UNIVERSIDAD
REY JUAN CARLOS

GRADO EN INGENIERÍA TELEMÁTICA

Curso Académico 2019/2020

Trabajo Fin de Grado

Mejoras en entorno de robótica educativa para niños

Autor : Rubén Álvarez Martín

Tutor : Dr. José María Cañas Plaza

Índice general

Capítulo 1

Introducción

El principal objetivo del proyecto es mejorar la plataforma Websim para disponer de más robots y ejercicios para incrementar su valor educativo dando más posibilidades de aprendizaje.

1.1. Tecnologías web

Las tecnologías web han ido evolucionando a los largo de la historia, pero todas se basan en un modelo cliente-servidor. Para lograr la comunicación entre cliente y servidor, se necesita un navegador por parte del cliente y un servidor web capaz de atender las solicitudes. El protocolo que es utilizado para esta comunicación se llama HTTP.

1.1.1. HTTP

1.1.2. Tecnologías en servidor

1.1.3. Tecnologías en cliente

1.2. Robótica

La robótica es una rama tecnológica encargada del diseño y construcción de aparatos que realizan operaciones y trabajos en sustitución de la mano de obra humana.

Un robot es un sistema autónomo programable capaz de realizar tareas de ayuda al ser humano y con aplicaciones en campos diversos como medicina, hogar, fábricas, etc.

Los robots se componen de sensores, controladores y actuadores:

- Sensores: son los encargados de recoger información del entorno. En este grupo se encuentran láser, cámaras, ultrasonidos u odómetros. Estos dispositivos equivalen a los sentidos humanos.

- Controladores: analizan los datos recogidos por los sensores y elaboran una respuesta que va a ser enviada a los actuadores. En los seres humanos equivale al cerebro.

- Actuadores: se encargan de transformar energía eléctrica, hidráulica o neumática en mecánica. Son los interactúan con el entorno y equivalen a los músculos humano

1.2.1. Historia

La palabra robot fue empleada por primera vez en 1920 por Karel Capek, un escritor checo que realiza una obra de teatro en la que un hombre fabrica un robot.

Años más tarde, en 1950, Isaac Asimov publica un libro llamado Yo, Robot en el que acuña la palabra robótica definiendo a la ciencia que estudia a los robots. En el también se incluyen tres leyes de la robótica:

- Un robot no hará daño a un ser humano ni permitirá que un ser humano sufra ningún tipo daño.

- Un robot debe cumplir las órdenes recibidas por un ser humano, salvo las que entren en conflicto con la primera ley.

- Un robot debe proteger su propia existencia mientras que no entre en conflicto con ninguna de las leyes anteriores.

Es a finales de la década de los 50 cuando la robótica sufre el mayor impulso al crearse el primer robot comercial en 1956 y, años más tarde, en 1961, se instala el primer robot industrial (Figura 1.1).

Figura 1.1: Unimate, primer robot industrial

En los años 70 se crea el primer robot autónomo, Shakey (Figura 1.2). Fue el primero que incorporaba inteligencia artificial y disponía de control de motores y sensores. Shakey era capaz de recoger toda la información de su entorno, crear un mapa y diseñar el camino más corto desde su ubicación al destino.

Figura 1.2: Shakey, primer robot autónomo

En 2000 la compañía Honda lanza el robot Asimo (Figura 1.3), un robot humanoide capaz de detectar múltiples objetos, reconocer caras e interpretar comandos de voz o

gestos.Asimo fue creado con el objetivo de, algún día, ofrecer asistencia a personas con necesidades especiales.

Figura 1.3: Asimo, primer robot humanoide

Es en 2011 cuando la robótica alcanza uno de sus mayores hitos gracias al robot explorador Curiosity (Figura 1.4). Este robot llego a Marte en 2012 y su cometido fue investigar el planeta para determinar si existió vida en Marte, caracterizar su clima y geología y preparar el entorno para su exploración humana.

Figura 1.4: Autorretrato del Curiosity en Marte

Hoy en día no solo se ven robots en entornos industriales, si no que aparecen cada vez más en entornos domésticos, educativos o automovilístico.

1.2.2. Tipos de robots

1.3. Robótica educativa

Capítulo 2

Objetivos

2.1. Objetivos

2.2. Metodología

Capítulo 3

Herramientas

En este capítulo se van a detallar las herramientas utilizadas en el desarrollo de este proyecto. Algunas se han elegido por facilidad de uso y otras por necesidad del entorno desarrollado.

3.1. JavaScript

JavaScript es un lenguaje interpretado de alto nivel que se encuentra bajo el estándar ECMAScript[1] y está basado en otros lenguajes de programación como Java o C.

