



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

ENERO 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: Enero 2020

Tribunal: _____
Dr./Dra.

Dr./Dra.

Dr./Dra.

“The true knowledge is ...”

Índice general

Índice de Tablas	XI
Índice de Figuras	XIII
Agradecimientos	XV
Resumen	XVII
Abstract	XIX
1. Introducción	1
1.1. Preludio	1
1.2. Introducción	2
1.3. Estado del Arte	4
1.4. Planteamiento de la Idea	9
1.4.1. Motivación	9
1.4.2. Proyecto Planteado	11
1.4.3. Punto de Partida	12
2. Objetivos y Metodología de Trabajo	15
2.1. Objetivos del Proyecto	15

2.2. Metodología Empleada	16
2.2.1. Metodología de Investigación	17
2.2.2. Metodología de Implementación	18
3. Infraestructura Utilizada	21
3.1. Herramientas utilizadas	21
3.1.1. Tecnologías Web	21
3.1.2. Proyecto Jupyter	24
3.1.3. ROS (Robot Operating System)	27
3.1.4. Plataforma end-to-end Docker	29
3.1.5. Simulador Gazebo	31
3.1.6. Framework Django	34
3.1.7. Biblioteca OpenCV	37
3.1.8. Lenguajes de Programación: Python, JavaScript	38
4. Implementación de la “Ejecución Mixta”	41
4.1. Diseño e Infraestructura de la Aplicación	41
4.1.1. Capa de Aplicación	43
4.1.2. Capa de Comunicación	44
4.1.3. Capa de Seguridad	45
4.1.4. Capa de Bajo Nivel	48
4.2. Proceso de Implementación	51
4.3. Funcionamiento	72
4.3.1. Flujograma	72
4.3.2. Mecanismo subyacente de la Herramienta	74
5. Validación Experimental	77
5.1. Servidor en Producción	77
5.2. Grupos de Pruebas: <i>betatesters</i>	79

6. Conclusiones y Líneas Futuras	81
6.1. Conclusiones	81
6.2. Análisis de prestaciones	84
6.3. Casos de Uso	85
6.4. Trabajos Futuros y Líneas Futuras de Investigación	85
6.5. Videos?	86
Referencias	87

Índice de Tablas

5.1. Tabla de Resultados de Parámetros de Red.	78
5.2. Tabla de Resultados de Sujetos de Pruebas.	79
6.1. Tabla de Análisis de Prestaciones.	84

Índice de Figuras

1.1.	Aplicaciones actuales de los robots.	2
1.2.	Robots Simulados soportados en Robot Ignite Academy.	6
1.3.	Simulación con AWS RoboMaker.	7
1.4.	Simulación de aplicación robótica con JdeRobot.	8
1.5.	Conexión Local de Colaboratory	13
2.1.	Modelo en Espiral del Proceso de Desarrollo Software.	19
3.1.	Evolución de las Tecnologías Web	22
3.2.	Estructura de Docker	31
3.3.	Motor de físicas de Gazebo	33
3.4.	Modelos de Simulación en Gazebo	34
3.5.	Arquitectura de Django	36
3.6.	Herramientas de procesado con OpenCV: Segmentación de caras.	37
4.1.	Arquitectura de la Ejecución Mixta para Aplicaciones de Robótica.	42
4.2.	Módulo de Acceso: Capa de Aplicación	43
4.3.	Módulo de Acceso: Capa de Comunicación	45
4.4.	Módulo de Gestión: Capa de Seguridad	45
4.5.	Volumen de Docker	47
4.6.	Módulo de Gestión: Capa de Gestión del Bajo Nivel	48

4.7.	Mensaje de Entrega del Cuadernillo o Notebook	49
4.8.	Mensaje de Respuesta de Petición de Ejecución	50
4.9.	Input: Fotograma de la Fuente de Vídeo	53
4.10.	Procesado de Imagen para el Filtro de Color	54
4.11.	UI de Jupyter para el Filtro de Color	55
4.12.	UI de la Aplicación en Django	56
4.13.	UI de Simulación de la Aplicación	56
4.14.	Capura de Paquetes del Mecanismo de Comunicación de Jupyter	58
4.15.	REST API de Jupyter	58
4.16.	Escenario de Simulación	66
4.17.	Flujograma	73

Agradecimientos

Thanks to all ...

Resumen

Esta tesis desarrolla ...

Abstract

This thesis develops ...

Capítulo 1

Introducción

1.1. Preludio

Si bien 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 los que son idóneos como las tareas relacionadas con logística de almacenes y envíos de mercancía o exploración espacial.

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

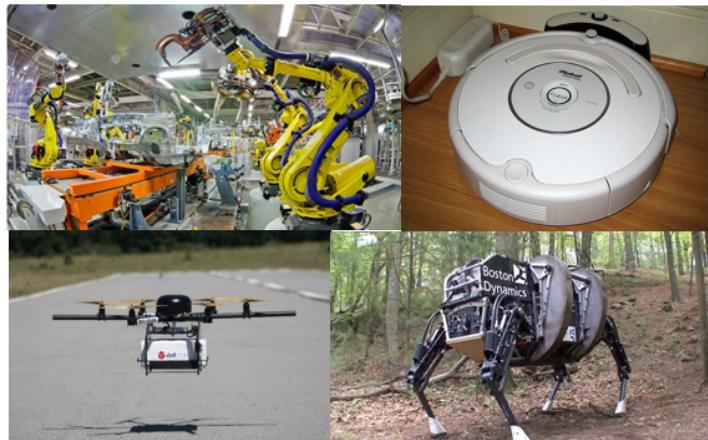


Figura 1.1: Aplicaciones actuales de los robots.

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, 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 impulsarán el desarrollo de este sector en búsqueda del beneficio social.

1.2. 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 paralelas 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 al mismo tiempo mayor importancia y mayor valor contributivo para perpetuar el camino de la robótica. Se llegó a una única conclusión que motivó el planteamiento de la idea que aquí se detalla, la cual si bien es difícil de resumir en una frase, hemos decidido bautizar como “hacer la robótica más accesible”.

Aunque, como ya se ha dicho, nuestro entorno actual engloba robots en grandes cantidades, 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 las circunstancias y 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 y de investigación 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. En base a mi experiencia reciente puedo decir que el énfasis se pone, inevitablemente, “sobre el papel” en lugar de “en la masa”.

Se observó 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.

Así las cosas, la conclusión a la que se llegó pasa por la creación de herramientas que simplifiquen el uso docente y de investigación de la robótica, poniendo los cimientos para que las nuevas y actuales generaciones puedan abstractarse en cierto sentido de los problemas generales asociados al trabajo con robots, y puedan centrarse en el objeto de su investigación o aprendizaje.

Esta memoria busca detallar de la manera más fiel posible el completo proceso de estudio y análisis de herramientas de naturaleza variada relacionadas con el desarrollo e implementación de aplicaciones asociadas al sector robótico y automático, 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**.

Se tratará de evaluar la validez de la idea que se propone y su viabilidad en forma de reseña crítica a través de la implementación de la lógica e infraestructura que permita cumplir el propósito, o al menos acercarse a él, y de la evaluación de los resultados que se obtenga de su empleo. 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.

1.3. Estado del Arte

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 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 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 o investigación 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:

- Una plataforma con un gran potencial es la “*Robot Ignite Academy*¹” de la empresa The construct². 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 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 servi-

¹<https://www.robotigniteacademy.com/en/teachers/>

²<https://www.theconstructsim.com/>

cios 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. 1.2).



Figura 1.2: Robots Simulados soportados en Robot Ignite Academy.

→ Otro gran ejemplo muy reciente es el entorno *AWS RoboMaker*³ de Amazon. Se trata de un servicio 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. 1.3). No ofrece soporte para robots reales, aunque sí proporciona un servicio de administración de flotas que simplifica en gran medida el paso de la simulación al sistema físico. Se trata de un entorno web.

→ Existen otro tipo de plataformas menos extendidas, en el ámbito nacional e

³<https://aws.amazon.com/es/robomaker/>

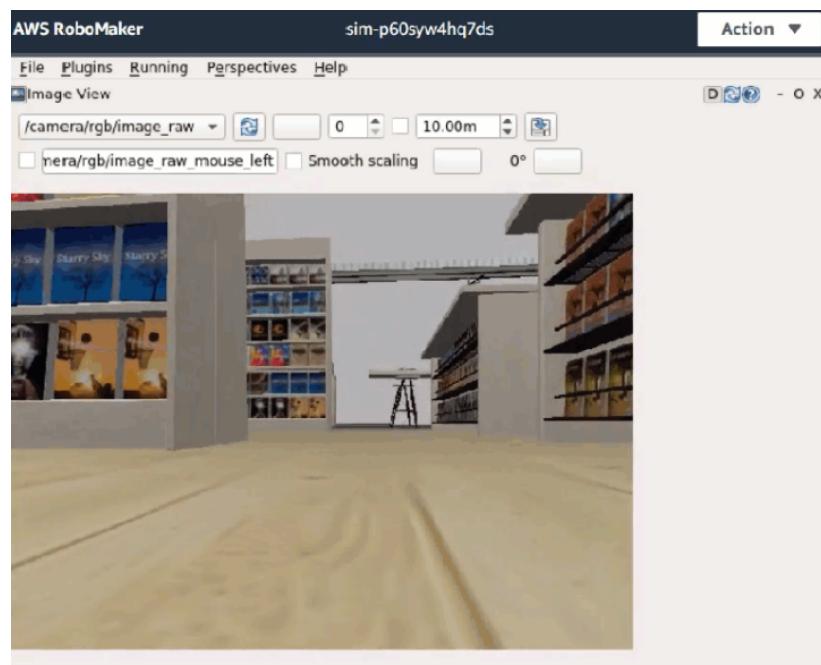


Figura 1.3: Simulación con AWS RoboMaker.

internacional, como JdeRobot⁴, 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 los 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 libre* ofrece calidad profesional (Fig. 1.4) 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.

