



GRADO EN INGENIERÍA TELEMÁTICA

Curso Académico 2020/2021

Trabajo Fin de Grado

Integración del Robot Lego Ev3 en una plataforma de Robótica Educativa

Autor : Daniel Pulido Millanes

Tutor : Dr. José María Cañas Plaza

Resumen

Este Trabajo de Fin de Grado tiene como fin la mejora de la plataforma de robótica educativa *Kibotics*. Plataforma que esta destinada a la educación de la robótica para alumnos desde niños hasta adolescentes, ya que cada vez es más común el uso de nuevas tecnologías en las etapas más tempranas del aprendizaje, porque está demostrado que mejora las habilidades en física, matemáticas y tecnología. *Kibotics* utiliza un simulador llamado *WebSim*, que esta basado en el entorno de *A-Frame*, para representar los ejercicios que contiene la plataforma

El proyecto se ha centrado en la integración del robot educativo *LEGO Ev3* en plataforma de *Kibotics*. Para ello, se han creado tres modelos tridimensionales con tres diferentes sensores para implementarlos en el entorno de *WebSim*. Se han creado los *drivers* en *JavaScript* y funciones al *RobotApi* necesarios para que el robot sea programable dentro de la plataforma. También he creado *Drivers* en robot real para que sea capaz de ejecutar código en *Python* que le llega al robot mediante peticiones de *HTML*. Todo lo implementado ha sido validado con la creación de ejercicios.

Las implementaciones en el *drivers* simulado se han realizado en *JavaScript*, tanto el servidor *Flask* como los *drivers* de robot real se han programado en *Python*.

Índice general

Lista de figuras	9
Lista de tablas	11
1. Introducción	1
1.1. Robótica	1
1.1.1. Aplicaciones robóticas	3
1.1.2. Software en robótica	6
1.1.3. Simuladores robóticos	7
1.2. Tecnologías web	8
1.2.1. FrontEnd	9
1.2.2. BackEnd	10
1.2.3. HTTP	10
1.3. Robótica educativa	11
1.3.1. Kibotics	15
2. Objetivos	17
2.1. Objetivos del TFG	17
2.2. Requisitos	18
2.3. Metodología	18
2.4. Plan de trabajo	20
3. Herramientas	21
3.1. Lego Ev3	21
3.2. Python	23

3.3.	HTML	23
3.4.	JavaScript	24
3.5.	A-Frame	24
3.6.	Blender	25
3.7.	Simulador WebSim	26
3.7.1.	Diseño	27
3.7.2.	Drivers de sensores	28
3.7.3.	Drivers de actuadores	29
3.8.	Django	30
3.9.	Flask	31
4.	Soporte Simulado	33
4.1.	Modelo 3D	33
4.2.	Interfaz de programación de sensores y actuadores	36
4.2.1.	Soporte de actuadores	36
4.2.2.	Soporte de sensores	37
4.3.	Validación experimental con ejercicios	44
4.3.1.	Sigue-líneas	45
4.3.2.	Choca-gira	46
4.3.3.	Atraviesa-bosque	46
5.	Soporte Físico	49
5.1.	Interfaz de programación de sensores y actuadores	49
5.1.1.	Soporte de actuadores	49
5.1.2.	Soporte de sensores	52
5.2.	Descarga de la plataforma	53
5.2.1.	Preparación ordenador a bordo	53
5.2.2.	Diseño	54
5.2.3.	WebBrowser	55
5.2.4.	Back-End	57
5.3.	Validación experimental con Ejercicios	59
5.3.1.	Bump and go	59

ÍNDICE GENERAL

7

5.3.2. Siguelineas	59
6. Conclusiones	61
6.1. Conclusiones	61
6.2. Mejoras futuras	62

Índice de figuras

1.1.	Imagen clásica de un robot	2
1.2.	Aspiradora robótica <i>Roomba</i>	4
1.3.	Robot médico <i>Da Vinci</i>	4
1.4.	Robot militar <i>Big Dog</i> creado por <i>Boston Dynamics</i>	5
1.5.	Vehículo <i>Waymo</i> de <i>Google</i>	6
1.6.	Robot <i>Perseverance</i> de la <i>NASA</i>	6
1.7.	Interfaz de Gazebo	8
1.8.	Concepto de Tecnologías Web	9
1.9.	Comunicación cliente-servidor en HTTP	10
1.10.	Modelo de Educación <i>STEM</i>	12
1.11.	Interfaz gráfica de Scratch	13
1.12.	Interfaz de LEGO WeDo	13
1.13.	Interfaz gráfica de LEGO Ev3	14
1.14.	Kit de piezas y sensores de LEGO Ev3	15
1.15.	Menu principal de <i>Kibotics</i>	16
1.16.	Ejercicio en Kibotics	16
2.1.	Metodología de modelo iterativo	19
3.1.	Conjunto Ev3	22
3.2.	Algunos ejemplos de <i>A-Frame</i>	25
3.3.	Interfaz gráfica de <i>Blender</i>	26
3.4.	Interfaz gráfica de <i>WebSim</i>	28
3.5.	Arquitectura Django	31

3.6. Logo Flask	32
4.1. Robot base	34
4.2. Robot con sensor de color	35
4.3. Robot con sensor tactil	35
4.4. Robot con sensor de ultrasonidos	35
4.5. Motores	36
4.6. Sensor color	37
4.7. Sensor de ultrasonidos	40
4.8. Raycaster	41
4.9. Sensor tactil	43
4.10. girosensor	44
4.11. Escenario para el ejercicio <i>LEGO EV3 sigue-líneas</i>	45
4.12. Solución en <i>Scratch</i> para el ejercicio sigue-líneas	45
4.13. Solución en <i>Scratch</i> para el ejercicio choca-gira	46
4.14. Escenario para el ejercicio atraviesa bosque	48
4.15. Solución en <i>JavaScript</i> para el ejercicio atraviesa bosque	48
5.1. Diseño hardware del Ev3	53
5.2. Ev3Dev Logo	54
5.3. Interfaz de la terminal de Ev3	54
5.4. Diseño para la integración del Ev3	55

Índice de cuadros

3.1. Métodos (HAL API) de los sensores del robot.	29
3.2. Métodos (HAL API) de los actuadores del robot.	30
4.1. Métodos (HAL API) de los actuadores del robot.	36
5.1. Métodos (HAL API) soportados por Ev3.	52

