



**GRADO EN INGENIERÍA EN SISTEMAS DE
TELECOMUNICACIÓN**

Curso Académico 2019/2020

Trabajo Fin de Grado

**Colaboración en el desarrollo software del entorno
web de aprendizaje de Kibotics**

Autor : Natalia Monforte Rodríguez

Tutor : Dr. José María Cañas Plaza

Índice general

Lista de figuras	5
Lista de tablas	7
1. Introducción	1
1.1. Robótica	1
1.2. Tecnologías web	4
1.2.1. Tecnologías web en el lado del cliente	5
1.2.2. Tecnologías web en el lado del servidor	5
1.3. Docencia robótica	7
1.4. Motores de físicas	9
2. Objetivos	11
2.1. Objetivos	11
2.2. Metodología y planificación	11
3. Herramientas	15
3.1. Lenguaje JavaScript	15
3.2. Lenguaje HTML	16
3.3. Lenguaje JSON	17
3.4. Lenguaje Python	18
3.5. Programación con Scratch	19
3.6. Blender	20
3.7. Simulador Websim	21
3.7.1. A-Frame	21

3.7.2. Sistema de físicas de <i>A-Frame</i>	22
4. Motor de físicas complementario	25
4.1. Estudio de las colisiones	25
4.1.1. Colisiones elásticas	25
4.1.2. Colisiones inelásticas	25
4.2. Estudio de la fricción	25
4.3. Físicas realistas	25
4.3.1. Mejoras en el movimiento en el eje vertical	25
4.3.2. Mejoras en el movimiento en el plano horizontal	25
5. Nuevos ejercicios	27
5.1. Roomba	27
5.2. Aparcamiento	27
5.3. Sigue-líneas sofisticado	27
5.4. Laberinto 3D para mBot	27
5.5. Laberinto para drone	27
5.5.1. Con señalización	27
5.5.2. Sin señalización	27
5.6. Fútbol competitivo	27
5.6.1. Evaluador	27
6. Conclusiones	29
6.1. Valoración de los resultados	29
6.2. Mejoras futuras	29

Índice de figuras

1.1. Ejemplos de robots en la actualidad	3
4figure.caption.5	
6figure.caption.6	
1.4. Aprendizaje de programación robótica con <i>Lenobotics</i>	7
1.5. Aprendizaje de programación robótica con <i>LEGO education</i>	8
1.6. Interfaz de programación en <i>Kibotics</i> de un ejercicio en <i>Scratch</i>	9
1.7. Robots soportados en la plataforma <i>Kibotics</i>	9
3.1. Interfaz de programación con <i>Scratch</i>	20
3.2. Interfaz de trabajo con <i>Blender</i>	20
23figure.caption.13	

Índice de cuadros

3.1. Parámetros configurables del sistema de físicas de <i>A-Frame</i>	24
--	----

Capítulo 1

Introducción

En este primer capítulo de la memoria se van a explicar los conceptos clave entorno a los cuales se ha desarrollado este Trabajo de Fin de Grado. Entender qué son la robótica y las tecnologías web y por qué son importantes es fundamental, ya que la combinación de ambos conceptos ha dado lugar a la docencia robótica.

Por otro lado, en este capítulo también se va a introducir el concepto de motor de físicas, ya que una importante parte del trabajo se ha basado en la generación de un motor de físicas complementario que permite recrear los movimientos realizados por los robots del entorno web de *Kibotics* con un mayor realismo.

1.1. Robótica

La robótica es la ciencia que estudia la creación de máquinas automatizadas capaces de recrear comportamientos humanos o animales en función del software que lleven incorporados. Estas máquinas son las que se denominan robots.

Haciendo un breve repaso de la historia de los robots, cabe destacar que desde el 85 a.C. ya se empezaron a crear los primeros robots en la Antigua Grecia. En esa época la creación de robots se basaba en el intento de replicar personas por medio de máquinas. De hecho, esas máquinas ni siquiera se denominaban robots. El término robot fue acuñado en 1920 por Karel

Capek como homenaje a su obra teatral Rossum's Universal Robots, que trataba de una empresa encargada de fabricar humanos artificiales para facilitar la realización de tareas a los trabajadores de las fábricas. Así, la palabra Robot procede de Robbota, que en checo significa trabajo forzado o servidumbre¹.

El primer robot del que se tiene contancia es el *Elektro*. Fue construido en 1937 y se le conoce como *el primer robot de la historia*. Este robot representaba a un humano de 2 metros de altura y 120 kg de peso. Era capaz de recrear movimientos humanos como caminar y de comunicarse utilizando hasta 700 palabras². Posteriormente, George Charles Devol creó el primer robot industrial en 1948. Por ello, George Charles Devol es considerado el inventor de la robótica. Junto a Joseph F. Engelberger, creó la empresa Unimation, que fue la responsable de la creación de gran parte de los primeros robots industriales de la historia³.

Hoy en día, los robots están presentes prácticamente en cualquier ámbito de la vida de cualquier persona. Ya se han creado robots capaces de recrear casi cualquier actividad realizada por el ser humano o que nos facilita la realización de las mismas. Algunos de los robots más populares en la actualidad son lo siguientes. En la Figura 1.1 se ofrece una representación gráfica de ellos.

- Robots que se encargan de la limpieza del hogar (por ejemplo, Roomba).
- Robots sociales encargados de hacer compañía (por ejemplo, Pepper).
- Robots de cocina capaces (por ejemplo, Thermomix).
- Robots especialistas en labores de rescate (por ejemplo, Atlas).
- Drones (por ejemplo, Tello).
- Coches autónomos (por ejemplo, Tesla).

