UNIVERSIDAD
REY JUAN CARLOS

INGENIERÍA EN SISTEMAS AUDIOVISUALES Y MULTIMEDIA

Curso Académico 2018/2019

Trabajo Fin de Grado

WEBSIM

SIMULADOR DE ROBOTS CON TECNOLOGÍAS WEB VR

Autor : Álvaro Paniagua Tena

Tutor : Dr. Jose María Cañas Plaza

Trabajo Fin de Grado

Título del Trabajo con Letras Capitales para Sustantivos y Adjetivos

Autor : Álvaro Paniagua Tena

Tutor : Dr. Jose María Cañas Plaza

La defensa del presente Proyecto Fin de Carrera se realizó el día de de 20XX, siendo calificada por el siguiente tribunal:

Presidente:

Secretario:

Vocal:

y habiendo obtenido la siguiente calificación:

Calificación:

Fuenlabrada, a de de 20XX

Dedicado a

mis padres, familia, y a mi pareja Cristina

II

Agradecimientos

En primer lugar dar las gracias a mis padres por apoyarme y animarme desde el primer momento. También agradecer a mis compañeros Roberto, Ángel y Ahmed por tantas horas de ayuda y clases particulares para que entendiese todo bien.

Gracias tambien a Jose María Cañas por darme la oportunidad de colaborar en el proyecto.

Por último dar las gracias a mi pareja Cristina, has sido un gran apoyo en estos últimos años y sin duda me has motivado a hacer mejor las cosas y superarme.

Resumen

El proyecto tiene una intención clara, la creación de un simulador de robots usando tecnologías web puras aprovechando el crecimiento que ha experimentado el framework *AFRAME*.

Para la realización del proyecto se han utilizado diversas tecnologías como *HTML5*, *JavaScript*, *CSS3*, *AFRAME*, *jQuery* y *Blockly (Scratch 3)*.

El proyecto pretende ser una herramienta para la plataforma JdeRobot Kids en la cual los alumnos pueden generar código en el simulador **WebSim** para resolver una serie de ejercicios introductorios sobre **visión artificial** y **programación de robots** brindandoles varias opciones para dicha generación de código:

- Escribiendo código en JavaScript puro a través de un editor de código embebido.
- Utilizando el lenguaje de bloques visuales Scratch 3, este a su vez se puede probar en el propio simulador ya que se traduce a JavaScript nativo o bien exportarlo para probarlo en el robot físico, esto es posible gracias a la traducción de Scratch 3 a *Python*.

Summary

This project has a clear intention, create a robot simulator using web tecnologies taking advantage of the *AFRAME* framework growth.

For the ralization of the project, different tecnologies have been used, such as *HTML5*, *JavaScript*, *CSS3 AFRAME*, *jQuery* and*Blockly (Scratch 3)*.

The project aims to be a tool for the JdeRobot Kids platform on which students can generate code on WebSim simulator to solve basic exercises on **artificial vision** and **robot programming** bringing them different options to generate this code:

- Writing code on JavaScript lenguage through the embedded code editor inside the web page.
- Using the visual block language Scratch 3, this in turn can be tested inside the simulator itself as it is translated to JavaScript language or can be export the code to test it on the physical robot translating Scratch 3 to *Python* code.

Índice general

Índice de figuras

Capítulo 1

Introducción

Este Trabajo Fin de Grado tiene como objetivo la creación de una herramienta de simulación de robots para la plataforma JdeRobot Kids.

1.1. Contexto

El desarrollo tecnológico ha generado cambios a nivel social facilitandonos a las personas la mayoría de las tareas que llevamos a cabo a lo largo del día tanto dentro del trabajo como pueden ser largas cadenas automáticas de fabricación de productos como en el ámbito del hogar como es el caso del aspirador Roomba.

La tendencia actual en robótica es la de tener robots que tengan un nivel de funcionalidad igual al que pueda tener un humano, con esto nos referimos a moverse de la misma manera que un humano, detectar los distintos objetos que conforman nuestro entorno e incluso detectar a las distintas personas de una sala. Este nuevo paradigma acarrea la necesidad de formar a nuevos expertos en éste área lo que conlleva la formación desde edades más tempranas.

1.1.1. Motivación

La motivación de este proyecto es la de sustituir el presente simulador *Gazebo* por WebSim, un simulador desarrollado íntegramente con tecnologías Web con peso computacional en el lado cliente lo que permite escalar el número de usuarios que lo utilizan de manera simultánea.

1.1.2. Motivación personal

Como motivación personal decir que, desde el inicio del grado me ha gustado el mundo de la programación y el desarrollo web.

Además, me gusta poder ayudar al aprendizaje de los demás por tanto este proyecto creo que encaja a la perfección ambos aspectos.

1.2. Estructura de la memoria

En esta sección se detalla la esctura de la memoria que constará de las siguientes partes:

- El capítulo 1 es una introducción al proyecto donde se explica la motivación del proyecto y la motivación personal.
- En el capítulo 2 se muestran los objetivos a completar para la elaboración del proyecto y la estructura del **roadmap**.
- En el capítulo 3 se presentan las tecnólogias que se han utilizado para el desarrollo del proyecto y se explica el porqué de dichas tecnologías y no otras.
- En el capítulo 4 se explicarán el diseño de la aplicación, soporte del robot, es decir, qué funcionalidades tiene y la conectividad que dispone el simulador.
- En el capítulo 5 se muestran las aplicaciones reales en las que se encuentra actualmente WebSim.
- Finalmente en el capítulo 6 se hace una valoración de todo lo que ha conllevado el proyecto y se proponen futuras implementaciones o mejoras de WebSim.

Capítulo 2

Objetivos

2.1. Objetivo general

Mi trabajo fin de grado consiste en crear la base de una herramienta educativa para simulación de robots para la plataforma JdeRobot Kids en la cual el peso computacional la lleve el lado cliente en lugar del lado servidor.

2.2. Objetivos específicos

El objetivo específico de el proyecto es la simulación del robot **PiBot** implementando así todo su conjunto de sensores y actuadores en los que se encuentran:

- **Motores**: son dos servomotores independientes que dotan de movimiento al robot.
- **Cámara**: una minicámara, ésto le da funcionalidad muy importante al robot como puede ser la detección de obstáculos.
- **Sensores IR**: dos sensores infrarrojos posicionados en la parte baja del chasis del robot, estos sensores permiten la detección de colores, objetos y formas.
- **Sensor de ultrasonidos**: Dos sensores de ultrasonido posicionados en la parte delantera del chasis del robot, su funcionalidad es la de sensores de proximidad lo que permite saber no solo si hay un objeto delante sino que también permite saber a qué distancia está dicho objeto.