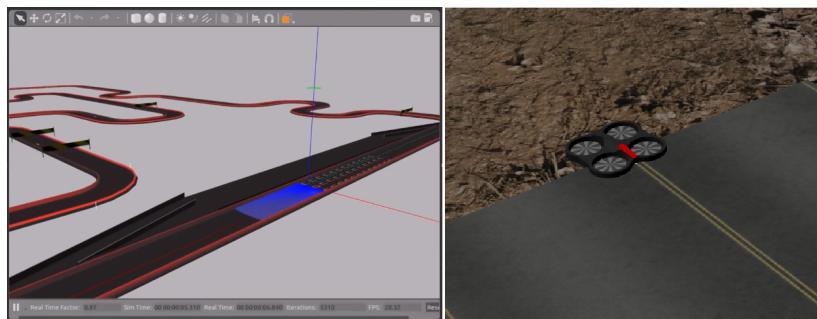


Figura 1.4: Simulación de aplicación robótica con JdeRobot.

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

⁴<http://jderobot.org>

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.

1.4. Planteamiento de la Idea

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.

1.4.1. Motivación

La propuesta de la idea para 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.

Por un lado 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 genera costes**, y que estos costes son elevados.

Por otro lado, 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 elemento *hardware* remoto y acceder a los resultados en tiempo de ejecución sería una ardua tarea. Por tanto, el análisis realizado destacó que **desarrollar aplicaciones robóticas requiere disponer de realimentación, visual y paramétrica, en tiempo real**, sólo alcanzable a través de la observación de sistemas electromecánicos reales.

Por último, 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, **carence de hardware específico. Sin embargo, sí que tiene acceso a sistemas más básicos**. Esta circunstancia resultó ser la tercera clave extraída del estudio previo.

Con todas ellas y sin perder de vista el objetivo de “Acercar la robótica a la gente”, se ideó un proyecto que busca sortear esos obstáculos. Quisimos proponer un método que permitiese a la vez minimizar los recursos consumidos por la aplicación robótica y reducir su curva de entrada, que permitiese trabajar indistintamente con entornos simulados y reales y que permitiese al usuario final

observar el resultado de su proyecto sobre el robot del que dispone.

1.4.2. Proyecto Planteado

La “**Ejecución Mixta**” para proyectos de robótica y visión artificial surge de la conjunción de las ideas anteriormente planteadas, persiguiendo incrementar 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.

Esta idea se planteó con una serie de características intrínsecas que dan forma a los objetivos que se quiere lograr. En tanto que se quiere facilitar el uso de la robótica, deberá existir 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 suponer costes sobre los constructores de las aplicaciones que necesitan usar para su trabajo o estudio y que tampoco quieren que se les cobre su utilización (debido en muchos casos a que no se pueden afrontar tasas a largo plazo sin resultados prometedores en plazo corto), la idea que se expone debe estar 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.

Queda de manifiesto que este proyecto busca 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.

1.4.3. Punto de Partida

El primer instinto al comenzar el proceso de investigación fue explorar si existía alguna aproximación previa a lo que queríamos conseguir, principalmente para verificar que la solución es posible y no comenzar a caminar por una vía sin salida, y en segunda instancia para partir de algún punto. Encontramos la herramienta *Colaboratory*⁵ de Google, un entorno que permite escribir 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. 1.5). Esto se aproxima en cierto modo a nuestros objetivos, aunque en nuestro caso debe extenderse el acceso a todo el *hardware* disponible en la máquina, principalmente. Sin embargo, al encontrar esta herramienta supimos que existía una manera de comunicarse con al menos un elemento *hardware* remoto desde una aplicación de cualquier carácter servida a través de un navegador. Adaptamos y centramos nuestros esfuerzos en esta línea de investigación.

Basándonos en esta estructura existente, continuamos la búsqueda de herramientas para implementar 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

⁵<https://colab.research.google.com/notebooks/welcome.ipynb>

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 análisis y resultados al ejecutarse en sus propios equipos y sistemas robóticos locales, quedando todas las tareas de bajo nivel involucradas en la aplicación robótica “escondidas” a nivel de usuario y solventadas por la aplicación 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 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.

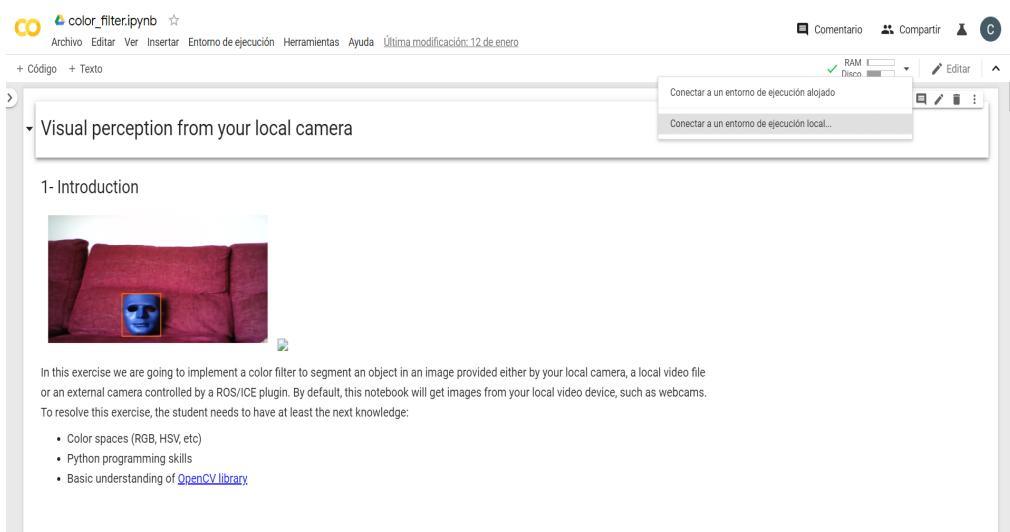


Figura 1.5: Conexión Local de Colaboratory

Capítulo 2

Objetivos y Metodología de Trabajo

Este capítulo detallará los objetivos impuestos tras el trabajo de análisis inicial que se pretende 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 otros sub-objetivos que surgieron durante el desarrollo reservando los detalles de los mismos para el punto en el que sea relevante. Así mismo, se expondrá la metodología bajo la que se organizó el proyecto.

2.1. Objetivos del Proyecto

Este trabajo está compuesto por cuatro grandes objetivos impuestos 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, componentes y entornos más utilizados en los proyectos robóticos.

- Se trata de analizar las características de cada agente de estudio para poder elegir o descartar con criterio su incorporación al proyecto.
2. Probar y seleccionar aquellas herramientas que puedan adaptarse al problema que se pretende resolver.
- Este hito pasa por construir diseños preliminares del entorno que engloba un cliente y un sistema remoto que se quieren interconectar.
3. Construir una nueva solución en forma de módulo que resuelva un problema para el que aún no existe una solución clara.
- 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 resultado contiene las características objetivo, que son:
- Accesible (Multiplataforma, Multilenguaje).
 - Soporte para simulación y sistemas reales.
 - Ejecución en el cliente.
 - Fácil de usar.
 - Versátil (Visión Artificial, Robots de cualquier clase).
4. Llevar a cabo un exhaustivo programa de pruebas para evaluar la validez y la viabilidad de la idea planteada y su implementación.

Estos objetivos establecen a su vez un esquema de trabajo que dictará la manera en que se desarrolla, realizando siempre una supervisión para garantizar que se cumple cada meta propuesta antes de pasar a la siguiente.

2.2. Metodología Empleada

Este proyecto se dividirá, principalmente, en dos partes bien diferenciadas: **etapa de investigación y etapa de implementación**. Estableceremos de ante-

mano una serie de reglas 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, 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, con lo cual se podían preparar las reuniones de antemano e incluso intercambiar ideas y sugerencias de manera anticipada².

2.2.1. 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. 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 de las ca-

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

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

racterísticas 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 encasen con los requisitos de nuestro proyecto, y luego se trató de descartar, en un último ciclo, 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 el set de características que habíamos establecido.

Esta filosofía se respetó 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.