¹<https://revistaderobots.com/robots-y-robotica/que-es-la-robotica/>

²Ibidem

³Ibidem



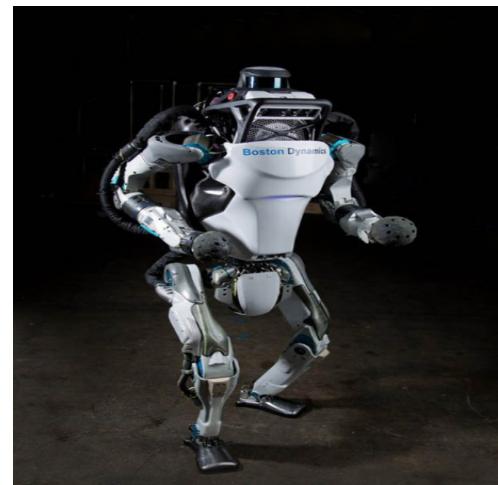
(a) Roomba



(b) Pepper



(c) Thermomix



(d) Atlas



(e) Tello



(f) Tesla

Figura 1.1: Ejemplos de robots en la actualidad

1.2. Tecnologías web

Las tecnologías web están en continúo desarrollo. Actualmente, existen tecnologías web tanto en el lado del cliente como en el lado del servidor. La idea de esta separación es marcar las diferentes partes de un sistema software para poder controlarlo de una forma más eficaz. Por este motivo, el frontend recoge los datos y el backend los procesa.

Por un lado, el frontend engloba todas aquellas tecnologías web del lado del cliente que se encargan de recopilar los datos. Principalmente, existen tres tecnologías de frontend: *HTML*, *CSS* y *JavaScript*. Estas tres tecnologías permiten al usuario interactuar con el servidor web, utilizando un navegador como intérprete. Por otro lado, el backend se encarga del almacenamiento de información en bases de datos, gestión de servidores y servir las vistas de las páginas web seleccionadas por el desarrollador en el lado del cliente. En el backend, el número de tecnologías es mucho más extenso. La programación backend incluye lenguajes como *PHP*, *Python*, *.NET* o *Java* y las bases de datos sobre las que se trabaja pueden ser *SQL*, *MongoDB* o *MySQL*. Todas estas tecnologías web permiten implementar comportamientos determinados de las aplicaciones web en el servidor. En la Figura 1.2 se muestra un esquema de la división de las tecnologías web entre ambos planos.

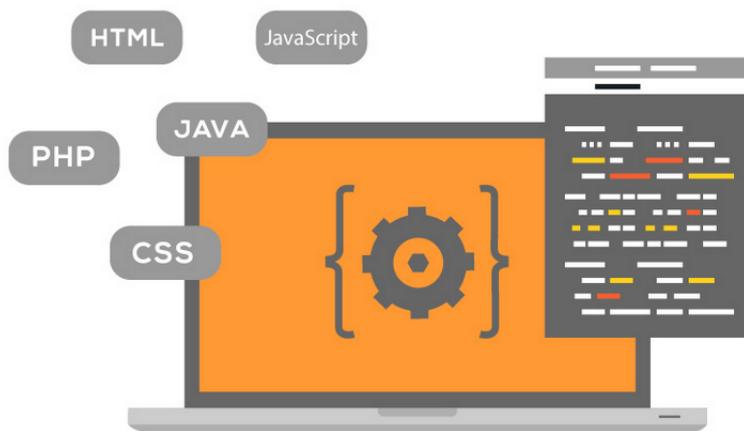


Figura 1.2: Tecnologías web en el lado del cliente y del servidor⁴

⁴<https://www.ingeniovirtual.com/conceptos-basicos-sobre-tecnologias-de-desarrollo-web/>

1.2.1. Tecnologías web en el lado del cliente

Las tres tecnologías web del frontend que permiten la interacción entre usuario y servidor web son las siguientes:

- **HTML:** es un lenguaje de marcado que permite diferenciar los contenidos y definir la estructura de un sitio web. Permite dividir una página web en diferentes secciones: títulos, texto, imágenes, pie de página, etc. Es la base de toda página web.
- **CSS:** es un lenguaje de hojas de estilo que permite modificar la apariencia de una página web.
- **JavaScript:** es un lenguaje de programación interpretado que permite definir el comportamiento de una página web (por ejemplo, al hacer click en un enlace). Por ello, gracias a JavaScript el usuario puede interaccionar con la página web.

1.2.2. Tecnologías web en el lado del servidor

Uno de los lenguajes más utilizados en la programación del backend es el lenguaje *JavaScript*. *JavaScript* se creó para su uso en el frontend en un principio; sin embargo, gracias al motor *NodeJS*, este lenguaje puede ser interpretado en el lado del servidor sin necesidad de un navegador.

El código *JavaScript* del cliente y el servidor es independiente el uno del otro. Sin embargo, resulta de especial interés el hecho de que se pueda desarrollar código en un mismo lenguaje tanto en el frontend como en el backend por las facilidades y reducción de tiempos y esfuerzo que esto supone para los desarrolladores.

Se puede utilizar el mismo lenguaje en todos los contextos del desarrollo: en el cliente de escritorio con DOM, en el cliente móvil con *Cordova o React Native*, en el servidor con *Node.js* o en la base de datos con *MongoDB*. En cuanto a la tecnología empleada, las herramientas que se utilizan en el backend son principalmente: editores de código, compiladores, depuradores de código y gestores de bases de datos.

La comunicación entre cliente y servidor se realiza utilizando el protocolo *HTTP* (protocolo de transferencia de hipertexto). Este protocolo funciona mediante la emisión de una serie de peticiones y respuestas entre el cliente y el servidor usando diferentes métodos. Existen numerosos métodos *HTTP*, pero los más comunes son los siguientes⁵:

- **GET:** solicitud de datos de un recurso concreto.
- **PUT:** reemplazo de las representaciones actuales del recurso de la petición.
- **POST:** envío de datos a un recurso concreto, normalmente provocando el cambio de estado del servidor.
- **DELETE:** eliminación de un recurso.
- **HEAD:** solicitud de datos de un recurso concreto tal y como ocurre con el método *GET*, pero la respuesta no incluye cuerpo.

