar ruido, como operaciones morfológicas o técnicas de post-procesado.

El proceso descrito se muestra gráficamente en la Fig. 5.2. De esta manera, nuestro cuadernillo de Jupyter tendría ya la lógica que resuelve el filtro, además de un recubrimiento de código auxiliar que contendría el *back-end* del ejercicio con los métodos de acceso a la fuente de vídeo, de recogida de imágenes, de visualización y control, etc. En la interfaz de Jupyter se pueden mostrar en todo momento las imágenes que reflejan el estado del proceso a través del API de programación de ejercicio y varias librerías gráficas (entre ellas OpenCV y matplotlib), como se ve en la Fig. 5.3.

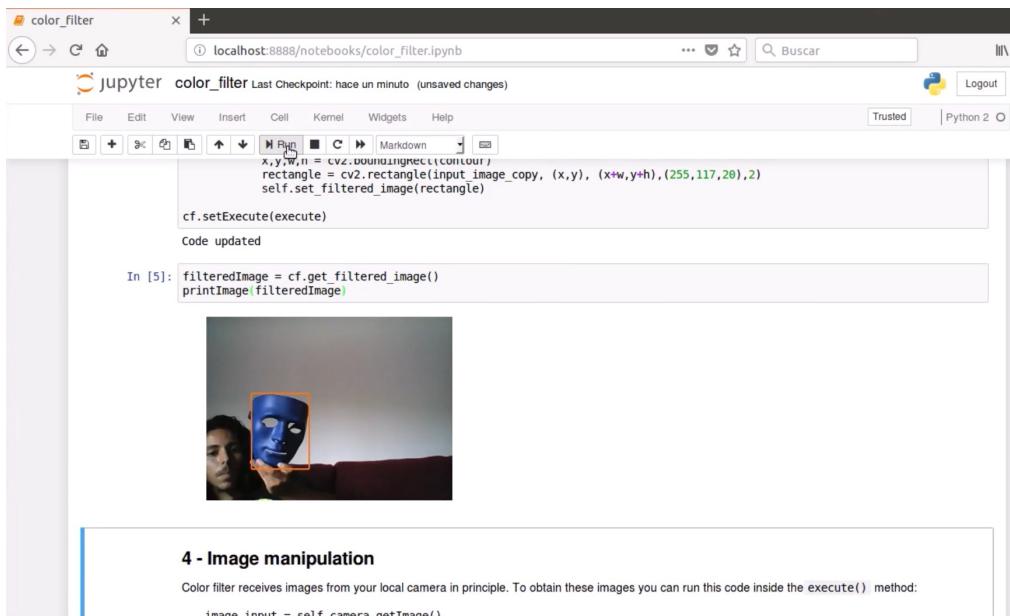


Figura 5.3: UI de Jupyter para el Filtro de Color

5.1.2. Ejercicio del Sigue Líneas con Fórmula 1

Este ejercicio, consistente en el clásico Sigue Líneas de la Visión Artificial, nos permitirá probar el mecanismo de simulación. Se trata de conseguir que un robot simulado con Gazebo, en este caso un Fórmula 1, procese las imágenes obtenidas con su cámara delantera para detectar una línea de determinado color e implementar un controlador que le permita seguirla.

La infraestructura de este ejercicio queda también recogida en un fichero de *back-end* que dispone, de manera similar al ejercicio anterior, un API de utilización del ejercicio que permite, en este caso, actuar sobre la simulación, es decir, acceder a las interfaces de actuación y sensorización del robot simulado y detener, pausar y reanudar la ejecución del código. Esta estructura de ejercicio incluye la disposición de los canales de ROS de intercambio de información entre Jupyter y el simulador, como ya se mostró en el capítulo anterior.