2.2.2. Metodología de Implementación

Para la consecución del resto de 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

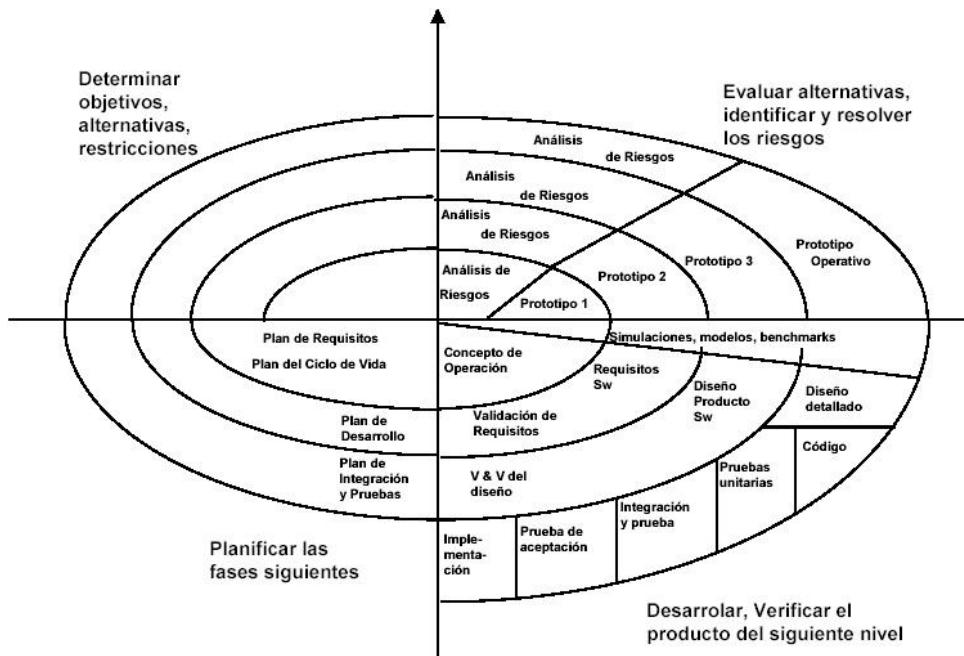


Figura 2.1: *Modelo en Espiral del Proceso de Desarrollo Software.*

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 el correspondiente *set* 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 que conformará la infraestructura del proyecto. Se introducirá cada una de las herramientas y se especificará tanto las razones bajo la elección de la misma para el proyecto como la función que cumplirá en él.

3.1. Herramientas utilizadas

El presente apartado es el resultado de los meses de investigación, en los cuales estudiamos numerosas herramientas, entornos y plataformas aplicables a la robótica que podían aportar algo a nuestros intereses. Finalmente se escogieron los que se presentan a continuación para formar parte de la solución de “Ejecución mixta” para aplicaciones robóticas.

3.1.1. Tecnologías Web

En los primeros días de exploración salió rápidamente a la luz el entorno en el que debíamos acotar la búsqueda e investigación. Recordemos que el fin último es aumentar la accesibilidad de la robótica, y que en accesibilidad no existe

ningún ámbito que supere a las tecnologías web. 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 si quiera superar los límites del navegador para acceder a dispositivos externos. Todo esto ha cambiado hasta tal punto en que se pueden crear aplicaciones tan exigentes como juegos en realidad virtual, procesamiento de imágenes, la comúnmente llamada “minería de datos” o KDD e incluso tareas de aprendizaje automático gracias a la habilitación del acceso 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. 3.1).

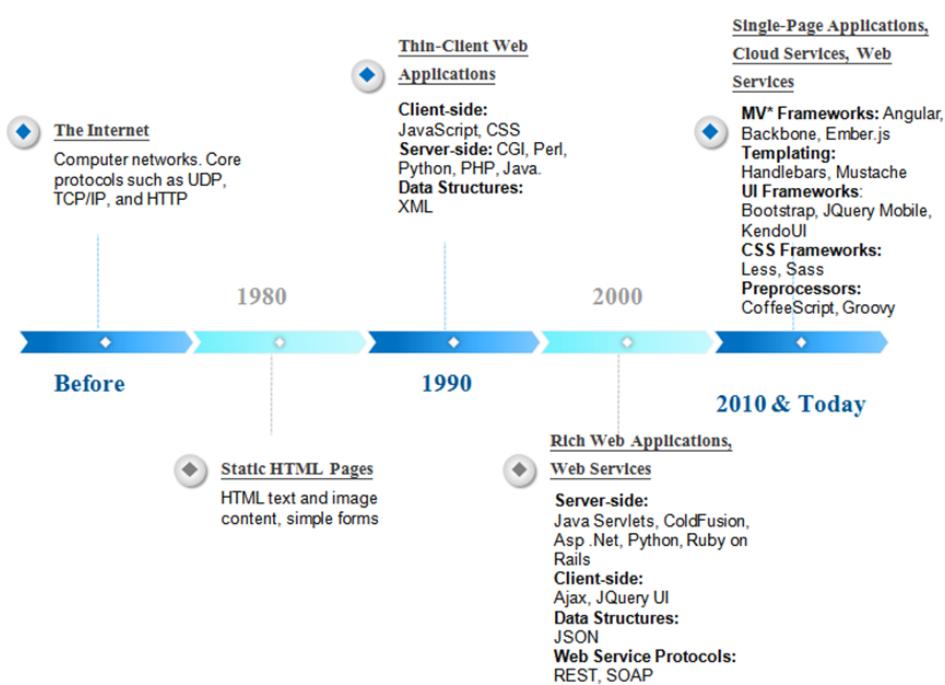


Figura 3.1: Evolución de las Tecnologías Web

En la actualidad, se puede construir aplicaciones para casi cualquier caso de uso, incluido el de la robótica, completamente formados 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. Precisamente por eso se escogió este tipo de tecnologías para establecer la base del proyecto, que nos permitirán conectar la parte cliente de la aplicación con el lado servidor para organizar un proceso de ejecución compartida utilizando protocolos preestablecidos que conozcan ambas partes, cumpliendo la función de mediador entre ellas y estableciendo la jerarquía que permita orquestar todo de manera transparente para el usuario.

Entre las funciones 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. Como se puede ver, todas ellas sirven para distintos tipos de tareas, de manera que no se puede decir que las tecnologías web formen una parte específica de este proyecto, sino más bien que actúan de base y “pegamento” para estructurar todas las que se exponen a continuación, además de facilitarnos un entorno de trabajo completo.

Por último, lo que hace ideal a la web para este proyecto es que su acceso, como ya se ha dicho, es 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. Los mecanismos básicos de seguridad de la red, como *firewalls* y NATs no permi-

ten ciertos tipos de comunicaciones que se consideran peligrosas, como aquellas que provienen de subredes externas o de sitios desconocidos. Con una aplicación remota no podemos actuar sobre el sistema del cliente, pero si podemos “hablar” con el navegador, que a su vez puede ejercer control sobre el sistema. En otras palabras, podemos utilizar el navegador como una especie de *proxy* que redirija los mensajes específicos de nuestra aplicación a su destinatario, siempre y cuando éste tenga un interfaz web o sea accesible a través de ella.

3.1.2. 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 elementos 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 bajo 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 navegador, 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 usuarios que pueden contener todo lo anteriormente mencionado, completamente legibles por usuarios humanos pero con ca-

¹<https://jupyter.org/>

pacidad 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.

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 ha 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 ha 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. Por tanto, mediante la combinación de estas dos características, obtenemos una herramienta que puede actuar de editor de texto para el código del usuario, y que nos proporciona herramientas para el acceso a su *hardware* de modo remoto, siempre y cuando sea el usuario quien escribe el código.

Así las cosas, la parte de ejecución de la aplicación quedaría resuelta con Jupyter, y se hace necesario desarrollar una mecanismo de comunicación que permita introducir en este *kernel* del cliente un código generado remotamente,

que contendría la aplicación robótica sobre la que un hipotético cliente querría construir, como parte de su instrucción o investigación. Dejaremos la solución adoptada para más adelante.

Los *Notebooks* que ofreceremos en este caso 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.

3.1.3. ROS (Robot Operating System)

ROS (Robot Operating System)² es un *middleware* flexible ideado para desarrollar *software* robótico. Proporciona multitud de bibliotecas, paquetes de código y herramientas para ayudar a los desarrolladores de software para aplicaciones relacionadas con la robótica en todos sus ámbitos a crear aplicaciones de robots. Su uso está muy extendido por proyectos de todo el mundo, desde caseros a profesionales, dadas sus latentes ventajas frente a otros soportes similares. Su extensión se debe en parte a que surgió en una etapa en la que no existía un único 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, de manera que el desarrollo de un proyecto en muchos casos se tornaba arduo, además de la complejidad presente a la hora de extender el proyecto, 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 evita todos estos problemas al ofrecer todos los módulos necesarios para implementar el código y llevarlo al robot, de principio a fin, todo en un mismo sitio distribuido bajo una licencia BSD de código abierto. Permite la abstracción de la capa *hardware*, proporciona controladores de dispositivo (*drivers* o *plugins*), bibliotecas, visualizadores, resuelve el intercambio de mensajes

²<https://www.ros.org/>

y facilita la administración de paquetes entre muchas otras cosas.

Resulta bastante lógica su elección para este proyecto, en el que precisamente queremos lograr esa abstracción del bajo nivel para poder ofrecer un entorno de desarrollo de aplicaciones de alto nivel para que el usuario se sienta cerca de la robótica, y no se vea forzado a abandonar un proyecto por falta de recursos o conocimientos específicos. Cabe destacar también que ROS ofrece soporte para distintos lenguajes, entre ellos Python, y que dispone canales de comunicación automáticos configurables y parametrizables que simplifican en gran medida la comunicación con el *hardware* y con aplicaciones externas.

En esta línea destaco otra de las fortalezas de ROS, que es su integración con el simulador Gazebo mediante la serie de paquetes *gazebo_ros_pkgs*³ que utilizan estructuras de tipo *ROS Messages* para permitir la simulación de robots y entornos que se comunican utilizando este *middleware*, además de ofrecer servicios sobre la simulación y reconfiguración dinámica. Esta ventaja pude ser muy útil para el soporte de simulación que queremos incluir en la solución de “Ejecución Mixta”, para aquellos casos en los que no se dispone de sistemas electro-mecánicos. ROS nos proveerá de los módulos *software* que nos permitan 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 gráfico 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 la tolerancia adicional a errores (quedan contenidos en nodos individuales) y la

³https://wiki.ros.org/gazebo_ros_pkgs

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 nuestro proyecto, 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.