Capítulo 1

Introducción

En este capítulo se introducen los conceptos básicos en robótica, de como esta nos ayuda en nuestro día a día, y cual es su estado actual. Y como puede ser un gran recurso en la educación. En lo que se basa este proyecto

1.1. Robótica

La robótica es una rama de las ingenierías y de las ciencias de la computación que se encarga del diseño, construcción, operación, estructura, manufactura y aplicación de los robots. El término *robot* se popularizó con el éxito de la obra R.U.R. (*Robots Universales Rossum*), escrita por Karel Čapek en 1920. En la traducción al inglés de dicha obra la palabra checa *roboťa*, que significa trabajos forzados o trabajador, fue traducida al inglés como robot. Un robot es una entidad virtual o mecánica artificial. Están diseñados con un propósito propio. La independencia creada en sus movimientos hace que sus acciones sean la razón de un estudio razonable y profundo en el área de la ciencia y tecnología. La palabra robot puede referirse tanto a mecanismos físicos como a sistemas virtuales de software, aunque suele aludirse a los segundos con el término de bots.



Figura 1.1: Imagen clásica de un robot

No hay un consenso sobre qué máquinas pueden ser consideradas robots, dentro de este proyecto tomaremos como definición que un robot es un sistema autónomo programable capaz de realizar tareas complejas. Además, todos los robots se componen de tres partes esenciales se componen de sensores, controladores y actuadores.

- **Sensores:** Son los sentidos del robot, con ellos ve, escucha y sabe lo que hay en el entorno. Recogen la información necesaria para que el robot realice la tarea En este grupo se encuentran láseres, cámaras, ultrasonidos u odómetros..

- **Controladores:** El equivalente al cerebro humano, utiliza los datos recogidos por los sensores para elaborar una respuesta para que la lleve a cabo los actuadores.
- **Actuadores:** Equivalen a los músculos humanos, son los que se encargan de interactuar con el entorno para llevar a cabo su tarea. Son brazos mecánicos, motores, etcétera...

1.1.1. Aplicaciones robóticas

Ahora que tenemos las bases de lo que es un robot asentadas podemos hablar de cuales son los principales propósitos de los robots hoy en día, aunque la mayor parte de ellos son utilizados por empresas en labores industriales. Aunque hay otros que podemos encontrar en nuestra vida cotidiana, en casas, hospitales, almacenes de tiendas... Esto es debido a la precisión de algunos trabajos, la eficiencia en el trabajo, la reducción de costes que supone o que pueden realizar acciones de alto riesgo para las personas. Los ejemplos más famosos en estos campos son los siguientes:

- Robots Domésticos: Creados para realizar las tareas del hogar. Los más famosos y destacados en el mercado son los Robots *Roomba*, aspiradores autónomos, y también el primer robot que se ha comercializado para todos los públicos y de manera global. Un gran paso para la robótica



Figura 1.2: Aspiradora robótica *Roomba*

- Robots médicos: Son robots diseñados para el uso en medicina para realizar tareas que requieren mucha precisión como en el caso de una cirugía, con el robot *Da Vinci* o robots diminutos que son capaces de navegar por las venas hasta llegar al corazón y allí realizar la cirugía necesaria.



Figura 1.3: Robot médico *Da Vinci*

- Robots militares: Son robots orientados a tareas militares, como reconocimientos de zo-

nas conflictivas o rescate de personas, desactivación de bombas. En los últimos años también se han desarrollado mucho los drones en combate.



Figura 1.4: Robot militar *Big Dog* creado por *Boston Dynamics*

- Vehículos autónomos: Es el campo de la robótica que más en auge esta ahora mismo. El objetivo de estos robots es usar la información que proporcionan sus sensores internos, como cámaras, sensores infrarrojos *Lidar*, y sensores externos como el GPS para llevar de un punto a otro un vehículo.



Figura 1.5: Vehículo Waymo de *Google*

- Robots Espaciales: Los famosos *Rover* de la *NASA* son robots diseñados para entornos donde el ser humano no puede llegar. Se centran en reconocimiento del terreno y análisis de las muestras que recogen.

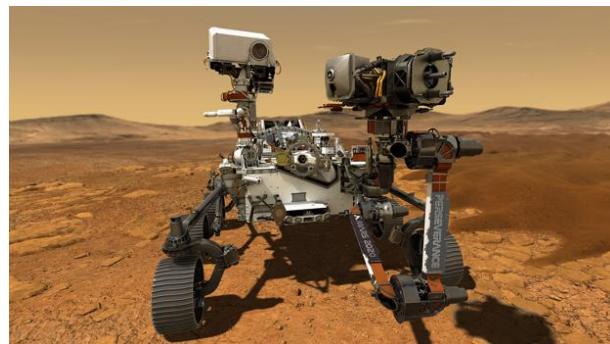


Figura 1.6: Robot *Perseverance* de la *NASA*

1.1.2. Software en robótica

Para dotar de esta inteligencia a los robots se necesitan herramientas que transformen los datos recibidos de los sensores en algo que puedan aplicar en los actuadores. Hace años, cada máquina tenía un software específico con sensores y actuadores únicos para ese robot y esa tarea a desarrollar. Esto hacía, que aunque hubieras implementado el software para otros robots anteriormente, tuvieras que repetir el proceso con cada nuevo robot. Con los años se desarrollaron plataformas de software que permiten desarrollar de manera genérica para todos los robots, y actuando de mediador entre el robot y el software del creador, estos son los llamados *middleware*

que hacen que te puedas abstraer de los *drivers* característicos de cada robot. Los middleware mas importantes a día de hoy son:

- **Robot Operating System (ROS)[?]**. Plataforma de *software* libre para el desarrollo de *software* de robots. Provee servicios estándar de un sistema operativo como la abstracción de *hardware*, control de dispositivos de bajo nivel, mecanismos de intercambio de mensajes entre procesos y mas herramientas vitales para el desarrollo del robot. Es el mas utilizado a día de hoy porque fue especialmente desarrollado para *UNIX* y luego se implemento para el resto de sistemas operativos
- **ORCA[?]**. Plataforma de *software* libre diseñado para crear aplicaciones mas complejas, ya que esta orientado a las componentes por separado
- **OROCOS[?]** Proyecto de *software* libre también orientado a componentes y basado en C++