Para resolverlo, se realiza una sencilla segmentación basada en el color para

obtener la línea de la imagen, aplicando un filtro y operaciones morfológicas de cierre y erosión para eliminar los falsos positivos. El resultado del procesado sería similar al siguiente:

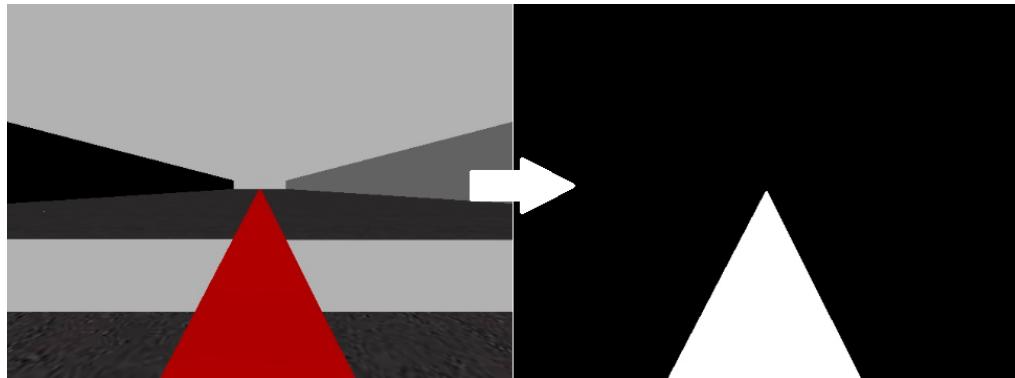


Figura 5.4: Procesado de la imagen del F1

De esta imagen resultante se obtiene el píxel central de la línea de la última fila de la misma, y con él se estiman las coordenadas en las que se encuentra dicho centro, objeto de seguimiento. Luego, se implementa un sencillo control basado en la desviación entre la posición actual y la posición objetivo (Listing 5.1)

```

1  if (deviation == 0):
2      self.motors.sendV(3)
3  elif (right_pixel_position[0] == 1000):
4      self.motors.sendW(-0.0000035)
5  elif ((abs(deviation)) < 85):
6      if ((abs(deviation)) < 15):
7          self.motors.sendV(1)
8  else:
9      self.motors.sendV(3.5)
10     self.motors.sendW(-0.000045 * deviation)
11 elif ((abs(deviation)) < 150):

```

```
12     if ((abs(deviation)) < 120):
13         self.motors.sendV(1)
14     else:
15         self.motors.sendV(1)
16         self.motors.sendW(-0.00045 * deviation)
17     else:
18         self.motors.sendV(1)
19         self.motors.sendW(-0.0055 * deviation)
```

Listing 5.1: Control basado en desviación

5.1.3. Ejercicio del Cuadrado con el dron Tello Real

Ese ejercicio se trata de realizar el más simple control de un robot real para que dibuje un cuadrado. En este caso, se utilizará un dron real que deberá despegar y aterrizar en el mismo sitio, después de haber completado las aristas de un cuadrado como recorrido.