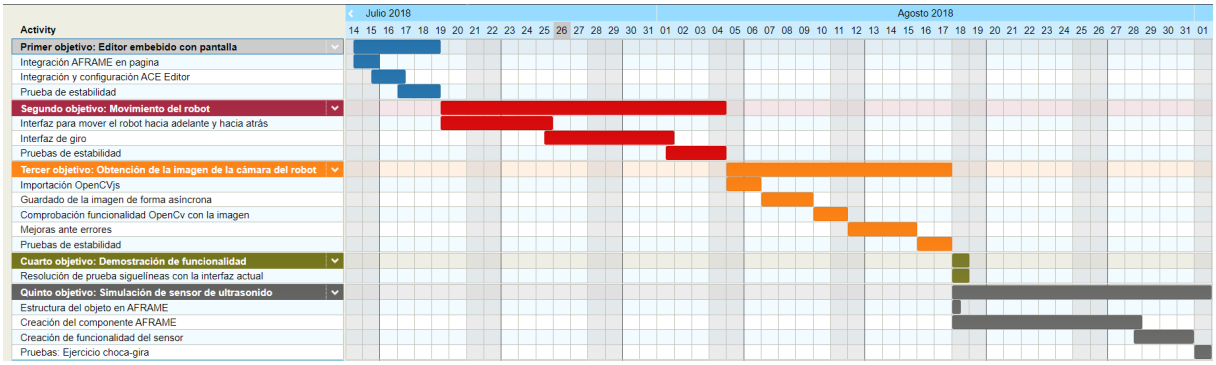

Figura 2.1: Diagrama de GANTT para el rango de fechas 14 de Julio de 2018 al 1 de Septiembre de 2018.

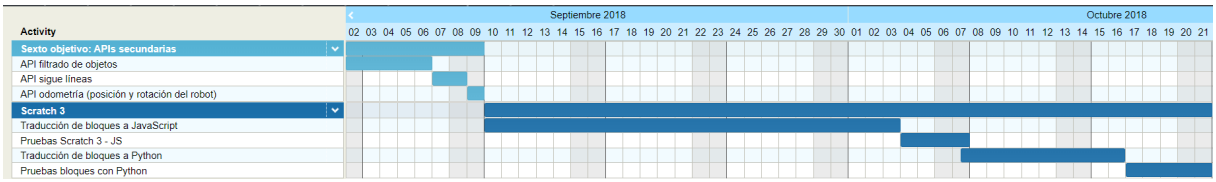

Figura 2.2: Diagrama de GANTT para el rango de fechas 2 de Septiembre de 2018 al 21 de Octubre de 2018.

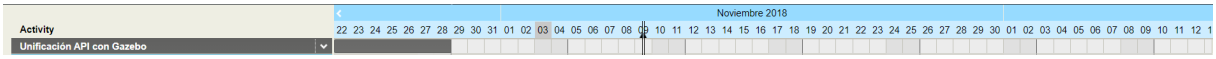

Figura 2.3: Diagrama de GANTT para el rango de fechas 22 de Octubre de 2018 al 28 de Octubre de 2018.

2.3. Planificación temporal

A continuación se muestran tres imágenes que representan la planificación temporal

El nivel de esfuerzo para este proyecto ha sido alto debido a la necesidad de aprender diferentes tecnologías como son AFRAME, jQuery y OpenCVjs. Se dedicaban alrededor de 3-4 horas al día cada día de la semana a excepción de los fines de semana que se añadían 2 horas más al anterior intervalo.

Capítulo 3

Estado del arte

3.1. ¿Por qué tecnologías WEB?

En la siguiente sección se tratarán de explicar las razones por las cuales se ha elegido desarrollar el proyecto con tecnologías web.

La principal y creo que más importante de las ventajas que ofrecen las tecnologías web es la ausencia de necesidad de instalación de paquetes de software, configuración y dependencias. Basta con tener un navegador, en este caso Firefox, y una conexión a internet, todos los archivos necesarios para la ejecución de la aplicación se sirven de manera automática al ingresar la URL dentro de la barra de navegación.

El cliente no tiene la necesidad de instalar actualizaciones ya que únicamente se actualiza la versión que proporciona el servidor, esto elimina las incompatibilidades entre versiones ya que todos los clientes usarán la misma versión. Desarrollo unificado, con esto hacemos referencia a que no se necesita desarrollar para los distintos sistemas operativos (Windows, MacOS, Linux/Ubuntu, etc.) así como conocer sus entornos gráficos y dependencias del sistema operativo, lo único necesario es saber HTML5, JavaScript y CSS3 , el navegador se encarga de interpretar los distintos lenguajes.

No todo son ventajas, como desventaja sabemos que las aplicaciones web son algo más lentas debido a la necesidad de descargar los recursos y no ser lenguajes binarios como C++ o Java sino interpretados, ésta desventaja cada vez va siendo menor debido a las mejoras en los navegadores y protocolos como por ejemplo *AJAX (Asynchronous JavaScript and XML requests* que trata de una técnica de peticiones ligeras para aplicaciones interactivas.

Además con los años ha aumentado el número de navegadores distintos y desarrollar la aplicación para todos ellos es costoso aunque existen frameworks que facilitan esta tarea como pueden ser Express y Loopback para el lenguaje JavaScript y Django para el lenguaje Python.

Como conclusión, las tecnologías web están avanzando cada vez más con el objetivo de llegar al rendimiento de las aplicaciones de escritorio hasta el punto de existir frameworks para el desarrollo de aplicaciones híbridas, es decir, desarrolladas con lenguaje web pero haciendo uso del sistema operativo como es el caso de *Electron*.

3.2. JavaScript

JavaScript fue creado por Brendan Eich en 1995 cuando trabajaba para Netscape Communications inspirado por el lenguaje Java.

JavaScript es un lenguaje con las siguientes características principales:

- Lenguaje de **alto nivel**, esto quiere decir que su sintaxis es similar a la escritura habitual de una persona, por ejemplo:

```
function myFunction(){
        console.log("Hello world");
}
```

- Lenguaje basado en **objetos**, esto es una estructura habitual en programación que se refiere a la encapsulación de operaciones y estados en un modelo de datos. Otros lenguajes orientados a objetos serían *Python, Ruby, Java, etc*. La figura 3.1 representa de manera visual la orientación a objetos.