3.1.4. Plataforma end-to-end Docker

Docker⁴ 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,

⁴<https://www.docker.com/>

eliminado 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 la de construir una unidad de contenedor sobre un 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).

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, dadas sus implicaciones de seguridad, principalmente, y en segunda instancia debido a que la usabilidad de una herramienta reside en gran medida en que el proceso de instalación y puesta a punto de la misma no sea tedioso y, si es posible, inexistente, además del hecho de que eliminar la necesidad de instala-

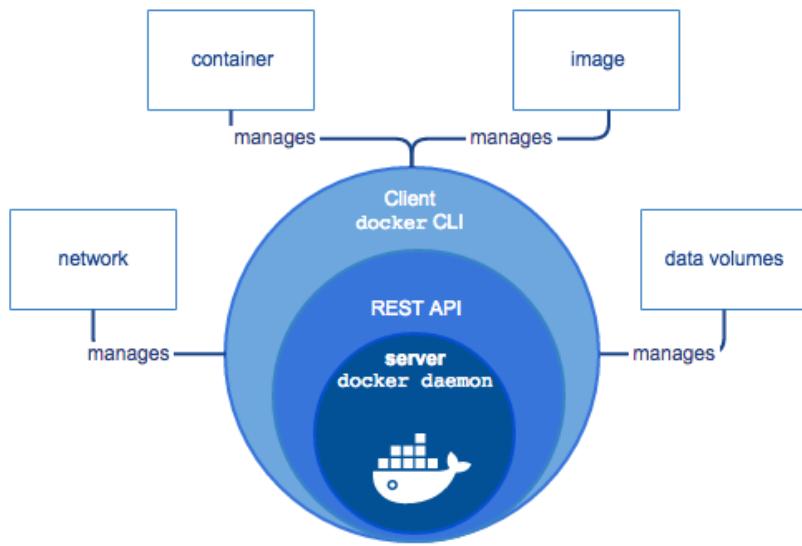


Figura 3.2: Estructura de Docker

ción supone que el rango de potenciales usuarios se amplia hasta prácticamente cualquiera, ya que se pasa a no depender del sistema ni del entorno del usuario. Por tanto, persiguiendo 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.

3.1.5. Simulador Gazebo

Gazebo⁵ 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

⁵<http://gazebosim.org/>

emular diversos escenarios tridimensionales customizables para robots autónomos. Esta diseñado 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 simulados 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 bajo su elección como simulador para este proyecto, además del hecho de que es de código abierto. Sin embargo, existen muchos otros simuladores con estas mismas características que hacen que Gazebo esté lejos del estándar de simulación robótica. A continuación veremos otras características que hemos analizado para elegir este 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 quitan el foco de construir simulaciones realistas para los usuarios humanos e 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 físicas⁶ (Fig. 3.3), 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 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 OpenGL⁷ que se adaptan bien a situaciones reales (Fig. 3.4).

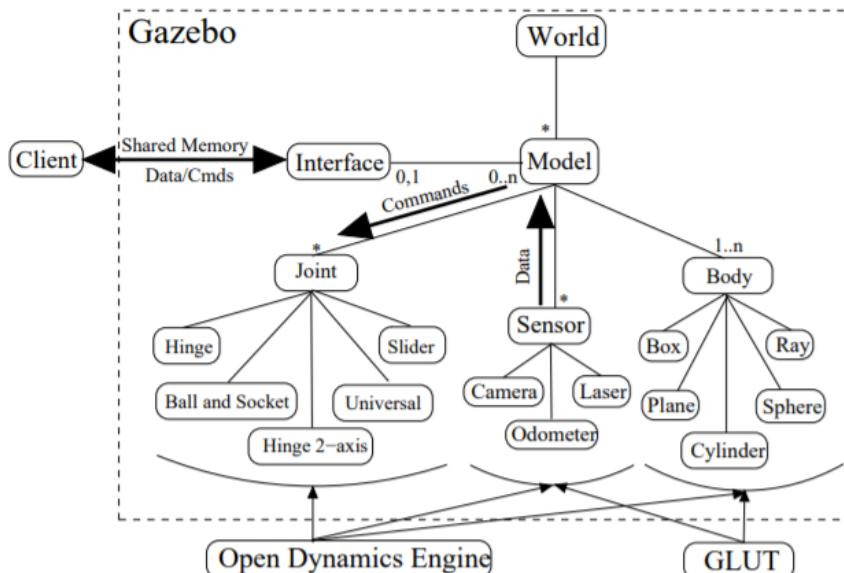


Figura 3.3: Motor de físicas de Gazebo

La configuración de mundos 3D con Gazebo se describe en ficheros con extensión “.world”, que son documentos de texto en formato de marcado XML (*Extensible Markup Language*) de documentos, con etiquetas definidas en el lenguaje

⁶<http://opende.sourceforge.net>

⁷<https://www.opengl.org/>

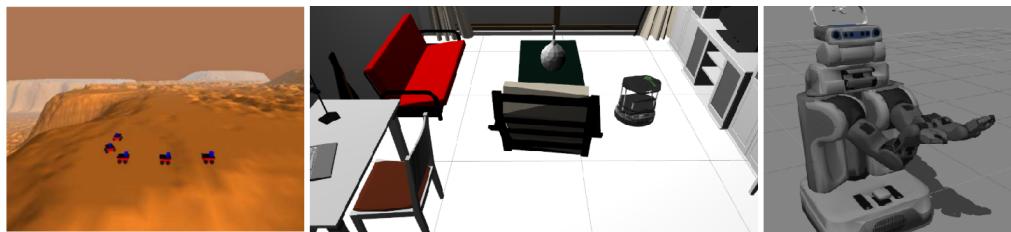


Figura 3.4: Modelos de Simulación en Gazebo

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 incluy 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 de inteligencia e interacción. Es aquí donde existe 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.

3.1.6. Framework Django

Para testar la herramienta que queremos construir, necesitamos una aplicación robótica que se conecte a ella y que ofrezca cierto servicio robótica, en nuestro caso orientado hacia la docencia o investigación. Este servicio debe usar el API que la “Ejecución mixta” proporcionará para montar el servicio sobre los mecanismos que se incluirán en el contenedor Docker del que dispondrá el cliente. Así, la aplicación remota 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. Esta aplicación remota será construida a través de Django.

Django⁸ 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 las molestias del desarrollo Web, por lo que 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 3.5, 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. 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 para la aplicación con la que testar nuestra herramienta, ya que construir una aplicación robótica no es el foco ni el objetivo de este trabajo, sino

⁸<https://www.djangoproject.com/>

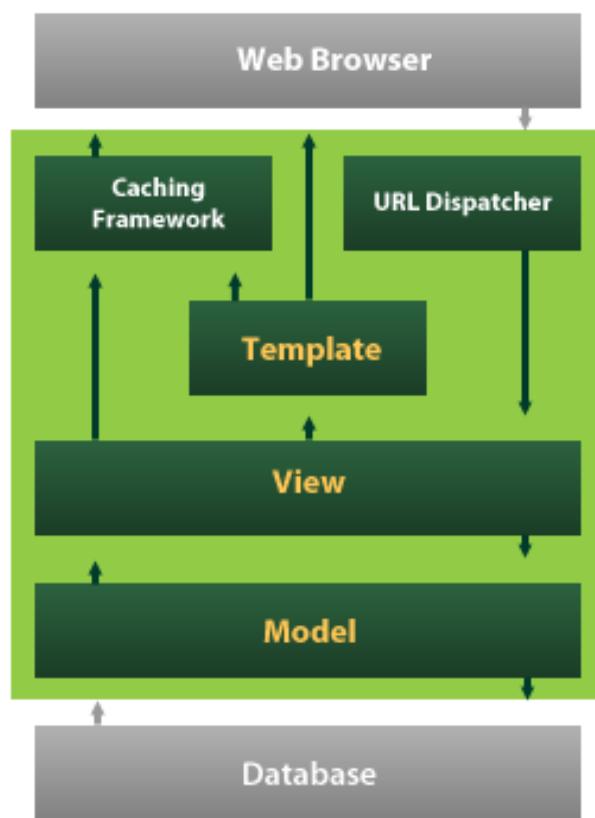


Figura 3.5: Arquitectura de Django

que es una tarea secundaria necesaria para demostrar la “Ejecución mixta”.

3.1.7. Biblioteca OpenCV

En primera instancia, la herramienta 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 herramienta soportase cualquier ámbito de la ciencia robótica, merece una mención especial la tecnología utilizada para el control de los sensores de imagen: las bibliotecas OpenCV⁹.

OpenCV es una librería de código abierto desarrollada 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 imágenes como para utilizarse en la aplicación robótica que se creará para tratar las matrices de información 2D.

⁹<https://opencv.org/>

3.1.8. 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 cliente de la aplicación 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 en-

¹⁰<https://www.python.org/>

torno 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 nos va a permitir realizar actividades complejas en páginas web, entorno en el que deberá operar la “Ejecución Mixta” para aplicaciones robóticas. Este entorno nos permitirá aprovecharnos de 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.

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

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

Implementación de la “Ejecución Mixta”

En este capítulo se explica cómo se implementó la idea planteada paso a paso, 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 describirá también el diseño de la herramienta en cuestión y la arquitectura de cada uno de los módulos que la componen.

4.1. Diseño e Infraestructura de la Aplicación

En esta sección abordaremos el diseño de las distintas capas que componen la herramienta, además de la relación existente entre las tecnologías seleccionadas y descritas anteriormente y las funciones que desempeñan en el mecanismo. Se comentará el diseño y funcionamiento finales de la herramienta, así como la jerarquía establecida.