El interés de este ejercicio radica en la inclusión al mecanismo de un *driver* para un robot real, en este caso un *driver* casero para el robot Tello (Listing. 5.2), un cuadricóptero de DJI e Intel cuyo precio de mercado está al alcance del consumidor medio. Una vez construido el *driver* en lenguaje Python, se creó un paquete PIP con el objetivo de poder utilizarlo desde el código del usuario, el cual lo importa como una librería normal y corriente y accede a sus funciones y métodos públicos para controlar el dron (Listing. 5.3).

```
1 #!/usr/bin/env python
2 # -*- coding: utf-8 -*-
3
4 import socket, threading, time, libh264decoder, cv2
5 import numpy as np
```

```
6  from math import pi as PI
7  from speed_thread import SpeedThread
8
9  MAX_VEL = 1.5 # m/s
10 MIN_VEL = 0.1 # m/s
11 MAX_ROT_VEL = 1 # deg/s
12 MAX_ROT_VEL = 360 # deg/s
13 ORANGE_MIN = np.array([117, 239, 76],np.uint8)
14 ORANGE_MAX = np.array([179, 255, 255],np.uint8)
15
16 class Tello:
17     """Wrapper class to interact with the Tello drone."""
18     def __init__(self, local_ip, local_port,
19                  command_timeout=.2, tello_ip='192.168.10.1',
20                  tello_port=8889):
21         # vels vector
22         # [
23         #     Right(+) / Left(-),
24         #     Forward(+) / Backward(-),
25         #     Up(+) / Down(-),
26         #     Yaw_right(+) / Yaw_left(-)
27         # ]
28         self._vels = [0, 0, 0, 0]
29         self.abort_flag = False
30         self.decoder = libh264decoder.H264Decoder()
31         self.command_timeout = command_timeout
32         self.response = None
33         self.frame = None # numpy array BGR
34
```

```
35      # socket for sending cmd
36      # -----
37      self.socket = socket.socket(socket.AF_INET,
38                                     socket.SOCK_DGRAM)
39      self.tello_address = (tello_ip, tello_port)
40      self.socket.bind((local_ip, local_port))
41      # -----
42
43      # thread for speed control
44      # -----
45      self.kill_event = threading.Event()
46      self.speed_thread = SpeedThread(self)
47      # -----
48
49      print("Conectando con Tello ..... ")
50      # to receive video -- send cmd: command, streamon
51      self.socket.sendto(b'command', self.tello_address)
52      print('[Tello] Preparando controlador')
53      self.socket.sendto(b'streamon', self.tello_address)
54      print('[Tello] Preparando flujo de vídeo')
55
56      # [...]
```

Listing 5.2: Snippet del Driver de Tello

```
1 from tello.tello_wrapper import Drone
2 tel = Drone('', 9005)
```

Listing 5.3: Uso del Driver

Como ya se dijo, este *driver* actuará como aplicación auxiliar dentro del contenedor de “Ejecución Mixta” que será lanzado por el secuenciador y añadido al mecanismo interno de paso de mensajes en un canal concreto, de tal manera que se consigue conectar el código que se ejecuta en Jupyter con el propio robot a través de la red interna.

En cuanto al algoritmo que permite solucionar el ejercicio, se trata simplemente de utilizar las funciones de control por posición incluidas en el *driver* implementado con los valores concretos de medida de la arista y ángulo de giro de 90°.

5.2. Experimentos Realizados

5.2.1. Servidor en Producción

Además de las pruebas realizadas sobre una aplicación que se servía de manera local durante el desarrollo se montó un entorno de pruebas sobre un servidor alojado en la universidad con un dominio accesible.

La motivación de ésta prueba es demostrar el funcionamiento compartido de la herramienta, y verificar que no existe barrera alguna en lo relativo a la ubicación física del cliente y del servidor de “Ejecución Mixta” y su *hardware* a controlar. Se puede testar también la capacidad de atravesar NATs y *firewalls* a través de Internet, además de los frecuentes problemas de origen cruzado de los protocolos de intercambio de datos. Se plantea por último el escenario para testar el enlace entre una aplicación robótica de naturaleza remota y el motor local de “Ejecución Mixta”.

Se analizaron los parámetros críticos relacionados con la comunicación a través de Internet desde distintos puntos de acceso (nacionales) y con máquinas de distintos rangos operacionales y distintas restricciones de acceso a la red. Los resultados se recogen en la siguiente tabla:

De la tabla de parámetros resultantes se deduce que el funcionamiento es in-

Navegador	Funcionamiento	Latencia	Ancho de Banda Consumido	Tiempo de Establecimiento de Comunicación
Chrome	100 %	38 ms	0.24 Mbps	2.8 s
FireFox	100 %	36 ms	0.23 Mbps	3.1 s
Opera	100 %	40 ms	0.24 Mbps	3.5 s
Safari	100 %	38 ms	0.28 Mbps	2.9 s
Internet Explorer	100 %	44 ms	0.13 Mbps	4.2 s

Tabla 5.1: Tabla de Resultados de Parámetros de Red.

diferente del navegador elegido, con mejor o peor desempeño en función de la conexión de red, y de la eficiencia de la tecnología de computación del propio navegador. Se puede ver que la latencia media no es para nada crítica, permitiendo un uso fluido de la herramienta en la web, y que en términos de ancho de banda el consumo es prácticamente equivalente al que se obtiene siendo cliente de un servicio de VOD, vídeo bajo demanda. Se demuestra con todo lo anterior que la herramienta de “Ejecución Mixta” está preparada para funcionar como parte de cualquier servicio con capacidad de recursos normal a través de la web.