1.1.3. Simuladores robóticos

La simulación es el proceso de diseñar un modelo de un sistema real y llevar a término experiencias con él, con la finalidad de comprender el comportamiento del sistema o evaluar nuevas estrategias. De esta forma los simuladores ahorran tiempo porque puedes detectar posibles errores en el código antes de lanzarlo en el *Hardware* y que pueda llevar mayores problemas. Algunos de los simuladores que más se utilizan y más nos interesan en este proyecto:

- **Gazebo**: Gazebo es un simulador de robótica 3D de código abierto. Gazebo fue un componente en el Proyecto Player desde 2004 hasta 2011. Gazebo integró el motor de física ODE, la representación de OpenGL y el código de soporte para la simulación de sensores y el control del actuador. También es importante destacar que tiene soporte para ROS lo que significa que puedes probar código real del robot en el simulador

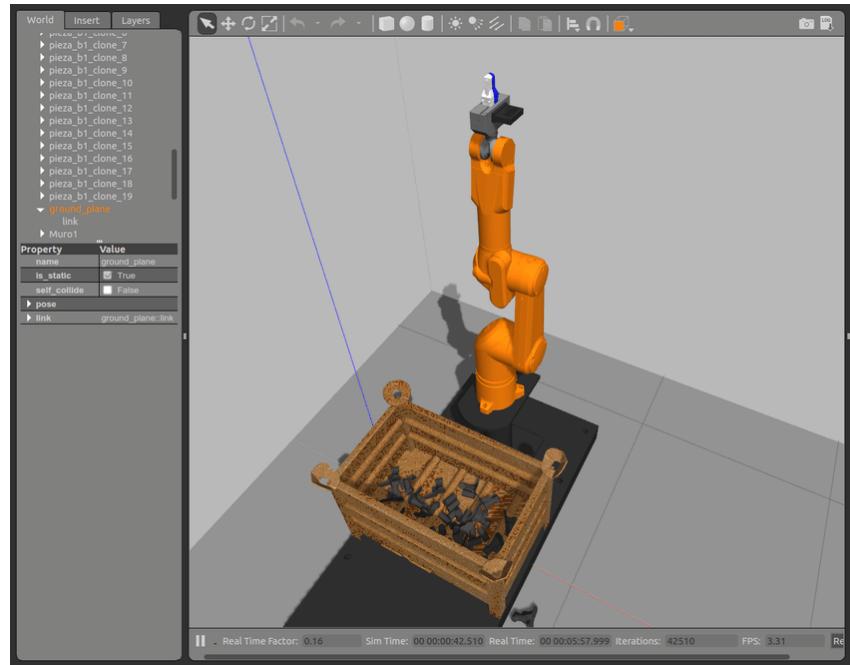


Figura 1.7: Interfaz de Gazebo

- **WebSim:** Es un simulador diseñado por alumnos de la universidad y que va escalando poco a poco. Hace uso de un entorno A-Frame y su diseño permite conectarlo con un editor que permite programar en diferentes lenguajes como *JavaScript* y *Python*, incluso un lenguaje de bloques llamado *Blockly*(equivalente a Scratch pero diseñado por Google). Permite acoplar una aplicación externa a través de comunicaciones ICE. Es la base de la plataforma de *Kibotics* y que vamos a utilizar en este proyecto.

1.2. Tecnologías web

Las tecnologías web sirven para acceder a los recursos de conocimiento que hay disponibles en *Internet* utilizando un *navegador*. Son herramientas que procesan y lo muestran mediante una interfaz gráfica lo almacenado en un servidor. Además gracias a la forma que esta estructurada la *World Wide Web* (WWW) hace sencillo saltar de un recurso a otro. WWW es una combinación de 4 ideas: Hipertexto, identificadores de recursos(URL y URI), el lenguaje de marcado y el modelo cliente servidor.

Lo que nos interesa de las *Tecnologías Web* es el modelo de cliente-servidor, también llamados, *FrontEnd* a todo lo que el usuario se encuentra directamente en la web, lo que es la parte

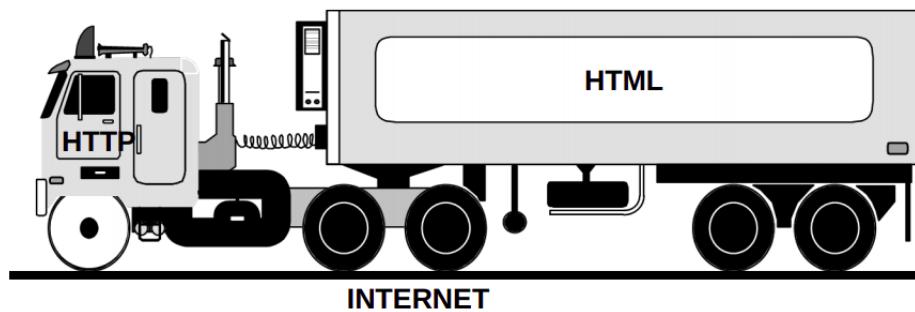


Figura 1.8: Concepto de Tecnologías Web

del cliente. Y *BackEnd* a la parte del interior de las aplicaciones que viven en el servidor, que es lo denominado .^{el} lado del servidor”. Modelo que utilizare para abastecer a mi robot de los programas que se desarrollan en *Kibotics*.

1.2.1. FrontEnd

Es la parte encargada de dar forma a la interfaz de usuario y de establecer la comunicación con el servidor. Una pieza importante del cliente es el navegador, ya que es el encargado de leer e interpretar la información recibida. Entre los navegadores *web* más empleados se encuentran *Firefox*, *Google Chrome* u *Opera*[?]. Las tecnologías que hacen posible esa comunicación son:

- **JavaScript.** Es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos. Es el lenguaje más utilizado para el desarrollo de aplicaciones Web y de páginas Web. La mayor parte de *Kibotics* esta desarrollado en *Javascript*. Por lo que en el siguiente capítulo desarrollaremos en más profundidad.
- **HTML.** siglas en inglés de HyperText Markup Language, hace referencia al lenguaje de marcado para la elaboración de páginas web. Es el estándar más utilizado a dia de hoy. HTML sólo sirve para indicar como va ordenado el contenido de una página web. Esto lo hace por medio de las marcas de hipertexto las cuales son etiquetas conocidas en inglés como tags. Las mejoras del diseño gráfico ya vienen dada de la versión actual, *HTML5*, que incluye muchas mejoras respecto a su predecesor: *canvas*, *websockets*, *WebRTC*, vídeo, audio, etc. Este lenguaje indica la estructura de una página web, para editar el estilo y presentación visual hay que hacer uso de otros elementos como *CSS*.