Se puede ver en el infograma (Fig. 4.1) 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:

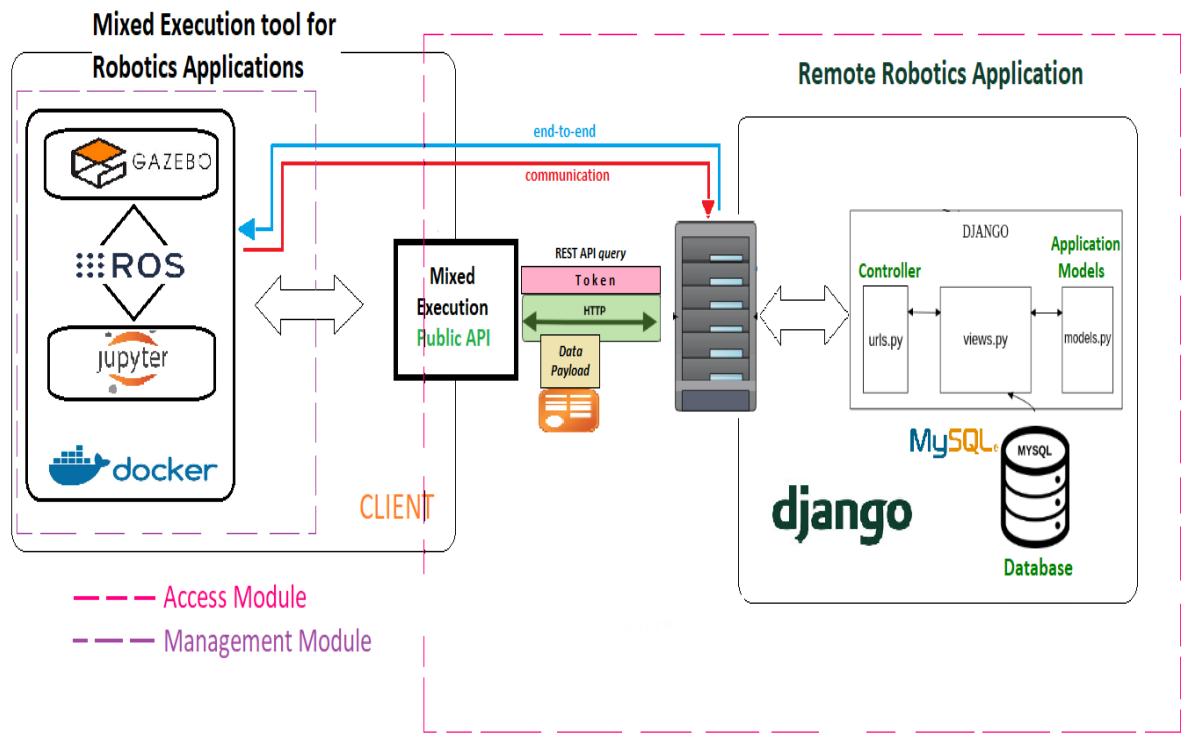


Figura 4.1: Arquitectura de la Ejecución Mixta para Aplicaciones de Robótica.

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

4.1.1. Capa de Aplicación

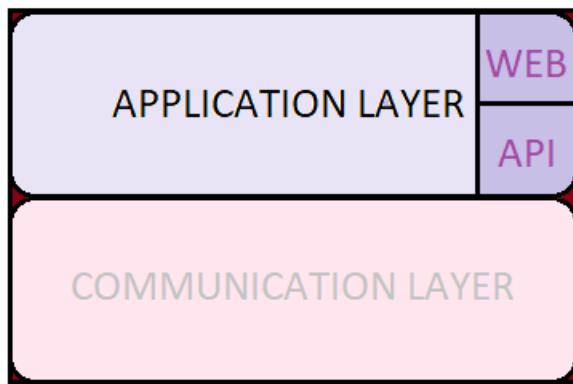


Figura 4.2: Módulo de Acceso: Capa de Aplicación

La capa de aplicación tendrá la tarea específica de emparejar la funcionalidad que ofrece la aplicación remota como servicio con la información que necesita de la ejecución en el lado cliente para dar soporte a dicha funcionalidad. Requerirá de la intervención de dos agentes para cumplir su tarea:

- 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. El cliente de la aplicación hace las funciones de servidor de la herramienta, ya que está pensada para ser colocada en el lado del cliente web.

- Con la conexión iniciada, esta capa debe solicitar la información que necesita 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”.

Así, el mecanismo completo de la capa de aplicación comenzará por establecer la conexión con la herramienta, utilizando uno o varios *endpoints* en función de los canales de comunicación que se quiera establecer para recopilar la información necesaria y generará un comando específico para configurar en la herramienta el tipo de enlace que se requiere y el conjunto de herramientas que se necesita para la ejecución.

Listing 4.1: Comando de configuración de Ejecución Mixta

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

4.1.2. Capa de Comunicación

Encargada del establecimiento de los canales de comunicación entre la aplicación robótica externa y la herramienta de “Ejecución Mixta” una vez iniciada la conexión. Se utilizará principalmente 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 utilizando los métodos clásicos: GET, POST, PUT, DELETE y OPTIONS.

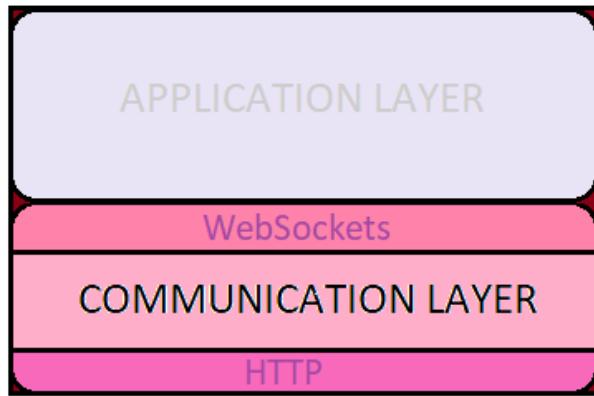


Figura 4.3: Módulo de Acceso: Capa de Comunicación

Adicionalmente se utilizará un medio WebSockets para el intercambio de mensajes ligeros, más relacionados con el interfaz de usuario de la aplicación y aquella información que requiere un refresco de carga ligera pero constante. Se utilizará también como canal auxiliar para la información que necesiten enviar los agentes y procesos en ejecución durante la “Ejecución Mixta”, en caso de necesitarse.

4.1.3. Capa de Seguridad

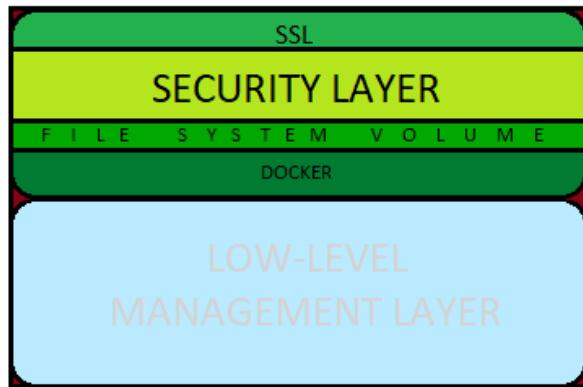


Figura 4.4: Módulo de Gestión: Capa de Seguridad

Debemos distinguir dos niveles de seguridad en el proceso de “Ejecución Mixta”:

- 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. Para esto utilizaremos la plataforma 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. 4.5) 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.

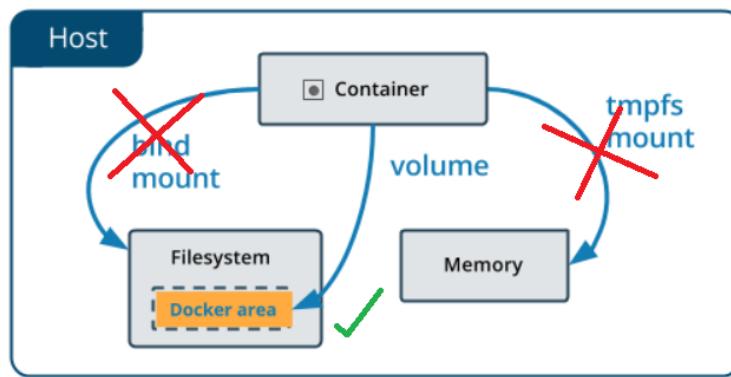


Figura 4.5: Volumen de Docker

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.

4.1.4. Capa de Bajo Nivel

La capa de menor nivel y mayor grado de restricción de la herramienta es la que hemos denominado Capa de Bajo Nivel, cuya puerta es sólo visible para el secuenciador del módulo de gestión.