5.2.2. Grupos de Pruebas: *betatesters*

Para no limitar el proceso de test a un único entorno cliente se reunió un conjunto de usuarios que accedieron a hacer las funciones de *betatesters* de manera voluntaria, lo que permitió ampliar el ámbito de experimentación y el alcance de las pruebas, además de su validez sustentada en la generalización, en medias aritméticas de las capacidades y en porcentajes de éxito y fracaso.

Cabe destacar que cada sujeto de pruebas disponía de su propio entorno, es decir, de su máquina con ciertas prestaciones y cierto sistema operativo a

cargo. Los *beta testers* tenían mayoritariamente distintas distribuciones de Linux y, en algunos casos, versiones de Windows. En la mayoría de los casos el cliente no disponía de instalación previa de ninguna de las herramientas de las que hace uso la “Ejecución Mixta” en su sistema, y en la totalidad de los casos no disponían de todas ellas.

Modalidad	Funcionamiento	Eficiencia	Carga media en Servidor	Carga media en Cliente
Simulación	100 %	60 %	15 %	85 %
Cámaras locales integradas	100 %	80 %	10 %	90 %
Robots Reales	80 %	95 %	0.5 %	99.5 %

Tabla 5.2: Tabla de Resultados de Sujetos de Pruebas.

Se observa en la tabla anterior que la totalidad de los usuarios pudieron acceder a la simulación a través de la aplicación derivando el cómputo a su propia máquina a través de la herramienta implementada. La eficiencia en este caso estuvo supeditada al *hardware* del que el cliente disponía para hacerse cargo de la ejecución, especialmente del *hardware* de aceleración gráfica. En todos los casos se experimentó un uso razonablemente bueno.

En cuanto al desempeño de la herramienta en conjunto con los sensores de visión integrados en la máquina del cliente se consiguió que funcionase para todos los sujetos. En este caso se liberaba de algo más de carga al servidor dado que la tasa de refresco de imágenes era más baja que la de la escena de simulación. Se comprobó que el procesado de imágenes que se intercambian por Internet es posible sin sufrir consecuencias temporales gracias al énfasis de la carga en el lado cliente. El acceso a las cámaras se garantizó en los sistemas operativos

testados.

Hubo una cantidad menor de pruebas realizadas frente a robots reales dada su escasez. No se consiguió funcionar en los SO basados en Windows dado que ningún usuario disponía de las versiones para las que Docker ofrece soporte. En el caso de los basados en Linux la eficiencia fue prácticamente máxima, además del grado de explotación de la herramienta, que si bien nació para soportar la ejecución local de algoritmos de visión artificial, parece tener mayores ventajas para el caso de uso de los robots, siempre teniendo en cuenta el tipo de aplicación sobre el que se está utilizando la herramienta.

Se infiere de lo anterior que la relación eficiencia-prestaciones fue en todos los casos al menos ligeramente positiva, y que la “Ejecución Mixta” funciona siempre para todos los supuestos para los que da soporte.

Conclusiones y Líneas Futuras

Este capítulo recogerá las inferencias extraídas durante el desarrollo de esta tesis, mayoritariamente provenientes del periodo de investigación y confirmadas durante la implementación, así como unos ilustrativos casos de uso a modo de conclusión. También dedicará una sección a listar las líneas futuras de investigación que podrían potenciar el desarrollo o abrir nuevos frentes para hacerlo más completo.