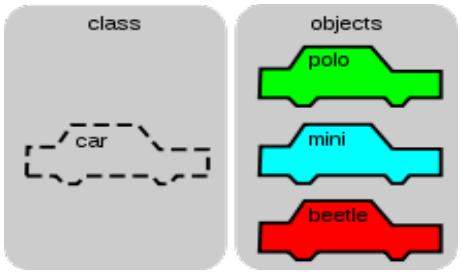

Figura 3.1: En la parte izquierda de la imagen se muestra el objeto **coche** del que heredan los 3 de la parte derecha de la imagen, los 3 coches tienen en común que son del objeto coche pero se diferencian en **marca** y **color.**

Es el lenguaje más utilizado para el desarrollo Web, permite que las aplicaciones web sean interactivas, es decir, permite hacer actualizaciones de contenido en el momento, mostrar mapas, animaciones 3D. Es el tercer pilar del estándar de tecnologías web compuesto por ***HTML, CSS y JS***.

Este lenguaje tiene mucho peso en el lado cliente, es decir, en el código que se ejecuta en el navegador pero gracias a la entrada de NodeJS cuando éste lenguaje ha obtenido más peso dentro del desarrollo web ya que NodeJS permite la creación de servidores de manera sencilla.

Actualmente **JavaScript** se utiliza en multitud de frameworks tanto de lado cliente como en el lado servidor, a continuación se enumeran algunos de los más usados:

- **NodeJS**: Tecnología en el lado servidor con código completamente JavaScript asíncrono y orientado a eventos, NodeJS fué diseñado para construir aplicaciones en red escalables. Su principal característica es la capacidad de gestionar multitud de conexiones simultáneas de una manera muy eficaz gracias a su arquitectura. Se basa en un hilo que escucha peticiones y las redirecciona a **hebras** distintas, cada una de estas hebras ejecuta el código necesario y una vez la hebra termina lanza un evento al hilo de peticiones indicando que ha terminado de ejecutar la tarea indicada entonces es el hilo de peticiones el que se encarga de devolver la respuesta.

- **Express y Loopback**: Ambos frameworks son extensiones de NodeJS y permiten la creación de un API en el lado servidor. Loopback es actualmente más importante que Express ya que ofrece muchas herramientas para la creación de API Rest de manera muy sencilla así como conectores a la mayoría de bases de datos como puede ser MongoDB, mySQL, sqlite3, etc.

- **AngularJS** Es un framework basado en el *Modelo Vista Controlador* para el desarrollo en la parte cliente que permite la creación de aplicaciones web **SPA** (*Simple-Page-Application*). AngularJS permite extender el lenguaje HTML con directivas y atributos sin perder la semántica.

- **AFRAME** Es un framework de creación de escenas de *Realidad Virtual*, se hablará de manera más extensa en el apartado 3.5. No es un framework muy extendido debido a su juventud pero cuenta con una gran comunidad de desarrolladores y es el pilar central del proyecto que se lleva a cabo.

Esta imagen creada de los frameworks disponibles para JavaScript sirve como respaldo ante la decisión de elegir el lenguaje. Además como se ha comentado en la subsección 1.1.1 el proyecto se orienta a la creación de una aplicación con peso en el lado cliente, por tanto sabiendo ésto y teniendo en cuenta que AFRAME está creado en el lenguaje JavaScript la elección de este lenguaje gana peso por sí sola.

3.3. HTML

HTML fue creado por *Tim Berners-Lee* en 1990 y es el acrónimo para *HyperText Markup Language* (Lenguaje de marcas de hipertexto).

Se utiliza para la creación de documentos electrónicos que se envían a través de la red global (internet). Cada documento tiene una serie de conexiones a otros documentos llamados ***hyperlinks*** que permiten la navegación entre distintos recursos.