- **Cascading Style Sheets (CSS)**. Lenguaje de diseño gráfico para definir y crear la presentación de un documento escrito en un lenguaje de marcado. De esta forma, se puede separar información y datos (en los documentos *HTML*) y todo lo relativo al diseño y presentación (en documentos *CSS*). Actualmente los navegadores usan la versión *CSS3*.

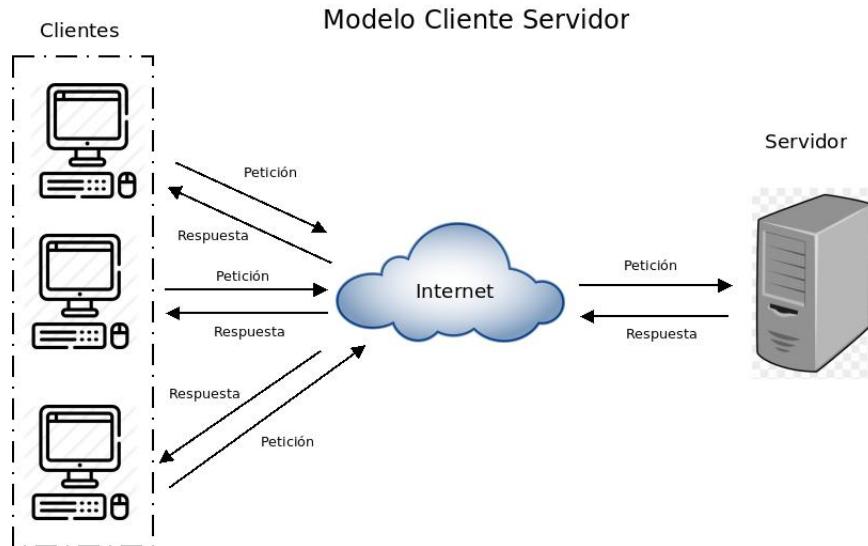


Figura 1.9: Comunicación cliente-servidor en HTTP

1.2.2. BackEnd

En la parte del servidor web se encarga de proveer los datos, hacer posible crear una aplicación que se le mostrará al cliente y facilitar herramientas como bases de datos y recursos compartidos.

- **Node.js**: Node.js es un entorno JavaScript que nos permite ejecutar en el servidor, de manera asíncrona, con una arquitectura orientada a eventos
- **Django**: Django es un framework web extremadamente popular y completamente funcional, escrito en *Python*. El módulo muestra por qué Django es uno de los frameworks de servidores web más populares.

1.2.3. HTTP

HiperText Transfer Protocol (HTTP) es el protocolo de nivel de aplicación utilizado para transferir recursos hipermedia entre ordenadores y sigue el esquema petición-respuesta entre

cliente y servidor (Figura 1.9). Las peticiones están definidas por el protocolo y tienen métodos concretos:

- GET: Solicita un recurso al servidor especificando su *URL*.
- HEAD: Método similar a GET con la diferencia de que únicamente solicita las cabeceras y no descarga el recurso completo.
- POST: Envía datos al servidor, normalmente un recurso específico que provoca un cambio de estado. Es también bastante similar GET pero la cabecera va en claro.
- PUT: Actualiza información sobre un recurso del servidor.
- DELETE: Elimina en el servidor un recurso.

Aunque estos son los principales métodos, el protocolo tiene flexibilidad para ir añadiendo nuevos e incorporar funcionalidad. El número de métodos ha ido aumentando con las nuevas versiones.

1.3. Robótica educativa

La robotica educativa ha ido tomando mas importancia con los años, ya que cada vez es mas importante que estudiantes de cualquier nivel estén familiarizados con la tecnología, tiene valores positivos como la implementación de pensamiento lógico, resolución de problemas y trabajo en equipo en las actividades académicas, que son ramas del conocimiento que se desarrollan poco en edades tempranas , con una componente en conocimiento matemático y físicos y ademas añade un atractivo que no tienen las asignaturas convencionales. Muchos estudios han demostrado que el uso de kits de robótica en la educación favorece a la capacidad de reflexión de los estudiantes. Cada año se crean mas cursos de robótica, y en 2015 la comunidad de Madrid introdujo la asignatura de robótica en los planes docentes de Enseñanza Secundaria con la asignatura “Tecnología, Programación y Robótica”[?] y en el curso 2020-2021 se empezará a implantar en Educación Primaria la asignatura “Programación y Robótica”[?].

Este tipo de educación es el llamado modelo **STEM**(siglas de *Science, Technology, Engineering y Mathematics*), término creado por Seymour Papert en la década de los 80 cuando creo uno

de los primeros juguetes con programación incorporada, creada para niños, el llamado "*Lego-Logo*", dando mucha importancia a juegos con engranajes, que consideraba que enriquecía el pensamiento. Justamente, el referente del protagonista de este proyecto.

Pero no fue hasta 2010, cuando este modelo de educación empezó a tomar relevancia, y se inicio la inclusión en la agenda escolar.

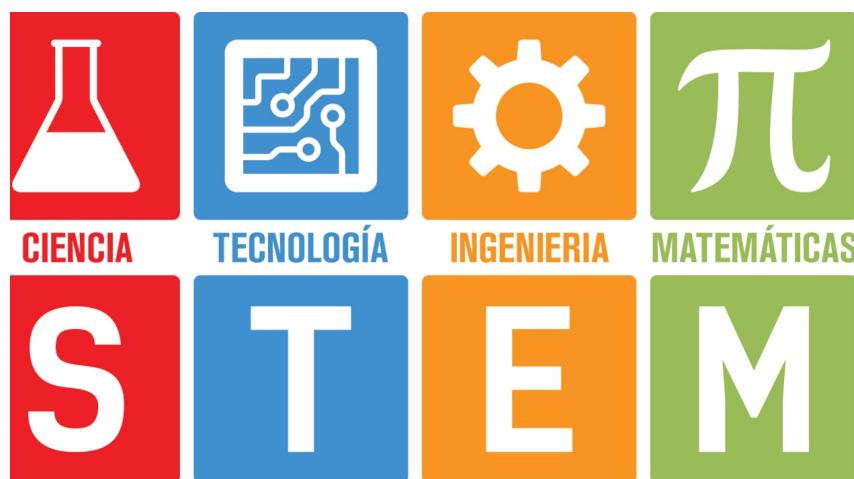


Figura 1.10: Modelo de Educación *STEM*

Una de las mayores partes de la robótica tiene que ver con la programación, que ademas de ser una habilidad muy importante para la sociedad actual, es algo complejo. Por lo que se utilizan lenguajes de programación visual, estos se tratan de lenguajes que abstraen en bloques las funciones o métodos de cualquier lenguaje de programación. Dentro de este tipo de lenguajes, los mas destacables son :

- **Scratch[?]:** proyecto liderado por el Grupo *Lifelong Kindergarten* del *MIT*, es utilizado por estudiantes para programar animaciones, juegos e interacciones. Su atractivo reside en lo facil que es de entender el pensamiento computacional debido a su sencilla interfaz gráfica y la implementación de sus bloques.