6.1. Conclusiones

Se distinguen dos grupos de conclusiones dada la estructura del presente proyecto que tienen relación con el planteamiento inicial de objetivos y el desarrollo e implementación en sí, respectivamente. En primer lugar cabe destacar que la herramienta desarrollada cumple los 4 sub-objetivos propuestos en primera instancia, así como todas las características deseables para un fácil y versátil acceso a aplicaciones robóticas. Además, la implementación final cuenta incluso con algunas propiedades no programadas en el planteamiento original, que lo hacen más robusto y seguro.

Se ha llevado a cabo un proceso completo de estudio de herramientas rela-

cionadas con la robótica y su accesibilidad, además de algunas otras con aparente carencia de dicha relación con el campo tecnológico robótico cuyos usos pueden ser replanteados para cumplir alguna función que sí tiene relación. Se ha desarrollado la capacidad de elección y la destreza necesarias para seleccionar un subconjunto de herramientas, aplicaciones y entornos que condujeron a la solución propuesta para un problema sin solución clara previa, además de la habilidad de discernir entre necesidad de desarrollar comportamientos y lógicas desde cero y la posibilidad de adaptar un programa o aplicación existente para que encajara en las necesidades del trabajo. Se consiguió descartar las vías que pudieran suponer no llegar a la solución o un cuello de botella en el desarrollo de la misma, llegando finalmente a la propuesta de un método válido y de funcionamiento verificado por distintas vías que constituye una buena solución para la idea expuesta entre todas las plausibles para este problema no abordado previamente. El cumplimiento de los objetivos demuestra que se ha desarrollado la capacidad de orientación, dirección y ejecución de un proceso de desarrollo asociado a una nueva idea, sin ser necesario el disponer de un guión preestablecido o un esquema de trabajo. En cuanto al periodo de implementación, se extrajo una serie de conclusiones más específicas acerca de la herramienta desarrollada, la coyuntura tecnológica en relación a la robótica y del modo de trabajar al enfrentarse a un proyecto robótico:

- Los primeros pasos de este proyecto desembocaron en un problema sin aparente solución, el cual fue abordable cambiando el enfoque inicial, algo muy común en proyectos tecnológicos. Esto arrojó luz sobre el método de “trabajo previo” que es altamente recomendable llevar a cabo antes de empezar un proyecto: plantear un estudio previo de la viabilidad del mismo y de cada uno de los módulos que se quiere incluir en el diseño. Así pues, se dedujo que no sólo la implementación debe estar sujeta a un proceso iterativo de verificación de calidad como el planteado inicialmente, sino que también el estudio debe estar sujeto a una constante supervisión en la

que se busque mejorar la idea o el método e incluso en la que se replanteen los objetivos por el bien final del proyecto, lo que permite lidiar con potenciales imprevistos e incrementar la robustez y calidad.

- Comprender el funcionamiento latente de los agentes que se utiliza en un desarrollo facilita mucho su integración y permite aprovechar la flexibilidad que pueden ofrecer. Así pues, a la hora de escoger entre distintas opciones se ha de ser estricto y restrictivo con los objetivos que se pretende alcanzar, dado que suele haber diferentes soluciones a un mismo problema. Personalmente considero que el software robótico es muy versátil hoy en día gracias a las arquitecturas distribuidas y modulares que abren la puerta a la integración en todo tipo de proyectos, permitiendo potenciar el alcance de cada nuevo desarrollo.
- Hay una necesidad social de robótica. Muchas aplicaciones laborales, domésticas y relacionadas con dar servicio a las personas están siendo completamente automatizadas y dotadas de la presencia de robots, desembocando en el gran incremento de la necesidad de formación en el campo, que hasta ahora resulta en parte complicado de acceder. La “Ejecución Mixta” supone un paso adelante en el acceso a la formación e investigación en robótica.
- Aunque hasta hace unos pocos años era impensable usar las tecnologías web en ámbitos robóticos dadas sus antiguas limitaciones, esto ha cambiado diametralmente hasta ofrecer un contexto tecnológico en el que esta vía de desarrollo es ideal para construir aplicaciones que permitan acercar la robótica a la gente.
- Finalmente se ha querido destacar el hecho de que, a pesar de la primera impresión, un proyecto robótico no sólo requiere exigir conocimientos en hardware y software robótico, sino que también requiere el uso y conocimiento de herramientas asociadas a todo tipo de materias. Para el caso concreto de esta tesis ha sido necesario un nivel alto de conocimientos en