HTML asegura el formato correcto de texto, imágenes y estilos para poder leer un documento con el navegador con la forma original con la que se generó el documento. En la figura 3.2 se muestra una página HTML muy simple y se explican sus distintas partes y su función.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset=utf-8 />
        <title> Mi primera pagina WEB</title>
    </head>
    <body>
      <h1> Hola Mundo! </h1>
      <p> Mi primera pagina HTML</p>
    </body>
</html>
```

Figura 3.2: Estructura de una pagina HTML simple con un título y un párrafo

Como vemos en la figura anterior,un documento HTML tiene una estructura de árbol donde la etiqueta **html** es el elemento raíz y cada nuevo elemento es una rama del anterior.

Como elementos principales del documento HTML tenemos la declaración documento **DOCTYPE html** que en este caso indica que estamos ante un documento HTML 5 que es última versión de HTML. Como hemos indicado en el párrafo anterior la etiqueta **html** marca la raíz del documento, dentro de ésta etiqueta tenemos dos etiquetas importantes:

- **HEAD** es la cabecera del documento, contiene los metadatos del documento como el título, la codificación de caracteres utilizada y links a otros recursos adicionales como pueden ser *scripts* y *hojas de estilos*.

- **BODY** es el contenido que se mostrará del documento, puede contener imágenes, enlaces a otros documentos, vídeos, menús de navegación, formularios, botones e incluso escenas animadas como la que se mostrará en el presente proyecto.

Las combinaciones de elementos son muy amplias, no existe una única estructura válida para un documento HTML sino que se genera una estructura en función de la aplicación.

Dentro del estándar HTML se ha escogido el estándar HTML5, a continuación se enumeran las nuevas características de éste nuevo estándar que tienen relación con el proyecto:

- **VIDEO**, esta etiqueta es una de las nuevas características de HTML5 y una de las más importantes, permite embeber vídeos dentro de una página web de manera nativa sin el uso de *plugins*.

- **NAV**, esta etiqueta declara un elemento de tipo *barra de navegación* en el cual se encuentra el menú con enlaces a otros tipos de recursos y secciones tanto dentro como fuera de la página. En nuestro caso esta etiqueta se ha utilizado para encapsular los botones de arranque/pare del código creado por el alumno y el botón que permite mostrar u ocultar la cámara del robot.

- **CANVAS**, permite la renderización de escenas gráficas a través de JavaScript. Es la etiqueta más importante dentro de nuestro proyecto ya que es la etiqueta que nos permite la creación de la escena en la cual tenemos nuestro robot simulado.

Existe una característica importante que afecta a todo el documento de HTML5 pero que, en general, no se está respetando en el desarrollo web y es que HTML5 tiene una tendencia semántica, es decir, las etiquetas como **SECTION, NAV y FOOTER** marcan claramente zonas dentro del documento HTML con el fin de poder conocer la estructura del documento de manera clara.

3.4. CSS

CSS o *Cascading StyleSheet* es un lenguaje que se usa para definir el aspecto visual de una página HTML. Su principal misión es la de separar la estructura y contenido del aspecto de la pagina HTML.

Con **CSS** podemos controlar incluso cómo se van a ver todos los documentos HTML de mi aplicación, es comúnmente utilizado por las empresas y diseñadores gráficos para crear de manera visual una identificación de la aplicación mediante tipos de letra, paleta de colores utilizada.

Deja atrás la gran necesidad de uso de JavaScript para fines de representación visual lo que hace que el rendimiento de la página se mejore al usar código JavaScript para otros fines. Además reduce la dependencia de software de edición gráfica como *Photoshop*, que sigue siendo utilizado pero su función es la edición más avanzada.

A continuación se muestra una imagen con una comparativa de la misma página web con y sin CSS.

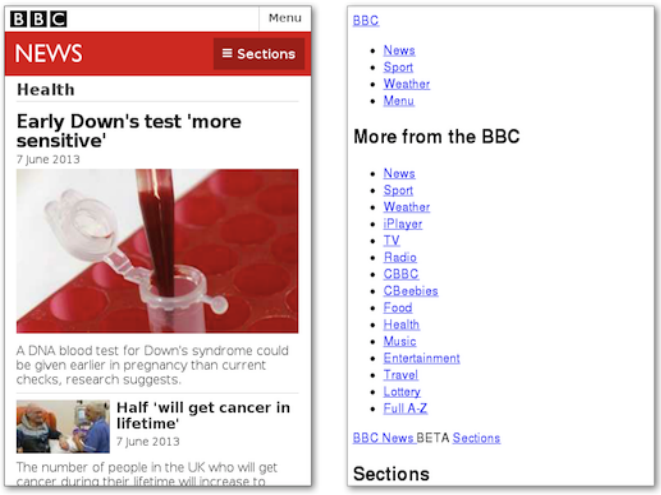

Figura 3.3: Imagen comparativa de la misma pagina web con y sin hojas CSS

Como vemos en la figura 3.3 las hojas de estilo CSS permite modificar completamente el aspecto de la pagina así como esconder menús y delimitar de manera visual las distintas partes de la página lo que permite mejorar la experiencia de usuario haciendo más accesible las partes importantes de la página.

Ésta última mención a la interfaz de usuario es importante ya que es uno de los puntos que más se cuidan en las empresas e incluso se hacen estudios para mejorar las interfaces , por ejemplo, debido al tamaño de los nuevos *smartphones* el menú de navegación ha cambiado su posición debido a que era difícil alcanzar la parte superior de la pantalla y por tanto se hacía molesto el uso de la aplicación.

3.5. AFRAME

AFRAME es un framework web para la construcción de escenas de realidad virtual, se creó con la intención de facilitar la creación de contenido de realidad virtual. Es un framework de código libre y tiene una de las comunidades de creadores de realidad virtual más grandes actualmente.

Soporta la mayoría de gafas de realidad virtual como *Vive, Rift, GearVR, etc.* además se puede usar no solo para realidad virtual sino para realidad aumentada. AFRAME fomenta la creación de escenas inmersivas completas de realidad virtual y va más allá de únicamente generar contenido en 360º, también implementa el uso de control de posición y controladores (mandos) que permiten interactuar con la escena, estos controles permiten al usuario tener una experiencia más inmersiva en la escena.

AFRAME además está soportado en los siguientes escenarios:

- Realidad virtual en aplicaciones de escritorio con *gadgets*.
- Realidad virtual en aplicaciones móviles con *gadgets*.
- Aplicaciones de escritorio convencionales.
- Aplicaciones de móvil convencionales.

A continuación se explican en subsecciones las características más importantes del framework AFRAME.

3.5.1. Primitivas y HTML

AFRAME está basado en HTML y el DOM *(Document Object Model)*, HTML es un lenguaje sencillo de leer y conocer la estructura, además no requiere de instalaciones únicamente se compone de texto y un navegador que muestre la página. AFRAME es compatible con la mayoría de frameworks que se utilizan actualmente en el desarrollo web como pueden ser Vue.js, React, AngularJS y jQuery.

Crear escenas de realidad virtual de manera muy simple, como se ve en la figura 3.4 únicamente se necesita una etiqueta **script** que haga referencia al código del framework y una etiqueta **a-scene** dentro del cuerpo del documento para crear una simple escena.

AFRAME ofrece un conjunto de elementos básicos para la escena llamados primitivas, estos elementos son figuras básicas como *cajas, esferas, cilindros, planos, cielo, etc.* AFRAME no solo ofrece figuras básicas, como hemos comentado anteriormente su intención es la de crear escenas inmersivas completas, por ello ofrece además etiquetas para la inyección de sonidos y vídeos dentro de la escena.

Este tipo de primitivas son bastante útiles para su uso en escenas simples, todas ellas heredan de la primitiva **a-entity** que, equiparandolo con HTML convencional, equivaldría con la etiqueta **div** del estándar HTML, la cual se utiliza de muchas maneras distintas.

Esta etiqueta **a-entity** representa por tanto el punto de partida de cualquier tipo de elemento de la escena que queramos crear al cual se le irán añadiendo *componentes* que le dotarán de cierta funcionalidad específica.

AFRAME permite además crear nuestras propias primitivas lo que nos permite seguir el principio de programación *DRY (Don't Repeat Yourself)* por el cual si tenemos un complicado elemento en la escena que está compuesto de varias entidades distintas no tenemos que copiar ese código *N* veces sino que podemos registrar la primitiva y hacer referencia a este elemento mediante el nombre de etiqueta que más convenga.

La figura 3.4 muestra todo el código necesario para crear la escena mostrada.