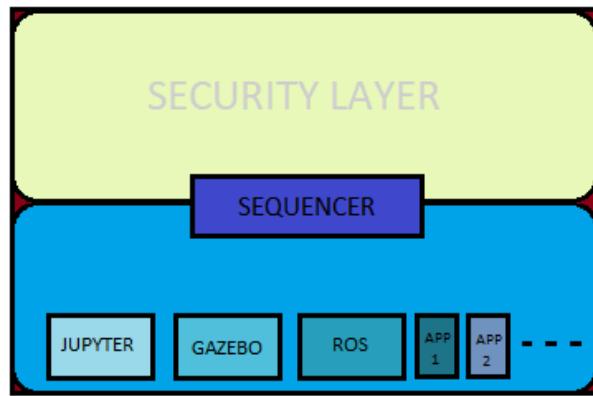


Figura 4.6: Módulo de Gestión: Capa de Gestión del Bajo Nivel

Una vez resuelta la comunicación y superada la seguridad, el motor de la “Ejecución Mixta” debe decidir qué se necesita en todo momento según las peticiones que recibe, y más importante, a quién redirige cada mensaje. Para la Capa de Comunicación, todos los mensajes tienen el mismo destinatario, en función de la conexión HTTPS que se abre para comunicar el lado servidor y el cliente. Este receptor único es un módulo secuenciador, configurado y levantado a través de un *entrypoint* que activará en cada momento la aplicación, herramienta o plataforma auxiliar que se necesite para satisfacer las peticiones de la aplicación robótica, y organizará el canal de bajo nivel para que no se produzcan colisiones durante el proceso. La forma de gestionar la secuenciación se basa en la apertura de nuevos sub-canales de comunicación, ya en un entorno puramente local o *localhost*, y en una redirección de la información obtenida en cada sub-canal hacia el canal HTTPS principal, formateando cada paquete de información según su fuente, su destinatario en la aplicación robótica y su prioridad,

a través de cabeceras y estructuras de datos JSON fácilmente procesables por una aplicación web, y accesibles a través del API público de la Capa de Aplicación (Figs. 4.7 y 4.8).

Listing 4.2: Ejemplo de la parte HTTP de los mensajes

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

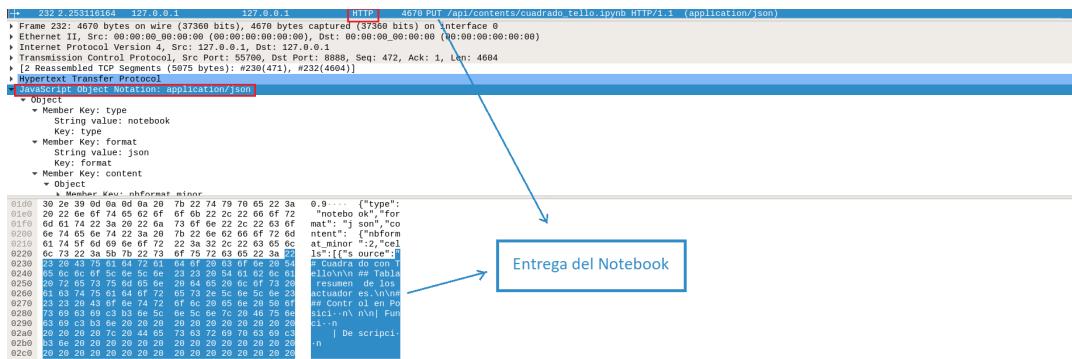


Figura 4.7: Mensaje de Entrega del Cuadernillo o Notebook



Figura 4.8: Mensaje de Respuesta de Petición de Ejecución

Además, los diferentes programas que ejecutan en el interior del contenedor Docker en el servidor de “Ejecución mixta” necesitan un mecanismo de intercomunicación de tipo p2p, sin pasar por todo el mecanismo de transmisión, recepción y procesado descrito anteriormente como parte de la arquitectura, para facilitar la colaboración entre ellos sin sucumbir a problemas de tipo *jitter*, grandes retardos de extremo a extremo o pérdidas de información y largas esperas de reenvíos que caracterizan a muchos enlaces de Internet basados en TCP/IP. Para implementar este mecanismo de comunicación interna aprovecharemos las herramientas de ROS, donde el secuenciador tendrá el rol de *MASTER* 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 de la misma gracias a la “Ejecución Mixta”.

4.2. Proceso de Implementación

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” sin una aplicación robótica a la que conectarla. Por tanto, el primer paso del proceso de construcción fue desarrollar el primer contenido que se quería servir a través de la “Ejecución Mixta”. Para ilustrar el funcionamiento y la orientación de la herramienta que queríamos crear, decidimos organizar el contenido a servir como una aplicación para el aprendizaje de distintos ámbitos de la robótica por medio de ejercicios con contenido académico. Dado el origen de la idea, resulta lógico que la temática del primer ejercicio tuviese que ver con la visión artificial. En la prueba de concepto que supone este primer prototipo no se quiso poner mucho énfasis en la calidad o utilidad del mismo, sino simplemente comenzar por obtener una primera versión completamente funcional para crecer a partir de ella, como se explicó en la sección de metodología. Por tanto, se comenzó por desarrollar a través de Jupyter un *Notebook* que debía actuar como interfaz de edición de la aplicación (IDE), el cual a su vez iba a ser el encargado de formar muchos de los mensajes y protocolos de ejecución del código del usuario en un entorno local. El comienzo por este contexto puramente local supone el asumir que el código servido por la aplicación robótica proviene de la misma máquina, en la misma red. Este primer cuadernillo se usaría posteriormente para desarrollar el resto de módulos y con fines depurativos. 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 *backend* 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, pero aún hay que montar sobre él un mecanismo que permita introducir la información del sensor en el contenedor Docker, desde donde quedará accesible para el *kernel* de Jupyter y, por tanto, utilizable y procesable por el usuario desde el editor de código dado, pero esto se explicará más adelante. 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 agente 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

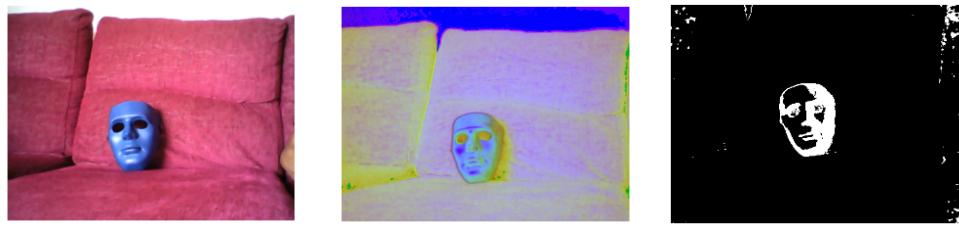
la construcción del primer ejercicio, se implementó una solución de referencia al mismo con una posible vía para obtener el resultado esperado, principalmente haciendo uso de las librerías OpenCV para el tratamiento digital. Aunque los algoritmos planteados para los ejercicios no son objeto de esta tesis, se describe brevemente a continuación el funcionamiento del filtro propuesto:

Ante la imagen de entrada



Figura 4.9: 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 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



(a) Suavizado de la Imagen

(b) Conversión a HSV

(c) Imagen Umbralizada

Figura 4.10: Procesado de Imagen para el Filtro de Color

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. 4.10. 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. 4.11.

En este punto, disponemos de una versión completamente local del prototipo de herramienta que queremos construir. Por tanto, y tras la investigación correspondiente, se procedió a la implementación de un *local runtime*, es decir, de un entorno de ejecución utilizando un *kernel* local. Había que desarrollar un servidor remoto que contuviese la aplicación y el modelo de ejercicio creado y

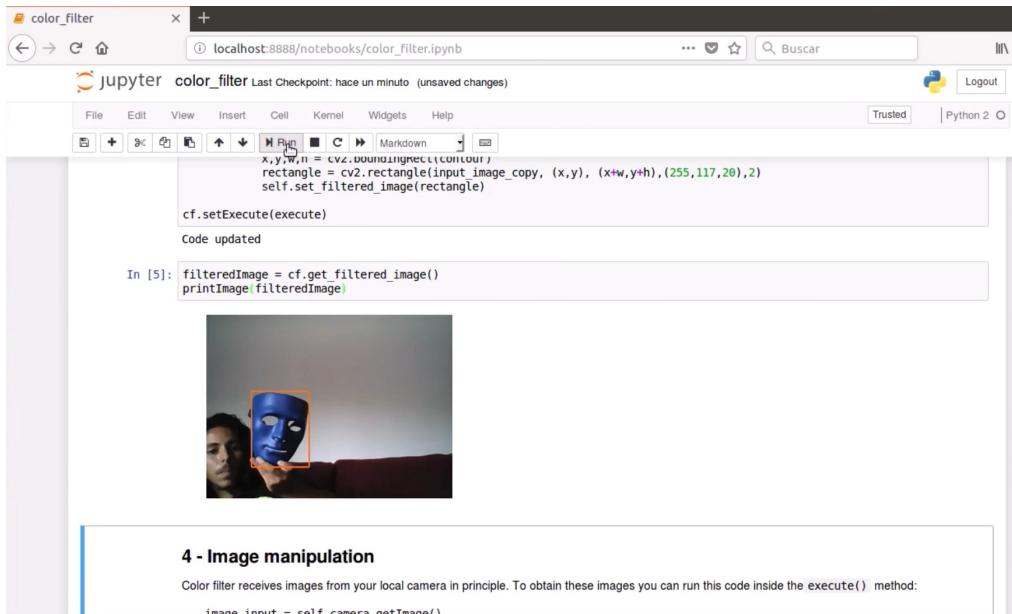


Figura 4.11: UI de Jupyter para el Filtro de Color

que, ante una nueva petición del ejercicio, estableciese 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.

Se construyó un pequeño servidor muy básico a través de Django (Fig. 4.12) con capacidad para atender a un cliente que solicitase un ejercicio por medio de un click en el interfaz web, y se colocó en una máquina externa. Debajo del botón sobre el que se hace click, sucedería todo el mecanismo de enlazado y comunicación para disponer el entorno mixto.