En su principio fue concebido como lenguaje para el lado cliente, implementado en un navegador web, permitiendo mejorar la interfaz de usuario y realizar páginas web dinámicas.

En la actualidad, JavaScript se ha ido extendiendo hacia el lado servidor y es por ello que es el lenguaje más utilizado para desarrollo web y todos los navegadores interpretan el código integrado en las páginas web.

3.1.1. Características

Las siguientes características tienen en común que todas se ajustan al estándar de ECMAScript:

[1]Especificación de lenguaje de programación el cual define tipos dinámicos y soporta características de programación orientada a objetos.

- Es un lenguaje estructurado, tiene gran similitud con C y comparte gran parte de su estructura (bucles, condicionales, sentencias...) a excepción del alcance de sus variables. En C su ámbito es el bloque en el que fue definida y, en su origen, JavaScript tenía un alcance global en las variables definidas. Es en ECMAScript 2015 cuando se añade la palabra clave let, que incorpora compatibilidad con block scoping (alcance de la variable en el bloque en la que es definida).

- Tipado débil, por el cual el tipo de datos está asociado al valor, no a la variable. Esto significa que una variable puede ser number o string en distintos momentos de ejecución.

- Formado en su totalidad por objetos, en los cuales los nombres de sus propiedades son claves de tipo cadena siendo objeto.a = 1 y objeto['a'] = 1 equivalentes.

- Lenguaje interpretado, es por esto que no requiere un compilador ni crear un fichero binario del código; cada navegador tiene su intérprete que se encarga de ejecutarlo.

- Evaluación en tiempo de ejecución gracias a la función eval, la cual evalúa un código en JavaScript representado como una cadena de caracteres.

3.2. A-Frame

A-Frame es un framework de código abierto destinado a crear experiencias de realidad virtual a partir de HTML de forma que sea sencillo de leer y comprender. De esta manera es accesible para crear una gran comunidad.

Además tiene compatibilidad con Vive, Rift, Windows Mixed Reality, Daydream, GearVR y CardBoard así como soporte para todos los controladores respectivos. También ofrece soporte para ordenadores de escritorio y para la mayoría de teléfonos inteligentes.

3.2.1. HTML y primitivas

A-Frame se basa en HTML y el DOM usando un polyfill[2] para elementos personalizados. HTML es un componente básico para Web y como tal, tiene una gran accesibilidad como lenguaje. Para crear una escena de realidad virtual con A-Frame no se requiere ninguna instalación y simplemente con la creación del HTML se puede abrir en el navegador. La mayoría de herramientas existentes(como React, Vue.js, d3.js y jQuery) funcionan en este framework.

HTML y DOM son solo la capa más externa del framework, debajo se encuentra el componente three.js en el que está basado A-Frame gracias al cual un componente puede ser utilizado en distintas entidades. De esta manera hace posible seguir el principio de programación Don't Repeat Yourself ya que, una vez registrada una primitiva, se puede hacer referencia al elemento todas las veces que sea necesario.

A-Frame proporciona elementos como a-box o a-sky llamados primitivas. Podemos crear un escenario a través de estas primitivas como el mostrado en la figura 3.1 con el siguiente código:

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <meta charset="utf-8">
5      <title>Escenario primitivas</title>
6      <script src="https://aframe.io/releases/0.9.2/aframe.min.js"></script>
7    </head>
8    <body>
9      <a-scene background="color: #FAFAFA">
10       <a-box position="-1 0.5 -3" rotation="0 45 0" color="#4CC3D9" shadow
             ></a-box>
11       <a-sphere position="0 1.25 -5" radius="1.25" color="#ff0000" shadow
             ></a-sphere>
12       <a-cylinder position="1 0.75 -3" radius="0.5" height="1.5" color="#
             FFC65D" shadow></a-cylinder>
13       <a-plane position="0 0 -4" rotation="-90 0 0" width="4" height="4"
```

[2]Fragmento de código en JavaScript utilizado para proporcionar una funcionalidad moderna en navegadores antiguos.