```
<html>
  <head>
    <script src="https://aframe.io/releases/0.8.0/aframe.min.js"></script>
  </head>
  <body>
    <a-scene>
      <a-box position="-1 0.5 -3" rotation="0 45 0" color="#4CC3D9"></a-box>
      <a-sphere position="0 1.25 -5" radius="1.25" color="#EF2D5E"></a-sphere>
      <a-cylinder position="1 0.75 -3" radius="0.5" height="1.5" color="#FFC65D"></a-cylinder>
      <a-plane position="0 0 -4" rotation="-90 0 0" width="4" height="4" color="#7BC8A4"></a-plane>
      <a-sky color="#ECECEC"></a-sky>
    </a-scene>
  </body>
</html>
```

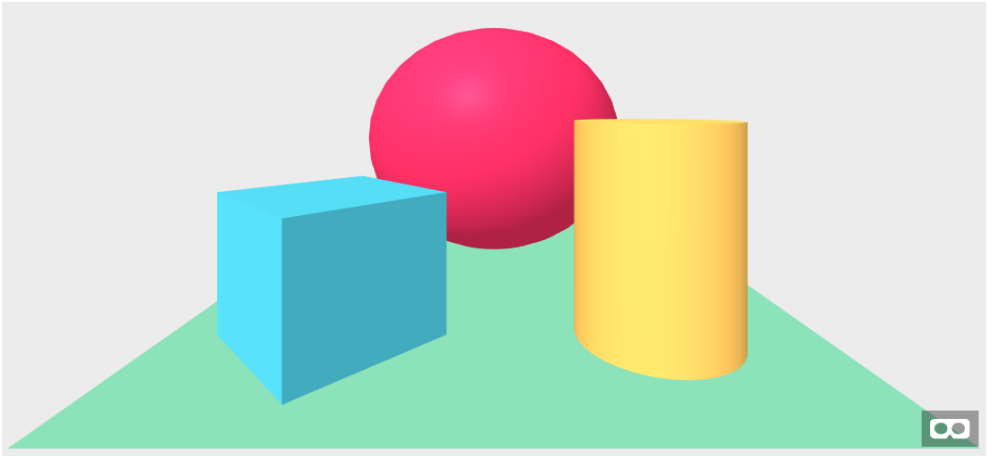

Figura 3.4: Ejemplo de código HTML que renderiza una escena básica de realidad virtual

3.5.2. Sistema Entidad-Componente

AFRAME se basa en el framework ***three.js*** y provee una estructura reutilizable de entidad-componente en la que un componente puede ser utilizado en distintas entidades de distinta clase. Se pueden generar componentes personalizados y vincularlos a cualquier tipo de entidad dándole una funcionalidad distinta. Esto permite una gran flexibilidad a la hora de generar distintos integrantes en la escena con funcionalidades diferentes pero heredando todos de una misma entidad.

La arquitectura entidad-componente es común en el desarrollo 3D y en el desarrollo de videojuegos y sigue el principio de composición por herencia. Los beneficios de este tipo de arquitectura son:

- Gran flexibilidad a la hora de crear objetos debido a la reutilización de componentes y el mezclado de distintos componentes.
- Elimina el problema de largas cadenas de herencia, cada componente es independiente.
- Diseño limpio gracias al desarrollo por módulos.

Figura 3.5: La siguiente figura muestra etiquetas que serían el equivalente en AFRAME a un componente, el conjunto de varios componentes dan forma a la caja

- Es la manera más escalable de generar complejas escenas de realidad virtual.
- Permite reutilizar y compartir componentes no solo en un mismo proyecto sino con la comunidad de desarrolladores.

En la figura 3.5 se muestra un esquema del *Sistema Entidad-Componente* en el cual tenemos una figura final con forma de caja la cual estaría compuesta de varios componentes distintos. Estos componentes son: **Posición, Geometría, Material y Color**.

A continuación se mostrará la API *(Application Program Interface)* que ofrece AFRAME para implementar el **Sistema Entidad-Componente**:

- **Entidad:** se representa en AFRAME mediante la etiqueta **a-entity**.
- **Componentes:** se representa en AFRAME como atributos de la etiqueta HTML. Estos componentes son objetos que contienen un esquema, manejadores y métodos. Éstos se registran mediante el método **AFRAME.registerComponent(nombre, definición);**. A continuación se muestra un componente que imprime un mensaje en la consola del navegador.

```
AFRAME.registerComponent('log', {
  schema: {
    message: {type: 'string', default: 'Hello, World!'}
  },
  init: function(){
    console.log(this.data.message);
  }
});
```

Figura 3.6: Componente básico que imprime por la consola del navegador el mensaje que se le pase por *message*

- **Sistema:** representado por la escena mediante la etiqueta **a-scene**. Los sistemas son similares a los componentes a la hora de definirlos, se registran mediante **AFRAME.registerSystem(nombre definición)**.

Además como hemos comentado AFRAME tiene dos tipos de implementaciones que le dan características adicionales al sistema entidad-componente y es que al implementarse sobre HTML y JavaScript tenemos dos características importantes:

- Referenciar una entidad mediante el método **querySelector** implementado en JavaScript lo que permite acceder a una entidad por su ID, clase o atributos. En la figura 3.7 se muestra un ejemplo de código JavaScript que accede a un elemento de AFRAME con ID *rightHand*.

