ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

MÁSTER EN INGENIERÍA DE LA TELECOMUNICACIÓN

TRABAJO FIN DE MÁSTER

Nuevas Prácticas Docentes en el Entorno Robotics-Academy

Autor: Pablo Moreno Vera

Tutor: José María Cañas Plaza

Curso académico 2019/2020

Agradecimientos

Resumen

Gracias al auge de la robótica en la actualidad, cada vez encontramos más productos desarrollados mediante la robótica a nuestro alrededor. Es por ello que se necesitan más especialistas en este campo. Debido a esto, surgen empresas dedicadas a la formación y aprendizaje de personas de todas las edades. Una de estas empresas es *Kibotics*. Esta empresa pone al alcance de los más pequeños un entorno seguro de aprendizaje, tanto con robots simulados como reales. Gracias a Kibotics, los niños pueden aprender a manejar robots desde el navegador y sin ningún tipo de requerimiento previo. El entorno facilitado a los niños ofrece dos lenguajes de programación, *Python* y *Scratch*.

La infraestructura de Kibotics está formada por un servidor *Apache* que soporta *Django*, todo ello en un *Docker* almacenado en *Amazon Web Services*. La parte del cliente está desarrollada en *JavaScript*, *HTML-5* y *CSS-3*.

En este *Trabajo de Fin de Máster*, se ha querido ir un poco más allá en la plataforma y, aprovechando la tecnología multirobot que ofrece, se han desarrollado prácticas multirobot y multiusuario. Gracias a la tecnología *WebRTC* se ha desarrollado un tipo de práctica donde dos usuarios pueden competir de manera simultánea, cada uno con su propio código. Además se ha incluido un chat en el que los contrincantes pueden hablar sobre *WebRTC*.

En otro aspecto abordado en este *TFM*, junto con los ejercicios multirobot y multiusuario (llamados *Juegos Compartidos*), se han desarrollado torneos en los que se pueden diferenciar dos clases:

1. **Torneos de ranking**. En estos torneos el usuario desarrollado su código en un tipo específico de ejercicio y guarda su código para competir de manera individual y por tiempos contra el resto.

2. **Torneos clasificatorios**. En estos torneos los usuarios compiten directamente contra otros usuarios en los Juegos Compartidos y el que pierda será eliminado, el ganador pasará a la siguiente ronda.

Con los torneos, se ha utilizado tecnología *Elastic-Search* para enviar los resultados al servidor y mostrarlos en la página correspondiente del torneo.

Índice general

1. Introducción **1**

1.1. Robótica 1

1.2. Software Robótico 4

1.2.1. Middlewares robóticos 4

1.2.2. Simuladores robóticos 5

1.3. Robótica educativa 6

1.4. Tecnologías Web 7

1.4.1. HTTP 7

1.4.2. Tecnologías en el cliente 9

1.4.3. Tecnologías en el servidor 9

2. Objetivos **11**

2.1. Objetivos 11

2.2. Requisitos 11

2.3. Metodología 12

2.4. Plan de trabajo 14

3. Infraestructura **15**

3.1. JavaScript 15

3.2. A-Frame 16

3.2.1. HTML y primitivas 16

3.2.2. Entidad, Componente y Sistema (ECS) 17

3.3. Gestores de paquetes 17

3.3.1. Node Package Manager (NPM) 17

3.3.2. Webpack 18
3.4. Simulador WebSim 18
3.4.1. Drivers de sensores 18
3.4.2. Drivers de actuadores 19
3.5. Django 19
3.6. Elastic-Search 20
3.6.1. Índice, Logstash y Kibana 21
3.7. WebRTC 22
3.8. Retransmisión en directo 23

4. Juegos Compartidos **24**
4.1. Refactorización de kibotics 24
4.2. Ejercicios 24
4.3. Plantillas 24
4.3.1. Infraestructura del servidor 24
4.4. WebRTC 24
4.4.1. Cliente 24
4.4.2. Servidor 24
4.5. Evaluadores 24

5. Torneos **25**
5.1. Plantillas 25
5.1.1. Infraestructura del servidor 25
5.2. Elastic-Search 25
5.3. Maestro de ceremonias 25
5.4. Automatización de los torneos 25
5.4.1. Evaluadores 25
5.5. Retransmisión en vivo 25

6. Conclusiones **26**

6.1. Conclusiones 26

6.2. Trabajos futuros 27

Bibliografía **28**

Capítulo 1

Introducción

El TFM descrito a continuación se encuadra en el entorno educativo *JdeRobot*, en su plataforma *Kibotics*, para la enseñanza de la programación de robots. La intención principal de su desarrollo ha sido el de mejorar la plataforma incluyendo ejercicios que fomenten la interacción social de sus usuarios.

En este capítulo se introducirá el contexto en el que se sitúa este proyecto y la motivación que ha llevado a su desarrollo. Para ello, es preciso comenzar con una explicación, a grandes rasgos, sobre qué es la robótica y sus aplicaciones para la sociedad.

Un robot está formado por dos componentes principales, el hardware, formado por la infraestructura física del propio robot y el software que se encarga de dotar de la inteligencia la robot. Esta última es la parte más importante de los robots y podemos encontrar diferentes elementos como bibliotecas de código, middlewares robóticos o, incluso, simuladores. Dentro del heterogéneo campo que es la robótica, este TFM se enmarca en la robótica educativa, destinada al aprendizaje en este campo.

1.1. Robótica

Durante toda la evolución del la humanidad el principal factor que la distingue del resto de seres vivos es su gran capacidad de pensamiento. Una de las principales muestras de inteligencia es el desarrollo de herramientas que facilitan el trabajo. La robótica nace en el momento en que el ser humano ha utilizado la informática y las máquinas para reducir el esfuerzo empleado. Así, la robótica es la rama de la tecnología basada en la utilización

de la informática para el diseño y desarrollo de sistemas automáticos que faciliten la vida al ser humano e, incluso, llegando a sustituirle en algunos de ellos. La robótica incluye campos tan distintos como la física, las matemáticas, la electrónica, la mecánica, la inteligencia artificial, la ingeniería de control, etc. Gracias a todas estas disciplinas, englobadas correctamente, se pueden diseñar máquinas que ejecuten comportamientos autónomos para el propósito que han sido desarrollados. Estas máquinas se denominan robots.