```
            color=" #1cde83" shadow></a-plane>
14        <a-sky color="#e7e1e0"></a-sky>
15      </a-scene>
16    </body>
17  </html>
```

Listing 3.1: Código con primitivas que representa un escenario

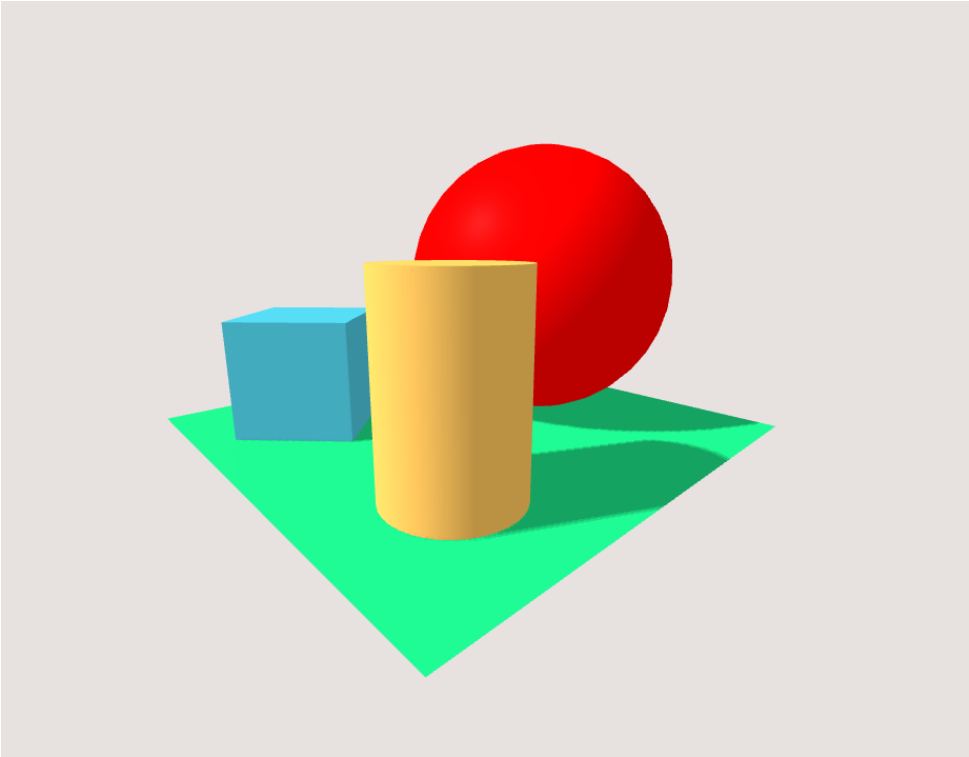

Figura 3.1: Escenario a-frame

A-Frame, además de disponer primitivas como las mostradas, hace posible la creación de primitivas para poder elaborar escenas lo más completas posible. También se pueden incluir entidades más complejas a partir de modelos 3D en formatos como gltf, obj o collada de los cuales se hablará en siguientes apartados.

3.2.2. Entidad, Componente y Sistema

Como ya se ha comentado, A-Frame es un framework three.js con una arquitectura de entidad-componente-sistema (ECS). Es un patrón común en 3D y desarrollo de juegos que siguen la composición sobre el principio de herencia y jerarquía. Algunos beneficios

que ECS aporta son mayor flexibilidad al definir objetos, gran escalabilidad o eliminación de problemas de largas cadenas de herencia. Existe un API que representa cada pieza de ECS:

- Una entidad se representa con la etiqueta a-entity.

```
<a-entity geometry="primitive: box" material="color: red">
```

En este ejemplo se hace uso de a-entity para crear una caja de color rojo.

- Un componente se representa como un atributo de HTML. Cada componente es un objeto que tiene un esquema, manejadores y métodos. Para registrar componentes se utiliza el método de A-Frame registerComponent.

```
AFRAME.registerComponent('example', {
  init: function () {
    var el = this.el;
    el.setObject3D('mesh', new THREE.Mesh());
    el.getObject3D('mesh');  // Returns THREE.Mesh that was
        just created.
  }
});
```

Listing 3.2: Código para registrar un componente

- Un sistema es el representado por atributos HTML mediante la etiqueta a-scene. Se registran de manera similar a un componente; gracias al método registerSystem.

3.3. Blockly

Blockly es una librería que añade un editor de código visual a aplicaciones web y móviles. Utiliza bloques gráficos para representar conceptos complejos de código de manera más sencilla. De esta manera permite a los usuarios aplicar principios de programación sin tener que preocuparse por la sintaxis y ayuda a iniciarse y a aprender a programar a estudiantes de temprana edad. Esta librería está diseñada por las personas que están detrás de Scratch[3] del MIT y construido sobre su base de código.

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

3.3.1. Traductor de código

Blockly es para desarrolladores y las aplicaciones Blockly son pensadas para estudiantes. Desde la perspectiva de usuario, Blockly es una forma visual e intuitiva de crear código y, desde la de desarrollador, es una interfaz de usuario preparada para crear un lenguaje visual que emite código y se puede exportar a otros lenguajes de programación como JavaScript, Python, PHP, Lua o Dart.

Estos generadores de código aportan las herramientas para crear funciones, condicionales, bucles, etc. El principal problema de esto es que en ocasiones se requiere del uso de APIs de otras dependencias. Blockly aporta una solución; un generador de bloques personalizados que traduce al código que deseemos. Esto aporta mucha flexibilidad y da la funcionalidad deseada para el desarrollo de este proyecto.

3.3.2. Bloques personalizados

Como se ha explicado, Blockly dispone de una gran cantidad de bloques predefinidos; desde funciones matemáticas hasta estructuras en bucle. Sin embargo, para interactuar con una aplicación externa, se deben crear bloques personalizados para formar una API. Según la documentación ofrecida por Google[4], la mejor forma de crear un bloque es buscar un bloque existente con una funcionalidad similar y modificarlo según se necesite. De otra manera, para generar un bloque personalizado, hay varios aspectos a tener en cuenta:

- Primero se define el bloque para determinar su aspecto gráfico y su comportamiento. Esto incluye el texto, color, forma o cómo conectarlos con otros bloques. La configuración de estos parámetros se puede realizar mediante JSON o JavaScript. El bloque mostrado en la figura 3.2 se puede configurar con los dos métodos de la siguiente manera:

```
1    {
2      "type": "block_test",
3      "message0": "%1 %2",
4      "args0": [
5        {
```

[4]https://developers.google.com/blockly/guides/create-custom-blocks/overview

```
6           "type": "field_label_serializable",
7           "name": "NAME",
8           "text": "custom_block"
9         },
10        {
11          "type": "input_value",
12          "name": "NAME",
13          "check": "Number"
14        }
15      ],
16      "inputsInline": false,
17      "previousStatement": null,
18      "nextStatement": null,
19      "colour": 120,
20      "tooltip": "This is a custom block",
21      "helpUrl": ""
22    }
```

Listing 3.3: Código en JSON para configurar un bloque personalizado

```
1   Blockly.Blocks['block_test'] = {
2     init: function() {
3       this.appendValueInput("NAME")
4           .setCheck("Number")
5           .appendField(new Blockly.FieldLabelSerializable("
                custom_block"), "NAME");
6       this.setInputsInline(false);
7       this.setPreviousStatement(true, null);
8       this.setNextStatement(true, null);
9       this.setColour(120);
10    this.setTooltip("This is a custom block");
11    this.setHelpUrl("");
12     }
```

```
13      };
```

Listing 3.4: Código en javascript para configurar un bloque personalizado

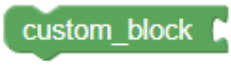

Figura 3.2: Bloque personalizado

- Configurar la traducción del bloque a la instrucción deseada en los distintos lenguajes necesarios.
- Inicializar el bloque para que sea visible en el editor visual de código.

  Aunque los parámetros del JSON son auto-descriptivos, es complicado generar un bloque desde cero. Es por ello que Google facilita una herramienta de creación de bloques, la cual se muestra en la figura 3.3. En ella se puede personalizar todos los aspectos del bloque: texto, color, variables de entrada, formas de conexión con otros bloques, etc. Además, permite exportar la configuración del bloque en formato JSON o JavaScript y muestra una función con la configuración básica para obtener el código en JavaScript(y permite seleccionar el lenguaje deseado entre los admitidos por Blockly).

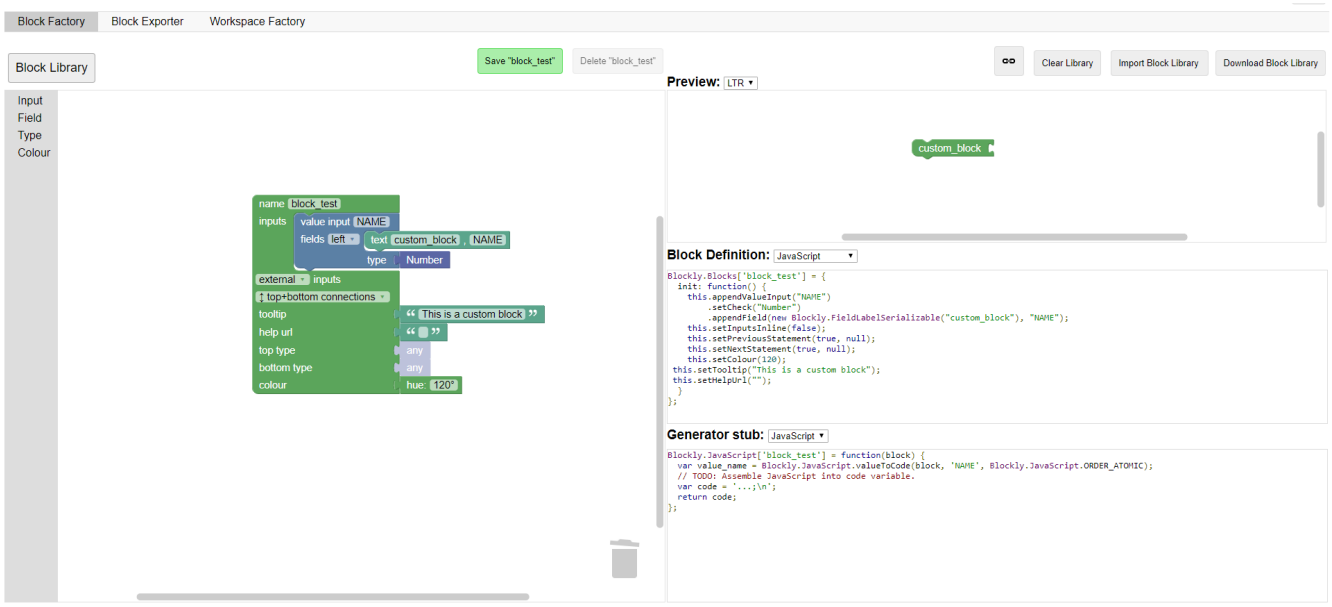

Figura 3.3: Herramienta para creación de bloques personalizados

3.4. Gestores de paquetes

Un gestor de paquetes es una herramienta para automatizar el proceso de instalación, actualización, configuración y eliminación de software. En este proyecto se han utilizado NPM y Webpack

3.4.1. NPM

Node Package Manager (NPM) es el sistema de gestión de dependencias por defecto para Node.js, que es un entorno de ejecución para JavaScript y permite, con la configuración de un fichero, descargar dependencias y paquetes necesarios para el correcto funcionamiento de la aplicación. Todo ello ha de estar definido en un fichero llamado package.json que debe ser escrito en JSON. Para usar NPM como instalador de paquetes únicamente hay que ejecutar npm install en el directorio en el que se encuentre el fichero package.json. Además, gracias a este gestor, podemos configurar pequeños scripts para ejecutar algún proceso como lanzar una aplicación después de la instalación o empaquetar la aplicación en bundles combinándolo con Webpack.