```
var rightHandElement = document.querySelector("#rightHand");
```

Figura 3.7: Ejemplo de acceso a un elemento de AFRAME, como se aprecia se hace de la misma manera que se accede a cualquier elemento de HTML convencional

- Comunicación entre las entidades mediante eventos, esta característica es hereditaria del lenguaje utilizado lo que permite registrar y suscribir eventos que permite que los elementos en la escena no se conozcan entre sí.

- Crear, eliminar y modificar atributos mediante el API del DOM, podemos utilizar los métodos **.setAttribute, .removeAttribute, .createElement y .removeChild** para modificar los elementos.

Por último pero no más importante, los componentes pueden hacer cualquier cosa, tienen acceso completo a **three.js, JavaScript y APIs Web** como pueden ser *WebRTC, AJAX, etc.*

3.5.3. Modelos 3-D

AFRAME ofrece la posibilidad de cargar modelos 3D más sofisticados en los formatos *glTF, OBJ, COLLADA*. Se recomienda el uso del formato *glTF* ya que es el modelo estándar para transmitir modelos 3D en la WEB. Los componentes se pueden escribir de manera que se pueda manejar cualquier tipo de formato que tenga un objeto en *three.js* para cargar el modelo.

Los modelos son archivos en texto plano y contiene vértices, caras, texturas, materiales y animaciones.

Como se ha repetido en varias ocasiones se trata de crear escenas, para ello es necesario implementar animaciones. Estas animaciones se implementan con el paquete de componentes creado por **Don McCurdy**, se puede localizar en `https://github.com/donmccurdy/aframe-extras/blob/master/src/loaders/animation-mixer.js`.

3.5.4. Herramientas de Desarrollo

AFRAME como hemos comentado se construye sobre JavaScript y HTML por tanto utiliza las mismas herramientas de desarrollo ya disponibles dentro del navegador. Además al crear escenas 3D se hace complicado depurar la escena y saber que todo está siendo representado en su posición correcta, para esta problemática AFRAME incluye un inspector visual que permite conocer la posición y los valores de los atributos para cada entidad de la escena. La figura 3.4 muestra la escena de nuestro simulador y los atributos de la cámara incluida dentro del robot, como se ve el inspector muestra el ángulo de visión de la cámara del robot y muestra los ejes de la escena respectivo al punto en el que se encuentra de la cámara.

Ofrece una representación en árbol de la escena siguiendo la estructura del documento HTML, el inspector ofrece la posibilidad de mover, rotar, añadir y borrar elementos de la escena así como copiar la etiqueta una vez movida para usarla en nuestro documento HTML. Un ejemplo de esto sería mover nuestro robot y orientarlo de cara a la pelota verde, sin el inspector tendríamos que ir haciendo pruebas modificando manualmente el atributo de la posición dentro del documento HTML pero con el inspector visual basta usar las herramientas para mover el elemento y copiar las coordenadas que aparecen en la ventana de la derecha.

Por último el inspector permite hacer capturas de movimiento lo que permite:

- Test más rápidos, no se necesitan utilizar los *gadgets* cada vez que se quiera hacer un test lo que acelera mucho el desarrollo de la aplicación.
- Múltiples desarrolladores pueden utilizar el mismo *gadget*, puedes grabar el movimiento y dejar de usar el *gadget* para que otros desarrolladores del mismo proyecto puedan usarlo.

- Mostrar errores del código.

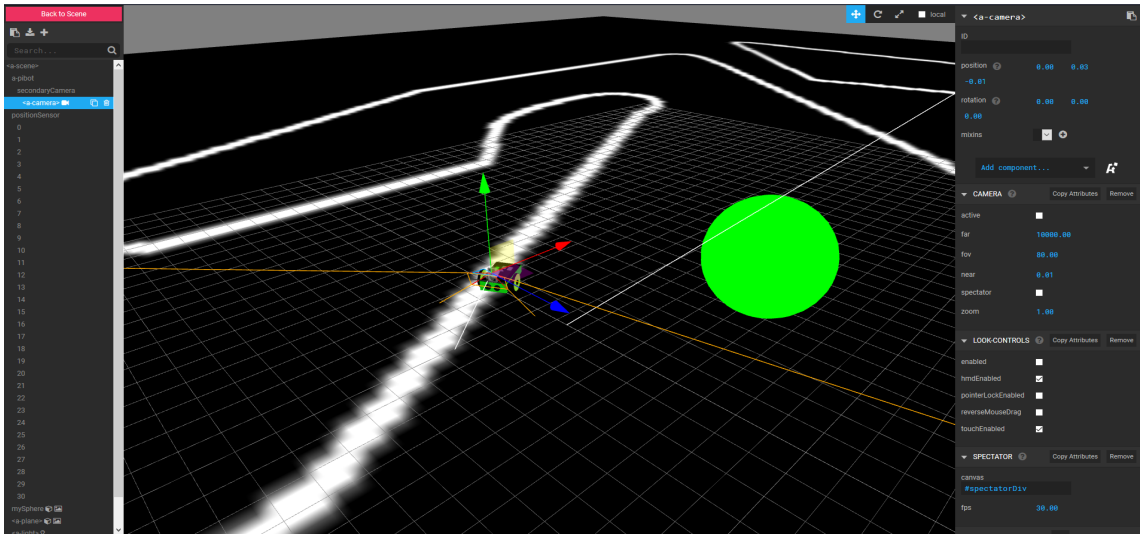

Figura 3.8: Inspector visual que ofrece el framework *AFRAME*

3.6. ACE Editor

ACE Editor es un editor de código embebido creado en *JavaScript*, implementa las características de los editores nativos como *Sublime Text, Vim, etc. ACE* es el editor usado en el servicio *AWS Cloud9 IDE*.

El editor ofrece la siguiente funcionalidad:

- Mantener el estado de la sesión como por ejemplo el *scroll*, selección de texto, etc.
- Resaltado de sintaxis para la mayoría de lenguajes de programación como *JavaScript, CSS , Python, Java, etc.*
- Permite crear tus propias reglas de resaltado.
- Indentación automática del código.
- Manejo de ficheros grandes, maneja miles de líneas sin problema.
- Resaltado de paréntesis.
- *Drag and Drop* de texto dentro del editor de código.
- Comprobación de sintaxis del lenguaje, esta característica es bastante útil ya que nos permite descartar errores en ejecución debido a la sintaxis del programa lo que acelera el desarrollo.

La característica más importante para nuestro proyecto es sin duda la facilidad para embeberlo dentro de nuestra aplicación, simplemente hace falta una etiqueta **script** y un pequeño código para configurar su carga en la página, el detalle del código se mostrará en el capítulo 4. El editor además provee de una sencilla *API* para poder obtener el código escrito en el editor a través de nuestro programa para poder manejarlo.

En la figura a continuación se muestra las posibles configuraciones que puede adoptar el editor.

El editor utiliza el DOM *(Document Object Model)* para el renderizado, concretamente la etiqueta **canvas** y no depende de librerías externas. La característica más destacable de el editor es que no es necesario instalar nada tu ordenador, el editor reside completamente en la página web lo que permite la creación de aplicaciones interactivas como la del presente proyecto en la