Desde 1950, estos sistemas autónomos han desarrollado un crecimiento exponencial en cuanto a su complejidad, versatilidad, autonomía y, sobre todo, su inclusión en una gran variedad de ámbitos. Esto es debido al gran desarrollo en cuanto a potencia y complejidad computacional que han sufrido los ordenadores en los últimos años, que han permitido el desarrollo de comportamiento más complejos en los robots. Tal es el grado de desarrollo alcanzado que la mayoría de sistemas operados por el ser humano comienzan a incorporar un sistema de control específico programable que permiten el desarrollo de tareas repetitivas con un gran riesgo para las persona, englobando tareas básicas pero de difícil realización (Figura 1.1).

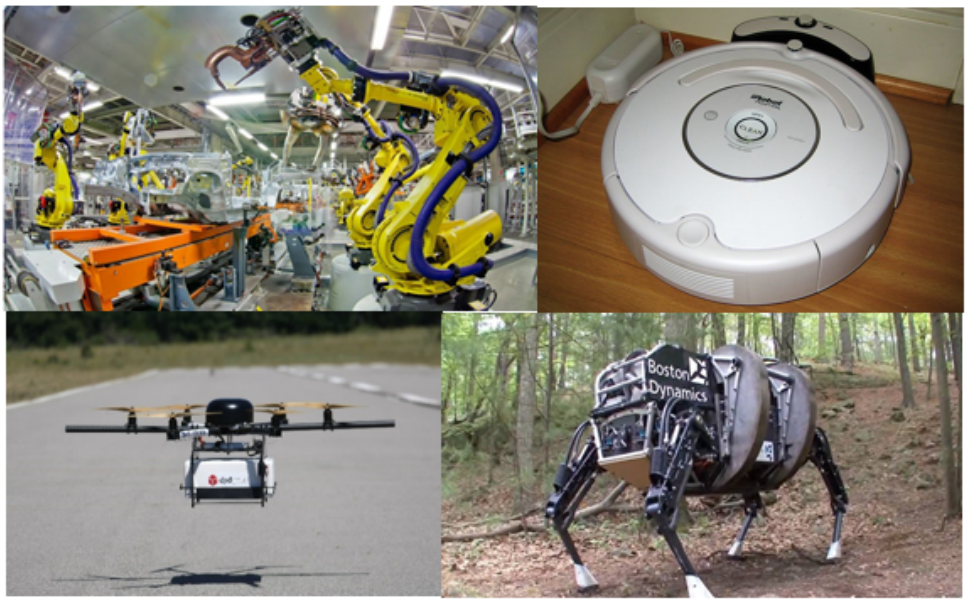

Figura 1.1: Robots modernos

El principal ejemplo del desarrollo de la robótica es su inclusión en la industria, pero tal es su auge en la actualidad que están presente en multitud de campos como los coches

autónomos, con el autopilot de Tesla, la logística, con los drones de paquetería o los robots de los almacenes de Amazon, la agricultura con drones que esparcen pesticidas o, incluso en entornos no especializados como el doméstico con los robots aspiradora de Roomba, y esenciales como la medicina con el robot DaVinci. También se han introducido robots en el ámbito militar para la desactivación de bombas, extinción de incendios o de rescate. Otro campo con gran desarrollo en los últimos años son los robots con IA que son capaces de interactuar con humanos e incluso, experimentar la locomoción bípeda, como el robot atlas.

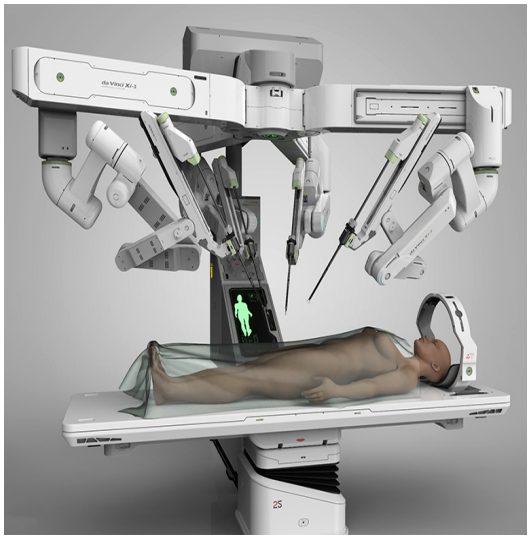

Figura 1.2: Robot DaVinci

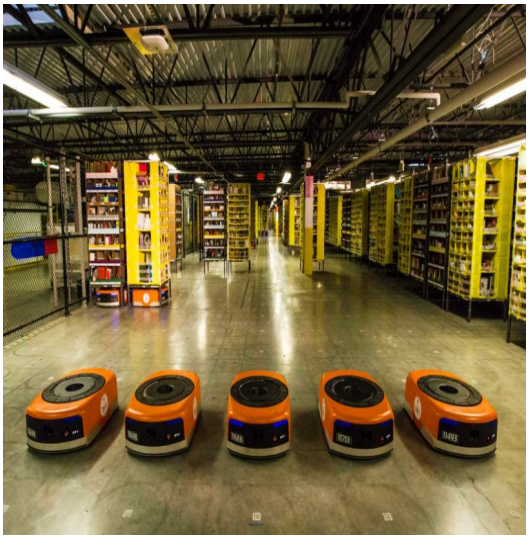

Figura 1.3: Flota de robot de Amazon

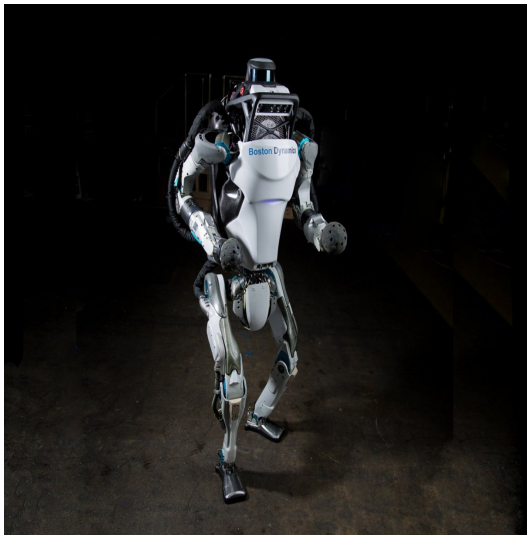

Figura 1.4: Robot Atlas

Figura 1.5: Robot Colossus

1.2. Software Robótico

Para que los robots puedan ser controlados de una manera efectiva, el comportamiento del software que los controla debe ser robusto. Para ello, el software robótico se divide en tres capas: drivers, middleware y aplicaciones, cuya arquitectura será distinta en función de su finalidad.