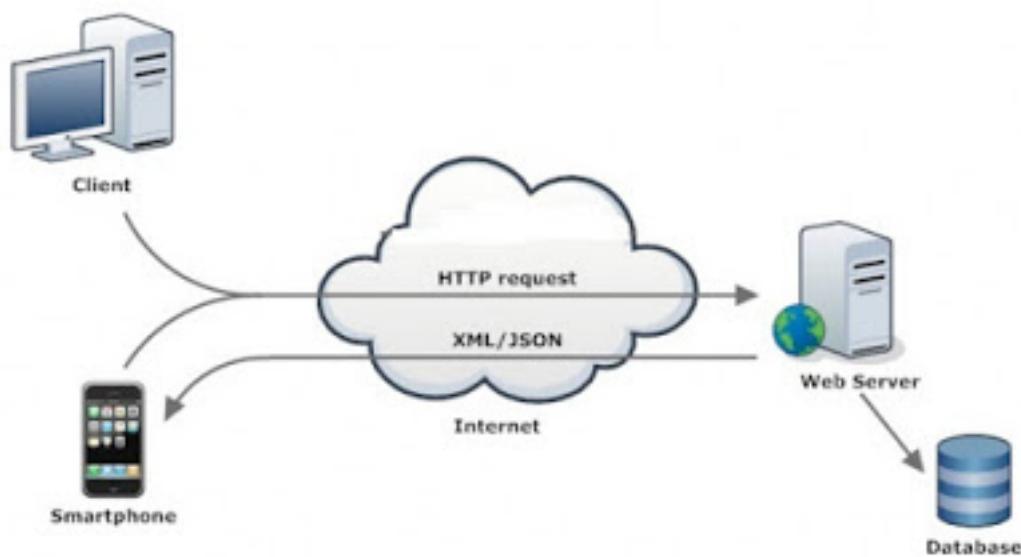


Figura 1.3: Ejemplo de una interacción *HTTP*⁶

⁵<https://developer.mozilla.org/es/docs/Web/HTTP/Methods>

⁶<http://dsanchezz.blogspot.com/2015/10/rest.html>

1.3. Docencia robótica

En la intersección entre la robótica y las tecnologías web se encuentra la docencia robótica. La docencia robótica tiene como objetivo el acercamiento a las tecnologías web de última generación que permiten el desarrollo software para la creación de robots desde edades tempranas.

Hoy en día, el plan de educación que se imparte en colegios e institutos no hace hincapié en la enseñanza de estas tecnologías, por lo que los niños no se encuentran especialmente motivados en el desarrollo de software robótico ya que desconocen su uso y posibilidades. En consecuencia, especialmente en los últimos años, han surgido algunas plataformas dedicadas a la docencia robótica que se encargan de impartir cursos de iniciación a la robótica desde edades muy tempranas para conseguir que los niños se sientan atraídos por esta rama desde el principio y aprendan a pensar como verdaderos programadores desde muy pequeños.

Un ejemplo de estas plataformas que se comentan es el de *Lenobotics*. *Lenobotics* es un programa que se encarga de impartir cursos de robótica en centros educativos para desarrollar las habilidades cognitivas de los niños que resultan necesarias para la programación⁷.



Figura 1.4: Aprendizaje de programación robótica con *Lenobotics*

⁷<https://lenobotics.com/>

Por su parte, *LEGO education* también ofrece a los más pequeños la posibilidad de iniciarse en la robótica a través de sus kits de robótica. Estos kits permiten aprender unas primeras nociones de electrónica, robótica y programación⁸.



Figura 1.5: Aprendizaje de programación robótica con *LEGO education*

El presente trabajo se va a centrar en la plataforma *Kibotics*⁹. *Kibotics* es un entorno web para docencia en robótica y programación que permite a niños y adolescentes aprender programando. Esto quiere decir que *Kibotics* apuesta por una enseñanza puramente práctica, ya que resulta mucho más atractiva tanto la enseñanza como el aprendizaje de este modo que únicamente con clases teóricas.

Kibotics se basa en la utilización del simulador *WebSim* que, a su vez, está basado en la tecnología *A-Frame* para representar los mundos. Este simulador permite la creación de diferentes ejercicios para los robots que tienen soporte en la plataforma: piBot, mBot, fórmula 1 y drone Tello. Estos ejercicios podrán solucionarse tanto en *Scratch* (especialmente indicado para aquellos alumnos que no hayan programado anteriormente) como en *Python* (para aquellos niños que cuenten con nociones previas).