Figura 1.11: Interfaz gráfica de Scratch

- **LEGO[?]:** Es el robot base de este proyecto, dispone de una amplia gama de robots programables y cada uno de ellos tiene un sistema gráfico, que es similar entre ellos pero también ligado a la edad el estudiante para el que esta diseñado el software.

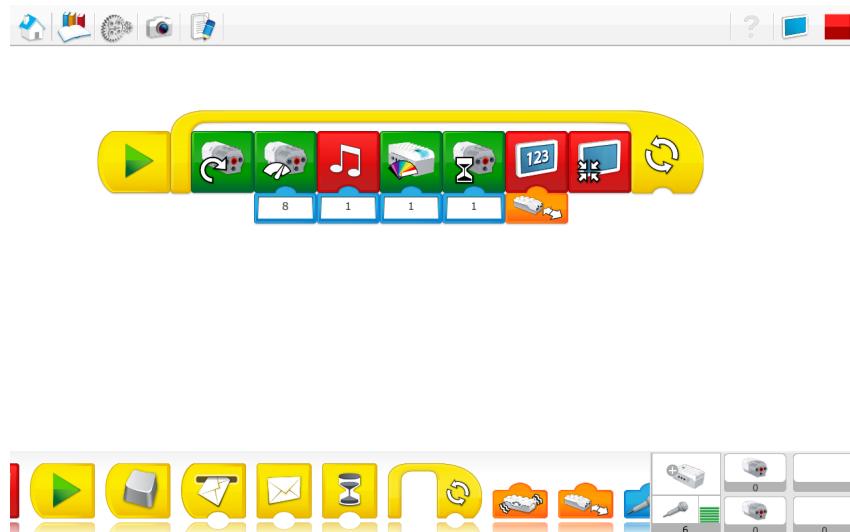


Figura 1.12: Interfaz de LEGO WeDo

Por ejemplo en la figura 1.12 se puede ver que la interfaz en este caso, es con colores vivos, los cuales representan distintas funcionalidades dentro del robot, es decir, el amarillo representa las acciones propias de programación, como: inicio de programa, fin de programa, bucles, esperar, etcétera. El color rojo representa los sensores del robot, todo lo que recoja datos. Y el color verde representa los motores que equivalen a los actuadores

en este robot. Como se puede observar es una abstracción muy simple para estudiantes de mas corta edad.

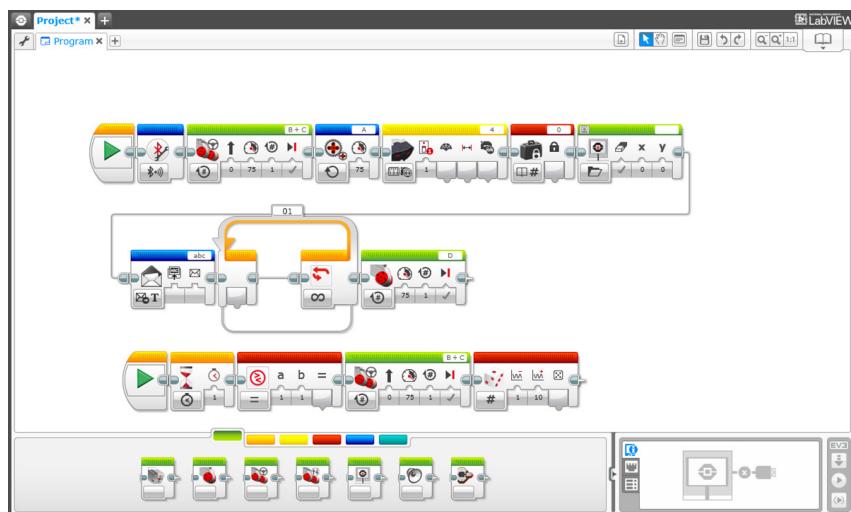


Figura 1.13: Interfaz gráfica de LEGO Ev3

En el caso del software para el **LEGO Ev3**, añade un grado de complejidad, incluyendo apartados para realizar operaciones matemáticas, envio de archivos entre robots, y añade mas actuadores, como la pantalla que integra el robot, o los altavoces.

En el caso de LEGO y en otros kits incorporan los elementos básicos para la construcción de un robot. En este en particular viene con lo indispensable para construir con piezas de LEGO. También incluye un microprocesador para ser programado, con Linux instalado, sensores (infrarrojos, táctiles y de color) y motores.



Figura 1.14: Kit de piezas y sensores de LEGO Ev3

En el siguiente capitulo, profundizare mas en lo que se puede hacer con el robot **LEGO Ev3** y explicare cuales van a ser los objetivos y porque elegir este robot.

1.3.1. Kibotics

Kibotics¹, es un entorno web desarrollado por la Asociación JdeRobot² para la docencia de robótica. Fue creada para impartir cursos de robótica, en los que destacan una iniciación muy sencilla, y una gran variedad de recursos a disposición del alumno. Es una plataforma en la que podemos encontrar distintos ejercicios de robots simulados que no son reales (como el Formula 1). Tiene los ejercicios divididos , en los robots con ruedas sobre una superficie, con cámaras, sensores de todo tipo, y los drones, con funcionalidad adicional, y otro tipo de ejercicios, y dentro de estos dos grupos, se dividen entre los dos lenguajes de programación.