Gracias al gran desarrollo de la robótica, los robots actuales ya no precisan del control del ser humano para su funcionamientos. En la actualidad tienen comportamientos autónomos que les permiten realizar las tares sin la mediación de terceros. Esto ha sido posible gracias al minucioso desarrollo del software que compone sus sistemas, algo parecido a una inteligencia autónoma. Aspectos importantes de este software robótico son los circuitos de realimentación, control, búsqueda de caminos, localización o procesado de imágenes.

De la mano del auge de la robótica, han aparecido multitud de plataformas de desarrollo de software robótico, también llamados middlewares robóticos. Otra herramienta muy importante en la robótica son los simuladores dado que permiten realizar pruebas y depurar fallos para programar una versión funcional del robot antes de fabricarlo.

1.2.1. Middlewares robóticos

Los middlewares robóticos pueden definirse como entornos o frameworks para el desarrollo de software para robots. Se trata de software que conecta aplicaciones o componentes software para soporta aplicaciones complejas y distribuidas. Para poder controlar los sensores y actuadores de los robots, estos middlewares incluyen drivers, arquitecturas software, bloques de funcionalidad, APIs, simuladores, visualizadores, etc. Es por todo esto que a los middlewares se les conoce como "pegamento para software". Una de las tareas más importantes del middleware es conectar el hardware (real o simulado) con las aplicaciones(software). El middleware más extendido es ROS.

Robotics Operating System[1] es un sistema operativo para el desarrollo robótico libre que proporciona toda la funcionalidad necesaria de un sistema operativo en un clúster heterogéneo como el control de dispositivos bajo nivel, mecanismos de intercambio de

[1] `http://www.ros.org/`

mensajes entre procesos y la abstracción hardware, necesarios para el desarrollo robótico. Aunque el framework ROS fue diseñado para sistemas UNIX, ha sido adaptado para ser soportado en otros sistemas operativos como Fedora, Debian, Windows, MacOS-X, Arch, Slackware, Gento u OpenSUSE, llegando a permitir aplicaciones multiplataforma. Gracias a todo esto, ROS es el middleware más extendido en el mundo.

Existen otros frameworks interesantes como Orocos[2] que permiten el control avanzado de máquinas y robots en C++, Orca[3] que está orientado a componentes, por lo que permite el dearrollo de aplicaciones más complejas y de software libre y Urbi[4] que es un middleware multiplataforma de código abierto en C++ que permite desarrollar aplicaciones en sistemas completos y complejos y trabaja de forma conjunta con ROS.

1.2.2. Simuladores robóticos

Debido al gran coste que supone la fabricación del hardware del robot, es preciso depurar el software al máximo antes de fabricar el hardware, de esta manera se reducen los costes de desarrollo y fabricación del robot. Para ello existen herramientas especializadas en reproducir el comportamiento del robot en entornos controlados de manera virtual. Estas herramientas son los simuladores.Los más utilizados son:

Gazebo[5]. Es el simulador más extendido de código abierto. Tiene gran importancia su motor de renderizado, sus motores de físicas y su soporte para plugins de sensores y actuadores, además de su amplio catálogo de robots. Otro hecho importante es su soporte para ROS, por lo que permite probar el software real desarrollado en un robot simulado.

Stage[6]. Es un simulador en dos dimensiones, integrable con ROS, que permite simular numerosos robots simultáneamente.

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

[2] `http://www.orocos.org/`
[3] `http://orca-robotics.sourceforge.net//`
[4] `https://github.com/urbiforge/urbi`
[5] `http://gazebosim.org/`
[6] `http://wiki.ros.org/stage`
[7] `https://www.cyberbotics.com/`

A-Frame[8]. Se trata de un simulador web orientado a la VR(Realidad Virtual). Utiliza el formato HTML para general figuras en un I-Frame propio.

1.3. Robótica educativa

Como se ha comentado en este TFM, la robótica educativa es un campo en auge debido al gran desarrollo de la robótica y la necesidad de especialistas y aprendizaje por parte de todo el mundo. En 2015, la *Comunidad de Madrid* introdujo la asignatura "Tecnología, Programación y Robótica" en el plan docente de Enseñanza Secundaria[1]. En el año 2020-2021 se prevé implantar la asignatura "Programación y Robótica" en el plan docente de Enseñanza Primaria[2].

Tanto ha sido la necesidad de comprender el campo de la robótica que se ha desarrollado una nueva forma de enseñanza, la educación STEM (Science, Technology, Engineering and Mathematics). Esta nueva forma de enseñanza promueve el pensamiento científico y la adquisición de conocimientos tecnológicos aplicables a situaciones reales permitiendo el desarrollo de competencias en la resolución de problemas.

Para poder impartir este tipo de asignaturas es necesario la infraestructura adecuada. Las plataformas como la de LEGO o Arduino, permiten una enseñanza y aprendizaje sencillos sobre la complejidad de la robótica resultando muy gratificante para el alumno por lo vistoso de los resultados y obteniendo una aplicación real inmediata.

Es importante acercar este conocimiento tan complejo de manera adecuada a edades cada vez más tempranas para permitir un mayor desarrollo en el conocimiento de esta tecnología. Es por ello que cada vez hay más plataformas que desarrollan software orientado a niños.

Algunos ejemplos de lenguajes sencillos son:

- **Scratch**[3]. Es un proyecto utilizado en docencia y liderado por el MIT[9] para programar animaciones, interacciones y juegos de manera sencilla por su interfaz visual. Toda la funcionalidad de un lenguaje de programación está embebida en bloques gráficas que agrupan funcionalidades específicas.

[8] `https://aframe.io/`

[9] `http://www.mit.edu/`

- **LEGO**[4]. Se trata de una plataforma que dispone de multitud de robots programables con su sistema gráfico.
- **Kodu**[5]. Es un sistema de programación visual para el desarrollo de videojuegos desarrollado por Microsoft[10]
- **Snap!**[6]. Se trata de un plataforma basada en Scratch pero con funcionalidad destinada a edades más avanzadas que permiten el desarrollo de funcionalidad más compleja.

Gracias a este lenguaje, se ha desarrollado la plataforma Kibotics, que ofrece distintos entornos simulados y controlados con robots a los alumnos. Con esto, los alumnos pueden programar la inteligencia de robots autónomos para que realicen distintas pruebas sin ningún riesgo. De esta manera es posible aprender robótica en casi cualquier edad de manera sencilla y con la única necesidad de acceso a internet.