```
selectionStyle: "line"|"text"
highlightActiveLine: true|false
highlightSelectedWord: true|false
readOnly: true|false
cursorStyle: "ace"|"slim"|"smooth"|"wide"
mergeUndoDeltas: false|true|"always"
behavioursEnabled: boolean
wrapBehavioursEnabled: boolean
// this is needed if editor is inside scrollable page
autoScrollEditorIntoView: boolean (defaults to false)
// copy/cut the full line if selection is empty, defaults to false
copyWithEmptySelection: boolean
useSoftTabs: boolean (defaults to false)
navigateWithinSoftTabs: boolean (defaults to false)
enableMultiselect: boolean   # on by default
```

Figura 3.9: Posibles parámetros de configuración para el editor

cual podemos programar el robot 'en vivo' sin la necesidad de tener que exportar el código de nuestro editor para hacer pruebas en la simulación.

3.7. Blockly

Blockly es una librería que permite aprender programación mediante el uso de bloques visuales que se pueden combinar y traducir posteriormente a distintos lenguajes. Permite a los usuarios programar en distintos lenguajes sin tener que conocer la sintaxis del lenguaje al detalle.

Es una forma de aprender a programar orientada a estudiantes de temprana edad, el objetivo es que el alumno pueda aprender la lógica de los algoritmos de programación pero sin tener que aprender todo el lenguaje como puede ser declaración de variables, tipo y en general la sintaxis propia del lenguaje que según el tipo de lenguaje puede ser pesado al principio.

Blockly es la base de otros entornos de programación visual como ***Scratch 3*** ya que es código libre lo que permite a cualquier desarrollador contribuir y utilizarlo en su propia aplicación. Ofrece métodos para importar y exportar el código de bloques además de métodos para modificar y personalizar el aspecto de los bloques para que sigan el estilo de la interfaz de la aplicación.

Es una tecnología orientada completamente al lado cliente y se puede utilizar mediante el fichero comprimido del repositorio oficial de *Blockly* llamado *blockly-compressed*, no utiliza dependencias y como ya hemos comentado es código libre.

3.7.1. Generadores de código

Blockly como hemos comentado provee de generadores de código para los siguientes lenguajes:

- JavaScript.
- Python.
- PHP.
- Lua.
- Dart

Estos generadores proveen las herramientas básicas para crear funciones, expresiones lógicas, bucles, etc. La problemática de esto es que a veces nuestras aplicaciones necesitan usar la API de otras dependencias como en nuestro caso, para solventar esto *Blockly* permite generar bloques personalizados que se traducirán a la instrucción necesaria dotando de mucha flexibilidad al entorno.

Además *Blockly* permite añadir palabras reservadas dentro de cada tipo de lenguaje lo cual nos permite un control de colisiones con las variables propias de la aplicación ya que en lugar de eliminar esta variable lo que hace *Blockly* es renombrar todas las apariciones de la variable en el código generado dinámicamente.

Se puede apreciar que el código generado a través de los bloques será correcto en su sintaxis pero hay una problemática presente en programación y es la aparición de *bucles infinitos*, la librería dota de un método para intentar aplacar esta problemática, en la figura 3.10 se muestra un simple código que cuenta el número de iteraciones del bucle, no es el mejor método para solventar el problema ya que, como es nuestro caso, necesitaremos bucles de ejecución continua pero dota a la aplicación de control de ejecución de código.

```
window.LoopTrap = 1000;
Blockly.JavaScript.INFINITE_LOOP_TRAP = 'if(--window.LoopTrap == 0) throw "Infinite loop.";\n';
var code = Blockly.JavaScript.workspaceToCode(workspace);
```

Figura 3.10: Implementación de un contador para evitar bucles infinitos al traducir lenguaje de *Blockly*

3.7.2. Bloques personalizados

Como se ha hecho referencia en la sección anterior, *Blockly* permite además de generar código en distintos lenguajes también crear bloques personalizados lo que permite generar código a través de bloques y conectarlos con cualquier tipo de API.

Para generar bloques personalizados *Blockly* hemos de configurar varios aspectos:

- Configuración de los parámetros de entrada y salida del bloque, conectores o parámetros

en línea y color. La configuración del bloque se permite mediante dos formas distintas a través de un JSON o mediante JavaScript registrando un nuevo bloque en el objeto **Blockly**. La figura 3.11 muestra ambos métodos para generar el mismo bloque.

```
{
  "type": "string_length",
  "message0": 'length of %1',
  "args0": [
    {
      "type": "input_value",
      "name": "VALUE",
      "check": "String"
    }
  ],
  "output": "Number",
  "colour": 160,
  "tooltip": "Returns number of letters in the provided text.",
  "helpUrl": "http://www.w3schools.com/jsref/jsref_length_string.asp"
}
```

```
Blockly.Blocks['string_length'] = {
  init: function() {
    this.appendValueInput('VALUE')
        .setCheck('String')
        .appendField('length of');
    this.setOutput(true, 'Number');
    this.setColour(160);
    this.setTooltip('Returns number of letters in the provided text.');
    this.setHelpUrl('http://www.w3schools.com/jsref/jsref_length_string.asp');
  }
};
```

Figura 3.11: Modos de configuración de un bloque personalizado en *Blockly*, como se puede apreciar la declaración de las distintas partes del bloque es bastante intuitiva y los parámetros son autodescriptivos.

- Configuración de la traducción del bloque a la instrucción de interés en los distintos lenguajes necesarios.

- Iniciar el bloque para que se renderice en el editor de bloques visual. La figura 3.12

```
Blockly.Blocks['move_combined'] = {
  init: function() {
    this.jsonInit(moveBlock);