telecomunicaciones y protocolos de intercambio de datos, así como adquirir conocimientos y destreza en el campo de la ingeniería de sistemas. Es por eso que se concluye que en proyectos de mayor calibre y envergadura se forma un grupo de desarrollo compuesto de expertos en distintos ámbitos, en tanto que la robótica agrupa muchos otros campos científicos y tecnológicos como la electrónica, mecánica, telecomunicaciones, sistemas, programación y física y matemáticas entre otras.

6.2. Análisis de prestaciones

Se recoge en la siguiente tabla (6.1) un análisis de los puntos favorables y desfavorables de la “Ejecución Mixta” desde el punto de vista de las aplicaciones robóticas que valoran si incorporar o no esta herramienta a su infraestructura:

Pros	Contras
Accesible (API)	Requiere conectividad de red
Multiplataforma	Depende del soporte de Docker (las capacidades en MacOS están algo limitadas por el momento)
Versátil (modular). Fácil de integrar	
Seguro	
Fácil de usar	

Tabla 6.1: Tabla de Análisis de Prestaciones.

6.3. Casos de Uso

Se ejemplifica a continuación algunos de los casos en los que el uso de la “Ejecución Mixta” es idóneo:

1. Se dispone de un servicio que se quiere prestar al público, pero no se dispone de soporte físico o recursos para prestarlo. La “Ejecución Mixta” pone el énfasis en el lado cliente, liberando totalmente al lado servidor de carga computacional.
2. Se pretende estudiar o entrar en contacto con el campo de la robótica, sin conocimiento ni habilidades suficientes para disponer un entorno práctico de aprendizaje. Esta herramienta envuelve toda la complejidad de bajo nivel en un API de sencilla utilización.
3. Se dispone de *hardware* robótico específico y se quiere conectar con alguna clase de aplicación. La solución de “Ejecución Mixta” es fácilmente ampliable con soporte para nuevos controladores, y sencillamente integrable con cualquier tipo de aplicación en la que se pueda actuar sobre el mecanismo de mensajería.
4. Se quiere trabajar con lógica robótica sin tener robots ni infraestructura típica. La “Ejecución Mixta” funciona sobre cualquier sistema operativo, y no requiere disponer de nada más que un navegador web para tratar con robótica, en el que se puede tener soporte de simulación.

6.4. Trabajos Futuros y Líneas Futuras de Investigación

Dada la “juventud” de la idea de este proyecto, su solución queda abierta a la alteración de algunas de sus partes para introducir mejoras e incrementar la eficiencia, o a la ampliación de sus características o de la funcionalidad que ofrece. Algunos ejemplos, escogidos por ser los de mayor prioridad, son los siguientes:

Actualmente la herramienta Docker utilizada sobre Windows tiene claras limitaciones en cuanto al acceso de *hardware* conectado a través de los puertos USB del sistema. Se hará necesario revisitar la herramienta para las futuras versiones de Docker, en las cuales se asegura la superación de estas limitaciones, para garantizar el completo y correcto funcionamiento sobre dicho sistema operativo.

Se pretende crear un REST API de “Ejecución Mixta” que permita a las aplicaciones ejercer cierto control sobre algunas de las tareas de la herramienta, pudiendo lanzarlas, interrumpirlas o detenerlas si se necesita. En relación con lo anterior, se planea construir un envoltorio profesional para la herramienta, en vistas a extender su uso.

En cuanto al enriquecimiento “en caliente” de la herramienta, estamos trabajando en la incorporación de un mecanismo que permita al usuario crear e incluir nuevos *drivers* para sus robots o incluso sus propias versiones de los controladores de los robots ya soportados. La idea es que en tiempo de ejecución (sin que sea necesario el lanzamiento de una nueva versión de la herramienta) cada usuario pueda customizar su “Ejecución Mixta”, adaptando el soporte al uso que quiere hacer de la aplicación robótica a la que accede.