3.4.2. Webpack

Webpack es un sistema de bundling usado para empaquetar una aplicación web en su fase de producción. Se puede considerar una evolución de Grunt[5] y Gulp[6] porque permite automatizar procesos principales tales como transpilar y preprocesar código.

3.5. Blender

Blender es un programa libre dedicado al diseño y animación 3D. Mediante una interfaz gráfica permite diseñar objetos, personajes y escenas en tres dimensiones con muy diversas técnicas. Cada uno de los elementos creados pueden ser animados mediante keyframing o animación por fotogramas clave. En su origen Blender fue distribuido como

[5]https://gruntjs.com/
[6]https://gulpjs.com/

una herramienta privada explotada por un estudio de animación, pero actualmente se encuentra bajo licencia GPL[7].

Entre las muchas características que ofrece Blender, las más útiles para el desarrollo del proyecto son:

- Modelados: es el proceso mediante el cual se crean objetos en un espacio 3D de manera digital. Están compuestos de líneas, puntos, cuadrados, triángulos, etc.

- Iluminación: existen varios tipos de luz que se pueden adaptar a la escena. Es posible aplicar a cada uno de los objetos para especificar la luz recibida por cada uno de ellos además de variar aspectos como la intensidad, el color o la posición en el escenario.

- Tracking: permite especificar el comportamiento y características de un determinado objeto en el escenario tridimensional.

- Animaciones: cada objeto se puede animar de manera independiente. Se realiza mediante un timelapse en la cual cada posición del timeline es un frame en el cual se especifica cualquier tipo de característica (rotación, movimiento, cambio de color, etc).

- Texturizado: son un medio para agregar o eliminar detalles a la superficie de un determinado objeto. Proyectando imágenes sobre la superficie del mallado se pueden aplicar patrones para personalizar el aspecto de la textura.

  Además permite exportar los modelos a los formatos más usados para A-Frame: glTF y COLLADA.

3.5.1. COLLADA

COLLaborative Design Activity (COLLADA) es un formato de archivo de intercambio para aplicaciones 3D interactivas. Se define como un esquema XML para intercambiar activos digitales entre varias aplicaciones de software de gráficos. Son definidos con la extensión .dae y pueden hacer referencia a archivos de imagen adicionales que actúan como texturas del objeto 3D.

[7]https://www.gnu.org/licenses/gpl-3.0.en.html

3.5.2. glTF

GL Transmission Format (glTF) es una especificación basada en el estándar JSON para transmisión y carga eficiente de escenas y modelos 3D para aplicaciones. Este formato minimiza el tamaño de los ficheros y su tiempo de ejecución necesario para desempaquetar y usarlos. Permite animación de los objetos que puede ser activada por A-Frame vía HTML o dinámicamente con javaScript. En la figura 3.4 se puede ver la figura añadida a un escenario de A-Frame via HTML:

```
1 <a-asset-item id="tree" src="assets/models/CartoonTree.dae"></a-asset-item
    >
2  <a-entity id="a-tree" collada-model="#tree" rotation="0 0 0" position="
     2.75 0.01 -2.27">
```

Listing 3.5: Código para añadir un modelo 3D personalizado a A-Frame

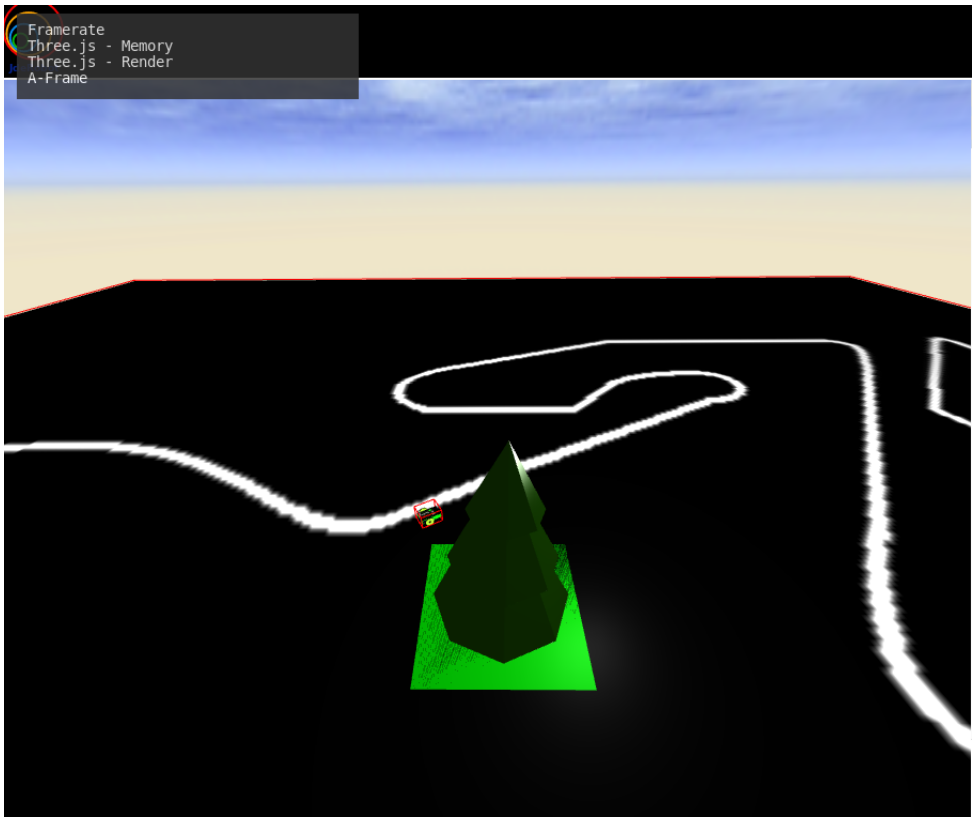

Figura 3.4: Modelo glTF añadido en un escenario de A-Frame

Además del software explicado en esta sección, existen multitud de herramientas para

convertir a glTF desde formatos como OBJ, COLLADA o PCD[8]. De esta manera es posible buscar modelos 3D en librerías[9] para editarlos e incluirlos en Websim.

3.6. WebSim

Websim es un simulador robótico diseñado para enseñar conceptos básicos de tecnología e iniciar a niños en robótica y programación de robots.

3.6.1. Diseño

El simulador hace uso del entorno A-Frame y permite conectar un editor de texto o un editor de bloques para poder programar en JavaScript o Blockly y conectar este código con el robot simulado. También permite acoplar una aplicación externa al navegador a través de comunicaciones ICE. Las principales funcionalidades del simulador son:

- Registrar los componentes principales para constituir un robot en A-Frame, los cuales son followBody,spectatorComponent e intersectionHandler. El primero se encarga de simular una cámara en el robot, el segundo maneja eventos de intersección de los láseres y el último permite anclar distintos elementos al robot simulado.
- Ofrece una interfaz en JavaScript para manejar el robot en el entorno simulado de A-Frame llamada Hardware Abstraction Layer (HAL API). Websim se encarga de enviar instrucciones al robot de manera sencilla sin necesidad de comunicarse con el motor de A-Frame.
- No es necesario que el usuario instancie ningún tipo de variable de la clase que contiene al objeto robot porque el simulador lo ofrece de manera directa. De esta manera el usuario puede mandar directamente instrucciones al robot simulado con el objeto myRobot y los métodos existentes de la clase RobotI.
- Permite manejar la ejecución de la simulación del robot. Es decir, permite lanzar o pausar la ejecución del robot e incluso reiniciar su posición para no tener que recargar la página en caso de querer probar distintos códigos con la misma simulación.

[8]https://github.com/KhronosGroup/glTF
[9]https://sketchfab.com

Además este control del entorno evita que la variable myRobot pierda el objeto instanciado porque el usuario cambie su valor.

Gracias a estas características, el simulador hace que los usuarios puedan programar de manera sencilla, ya que solo tienen que acceder a la información que ofrecen los sensores del robot y mandar órdenes sobre los actuadores del mismo. Es decir, solo se tienen que encargar de programar la lógica del robot para resolver los ejercicios propuestos que se explicarán en próximos capítulos.

3.6.2. Clase Robot

Para poder tener una entidad en A-Frame que pueda ser programada por el usuario existe un clase en JavaScript llamada RobotI. El constructor de esta clase es un método obligatorio y es al primero que se llama cuando se instancia un objeto. Tiene un parámetro de entrada que es un string con el identificador de la etiqueta HTML en la que se encuentra el robot simulado. Además de inicializar el robot, la clase contiene una serie de variables para su configuración:

- defaultDistanceDetection: declara la distancia máxima en la que los sensores de ultrasonido detectan un objeto.
- defaultNumOfRays: declara el número de rayos que se simulan para detectar objetos. Abarca un ángulo de 180 grados y, por lo tanto, cuanto mayor sea el número, más precisión habrá en la detección de objetos.
- robot: crea la unión entre la clase y la entidad del robot simulado en A-Frame.
- initialPosition: en la inicialización del robot, obtiene del HTML la posición del robot y la guarda en esta variable.
- initialRotation: en la inicialización del robot, obtiene del HTML la rotación del robot y la guarda en esta variable.
- activeRays: es una variable de tipo boolean y permite saber si los rayos del sensor de ultrasonidos están activos no.

- distanceArray. Es un objeto en JavaScript con tres variables tipo array y almacena la distancia con los objetos detectada por los sensores de ultrasonido. Las variables están diferenciadas en derecha, centro e izquierda y permite conocer donde se encuentra la ubicación del objeto detectado.

- understandedColors: variable que permite asociar un color como tipo string a sus valores de filtro en el espacio RGB. Esto permite hacer más simple la detección de los objetos para los usuarios ya que con el nombre del color se puede realizar el filtro sin necesidad de tener conocimientos sobre visión artificial.