1.4. Tecnologías Web

Las tecnologías web han evolucionado en los últimos años, sin embargo han mantenido el modelo "cliente-servidor". En la actualidad, lo más común son *aplicaciones-web* que son ejecutadas en un navegador y los datos se almacenan y procesan en el servidor. Algunos tipos de aplicaciones son mensajería (Gmail), tiendas(Amazon, AliExpress) o distribución de contenidos (Netflix, HBO) o redes sociales (Facebook, Instagram).

1.4.1. HTTP

El protocolo de transmisión de recursos hipermedia entre máquinas HTTP (HiperText Transfer Protocol) trabaja a nivel de aplicación y tiene un esquema de petición-respuesta entre el cliente y el servidor (figura 1.6).

[10] `https://www.microsoft.com/es-es`

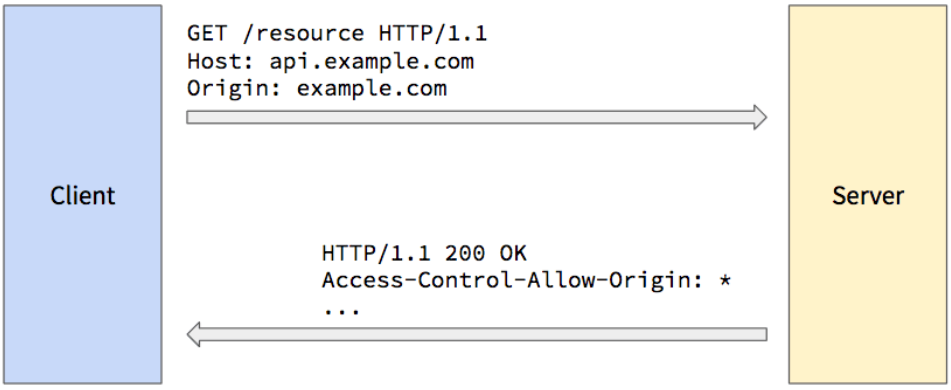

Figura 1.6: Protocolo HTTP

En este tipo de comunicación es el cliente el que abre la conexión pidiendo un recurso y el servidor responde con el recurso o un error. Una vez respondido el recurso, el servidor cierra la conexión, por lo que HTTP no guarda estado, es decir, el servidor responde cada petición de manera independiente y no almacena información sobre las peticiones. Los tipos de peticiones están definidos en el protocolo y son los siguientes:

- **GET**: Solicita un recurso al servidor con una URL específica.
- **POST**: Envía datos al servidor, suele provocar un cambio de estado.
- **HEAD**: Método parcido al GET pero sólo pide las cabeceras, no el recurso completo.
- **PUT**: Actualiza el recurso en el servidor.
- **DELETE**: Elimina el recurso en el servidor.

Aunque estos son los métodos principales, los métodos más básicos soportado y que pueden englobar al resto son los métodos GET y POST (llamado API-REST). Además, el protocolo es flexible y admite la incorporación de nueva funcionalidad.

La versión actual de HTTP es HTTP/2.4.38 pero la primera versión es de 1991, cuando aparece como estándar para las páginas de HTML. En el último año se ha desarrollado HTTP-3 y cuenta con soporte en Google y Firefox.

1.4.2. Tecnologías en el cliente

Se trata de la parte encargada de realizar las peticiones e iniciar la comunicación con el servidor. Las tecnologías que hacen posible esta comunicación son:

- **Navegadores**. Se encargar de procesar y mostrar la información recibida del servidor.
- **HTML (HiperText Markup Language)**. Estándar más utilizado en el desarrollo de páginas web. Tiene un modelo DOM (Document Object Model), en el que se define la manera en que se comunican los objetos con la interfaz de representación de documentos. El DOM permite el acceso dinámico a los datos del HTML con lenguajes como JavaScript y define cómo se comunican objetos y elementos con el navegador. En la actualidad los navegadores utilizan la versión de HTML, HTML-5 que proporciona mejoras como elementos *canvas, websockets, WebRTC*, etc. Este lenguaje de marcado proporciona al navegador la estructura de los datos a representar únicamente.
- **CSS (Cascading Style Sheets)**. Se trata de un lenguaje de diseño gráfico para definir y crear la representación de los datos recibidos en el navegador. La versión actual de CSS es CSS-3.
- **JavaScript**. Lenguaje de programación orientado a eventos destinado a programa la lógica, dinamismo e interacción con el usuario.

1.4.3. Tecnologías en el servidor

Son las encargadas del diseño del servidor permitiendo el acceso a la base de datos, conexiones de red o recursos compartidos. Las más utlizadas son:

- **Node.js**. Se trata de un entorno de ejecución de JavaScript en el servidor. Este entorno es capaz de compilar y ejecutar a gran velocidad, aspecto muy importante en tecnologías web. Esta velocidad se produce porque compila en código máquina nativo en lugar de interpretarlo o ejecutarlo.

- **Django**. Se trata de un entorno web de alto nivel programado en Python. Proporciona seguridad, escalabilidad y rapidez. Además incluye una interfaz de acceso a base de datos en Python y admite plugins para aumentar su funcionalidad.
- **Spring**. Se trata de una herramienta en Java para la simplificación del desarrollo de aplicaciones web puesto que facilita la configuración de la aplicación y el despliegue del servidor. Se basa en servicios REST.

Capítulo 2

Objetivos

Una vez introducido todo el contexto sobre el que se ha desarrollado este trabajo, es momento de profundizar en los objetivos que se han intentado alcanzar, los requisitos para las soluciones planteadas y la metodología para conseguirlos.

2.1. Objetivos

La principal meta planteada en es TFM ha sido la mejora de la plataforma *Kibotics* con los *Juegos-Compartidos* y los *Torneos*. Para alcanzar éste objetivo principal, se ha subdividido en dos objetivos secundarios.

- La introducción del elemento social en la plataforma con el chat y los Juegos-Compartidos.
- El desarrollo del aprendizaje más avanzado gracias al perfeccionamiento del código con los torneos y su retransmisión.

2.2. Requisitos

Además de los objetivos descritos en la sección anterior, lo requisitos para el desarrollo de este trabajo han sido los siguientes:

- Para el desarrollo de código en la plataforma en el cliente, se ha utilizado *HTML-5*, *CSS-3*, *JavaScript* y *JQuery-3.3.1*.