En cuanto a los lenguajes reconocidos por esta plataforma, encontramos *Blockly*,el equivalente de *Scratch* pero creada por *Google*, un lenguaje de bloques. Su uso en colegios, para estudiantes de corta edad es cada vez más frecuente. También encontramos *Python* un lenguaje

¹<https://kibotics.org/>

²<https://jderobot.github.io/>

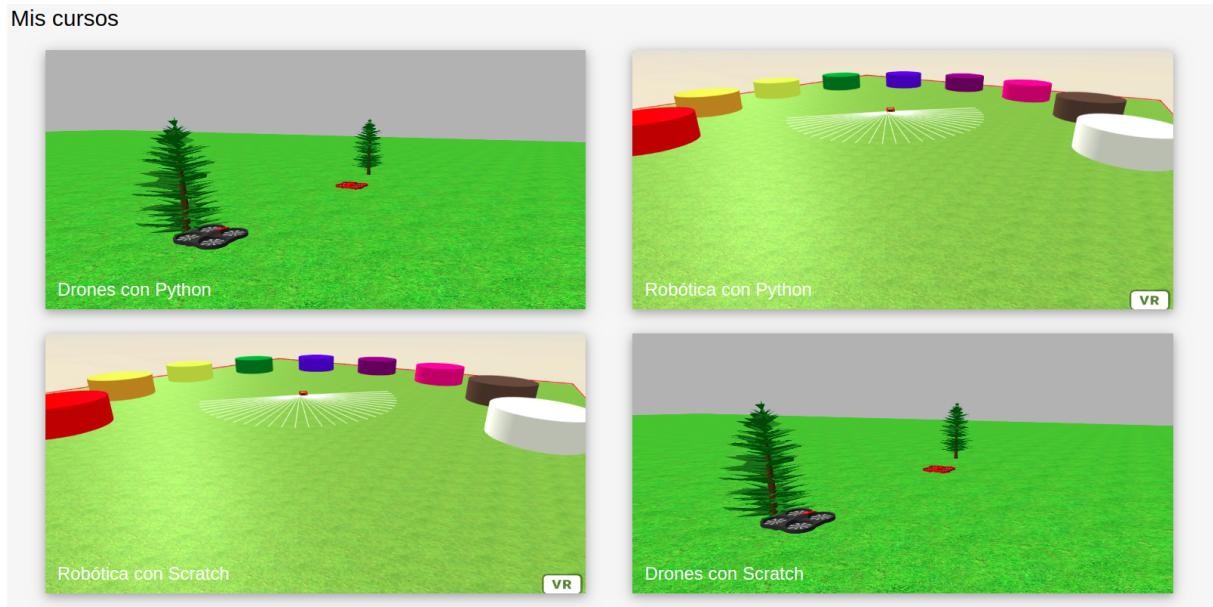


Figura 1.15: Menu principal de *Kibotics*

de alto nivel, de los más populares actualmente. Es el que emplean los estudiantes más mayores.

The image shows the exercise interface for 'Cuadrado con el Mbot' in Kibotics. The interface includes:

- Mbotics** logo and navigation icons.
- Cuadrado con el Mbot** title.
- Presentación** and **Teoría** tabs (with **Teoría** currently selected).
- Tiempo de estudio/Dificultad** section indicating 2 horas and a gear icon.
- El objetivo de este ejercicio es lograr que el Mbot realice un cuadrado.** (The objective of this exercise is to make the Mbot draw a square.)
- Pero... ¿Cómo podremos estar seguros de que la trayectoria del robot es un cuadrado perfecto?** (But... How can we be sure that the robot's trajectory is a perfect square?)
- El Mbot dispondrá de ayuda "visual". Utiliza las marcas rojas del terreno para guiar al robot en todo momento.** (The Mbot will have visual aid. Use the red markers on the ground to guide the robot at all times.)
- No hay restricciones para realizar la trayectoria, de modo que el cuadrado puede ser del lado que tú quieras, empezando por la marca que más te guste.** (There are no restrictions for the trajectory, so the square can be as large or small as you want, starting from the marker you like most.)
- Mbot** image.
- Simulator preview** showing the Mbot drawing a square path on a green field with red markers.

Figura 1.16: Ejercicio en Kibotics

Cada ejercicio viene con una parte teórica, donde se explica la lección que se va a aprender y los pasos para seguir para poder completar el ejercicio, la siguiente pestaña, es donde se programa el ejercicio, y la última, es donde se encuentra el simulador que ejecuta el código que acabas de programar.

Capítulo 2

Objetivos

Una vez explicado en el ámbito en el que se realiza este proyecto, en este capítulo explicaremos los objetivos que se han tratado de alcanzar y el método de trabajo que se ha seguido para lograrlo

2.1. Objetivos del TFG

El objetivo de este trabajo es dar soporte en la plataforma de *Kibotics*¹ a un robot bastante popular para la robótica educativa como es el *LEGO Ev3*, esto significa integrarlo de forma que se pueda programar dentro de la plataforma, y que también funcione para el robot real.

Soporte Simulado

- Añadir una simulación 3D realista sabiendo que hay modelos de robots prediseñados por *LEGO* con estructuras que cambian dependiendo de los sensores que lleven. Por lo que, al menos, tendremos que crear un modelo por cada tipo de sensor que pueda llevar. Y que esta simulación tenga sentido físico dentro del entorno.
- Añadir la infraestructura necesaria como *drivers*, y funciones al *Robot API* para que el robot sea programable en cualquier lenguaje soportado por la plataforma, y funcione en el robot real

Soporte Real

¹<https://kibotics.org/>

- Crear los *drivers* de *Kibotics* que sean necesarios para que el robot *LEGO EV3* real para que sea capaz de ejecutar código en Python, que implemente funciones del *HAL API* de *Kibotics*.
- Instalar un software necesario en el robot para lanzar un servidor en el robot capaz de recibir mensajes con el código, y lo transforme en un archivo y lo ejecute dentro de la máquina. Lo que conlleva instalar una imagen de un sistema operativo en el *Lego ev3* en este caso, una distribución basada en *Debian Linux*, e instalar lo necesario para que se ejecuten programas en *Python*.

Validación Experimental

Toda la funcionalidad que implemente debe ser comprobada mediante ejercicios que se realicen tanto en la plataforma, como en el robot real. Y serán mostrados al final de cada subobjetivo anterior.

2.2. Requisitos

Para completar la integración del robot **LEGO Ev3** se necesitan ciertos requisitos que cumplir:

- Dentro del **LEGO Ev3** debe correr una distribución de *Linux* .
- Se deben crear todas las funciones necesarias para que todos los sensores incluidos en *LEGO Ev3*, tengan equivalencia en la plataforma de *Kibotics*.
- El resultado final debe ser lo más sencillo posible, no debe requerir configuraciones adicionales.