Como se puede ver en la imágenes (Fig. 4.13) de la interfaz que ofrece el servidor web a su cliente, es necesario para el correcto funcionamiento de la herramienta que el receptor principal de los mensajes de “Ejecución Mixta” sea Jupyter, y por tanto también tiene sentido que se use como editor de código. Tras la conexión HTTPS con el servidor web, el cliente especifica su dirección IP y el puerto en que está corriendo el servidor de Jupyter en su máquina, de tal manera que se puede establecer fácilmente un puente que permita que el servidor

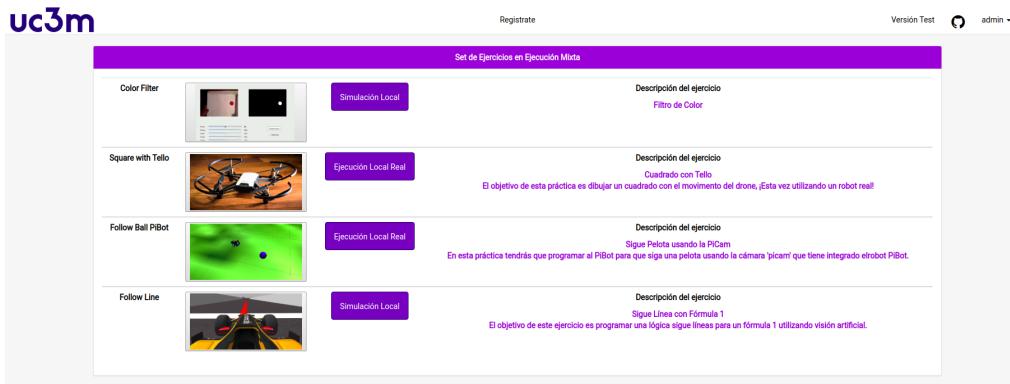


Figura 4.12: UI de la Aplicación en Django

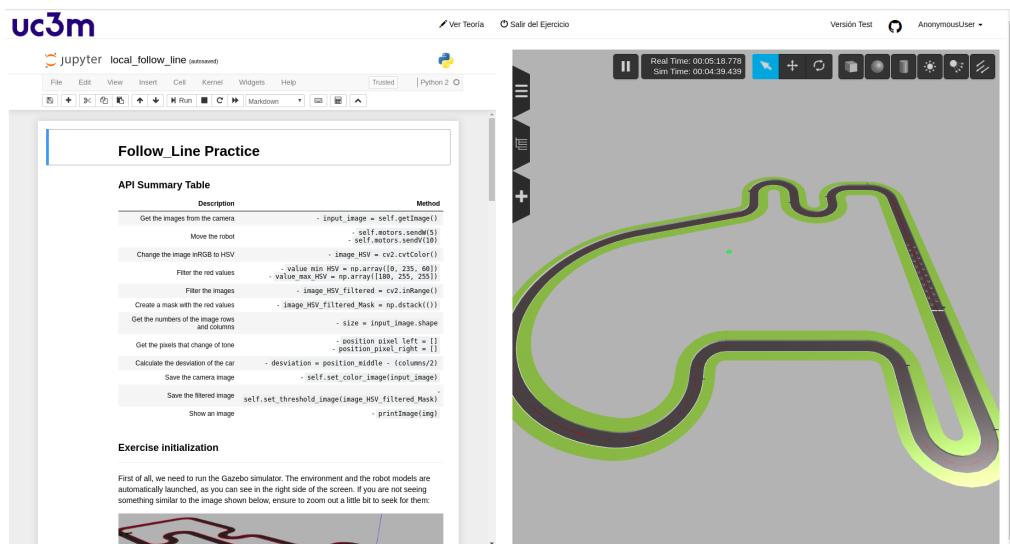


Figura 4.13: UI de Simulación de la Aplicación

acceda a los datos generados por Jupyter en un punto remoto, y los ofrezca al usuario como parte del interfaz, a través del mecanismo de paso de mensajes de actualización periódicos. 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. Luego, como paso previo a la visualización y 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. Aquí comienza el mecanismo de establecimiento y supervisión de la conexión HTTPS de la Capa de Comunicación. La conexión con Jupyter puede hacerse de manera sencilla mediante el envío de un simple mensaje bajo el método OPTIONS, al que el servidor responderá con toda la información que necesitamos para el establecimiento de la conexión 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.

Estudiamos el mecanismo de comunicación de la plataforma, 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. 4.14). Este paso fue vital para comprender el procedimiento que debíamos seguir para iniciar una sesión de Jupyter completa y válida.

¹What is CSRF and how to prevent it?

²<https://zeromq.org/>

³Wireshark Sniffer



Figura 4.14: Captura de Paquetes del Mecanismo de Comunicación de Jupyter

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. 4.15) 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.

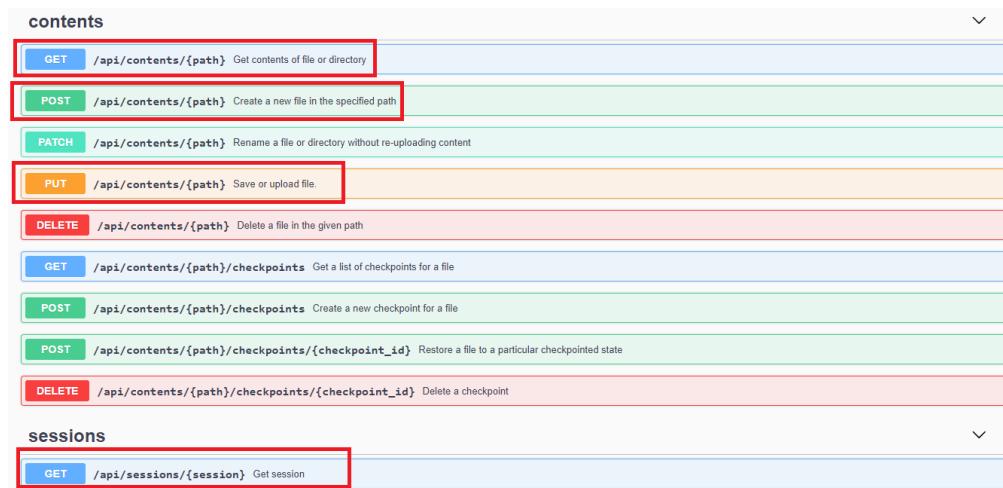


Figura 4.15: REST API de Jupyter

En base al conocimiento obtenido del motor de comunicación de Jupyter, seleccionamos aquellos métodos del REST API que necesitaríamos, marcados

⁴What is a RESTfull Service?

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 4.3: 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. 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.

El proceso de comunicación con Jupyter comenzaría por el establecimiento de la conexión, donde se crea una sesión de ejecución, continuaría con el envío 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 cliente 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.

En este punto del desarrollo, hemos superado el principal obstáculo de la “Ejecución Mixta” e implementado un mecanismo capaz de ejecutar código entre redes, con actualización de los resultados en tiempo real en ambos lados servidor y cliente web. Se completa así el primer ciclo de trabajo, no sin antes realizar los tests convenientes de funcionamiento y desempeño y el análisis de los resultados. La herramienta ya es capaz de conectar un código remoto (aplicación robótica) con un “robot” local (cámara). Se propusieron los siguientes pasos y analizaron los riesgos, donde salió a la luz la necesidad de utilizar contenedores Docker (lo cual cambiará sustancialmente el mecanismo de comunicación) y la posibilidad de enriquecer la aplicación robótica con otros procesos dentro de él, para dar soporte de simulación y ampliar el *hardware* utilizable.

Hubo que estudiar 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 el 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 comunicación para que utilizase el navegador del cliente web como intermediario entre el usuario y la aplicación robótica, 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 Jupyter, y éste actúa como *proxy* reenviando

cada mensaje donde corresponde a través de una secuenciación programada en 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 el módulo secuenciador de “Ejecución Mixta”, 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. 4.4) y entregando las órdenes de ejecución a quien corresponda (Listing. 4.5). Ahora se tiene un prototipo completo y seguro, capaz de funcionar sobre Docker y a través de Internet, con ambos extremos situados en cualquier punto.

```
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 4.4: 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 4.5: Reenvío de Mensajes

El API de inicio de la “Ejecución Mixta” fue enriquecido con la aparición de los parámetros característicos de Docker que permiten crear un puente seguro entre un elemento virtualizado y su homólogo en la máquina anfitrión. Un ejemplo de esto es el mecanismo que ya anticipábamos para el 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`).

Llegados a este punto, se trata de enriquecer la “Ejecución Mixta” para que además de ejecutar lógica pueda simular y usar aplicaciones auxiliares como *plugins* o *drivers* robóticos. Utilizamos ROS dada su fácil integración con Gazebo y el lenguaje Python, que nos permite construir una aplicación completa y autocontenido asociada al código del ejercicio que, lanzada en el interior del contenedor de “Ejecución Mixta”, creará el mecanismo de inter-comunicación a través de mensajes de ROS que orquestará el funcionamiento de un código que se ejecuta en una aplicación para que sus resultados se reflejen en la simulación y en la aplicación web al mismo tiempo. Lo mismo sucederá con robots reales sustituyendo el simulador por un controlador del dispositivo *hardware*, empleando el mismo mecanismo interno de mensajes (Listings. 4.6 y 4.7). Cada mensaje que deba viajar hacia la aplicación con información relevante será recogido a través de un canal WebSockets por el secuenciador del módulo de gestión situado en el navegador, envuelto con una capa HTTPS y enviado al servidor remoto para

que este lo procese (Fig. 4.1).