- Para el desarrollo de código en la plataforma en el servidor, se ha utilizado *Python-3*, *Django 1.11*, *WebRTC*, *WebSockets*, *Django-channels 1.5*, *Elastic-Search*, entre otros.
- Para la puesta en producción del servidor, se ha utilizado *Apache*, *Docker* y *AWS*.
- Para el simulador, se utiliza un bundle con el paquete de *Kibotics*, *Kibotics-WebSim*, basado en *A-Frame* y con mejoras específicas de *Kibotics*.
- En cuanto a los ejercicios, se utilizan ficheros de configuración *JSON* con los modelos a incluir en el simulador y se cuenta con repositorios de *GitHub* donde se almacena el código del usuario.

2.3. Metodología

El desarrollo de este TFM puede descomponerse en un conjunto de iteraciones con distintas fases. A este modelo de desarrollo sele conoce como *Modelo de desarrollo en espiral*. Este modelo es típico en la *Metodología de desarrollo ágil*, muy frecuentes en desarrollo software.

Según éste modelo, se han desarrollado reuniones cada 3 días en las que se determinaban los subobjetivos más esenciales e indivisibles en los que se podían subdividir los objetivos secundarios para, de esta manera, alcanzar el objetivo final.

Gracias a este modelo de desarrollo en espiral se puede planificar cómo abordar los subobjetivos establecidos, intentar abordar los problemas surgidos o que vayan a surgir lo más anticipadamente posible e intentar la consecución de los mismo en el menor tiempo posible.

Además de las reuniones semanales, se ha podido hacer un seguimiento de las primeras fases del desarrollo con herramientas externas como la bitácora semanal en la Wiki de *Robotics-Academy*[1] en el que se redactaban los avances principales. Además, se ha contado con un repositorio en *GitHub* de apoyo para desarrollar software en paralelo a incluir después en la plataforma[2].

[1] `https://roboticslaburjc.github.io/2019-tfm-pablo-moreno/logbook/`

[2] `https://github.com/RoboticsLabURJC/2019-tfm-pablo-moreno`

El modelo de desarrollo en espiral escogido ha sido el modelo creado por *Barry Boehm* (Figura 2.1). Al tratarse de un modelo en espiral, se adapta a la perfección a nuestras necesidades, permitiendo disponer de flexibilidad ante cambios en los requisitos semanales, algo común mientras se avanzaba en este desarrollo. En paralelo, nos ha permitido dividir el objetivo final en otros más sencillos, secundarios, y, a su vez, en otros indivisibles y abordables semanalmente. Con esto, se ha conseguido una mitigación del riesgo en el proyecto gracias a la temprana identificación de potenciales riesgos en el mismo, consiguiendo así, une mejor calidad final.

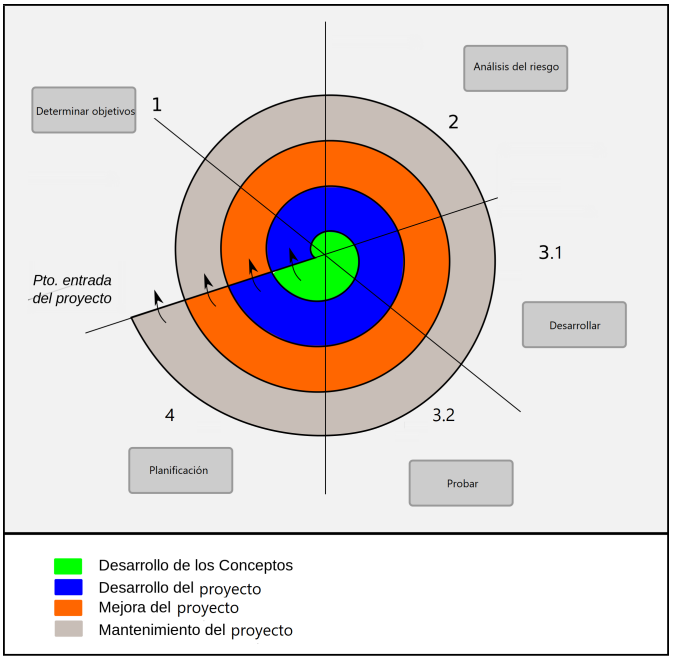

Figura 2.1: Modelo de desarrollo en espiral

La ventaja con este ciclo de vida es que permite la obtención temprana de prototipos funcionales, la optimización progresiva de estos prototipos y, en última instancia, pulir los detalles para abarcar la totalidad de los requisitos especificados. De esta manera, el trabajo se desarrolla de manera incremental con cuatro fases bien definidas:

- **Determinación de objetivos**: Esta primera fase del ciclo está formada por la definición de las metas.
- **Análisis del riesgo**: En esta etapa se evalúan los posibles problemas iniciales al desarrollo y las soluciones a los mismos.

- **Desarrollar y probar**: En esta tercera fase se procede al desarrollo del trabajo propiamente dicho, junto con una serie de pruebas para verificar su funcionamiento.
- **Planificación**: En esta última fase del ciclo de vida se valoran los resultados obtenidos y se planifican las siguientes etapas del proyecto.

2.4. Plan de trabajo

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

1. **Estudio de la plataforma Kibotics**: En esta primera fase de toma de contacto se han modificado las plantillas de la plataforma con una primera versión. Además la puesta en funcionamiento en local del servidor, el simulador y los ejercicios.
2. **Estudio del simulador *WebSim* y de la tecnología *WebRTC***: En esta fase se ha estudiado el funcionamiento del simulador y se han modificado pequeñas funcionalidades. En cuanto a la tecnología de retransmisión de vídeo/audio, se ha desarrollado un primera versión para la plataforma.
3. **Desarrollo de los *Juegos-Compartidos***: Una vez adquiridos lo conocimientos necesarios, se ha realizado una refactorización completa de las plantillas el servidor y se ha desarrollado toda la infraestructura para los *Juegos-Compartidos*.
4. **Desarrollo de los torneos**: Tras completar el desarrollo de los *Juegos-Compartidos*, se ha creado la infraestructura para los torneos, modificando por completo los evaluadores y generando el modelo de datos en la base de datos del servidor con *Elastic-Search*.

Capítulo 3

Infraestructura

En este capítulo se presentan todos los componentes y software que han servido de apoyo en el desarrollo del TFM. En este punto se dará una explicación introductoria a el simulador *WebSim*, tecnologías como *WebRTC*, *Elastic-Search*, *JavaScript*, *A-Frame*, *NPM*.

3.1. JavaScript

Se trata de un lenguaje interpretado de alto nivel bajo el estándar EC-MAScript[1] y basado en *Java* y *C*. Fue creado como lenguaje de aplicaciones web en el cliente. Se interpreta con el navegador web para poder mejorar su interfaz y obtener páginas web dinámicas. En la actualidad se ha extendido al servidor con *Node.js* y se ha convertido en el lenguaje de desarrollo web más extendido.