2.3. Metodología

La metodología para completar el trabajo de fin de grado se puede dividir en diferentes fases que se iban repitiendo cada cierto tiempo, en cada una de ellas, semanalmente, tenía lugar una reunión con el tutor del trabajo para determinar los siguientes objetivos a cumplir y evaluar las tareas propuestas en anteriores sesiones. Esto ayuda mucho en proyectos como este, en constante desarrollo.

Este proyecto se lleva a cabo con un equipo de trabajo, que se ocupa de la plataforma de *Kibotics*, cada uno con sus labores y ocupaciones. Por lo tanto, es necesaria la comunicación y realimentación con el resto de integrantes. Para ello se utiliza la herramienta *Slack*² en la que los desarrolladores están en contacto en todo momento, no solo para comunicar avances, si no también para ayudar en todo momento si surge algún contratiempo en el desarrollo.

Para trabajar en local, con el repositorio original me hice *git clone* de los repositorios que necesitaba, e iba trabajando sobre ellos, guardando los cambios en un repositorio creado solo para el trabajo de fin de grado³ hasta que la funcionalidad estaba completa, que era cuando se subían al repositorio principal. Esta metodología, es el llamado modelo iterativo de desarrollo de software.

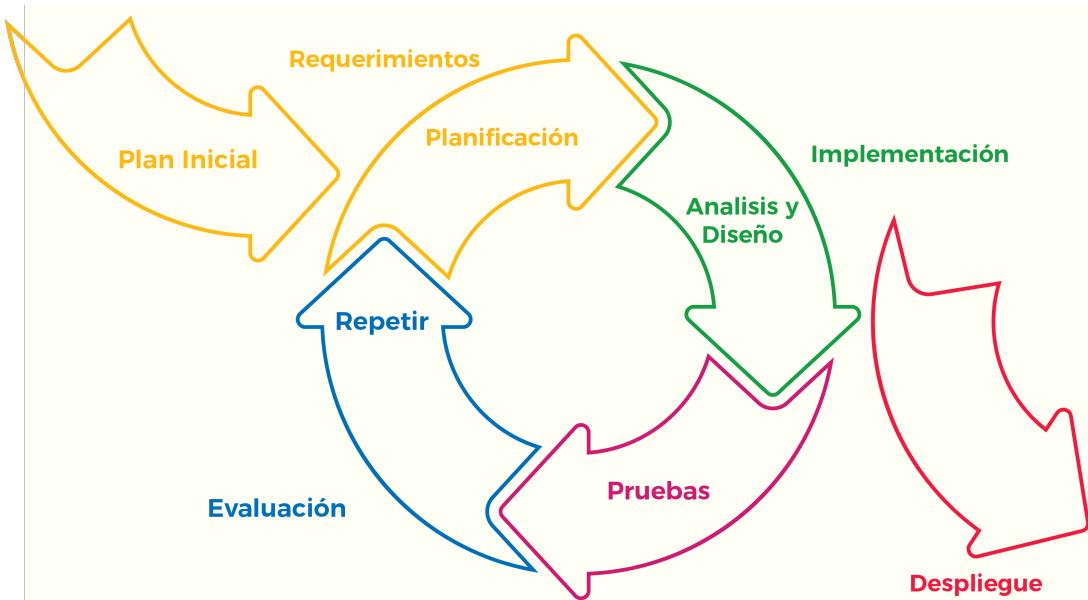


Figura 2.1: Metodología de modelo iterativo

Cuando los cambios eran revisados por el tutor, se seguía una dinámica de *Incidencia* y *parche* en el repositorio principal. Para ello se creaba una incidencia (*issues*) con el tema que se iba a solucionar o añadir en la funcionalidad de la plataforma, y una vez resuelto en local y para cerrarla, se creaba una rama (*branch*) creando parche (*pull request*) para que un desarrollador principal de *Kibotics*, lo aceptará y fusionará con la rama principal y así arreglar la incidencia. Esto se hace para registrar todos los cambios, y comprobarlos antes de que se trabaje

²<https://slack.com/>

³<https://github.com/RoboticsLabURJC/2020-tfg-daniel-pulido>

directamente en el repositorio

2.4. Plan de trabajo

El plan de trabajo a seguir para conseguir el objetivo se puede dividir en los siguientes pasos:

- *Paso 1:* Aterrizaje en *Kibotics*. Lo primero que hay que hacer es familiarizarse con el entorno con el que se va a trabajar. *Kibotics* es una plataforma web en la que entraremos mas en detalle en el siguiente capítulo
- *Paso 2:* Comienzo de la creación del robot simulado. Comenzaremos por la creación de varios modelos 3D para usarlos en el entorno simulado, tenemos que crear varios modelos porque *LEGO EV3* esta diseñado para poder construir tu robot en base a la tarea que vaya a realizar, esto significa, un modelo por cada sensor que utilices en el robot. Además de la creación de los ejercicios que lleven a probar las funciones implementadas, para probar su funcionamiento, así como para añadirlos a la plataforma.
- *Paso 3:* Desarrollo de las funciones y ejercicios dentro de la aplicación de *Kibotics* para crear una dinámica de trabajo con el nuevo robot.
- *Paso 4:* Dar soporte al robot real para poder programarlo en *python* desde el exterior, y la instalación de un server para que pueda recibir código y ejecutarlo en local.
- *Paso 5:* Creación de los *drivers* que hagan de traductor entre lo que creas en la plataforma, y lo que entiende el robot. Validar que funciona este envio de programas,con la ejecución de ejercicios programados en el editor del navegador.

Capítulo 3

Herramientas

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

3.1. Lego Ev3

Dentro de los robots de *LEGO* el que mas versatilidad, además de más potencia de procesamiento y posibilidades en las opciones de creatividad a la hora de crear robots es *LEGO MINDSTORMS Education*(Pack en el que viene integrado el *LEGO Ev3*) tambien trae una mayor gama de sensores, por no decir que es uno de los lideres en la educación STEM (siglas en inglés de Ciencias, Tecnología,Ingeniería y Matemática), En el centro de *LEGO MINDSTORMS Education* se encuentra el Bloque EV3, el bloque inteligente programable que controla motores y sensores y además proporciona comunicación inalámbrica como quiera que sea.