También se está trabajando en una segunda versión del contenedor Docker que reduzca casi a la mitad su peso actual (espacio en disco) a través de la utilización eficiente de las capas de construcción de Docker, cuyo tamaño se puede reducir mediante la recolocación inteligente de las instrucciones de instalación de herramientas afines o similares en grupos.

6.5. Vídeos Demostrativos

En los siguientes vídeos se puede encontrar un ejemplo de Ejecución Mixta para cada ejercicio implementado que demuestra su funcionamiento.

1. Ejercicio del Filtro de Color

<https://www.youtube.com/watch?v=QnhI0jsEtp4>

2. Ejercicio del Sigue-Líneas con Fórmula 1

<https://www.youtube.com/watch?v=39MVVn3u8SE>

3. Ejercicio del Cuadrado con el dron Tello

https://www.youtube.com/watch?v=4GVvF_ce6Ko

Referencias

- Boehm, B. W. (1986). A Spiral Model of Software Development and Enhancement . *Special Interest Group on Software Engineering (ACM SIGSOFT)*, 11, 14–24.
- Construcción de Contenedores Docker.* (s.f.). <https://docs.docker.com/develop/>.
- Desarrollo con GzWeb [Manual de software informático]. (s.f.). Descargado de <http://gazebosim.org/gzweb.html>
- Documentación Apache [Manual de software informático]. (s.f.). Descargado de <https://httpd.apache.org/docs/2.4/>
- Documentación Django [Manual de software informático]. (s.f.). Descargado de <https://docs.djangoproject.com/en/1.11/>
- Documentación Docker [Manual de software informático]. (s.f.). Descargado de <https://docs.docker.com/docsarchive>
- Documentación Jupyter [Manual de software informático]. (s.f.). Descargado de <https://jupyter.org/documentation>
- Documentación OpenCV [Manual de software informático]. (s.f.). Descargado de <https://docs.opencv.org/2.4.13.7/>

Documentación ROS [Manual de software informático]. (s.f.). Descargado de <http://wiki.ros.org/>

Formato SDF. (s.f.). <http://sdformat.org/>.

Gazebosim.org. (s.f.). *Tutoriales de Gazebo.* <http://gazebosim.org/tutorials>.

Hintjens, P. (s.f.). *Guía del Protocolo 0MQ.* <http://zguide.zeromq.org/page:all>.

J.M. Cañas, E. P. F. R., A. Martí, y Calvo, R. (2016). Entorno docente para la programación de la inteligencia de los robots. *Revista Iberoamericana de Automática e Informática Industrial*, 15(4), 404–415.

JupyterTeam. (2019-10-22). *Arquitectura interna de Jupyter.* https://jupyter.readthedocs.io/en/latest/architecture/how_jupyter_ipython_work.html.

Koenig, N., y Howard, A. (2004, Sep). Design and use paradigms for gazebo, an open-source multi-robot simulator. En *Ieee/rsj international conference on intelligent robots and systems* (p. 2149-2154). Sendai, Japan.

Migliavacca, M., y Ceriani, S. (2012). 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*.

Proyecto Jupyter [Manual de software informático]. (s.f.). Descargado de <https://jupyter-notebook.readthedocs.io/en/stable/notebook.html>

Sailer, Z. (27 Nov 2018). *Documentación del REST API de Jupyter.* <https://github.com/jupyter/jupyter/wiki/Jupyter-Notebook>

-Server-API y <http://petstore.swagger.io/?url=https://raw.githubusercontent.com/jupyter/notebook/master/notebook/services/api/api.yaml>.

SDKRobotics. (s.f.). *Prototipo de driver para el dron Tello.* <https://github.com/dji-sdk/Tello-Python>.

W3Schools. (s.f.). *Guía de JavaScript para Tecnologías Web.* <https://www.w3schools.com/js/>.