En este proyecto se ha utilizado ECMAScript-6 para el desarrollo del lado del cliente. De esta manera, se ha desarrollado la inteligencia de las plantillas que corre en el navegador. Las principales características de ES-6 son:

- Es un lenguaje estructurado similar a C (comparte gran parte e su estructura). Le diferencia con *JavaScript* en el alcance de las variables definidas, ya que incorpora la palabra reservada "let" para tener compatibilidad *block-scoping*.

[1]Especificación de lenguaje de programación que define tipos dinámicos y soporta programación orientada a objetos

- Tiene tipado débil, por lo que los datos se asocian al valor en lugar de la variable. Esto permite que la misma variable sea de distintos tipos en distintas parte del código.
- Se trata de un lenguaje interpretado, por lo que no requiere compilación ni fichero binario de código y cada navegador tiene su intérprete para ejecutarlo.
- Está formado por objetos en su totalidad en los que los nombres de sus propiedades son claves de tipo cadena.
- Tiene evaluación en tiempo de ejecución con la función "eval" que evalúa un código en *JavaScript* representado como una cadena de caracteres.

3.2. A-Frame

Es un simulador de código abierto para crear entornos de realidad virtual a partir de HTML para que sea sencillo de leer y comprender. La versión de *Kibotics* es la última estable (v0.9.2), aunque está en constante desarrollo. Además, tiene compatibilidad con otros entornos como *Vive, Rift, Windows Mixed Reality, Daydream o GearVR* y soporte tanto para ordenadores como para *smartphones*.

3.2.1. HTML y primitivas

A-Frame se basa en HTML y DOM con *polyfill*[2]. Para la creación de una escena solo es necesario el fichero HTML con su descripción. Además, la mayoría de las herramientas de HTML (como *React, Vue.js o JQuery* tienen soporte.

Tanto HTML como DOM, forman la capa más externa del entorno, por debajo está el componente *Three.js* gracias al cual un componente puede ser utilizado en otras entidades. Con esta característica no es necesario repetir código, ya que el elemento puede ser referenciado las veces que sea necesario.

Además, *A-Frame* proporciona primitivas para la declaración de los objetos en el escenario. Éstas no son más que los distintos elementos de los que va a constar el escenario.

[2]Fragmento de código en *JavaScript* para dar soporte a funcionalidad moderna en navegadores antiguos.

También permite primitivas complejas para la inclusión de elementos como robots.

3.2.2. Entidad, Componente y Sistema (ECS)

Esta arquitectura forma *A-Frame*, y es muy utilizada en entornos de desarrollo de videojuegos y en 3D. Se trata de una jerarquía que está caracterizada por el principio de herencia, con lo cual los elementos de más bajo nivel, heredan las características de los elementos de mayor nivel de los que dependen.. Esto aporta mayor flexibilidad al definir objetos, gran escalabilidad o eliminación de largas cadenas de herencia. El API para definir cada tipo es el siguiente:

- Entidad: Se representa con *a-entity*.
- Componente: Se representa como un atributo HTML y está formado por esquema, manejadores y métodos. Para declarar un componente se utiliza el métodp de *A-Frame registerComponent*.
- Sistema: Se representa con atributos HTML mediante la etiqueta *a-scene* y se registran con el método de *A-Frame resgisterSystem*.

3.3. Gestores de paquetes

Un gestor de paquetes es un herramienta software que permite automatizar los procesos de instalación, actualización y eliminación de otro software. Para este proyecto se han utilizado NPM (v3.5.2) y Webpack (v4.41.2) para la generación de los bundles referentes a toda la funcionalidad de *WebSim*.

3.3.1. Node Package Manager (NPM)

Node Package Management[7] es un sistema de gestión de dependencias para *Node.js*, que es un entorno de ejecución de *JavaScript*. Permite, configurando un fichero (package.json), descargar los paquetes necesarios para el correcto funcionamiento de la aplicación. Para instalar los paquetes declarados en el fichero basta con ejecutar “npm install” en el directorio del fichero e, incluso se puede configurar para que se lancen scripts

una vez terminada la instalación o el empaquetamiento de los *bundles*. si se combina con *Webpack*.

3.3.2. Webpack

Webpack[8] es un sistema de *bundling* para empaquetar una aplicación web en producción. Se puede considerar la evolución de *Grunt*[9] y *Gulp*[10] dado que permite automatizar procesos como compilar y transpilar. En este proyecto se ha utilizado para hacer el *bundling* del API de los editores y del simulador.

3.4. Simulador WebSim

WebSim es un simulador diseñado para entender, de manera sencilla, conceptos básicos de tecnología enfocada a niños. La primera versión de este simulador fue desarrollado por Álvaro Paniagua Tena[11] y ha sido mejorado por Rubén Álvarez Martín[12].

Utiliza el entorno *A-Frame* para conectar el editor de bloques (Scratch) o editor de texto (Python) con el robot simulado. También permite insertar una comunicación externa mediante *ICE*[3]. Las principales funcionalidades que posee este simulador son:

- Registrar los componentes principales para construir el robot en *A-Frame*.
- Ofrecer la interfaz *JavaScript* para manejar el robot.
- Controlar la ejecución del simulador.

3.4.1. Drivers de sensores

El robot está formado por distintos sensores en *A-Frame*, a los que se puede acceder con *JavaScript*. Los tipos de sensores soportados son los siguientes:

- Ultrasonido
- Cámara

[3] `https://en.wikipedia.org/wiki/Internet_Communications_Engine`

- Infrarrojos
- Odómetro

3.4.2. Drivers de actuadores

El robot también está formado por actuadores en *A-Frame*, con los que se puede dotar de movimiento al robot y a los que se accede mediante *JavaScript*. Además, como las instrucciones son bloqueantes, no es necesario enviar las órdenes de manera constante.

3.5. Django

Django[13][14] es un framework de alto nivel en Python que fomenta el desarrollo rápido y limpio y un diseño pragmático. Creado por desarrolladores expertos, se ocupa de lo relacionado con el desarrollo web, para que puedas centrarte en escribir tu aplicación. En su comienzo se utilizó en producción y, en 2005, se liberó bajo una "licencia BSD". Originalmente fue creado para ayudar en el desarrollo de "World online", para administrar sitios web de noticias.