  }
};
```

Figura 3.12: Muestra del código que inicia el bloque para que se muestre en el editor visual, *moveBlock* representa un objeto JSON con la configuración del bloque

Como se ve aunque los parámetros del JSON son autodescriptivos y sencillos de entender es complicado y lento generar un bloque desde cero por tanto *Google* ofrece unas herramientas para desarrolladores en línea lo que permite acelerar esta generación de código.

En la figura 3.13 se muestra una imagen del entorno de creación de bloques personalizados que provee *Blockly* en el cual se puede observar en la parte de la derecha el archivo en formato *JSON (JavaScript Object Notation)* de configuración del bloque en el que se definen las entradas que toma, el color con el que se mostrará entre otra serie de parámetros. Además, muestra la función con la configuración básica para obtener los parámetros que se utilizaran para generar código real JavaScript.

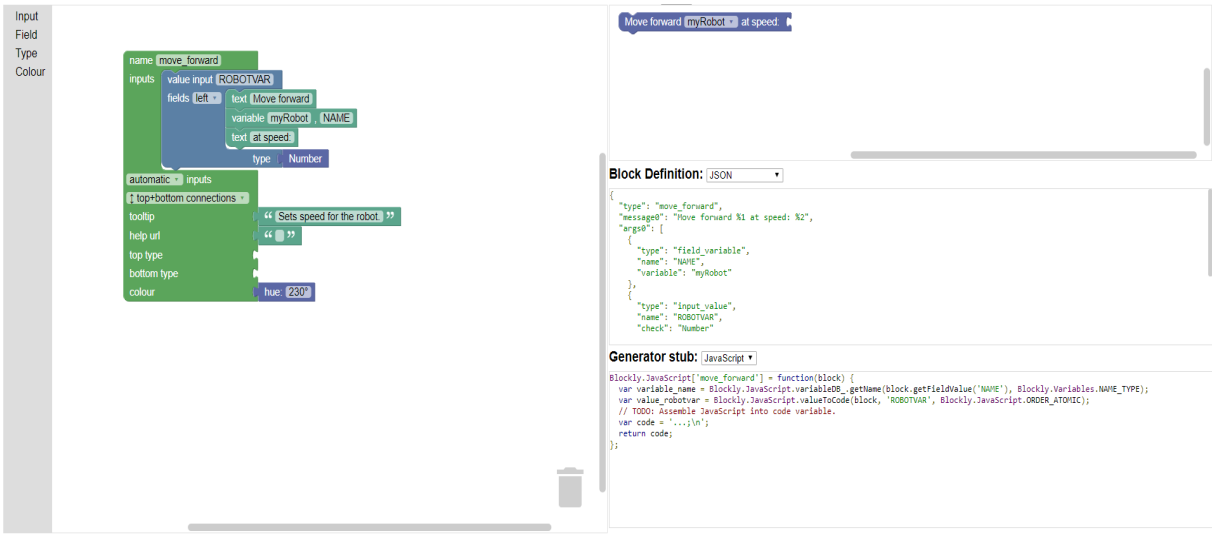

Figura 3.13: Herramienta de desarrollo para generación de bloques personalizados.

Como se ve en la figura 3.13 en la parte superior derecha la herramienta te muestra cómo se verá el bloque final en tu aplicación bajo la configuración visual por defecto que trae *Blockly*

3.7.3. Menú de bloques, *Toolbox*

El editor de *Blockly* provee además de una barra de herramientas en la cual se muestran los bloques que podrán ser usados a través del editor, estos bloques se configuran a través de un fichero XML en el cual podemos tener distintos tipos de etiquetas que se muestran a continuación.

Como vemos en la figura 3.14 tenemos tres bloques distintos, la raíz del XML marcada

```
<xml id="toolbox" style="display: none">
  <category name="Variables" custom="VARIABLE"></category>
  <category name="Text" colour="%{BKY_MATH_HUE}">
    <block type="text"></block>
```

Figura 3.14: Ejemplo de las etiquetas posibles de configuración de la *toolbox* del editor

por la etiqueta *xml*, la etiqueta *category* que nos permite hacer divisiones de bloques por tipos distintos, en la imagen tenemos dos categorías *variables y texto*, la categoría *variables* como vemos no tiene bloques dentro ya que se generan dinámicamente. En la categoría *texto* vemos declarado un bloque de tipo *text* que representaría un *String* en lenguaje JavaScript. Como se puede apreciar la configuración del menú de bloques no es fija, se puede crear distintas categorías en función del diseño de la interfaz de usuario.

3.8. jQuery

Capítulo 4

Diseño e implementación

Aquí viene todo lo que has hecho tú (tecnológicamente). Puedes entrar hasta el detalle. Es la parte más importante de la memoria, porque describe lo que has hecho tú. Eso sí, normalmente aconsejo no poner código, sino diagramas.

4.1. Arquitectura

Si tu proyecto es un software, siempre es bueno poner la arquitectura (que es cómo se estructura tu programa a "vista de pájaro").

Por ejemplo, puedes verlo en la figura 4.1.

Si utilizas una base de datos, no te olvides de incluir también un diagrama de entidad-relación.

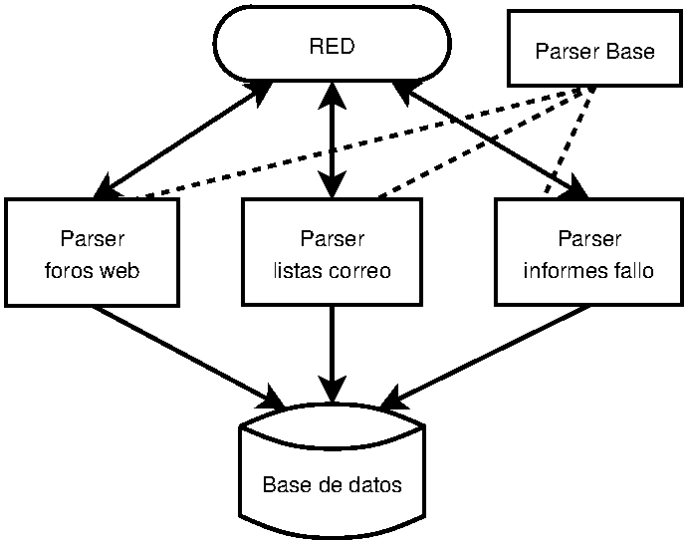

Figura 4.1: Estructura del parser básico

4.2. Objeto Robot

4.2.1. Motores

4.2.2. Sensores

4.2.3. Cámara

4.3. Interfaz

4.4. Funcionalidad del servidor

Capítulo 5

Usos del simulador

5.1. Ejercicios JdeRobot Kids

5.2. Robot real

Capítulo 6

Conclusiones

Esta sección es la sección espejo de las dos primeras del capítulo de objetivos, donde se planteaba el objetivo general y se elaboraban los específicos.

Es aquí donde hay que debatir qué se ha conseguido y qué no. Cuando algo no se ha conseguido, se ha de justificar, en términos de qué problemas se han encontrado y qué medidas se han tomado para mitigar esos problemas.

6.1. Valoración objetivo final

6.2. Aplicación de lo aprendido

Aquí viene lo que has aprendido durante el Grado/Máster y que has aplicado en el TFG/TFM. Una buena idea es poner las asignaturas más relacionadas y comentar en un párrafo los conocimientos y habilidades puestos en práctica.

1. a
2. b

6.3. Mejoras futuras

Ningún software se termina, así que aquí vienen ideas y funcionalidades que estaría bien tener implementadas en el futuro.

Es un apartado que sirve para dar ideas de cara a futuros TFGs/TFMs.

6.4. Autovaloración del proyecto

Aquí viene lo que has aprendido en el Trabajo Fin de Grado/Máster.

1. a
2. b

Apéndice A

Manual de usuario

Apéndice B

Enlaces a tutoriales de uso

Bibliografía