⁸<https://education.lego.com/es-es>

⁹<https://kibotics.org/>

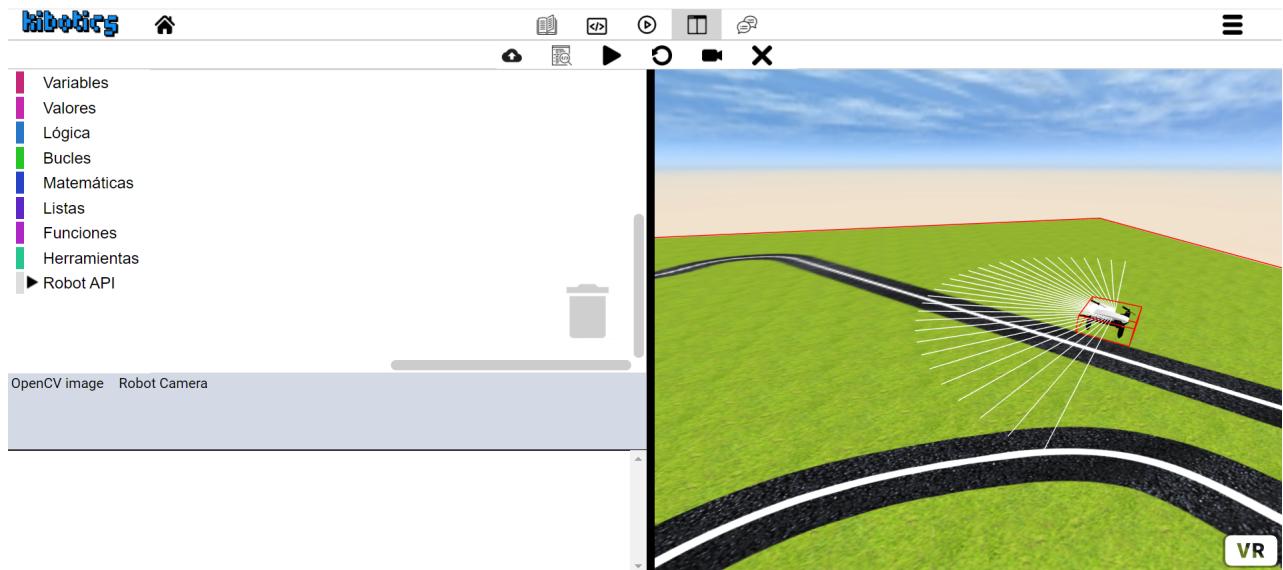


Figura 1.6: Interfaz de programación en *Kibotics* de un ejercicio en *Scratch*

Todos los robots cuentan con cámaras incorporadas en el hardware, lo que permite la creación de ejercicios que se deben solucionar mediante la utilización de la visión artificial además de los ejercicios que se puedan solucionar mediante el uso de los sensores de los robots (sensores infrarrojos, por ejemplo). La Figura 1.7 muestra los robots que soporta actualmente la plataforma.

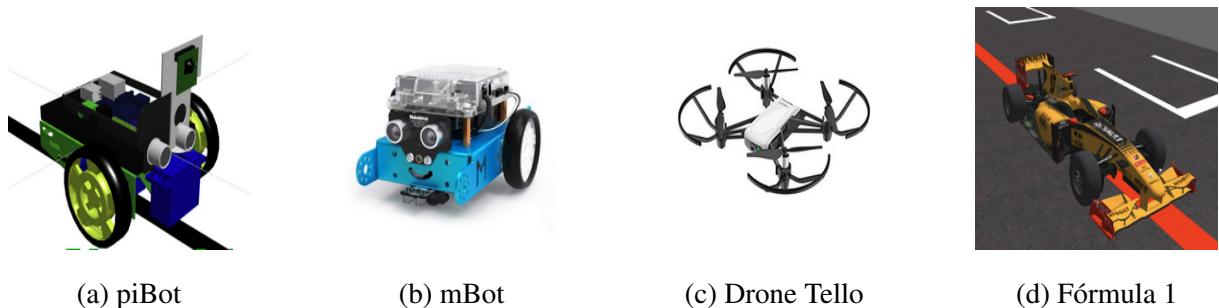


Figura 1.7: Robots soportados en la plataforma *Kibotics*

1.4. Motores de físicas

Un motor de físicas es un software capaz de realizar simulaciones de ciertos sistemas físicos como la dinámica del cuerpo en movimiento, la fricción y la elasticidad de una colisión.

Se emplean con mucha frecuencia en los videojuegos, para recrear con un mayor realismo el movimiento de los personajes.

Existen numerosos motores de físicas como *Box2D*, *Cocos2D*, *Ammo.js* o *CANNON*. El presente trabajo se va a centrar en este último, ya que es el que emplea *A-Frame* en la actualidad. En el capítulo 3: Herramientas se explicará con mayor detalle las peculiaridades de este motor de físicas en cuestión.

Capítulo 2

Objetivos

En este capítulo se explican los objetivos del presente trabajo, la metodología que se ha seguido para alcanzar dichos objetivos y la planificación que se ha llevado durante el proceso de investigación.

2.1. Objetivos

El principal objetivo de este Trabajo de Fin de Grado es la mejora de las habilidades de desarrollo software mediante diversas actividades de programación.

Con este fin, se han abordado las siguientes tareas:

- Exploración del simulador *WebSim* y creación de nuevos mundos para el desarrollo de ejercicios más variados.
- Exploración de la tecnología *A-Frame* para la construcción de nuevos escenarios.
- Exploración del motor de físicas de *CANNON* para desarrollar unas físicas más realistas para el simulador *WebSim*.

2.2. Metodología y planificación

Con el fin de asegurar el correcto desarrollo del Trabajo de Fin de Grado se estableció una reunión semanal con el tutor para compartir los progresos realizados durante la semana y en la

que el tutor me pudo orientar sobre dónde dirigir los esfuerzos cada semana. Paralelamente a las reuniones semanales, también se ha contado con un canal de slack en el que se encuentran todos los contribuyentes del entorno *Kibotics* en el que se han podido plantear todo tipo de dudas durante el proceso de aprendizaje.

El proceso de elaboración del Trabajo de Fin de Grado se ha dividido en cinco fases distintas:

- **FASE 0:** aprendizaje y primera toma de contacto con las tecnologías web necesarias para la elaboración del trabajo. Especialmente *A-Frame* y *JavaScript*.
- **FASE 1:** estudio del código de *Kibotics-WebSim*.
- **FASE 2:** creación de los primeros mundos utilizando las funcionalidades proporcionadas por *Blender* y *A-Frame*.
- **FASE 3:** estudio de las físicas de *A-Frame* (motor de *CANNON*) y elaboración de un motor de físicas complementario para el simulador *WebSim*.
- **FASE 4:** creación de nuevos ejercicios para incluir en la plataforma.