Se fundamenta en la creación sencilla de sitios web, poniendo énfasis en la conectividad y extensibilidad de sus componentes, la re-utilización, el desarrollo dinámico y en principio "Don't Repeat Yourself". Tambíen soporta el uso de bases de datos como *MySQL, PosgreSQL y SQLite-3*. En cuanto a servidores web soportados, aunque se recomienda *Apache*, soporta la especificación *WSGI*. El único requerimiento de Django es Python 2.5 como mínimo.

Algunas características de Django son:

- Mapeador objeto-relacional. Es capaz de crear una BD virtual orientada a objetos sobre la BD relacional utilizada.
- Soporta aplicaciones instalables en cualquier página gestionada con Django.
- API de BD robusta.
- Sistema de vistas genéricas para ciertas tareas comunes.

- Sistema extensible de plantillas con herencia y etiquetas.
- *Dispatcher* de URLs basado en expresiones reuglares.
- Sistema *middleware* para el desarrollo de características adicionales como el cacheo, la compresión de salida, la normalización de URLs, la protección CSRF y soporte de sesiones.
- Soporte de internacionalización, incluyendo traducciones.
- Documentación incorporada a través de la aplicación administrativa.

3.6. Elastic-Search

Elasticsearch[15] es un motor de analítica y análisis distribuido y open source para todos los tipos de datos, incluidos textuales, numéricos, geoespaciales, estructurados y desestructurados. Está desarrollado en Apache Lucene y fue presentado por primera vez en 2010 por Elasticsearch N.V. (ahora conocido como Elastic). Es conocido por sus API REST simples, naturaleza distribuida, velocidad y escalabilidad. Es el componente principal del Elastic Stack, un conjunto de herramientas open source para la ingesta, el enriquecimiento, el almacenamiento, el análisis y la visualización de datos. Comúnmente referido como el ELK Stack (por Elasticsearch, Logstash y Kibana), el Elastic Stack ahora incluye una gran colección de agentes de envío conocidos como Beats para enviar los datos a Elasticsearch.

La velocidad y escalabilidad de Elasticsearch y su capacidad de indexar muchos tipos de contenido significan que puede usarse para una variedad de casos de uso:

- Búsqueda de aplicaciones
- Búsqueda de sitio web
- Búsqueda Empresarial
- Logging y analíticas de log
- Métricas de infraestructura y monitoreo de contenedores
- Monitoreo de rendimiento de aplicaciones

- Análisis y visualización de datos geoespaciales
- Analítica de Seguridad
- Analítica de Negocios

Entre sus ventajas destacan:

- **Rapidez**. Como Elasticsearch está desarrollado sobre Lucene, es excelente en la búsqueda de texto completo. Elasticsearch también es una plataforma de búsqueda en casi tiempo real, lo que implica que la latencia entre el momento en que se indexa un documento hasta el momento en que se puede buscar en él es muy breve: típicamente, un segundo. Como resultado, Elasticsearch está bien preparado para casos de uso con restricciones de tiempo como analítica de seguridad y monitoreo de infraestructura.
- **Distribuido**.Los documentos almacenados en Elasticsearch se distribuyen en distintos contenedores conocidos como shards, que están duplicados para brindar copias redundantes de los datos en caso de que falle el hardware. La naturaleza distribuida de Elasticsearch le permite escalar horizontalmente a cientos (o incluso miles) de servidores y gestionar petabytes de datos.
- **Amplio conjunto de características**. Además de su velocidad, la escalabilidad y la resistencia, Elasticsearch tiene una cantidad de características integradas poderosas que contribuyen a que el almacenamiento y la búsqueda de datos sean incluso más eficientes, como data rollup y gestión de ciclo de vida del índice.
- **Simple**.La integración con Beats y Logstash facilita el proceso de datos antes de indexarlos en Elasticsearch. Y Kibana provee visualización en tiempo real de los datos de Elasticsearch así como UI para acceder rápidamente al monitoreo de rendimiento de aplicaciones (APM), los logs y los datos de métricas de infraestructura.

3.6.1. Índice, Logstash y Kibana

Un índice de Elasticsearch es una colección de documentos relacionados entre sí. Elasticsearch almacena datos como documentos JSON. Cada documento correlaciona un

conjunto de claves (nombres de campos o propiedades) con sus valores correspondientes (textos, números, Booleanos, fechas, variedades de valores, geolocalizaciones u otros tipos de datos).

Elasticsearch usa una estructura de datos llamada índice invertido, que está diseñado para permitir búsquedas de texto completo muy rápidas. Un índice invertido hace una lista de cada palabra única que aparece en cualquier documento e identifica todos los documentos en que ocurre cada palabra.

Durante el proceso de indexación, Elasticsearch almacena documentos y construye un índice invertido para poder buscar datos en el documento casi en tiempo real. La indexación comienza con la API de índice, a través de la cual puedes agregar o actualizar un documento JSON en un índice específico.

Logstash, uno de los productos principales del Elastic Stack, se usa para agregar y procesar datos y enviarlos a Elasticsearch. Logstash es una pipeline de procesamiento de datos open-source y del lado del servidor que te permite introducir datos de múltiples fuentes simultáneamente y enriquecerlos y transformarlos antes de que se indexen en Elasticsearch.

Kibana es una herramienta de visualización y gestión de datos para Elasticsearch que brinda histogramas en tiempo real, gráficos circulares y mapas. Kibana también incluye aplicaciones avanzadas, como Canvas, que permite a los usuarios crear infografías dinámicas personalizadas con base en sus datos, y Elastic Maps para visualizar los datos geoespaciales.

3.7. WebRTC

WebRTC[16] es un proyecto libre de código abierto bajo una licencia BSD que proporciona a los navegadores web y a las aplicaciones móviles comunicación en tiempo real (*Real Time Connection* (RTC)) a traves de APIs. Permite que la comunicación audio, vídeo y datos funcione con conexión *Peer To Peer*, es decir entre máquinas sin necesidad del uso del servidor o de plugins. Esta plataforma está siendo estandarizada por medio de W3C[4] (World Wide Web Consortium) y del IETF[5] (Internet Engineering Task Force).

[4] `https://www.w3.org/`

[5] `https://www.ietf.org/`

WebRTC tiene soporte en Apple, Google, Microsoft y Mozilla, entre otros. Es mantenido por Google. Su implementación se realiza en JavaScript, y sus principales componentes son:

- **getUserMedia**. Para permitir al navegador acceder a la cámara y el micrófono.
- **PeerConnection**. Para el establecimiento de las llamadas de vídeo y audio.
- **DataChannels**. Para el establecimiento de la conexión de transmisión de datos.