- velocity: variable que guarda la velocidad del robot en los distintos ejes (x,y,z). En el eje X se almacena la velocidad en el plano horizontal y en el eje Z la velocidad de giro. La velocidad del eje Y será útil en el desarrollo del proyecto ya que es la que otorga velocidad de elevación.

Además, el constructor llama a los métodos para inicializar los motores y sensores los cuales tienen sus propios métodos para su manejo.

3.6.3. Drivers de sensores

El robot consta de sensores simulados con A-Frame. Los drivers permiten que el usuario pueda acceder a estos sensores a través del código programado y obtener su información. En este entorno disponemos de los siguientes sensores:

- Ultrasonido o láser: permite detectar obstáculos en un radio de 180 grados en la parte frontal del robot. En A-Frame se simula gracias al atributo raycaster el cual permite conocer el punto de intersección entre un rayo emitido y un objeto. Para la simulación de este sensor se usan los componentes followBody (para anclar un raycaster al robot y que pueda seguir la posición del robot) e intersectionHandler (que crea un manejador para controlar el evento que lanza un raycaster cuando un objeto se interpone en el rayo que emite). Además, este componente es capaz de detectar el identificador del raycaster y la distancia a la que es detectado.

- Cámara: obtiene una imagen con lo que capta el robot de la escena. Para su simulación se ha creado el atributo spectator y permite mostrar la visión del robot en

primera persona.

- Infrarrojos: dispositivo que está compuesto de un LED infrarrojo y un fototransistor siendo capaz de detectar si el objeto con el que choca la luz emitida por el LED es una superficie blanca (la luz rebota y llega hasta el fotoreceptor) o negra (la superficie absorbe toda la luz y no llega al fotoreceptor). Para simular este sensor se recorta la imagen de la cámara para quedarse con los píxeles de la parte inferior (con una dimensión de 5x150px). La función del HAL API que hace referencia a este sensor es readIR(color) siendo el parámetro pasado a la función un color en formato de string que toma como variable undestandedColors para obtener los valores del filtro correspondiente. De tal manera, después del filtrado se obtiene una imagen según la posición en la que se encuentre la línea a seguir y según donde esté el centro devuelve distintos valores: 0 si la línea se encuentra entre los píxeles 57 y 93 de la imagen, 1 entre los píxeles 0 y 57, 2 si la línea está entre 93 y 150 y 3 si no detecta la línea.

- Odómetro: obtiene la posición absoluta del robot durante la ejecución del código. Para la simulación de los sensores de odometría se ha hecho uso del sistema de coordenadas y rotación de A-Frame y se ha hecho el cálculo a través de su API en JavaScript.

En la tabla 3.1 se explican todas las funciones del HAL API que hacen referencia a los sensores.

Cuadro 3.1: Métodos (HAL API) de los sensores del robot.

Método	Descripción	Salida
.getDistance()	Devuelve la distancia entre el robot y la intersección del raycaster en el centro	number(metros)
.getDistances()	Devuelve la distancia entre el robot y la intersección con cada una de los raycaster	list number(metros)
.readIR(color)	Recorta la imagen, filtra y calcula el centro del objeto con el color pasado como argumento	number
.getRotation()	Retorna un objeto con la orientación del robot en los 3 ejes	{x:number, y:number, z:number}
.getPosition()	Obtiene la posición del robot en la escena	{x:number, y:number, z:number}
.getImage()	Devuelve la imagen de la cámara del robot	cv.Mat()
.getObjectColor(color)	Devuelve un objeto con la posición del objeto detectado por la cámara del color pasado por parámetro	{center:[x.y], area: int}
.getObjectColorRGB(valorBajo,valorAlto)	Devuelve un objeto con la posición del objeto detectado por la cámara con los valores pasados por parámetro	{center:[x.y], area: int}

3.6.4. Drivers de actuadores

La función de los actuadores es otorgar movimiento al cuerpo del robot simulado en A-Frame. Con los métodos creados, además, no es necesario mandarle ordenes constantemente, si no que es suficiente con mandar la instrucción una vez y el robot seguirá ejecutándola hasta que reciba una nueva.

Para que el robot siga las ordenes mandadas existen los siguientes métodos:

- getV: Método para conocer la velocidad lineal ordenada al robot.
- getW: Método que devuelve la velocidad angular.
- setV: Método que permite otorgar velocidad lineal al robot.
- setL: Método para que el usuario comande la velocidad angular del robot.
- move: Método que sirve para otorgar tanto velocidad lineal como angular.

Después de llamar a cada uno de estos métodos, se guarda la velocidad pasada por parámetro en la variable interna llamada velocity y, para calcular la posición en el escenario

simulado con A-Frame es necesaria la siguiente función:

```
updatePosition(rotation, velocity, robotPos){
  let x = velocity.x/10 * Math.cos(rotation.y * Math.PI/180);
  let z = velocity.x/10 * Math.sin(-rotation.y * Math.PI/180);
  robotPos.x += x;
  robotPos.z += z;
  return robotPos;
}
```

Listing 3.6: Función para actualizar la posición del robot en el escenario

En esta función se establece la nueva posición relativa para el objeto. En cada iteracción se calcula y se establece nueva posición y rotación del robot. Además, gracias a la función de javaScript llamada setTimeout, se establece un temporizador para actualizar la posición del robot llamando a la función updatePosition iterativamente cada cierto periodo de tiempo. Esto permite simplificar el código ya que hace que no es necesario mandar ordenes al robot constantemente.

En la tabla 3.2 se explican todas las funciones del HAL API que hacen referencia a los actuadores.

Cuadro 3.2: Métodos (HAL API) de los actuadores del robot.

Método	Descripción
.setV(integer)	Mueve hacia delante o atrás el robot.
.setW(integer)	Hace girar al robot.
.move(integer, integer)	Mueve el robot hacia delante/atrás y gira al mismo tiempo.
.getV()	Obtener la velocidad lineal configurada en el robot.
.getW()	Obtener la velocidad angular configurada en el robot.

Capítulo 4

Mejoras a WebSim

4.1. Modelos y mundos

4.1.1. Drone

4.1.2. F1

4.1.3. mBot

4.2. Teleoperadores

4.2.1. Archivos de configuración

4.2.2. Diseño

4.3. Ejercicios competitivos

4.3.1. Escenarios

4.3.2. Evaluador automático

Capítulo 5

Conclusiones

5.1. Conclusiones

5.2. Mejoras futuras

Bibliografía

[1] Documentación JavaScript: https://developer.mozilla.org/es/docs/Web/JavaScript

[2] Documentación A-Frame: https://aframe.io/

[3] Documentación Blockly: https://developers.google.com/blockly

[4] Desarrollo web: https://www.w3schools.com/

[5] Documentación Blender: https://www.blender.org/

[6] Información sobre glTF: https://www.khronos.org/gltf/ @bookperez2011manual, title=Manual de modelado y animación con Blender, author=Pérez, Pablo Suau, year=2011, publisher=Universidad de Alicante