También se ha elaborado un blog en el que se han ido compartiendo los resultados y el trabajo que se ha realizado cada semana. El blog se ha implementado gracias al dominio gratuito que ofrece Github para crear un blog¹. En el README de mi Github se ha incluido un enlace pinchable para acceder a dicho blog².

Para integrar el código de las mejoras o aportaciones realizadas al código fuente de *Kibotics*, cabe destacar que se ha utilizado el sistema que ofrece Github para integrar código mediante la creación de nuevas ramas y parches. Para que los desarrolladores pudiesen añadir las nuevas funcionalidades al código fuente de *Kibotics*, se creaba una nueva rama actualizada con los últimos cambios de la rama principal. Sobre esta rama se desarrollaba la solución a cada incidencia. Una vez incluidos los cambios se explicaban en un comentario o commit y se subían a la nueva rama creada del repositorio de *Kibotics*. El siguiente paso consistía en solicitar la fusión de los cambios de esta rama con la rama principal, abriendo peticiones pull request o parches. Tras la

¹<https://roboticslaburjc.github.io/2019-tfg-natalia-monforte/>

²<https://github.com/RoboticsLabURJC/2019-tfg-natalia-monforte>

solicitud de la fusión o parche los desarrolladores que cuentan con más experiencia verifican que los cambios son correctos y, si es así, integran los cambios a la rama maestra oficial, dando por resuelta la incidencia. Los comandos necesarios para realizar la integración del código a la rama creada son los siguientes:

```
git checkout -b issue-XXX  
git add -ruta-del-fichero-a-añadir  
git commit -m "Comentario para el commit"  
git push -u origin issue-XXX
```


Capítulo 3

Herramientas

En este capítulo se explica con un mayor grado de detalle qué herramientas han sido necesarias para el desarrollo del trabajo. Principalmente, se han utilizado los lenguajes de programación *JavaScript*, *HTML*, *JSON*, *Python* y *Scratch*. Por otro lado, se han utilizado aplicaciones como *Blender* para el modelado de objetos 3D y el simulador *WebSim* para la recreación de los mundos tridimensionales.

3.1. Lenguaje JavaScript

JavaScript es un lenguaje de programación interpretado de alto nivel que se encuentra bajo el estándar *ECMAScript*¹. Este lenguaje es comúnmente conocido por su uso en los scripts de las páginas web. No obstante, dada su orientación a objetos y a ser un lenguaje de programación basada en prototipos y de un solo hilo, es usado en otros muchos entornos externos de la página web: *Node.js*, *Apache CouchDB* o *Adobe Acrobat*².

La sintaxis es similar a la utilizada en *Java* y *C++*. De esta manera, se facilita el aprendizaje del lenguaje ya que está basado en conceptos ya conocidos por el programador. Las estructuras del lenguaje, tales como sentencias condicionales (if y switch) y bucles (while y for) funcionan de una manera muy similar a como lo hacen en los otros lenguajes de programación³.

¹Especificación de lenguaje de programación en el que se definen tipos dinámicos y soporte de programación orientada a objetos

²<https://developer.mozilla.org/es/docs/Web/JavaScript>

³Ibidem

Las siguientes características son las principales de *ECMAScript*:

- Lenguaje escruturado similar a la estructura utilizada en *Java* y *C++*.
- *ECMAScript 2015* añadió la palabra clave *let*, que permite que el alcance de la variable se corresponda con el bloque en el que esta se haya definido (*block scoping*).
- Tipado débil, es decir, el tipo de datos se asocia al valor de la variable en un preciso momento.
- El lenguaje está formado por objetos.
- Lenguaje interpretado, es decir, se compila justo-a-tiempo. No es necesario disponer de un compilador adicional, ya que cada navegador incluye un intérprete que se encarga de ejecutar el código.

El proyecto está enteramente programado en *JavaScript*, ya que se trata de una aplicación web que corre en el lado del cliente. Por ello, este lenguaje es el que mejores prestaciones y más necesidades cubre durante el desarrollo de la aplicación.

3.2. Lenguaje HTML

HTML es un lenguaje de marcado que define la estructura de una página web. *HTML* ofrece una serie de elementos que permiten clasificar diferentes partes de una misma página web en una misma clase para otorgarles una misma apariencia. Además, *HTML* permite cambiar el estilo de las palabras (por ejemplo, a cursiva, a negrita, agrandar o reducir el tamaño de letra, etc)⁴.

Las partes principales del elemento *HTML* son las siguientes:

- **Etiqueta de apertura:** se trata del nombre del elemento y se incluye entre paréntesis angulares (*<*). Indica el inicio del elemento.
- **Etiqueta de cierre:** similar a la etiqueta de apertura salvo que incluye, además, una barra de cierre (*/*) precediendo al nombre de la etiqueta. Indica el fin del elemento.

⁴https://developer.mozilla.org/es/docs/Learn/Getting_started_with_the_web/HTML_basics

- **Contenido:** todo aquello que se incluye entre la etiqueta de apertura y la de cierre.
- **Elemento:** es el conjunto formado por las etiquetas de apertura y cierre y el contenido del elemento.