3.8. Retransmisión en directo

La retransmisión en directo o *Streaming*, es un término utilizado para definir la visualización de vídeos y audio en tiempo real. Básicamente hay dos tipos de streaming:

- **Streaming real**. También conocido como "Live Streaming". Requiere un servidor especial que difunde la información de audio/vídeo en tiempo real. El reproductor del cliente, lo recibe y visualiza de manera instantánea. Para soportar esta tecnología, se necesitan servidores dedicados potentes con una gran cantidad de recursos y ancho de banda. Este tipo de transmisión utiliza el protocolo UDP[6] para la transmisión de datos de vídeo/audio porque soporta la pérdida de paquetes y está orientado a la velocidad de transmisión.
- **Pseudo-streaming**. También conocido como "Streaming HTTP", es una alternativa al *Live streaming* cuando no se quieren gastar tantos recursos y dinero en servidores dedicados de difusión. Para conseguir el efecto de difusión en vivo se utilizan buffers de datos del vídeo. Para ésto, se utiliza el protocolo TCP[7]. Este protocolo no es eficaz en transmisiones en vivo porque tiene controles de pérdidas de paquetes, ya que no soporta estas pérdidas. Este inconveniente se contrarresta al existir el *buffering*, por lo que no existirán parones en el transmisión y la calidad será mejor aunque exista un cierto retardo.

[6] https://www.ionos.es/digitalguide/servidores/know-how/udp-user-datagram-protocol/
[7] https://es.wikipedia.org/wiki/Protocolo_de_control_de_transmisi%C3%B3n

Capítulo 4

Juegos Compartidos

4.1. Refactorización de kibotics

4.2. Ejercicios

4.3. Plantillas

4.3.1. Infraestructura del servidor

4.4. WebRTC

4.4.1. Cliente

4.4.2. Servidor

4.5. Evaluadores

Capítulo 5

Torneos

5.1. Plantillas

5.1.1. Infraestructura del servidor

5.2. Elastic-Search

5.3. Maestro de ceremonias

5.4. Automatización de los torneos

5.4.1. Evaluadores

5.5. Retransmisión en vivo

Capítulo 6

Conclusiones

Una vez documentada toda la información sobre este TFM, se dedica este capítulo a la comprobación de los objetivos alcanzados, así como a la explicación de los conocimientos adquiridos y una breve exposición de posibles mejoras en la plataforma y de las líneas de actuación futuras.

6.1. Conclusiones

El objetivo principal de la refactorización de la plataforma y la incorporación de los *Juegos-Compartidos* y los *Torneos* ha sido alcanzado con éxito. Este objetivo se ha subdividido en dos objetivos secundarios.

El primer objetivo secundario ha sido el desarrollo de los *Juegos-Compartidos* y ha sido alcanzado satisfactoriamente. Gracias a la tecnología *WebRTC* ha sido posible desarrollar cuatro nuevas prácticas para la plataforma en las que los usuarios pueden competir. Para lograr este objetivo ha sido necesario una implementación nueva de las plantillas, la creación del chat y el soporte para el mismo en el lado del servidor con *Django-Channels*.

El segundo objetivo secundario ha sido la creación de los *Torneos*. Este objetivo también ha sido alcanzado y para su consecución ha sido necesaria la tecnología *Elastic-Search* en el servidor y la generación de una nueva figura llamada maestro de ceremonias, el cual ejecuta los códigos de los participantes de manera automática con un fichero *JSON*. Además se han realizado los pasos necesarios para incluir la retransmisión en directo de *Twitch* en un *IFrame* empotrado en la página de los torneos para que se puedan ver en

tiempo real así como un enlace a *Youtube-Live*.

Para la consecución de todos estos objetivos, y de manera personal, se ha alcanzado el objetivo de una mejora de conocimiento en un campo tan heterogéneo como el desarrollo web. Gracias a ello, se han adquirido conocimientos en campos tan diversos como *JavaScript*, *HTML*, *CSS*, *JQuery*, *WebRTC*, *Django*, *Django-Channels*, *Twitch y Youtube API*, *Elastic-Search*, entre otros.

6.2. Trabajos futuros

Durante el desarrollo de este TFM la plataforma ha sufrido diversas mejoras importantes (se ha pasado de la versión 1.3 a la versión 2.2.2). Estas mejoras han venido de la mano de este TFM y del resto de equipo que conforma *Kibotics*. Aún así hay muchas mejoras abordables para seguir mejorando la plataforma. Algunas de estas mejoras, relacionadas con este trabajo son:

- **Prácticas multirobot**: La inclusión de prácticas con más de dos robots es un punto muy interesante para poder hacer torneos masivos.
- **Desarrollo de un vídeo-chat**: De esta manera los contrincantes pueden hablar entre ellos en lugar de utilizar el chat únicamente.
- **Nuevos torneos**: En los que los participantes jueguen directamente y no mediante el maestro de ceremonias.
- **Nuevos Juegos-Compartidos**: La introducción de nuevos juegos, proporcionaría una mayor diversidad en el elenco de prácticas existentes en *Kibotics*.

Bibliografía

[1] Tecnología, programación y robótica en eso. https://n9.cl/7lqc.

[2] Programación y robótica en ep. http://www.telemadrid.es/noticias/madrid/madrilenos-impartiran-Programacion-Robotica-Primaria-0-2158584125--20190914103816.html.

[3] Scratch. https://scratch.mit.edu/.

[4] Lego. https://www.lego.com/es-es/categories/coding-for-kids.

[5] Kodu. https://www.kodugamelab.com/.

[6] Snap. https://snap.berkeley.edu/.

[7] Isaac z. schlueter. documentación oficial de npm. https://www.npmjs.com/.

[8] Webpack. documentación oficial de webpack. https://webpack.js.org/.

[9] Grunt. documentación oficial de grunt. https://gruntjs.com/.

[10] Gulp. documentación oficial de gulp. https://gulpjs.com/.

[11] Álvaro paniagua tena. websim, simulador de robots con tecnologías web vr. (2018). https://github.com/RoboticsLabURJC/2018-tfg-alvaro_paniagua.

[12] Rubén Álvarez marín. mejoras en entorno de robótica educativa para niños. (2019). https://github.com/RoboticsLabURJC/2018-tfg-alvaro_paniagua.

[13] Django. documentación oficial de django. https://www.djangoproject.com/.

[14] Django. documentación de wikipedia de django. https://es.wikipedia.org/wiki/Django_(framework).

[15] Elasticsearch. documentación oficial de elasticsearch. https://www.elastic.co/es/.

[16] Webrtc. documentación oficial de webrtc. https://webrtc.org/.