```
1 ~$ rostopic list
2 /F1ROS/cameraL/camera_info
3 /F1ROS/cameraL/image_raw
4 /F1ROS/cameraL/parameter_descriptions
5 /F1ROS/cameraL/parameter_updates
6 /F1ROS/cmd_vel
7 /F1ROS/odom
8 /clock
9 /gazebo/link_states
10 /gazebo/model_states
11 /gazebo/parameter_descriptions
12 /gazebo/parameter_updates
13 /gazebo/set_link_state
14 /gazebo/set_model_state
15 /rosout
16 /rosout_agg
17 /tf
```

Listing 4.6: Topics de ROS asociados a los Canales de Comunicación Internos

```
1 # follow_line.yml
2 Camera:
3   Topic: "/F1ROS/cameraL/image_raw"
4   Name: follow_line_camera
5
6
7 Motors:
8   Topic: "/F1ROS/cmd_vel"
9   Name: follow_line_motors
10  MaxV: 40
11  MaxW: 2
```

```
12  
13 Websockets:  
14     Host: 0.0.0.0  
15     Port: 9002  
16     SSL: False  
17     Cert: '/etc/certs/fullchain1.pem'  
18     Key: '/etc/certs/privkey1.pem'
```

Listing 4.7: Configuración de Canales del Secuenciador en formato YAML

Una vez dispuesto el contenedor Docker, completamos otro ciclo y verificamos que la “Ejecución Mixta” ya no supone ninguna carga de cómputo para el lado servidor de la aplicación web, y también que podemos eliminar 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, por parte del servidor remoto a través del secuenciador o *proxy* web. 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. La aplicación robótica le indicará al secuenciador qué sub-procesos necesita para el funcionamiento de un ejercicio robótico concreto.

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. 4.16), 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. 4.9), 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. 4.8).

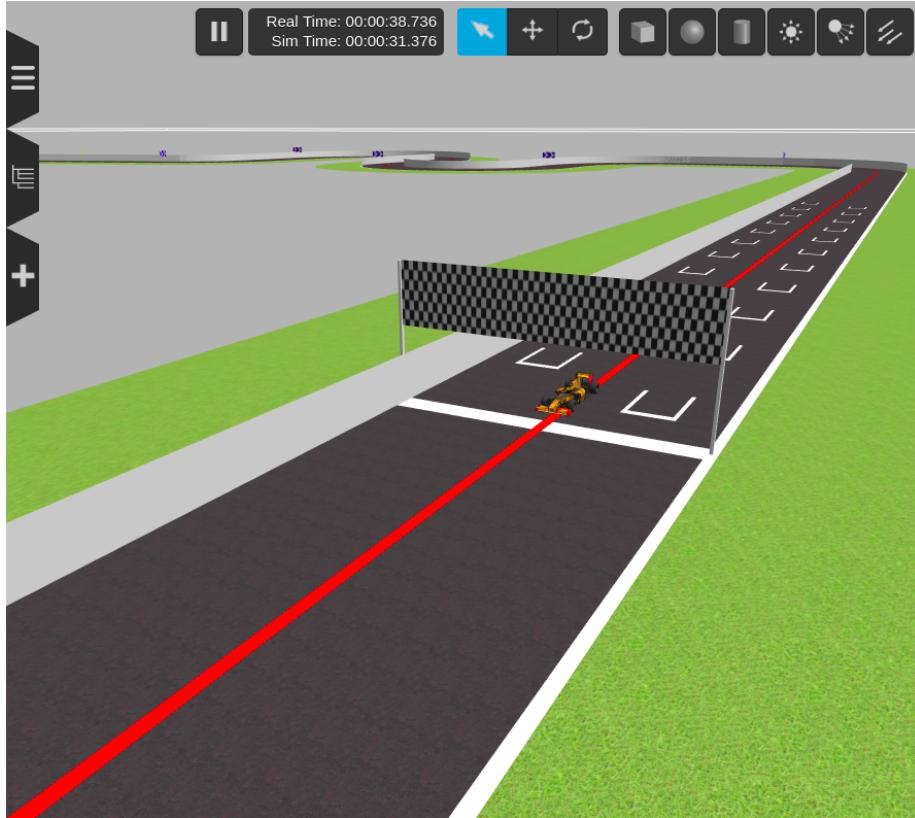


Figura 4.16: Escenario de Simulación

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <launch>
3   <!-- We resume the logic in empty_world.launch, changing only the
      name of the world to be launched -->
4   <include file="$(find gazebo_ros)/launch/empty_world.launch">
5     <arg name="world_name" value="f1_1_simplecircuit.world"/> <!-- Note:
      the world_name is with respect to GAZEBO_RESOURCE_PATH
      environmental variable -->
6     <arg name="paused" value="false"/>

```

```
7 <arg name="use_sim_time" value="true"/>
8 <arg name="gui" value="true"/>
9 <arg name="headless" value="false"/>
10 <arg name="debug" value="false"/>
11 <arg name="verbose" default="false"/>
12 </include>
13 </launch>
```

Listing 4.8: 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 4.9: 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. 4.10), 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         rospy.init_node("FollowLineF1")
27
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)
```

51 # [. . .]

Listing 4.10: Creación de la Red Interna de Comunicación

Con todo lo anterior ya resuelto, decidimos implementar también el soporte de ejecución compartida haciendo uso de robots reales. 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. Así, se implementó un driver casero para el robot Tello (Listing. 4.11), un cuadricóptero de DJI e Intel cuyo precio de mercado está al alcance del consumidor medio. Una vez hecho 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. 4.12).

```

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         # -----
```

```

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 4.11: Snippet del Driver de Tello

```

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

```

Listing 4.12: Uso del Driver

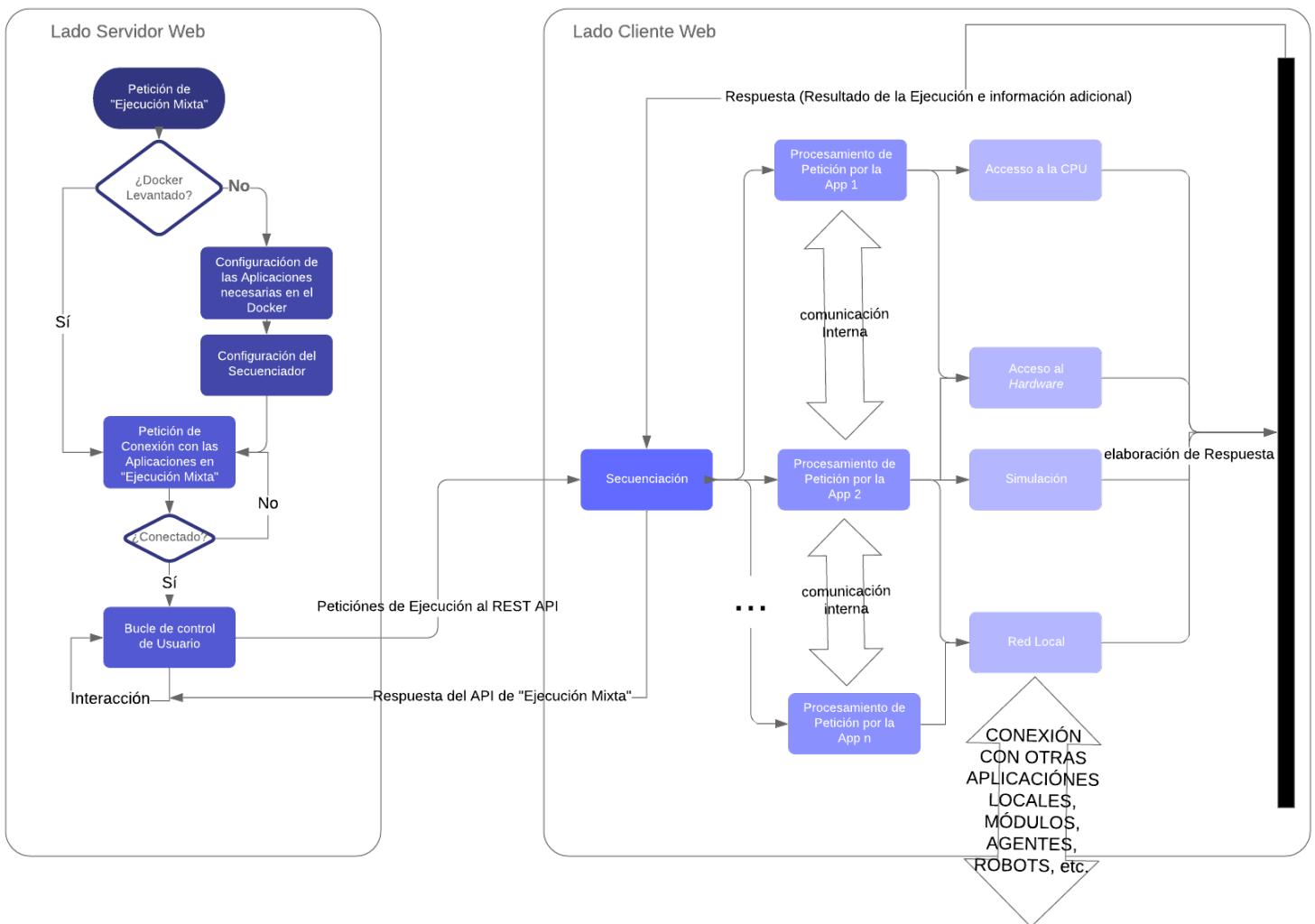
De manera análoga a lo ya construido, se diseñaron algunos ejercicios más para probar la herramienta bajo distintos pretextos.

4.3. Funcionamiento

4.3.1. Fluograma

1

MECANISMO SUBYACENTE EN LA
"EJECUCIÓN mIXTA"



4.3.2. Mecanismo subyacente de la Herramienta

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. 4.17), 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 des-

de 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 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. 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:

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.

De la tabla de parámetros resultantes se deduce que el funcionamiento es igual para todos los navegadores, 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. 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 *betatesters* 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 guion 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 MacOS 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.

6.5. Videos?

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 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 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/>.