Además, cada elemento puede incluir uno o más atributos que permiten añadir mas información acerca de ese elemento. Por ejemplo, añadir información sobre el estilo del elemento. A continuación, se incluye un fragmento de código *HTML* a modo de ejemplo.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Mi código de prueba</title>
  </head>
  <body>
    <p class="editor-note">Esto es<strong>una prueba</strong></p>
  </body>
</html>
```

Kibotics emplea *HTML* para crear las plantillas de las diferentes páginas que sirve la aplicación web.

3.3. Lenguaje JSON

El acrónimo *JSON* significa JavaScript Object Notation (Notación de Objetos de JavaScript). Se trata de un formato para el intercambio de datos. Es un lenguaje sencillo para la escritura y lectura humana y, al mismo tiempo, resulta fácil para las máquinas interpretarlo y procesarlo. *JSON* está constituido por dos estructuras: una colección de pares nombre - valor y una lista ordenada de valores. Dado que estas convenciones son conocidas por otros lenguajes como *C*, *C++*, *C*, *Java*, *JavaScript*, *Perl*, *Python*, se trata de un lenguaje ideal para el intercambio de datos⁵.

⁵<https://www.json.org/json-es.html>

En *JSON*, estas estructuras se presentan de la siguiente forma:

- **Objeto:** conjunto desordenado de pares nombre - valor. Un objeto va encerrado entre llaves (). La sintaxis es la siguiente:

```
objeto {  
    nombre1: valor1,  
    nombre2: valor2,  
    ...  
}
```

- **Array:** colección de valores. Van encerrados entre corchetes []. Un valor puede ser una cadena de caracteres con comillas dobles, un número, true, false o null, un objeto o un array. Estas estructuras pueden anidarse.

La aplicación de *Kibotics* emplea *JSON* para crear los ficheros de configuración de los diferentes escenarios utilizados en los ejercicios que ofrece la aplicación. Mediante un parser se recopila la información necesaria del fichero de configuración *JSON* para poder construir el mundo.

3.4. Lenguaje Python

Python es un lenguaje de programación muy popular hoy en día. Fue creado por Guido van Rossum y lanzado en 1991. La popularidad de este lenguaje reside en su fácil comprensión para el usuario y en las facilidades que este ofrece a la programación. Las principales características son las siguientes⁶:

- Python es compatible con diferentes sistemas operativos (Windows, Mac, Linux, Raspberry Pi, entre otros).
- Su sintaxis se asemeja en gran medida al habla inglesa, lo que facilita su comprensión.

⁶https://www.w3schools.com/python/python_intro.asp

- Su sintaxis permite a los programadores escribir códigos con menos líneas que con otro lenguaje, lo que facilita la labor de depurado.
- Python es un lenguaje interpretado, lo que significa que el código puede ser ejecutado tan pronto como se escribe. No es necesario disponer de un compilador externo.
- Este lenguaje puede ser tratado de una manera procedural, una manera orientada a objetos o una manera funcional.
- Se basa en la indentación. Utiliza los espacios en blanco para determinar el alcance de las estructuras.

Valorando todas las ventajas que ofrece *Python*, se ha seleccionado este lenguaje como uno de los que ofrece la plataforma *Kibotics* para que los usuarios puedan dar solución a los ejercicios que se plantean.

3.5. Programación con Scratch

Scratch es un lenguaje de programación visual que fue desarrollado por el Grupo Lifelong Kindergarten del MIT Media Lab. Hoy en día, se utiliza frecuentemente en la educación de niños, adolescentes y adultos ya que permite el aprendizaje de la programación sin tener un amplio conocimiento del código.

Scratch ofrece al usuario la posibilidad de programar construyendo una secuencia de código a partir de diversos bloques de acciones. El programador es capaz de construir la secuencia de código con rapidez y facilidad, ya que cada bloque incluye una secuencia de texto que explica la función que desempeña. Por ello, la secuencia de código finalmente podrá ser leída e interpretada como si de un texto se tratase.

Este es el segundo lenguaje ofrecido por *Kibotics* para dar solución a los ejercicios. Gracias a la interfaz gráfica de la que dispone, Scratch es sin duda una de las mejores alternativas para aquellas personas que no disponen de nociones previas de programación.

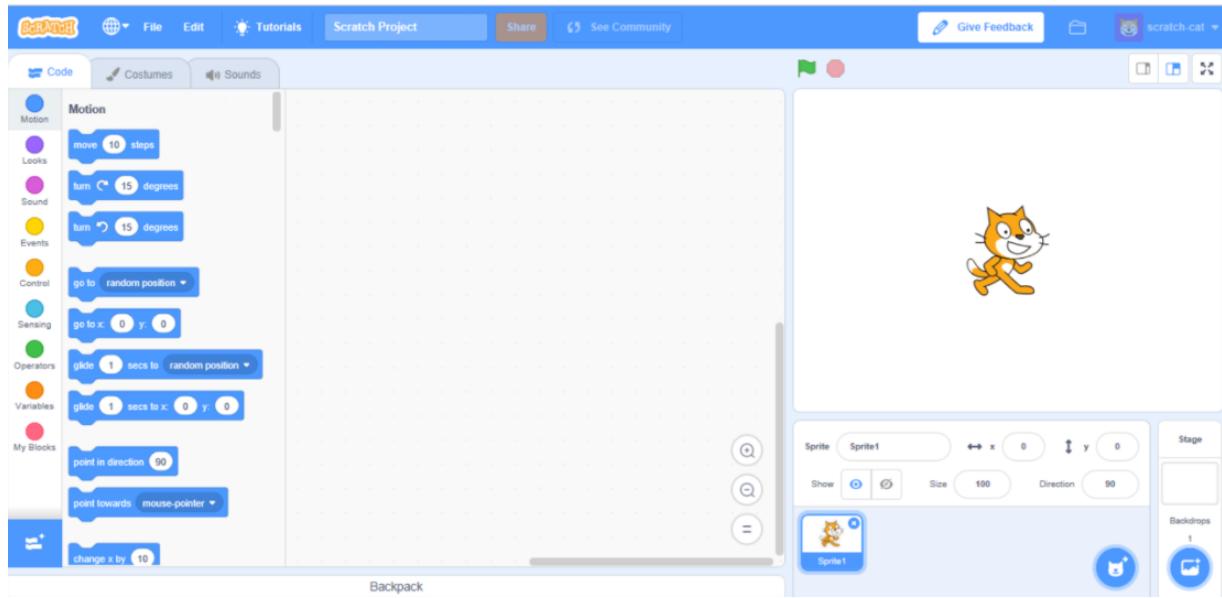


Figura 3.1: Interfaz de programación con *Scratch*

3.6. Blender

Blender es un programa informático multi plataforma, es decir, compatible para distintos sistemas operativos como Windows o Linux. Las funciones principales que se pueden realizar con *Blender* son el modelado, iluminación, renderizado, animación y creación de gráficos tridimensionales. También se pueden realizar actividades relacionadas con la composición de vídeo.

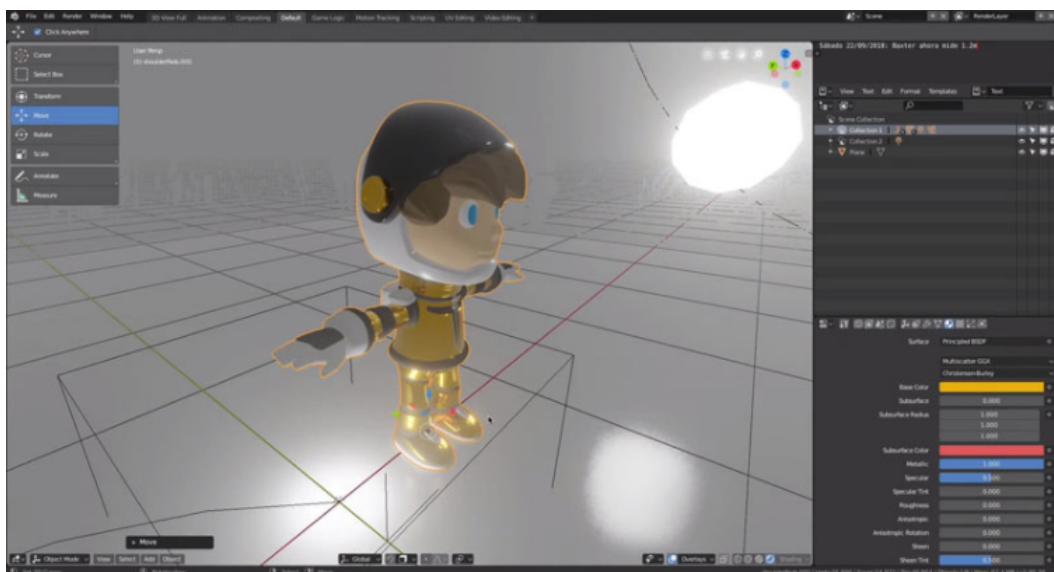


Figura 3.2: Interfaz de trabajo con *Blender*

Durante el presente trabajo se ha utilizado esta herramienta para modificar la rotación y apariencia de los robots de los que dispone la plataforma *Kibotics*. Además, *Blender* permite exportar los modelos en formato glTF (GL Transmission Format). glTF es un formato de archivo basado en el estandar *JSON*. Pemite la compresión de escenas y modelos 3D para minimizar el tiempo de ejecución de los programas en los que posteriormente se utilicen.

3.7. Simulador Websim

Websim es un simulador diseñado para el aprendizaje de conceptos básicos de programación de robots especialmente para niños. El simulador permite que los usuarios puedan programar fácilmente los movimientos de los robots, ya que simplemente tienen que acceder a la información que recogen sus sensores y enviar las órdenes precisas a los actuadores del robot. Estas órdenes se deberán programar, en *Python o Scratch*, dentro del editor que incorpora la interfaz de *Websim*.

El simulador está diseñado basándose en el uso del entorno *A-Frame*. A su vez, *A-Frame* se sirve del motor de físicas de *CANNON* para materializar los movimientos de los cuerpos dinámicos en la escena. A continuación, se explican con mayor detalle ambas tecnologías.

3.7.1. A-Frame

A-Frame es un marco web para crear escenas de realidad virtual. Sus principales características son las siguientes⁷:

- **Permite un uso sencillo de la realidad virtual:** para usar *A-Frame* basta con colocar las etiquetas `<script>`y `<a-scene>`. *A-Frame* se encarga del modelado 3D y la realidad virtual, no es necesaria la instalación de ningún paquete externo.
- **HTML declarativo:** *A-Frame* está basado en *HTML*, por ello es fácil y accesible para cualquier programador, puesto que *HTML* es un lenguaje ampliamente conocido.

⁷<https://aframe.io/docs/1.0.0/introduction/features>

- **Arquitectura de componente de entidad:** *A-Frame* sigue el patrón ECS (entidad-componente-sistema). Se trata de un patrón de desarrollo de juegos basado en el principio de composición sobre herencia. De esta manera, se otorga una mayor flexibilidad en la definición de entidades ya que cada objeto de la escena se corresponde con una entidad y cada entidad, a su vez, está compuesta por uno o más componentes que contienen datos y estado de la entidad. Por tanto, una entidad puede verse modificada en tiempo de ejecución si alguno de los componentes que agrega modifica sus datos.
- **Multiplataforma:** *A-Frame* es compatible con plataformas tan variadas como *Vive*, *Rift*, *Windows Mixed Reality*, *Daydream*, *GearVR* y *Cardboard*.
- **Rendimiento:** las actualizaciones de *A-Frame* se realizan en la memoria y con poco gasto energético. *A-Frame* se encuentra optimizado para *WebVR*.
- **Inspector visual:** *A-Frame* cuenta con un inspector visual integrado. Este se despliega al presionar la combinación de teclas `ctrl + alt + i`. El inspector permite detectar el origen de problemas o desarrollar una mejor distribución de la escena con menos esfuerzo.
- **Componentes:** *A-Frame* cuenta con una gran cantidad de componentes con los que trabajar. Esta amplia variedad va desde componentes geométricos básicos o materiales hasta componentes como la teletransportación, la realidad aumentada o componentes personalizados por el usuario.
- **Contribuciones con otras empresas:** *A-Frame* ha sido utilizado por empresas como Google, Disney, Samsung, Toyota o CERN, entre otras. Además, algunas de ellas, como Google, Microsoft, Oculus y Samsung han llegado a realizar contribuciones.

3.7.2. Sistema de físicas de *A-Frame*

Las físicas de *A-Frame* soportan dos motores de físicas: *Ammo Driver* y *CANNON*. Actualmente, el motor que está en uso es el de *CANNON*⁹. No obstante, ya ha sido añadido el soporte de *Ammo Driver* al sistema de físicas, ya que se preveé que *CANNON* acabe quedando obsoleto

⁸<https://aframe.io/>

⁹<https://github.com/donmccurdy/aframe-physics-system>

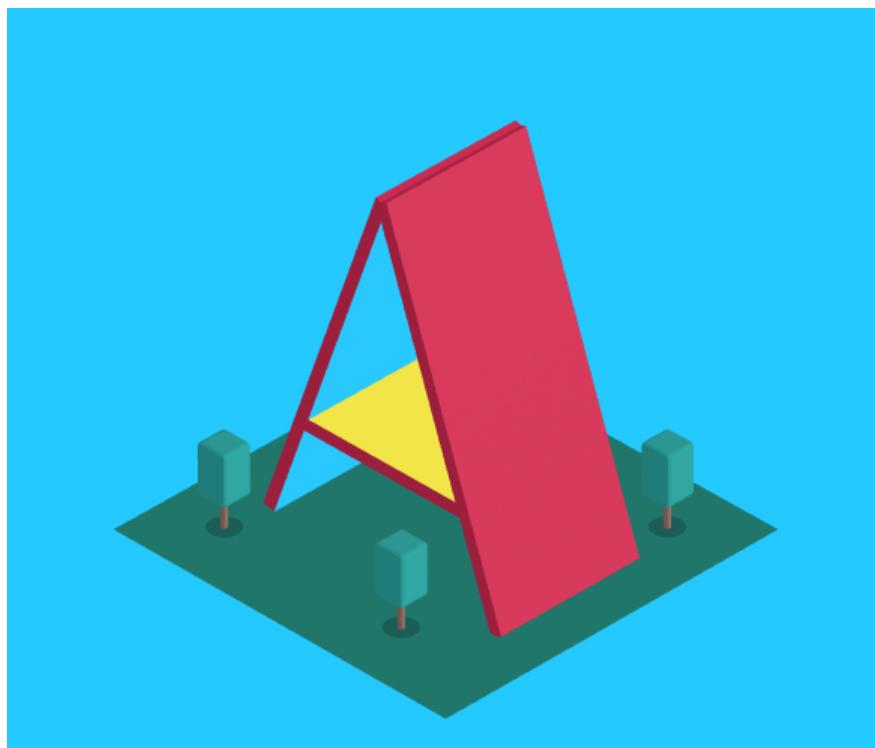


Figura 3.3: Ejemplo de construcción 3D con *A-Frame*⁸

con el paso del tiempo.

Para la instalación del sistema de físicas de *A-Frame* basta con incluir el siguiente script en el código *HTML* de la aplicación:

```
<script src="//cdn.rawgit.com/donmccurdy/aframe-physics-system/v4.0.1/dist/aframe-physics-system.min.js"></script>
```

El sistema de físicas de *A-Frame* cuenta con dos tipos de cuerpos: dinámicos y estáticos.

- **Cuerpo dinámico:** aquellos objetos de la escena que presentan libertad de movimiento. Estos objetos se ven afectados por la gravedad, la fricción y las colisiones.
- **Cuerpo estático:** aquellos objetos de la escena que no necesitan modificar su posición en la misma. Son cuerpos fijos y sin animaciones. Otros cuerpos dinámicos podrán colisionar con un cuerpo estático, pero el cuerpo estático no verá modificada su posición tras la colisión.

El sistema de físicas ofrece la posibilidad de añadir una malla de colisión a un objeto de la escena. Existen mallas de colisión de diferentes formas, por lo que se debe seleccionar aquella que se ajuste mejor al objeto en cuestión. Se puede elegir entre el ajuste automático, una caja, un cilindro, una esfera, un cuerpo convexo o una primitiva (plano, cilindro o esfera seleccionadas automáticamente en función de la primitiva *A-Frame* correspondiente).

Cada escena de *A-Frame* admite una serie de parámetros a los que se les puede modificar el valor para ajustar el mundo a las características deseadas. Si no se especifica el valor que se desea de un parámetro, este tomará el valor por defecto. Algunos de los parámetros que admite una escena son *debug*, que cuando está a true muestra las mallas de colisión de los objetos de la escena o *gravity*, *friction* y *restitution*, que se corresponden con la gravedad, fricción y coeficiente de restitución, respectivamente, del mundo simulado.

Atributo	Valor por defecto
debug	true
gravity	-9.8
friction	0.01
restitution	0.3

Cuadro 3.1: Parámetros configurables del sistema de físicas de *A-Frame*

Capítulo 4

Motor de físicas complementario

4.1. Estudio de las colisiones

4.1.1. Colisiones elásticas

4.1.2. Colisiones inelásticas

4.2. Estudio de la fricción

4.3. Físicas realistas

4.3.1. Mejoras en el movimiento en el eje vertical

4.3.2. Mejoras en el movimiento en el plano horizontal

Capítulo 5

Nuevos ejercicios

5.1. Roomba

5.2. Aparcamiento

5.3. Sigue-líneas sofisticado

5.4. Laberinto 3D para mBot

5.5. Laberinto para drone

5.5.1. Con señalización

5.5.2. Sin señalización

5.6. Fútbol competitivo

5.6.1. Evaluador

Capítulo 6

Conclusiones

6.1. Valoración de los resultados

6.2. Mejoras futuras