Descripción general

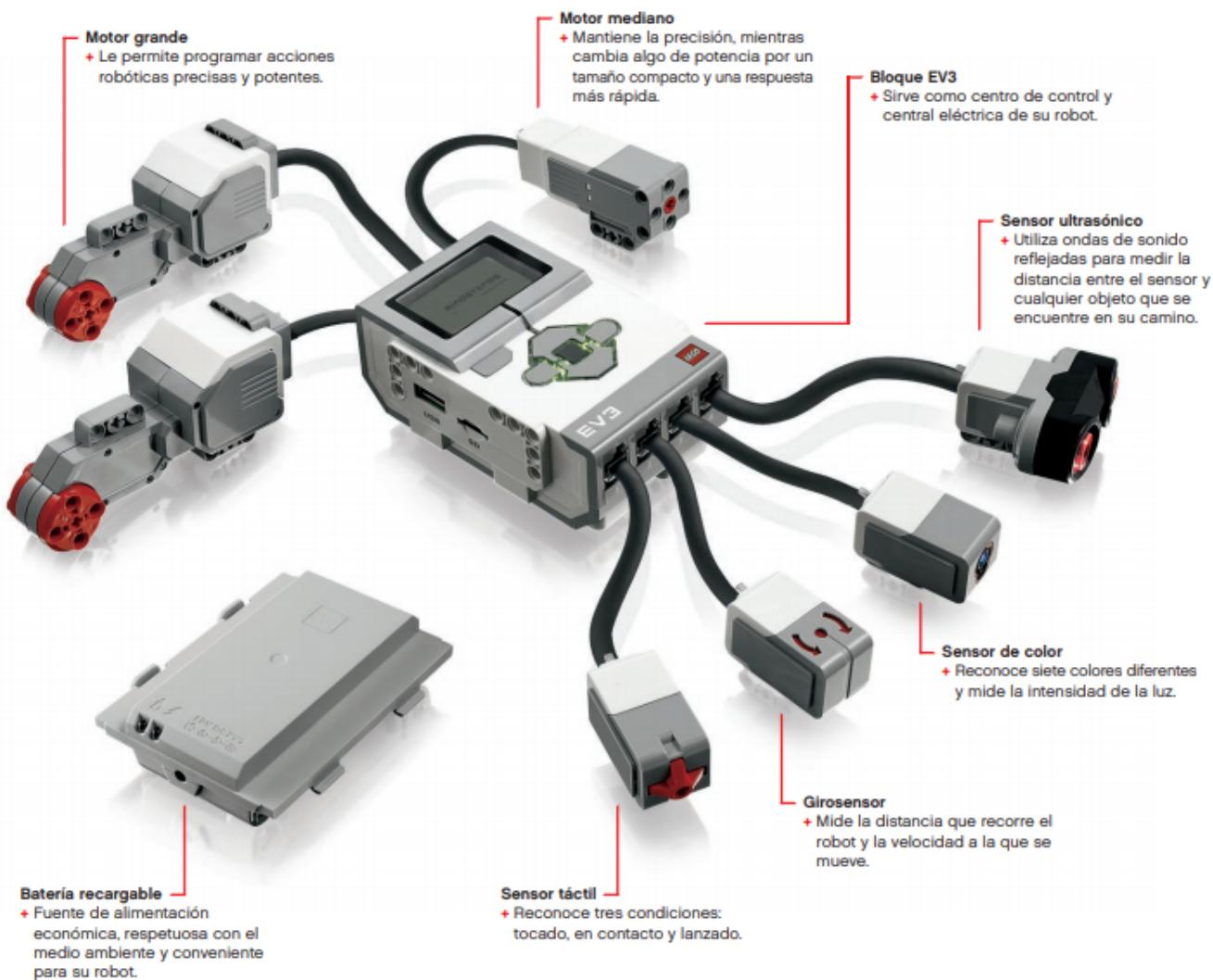


Figura 3.1: Conjunto Ev3

Esto quiere decir que las posibilidades a la hora de crear diferentes robots, con diferentes configuraciones de sensores, son prácticamente infinitas. Pero vamos a centrarnos en los casos que mas funcionalidad tienen. El robot que mas versatilidad presenta a la hora de superar ejercicios, es un robot triciclo, con dos ruedas delanteras y una pivotante trasera. Así que este será nuestro modelo para el robot. Y ademas como el *kit de LEGO MINDSTORMS* viene con tres sensores diferentes, crearé tres modelos para integrarlos por separado en la plataforma.

3.2. Python

Python es un lenguaje de programación administrado por la Python Software Foundation. Posee una licencia de código abierto, denominada Python Software Foundation License. Es un lenguaje interpretado, por lo que no se necesita compilar el código fuente para poder ejecutarlo. Esto ofrece ventajas como la rapidez de desarrollo e inconvenientes como una menor velocidad. En ciertos casos, cuando se ejecuta por primera vez un código, se producen unos bytecodes que se guardan en el sistema y que sirven para acelerar la compilación implícita que realiza el intérprete cada vez que se ejecuta el mismo código. Lenguaje muy popular en los últimos años gracias a la cantidad de librerías que contiene, tipos de datos y funciones incorporadas en el propio lenguaje, que ayudan a realizar muchas tareas habituales sin necesidad de tener que programarlas desde cero. Además, su filosofía hace hincapié en una sintaxis que favorezca un código legible, facilitando su aprendizaje. Este lenguaje nos es muy importante en este proyecto ya que es el lenguaje nativo del robot y en el que van a ir programados los *drivers* para el robot real.

3.3. HTML

HTML (HyperText Markup Language) fue desarrollado en 1991 por Tim Berners-Lee mientras trabajaba en la Organización Europea para la Investigación Nuclear (CERN), y popularizado por el navegador Mosaic desarrollado en NCSA (Raggett, Le Hors y Jacobs, 1999).

Es un lenguaje de marcado que sirve para estructurar una página Web. Permite poner etiquetas a diferentes partes de la página para darles la misma apariencia. También tiene tipos de funciones para personalizar el tipo de letra y la forma que tienen en pantalla.

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. Estas ramas se pueden ir extendiendo sea necesidad del proyecto web.

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="utf-8">
5     <title>Mi pagina de prueba</title>
6   </head>
7   <body>
```

```
9     
10    </body>
11 </html>
```

Listing 3.1: Ejemplo de funcionamiento HTTP

3.4. JavaScript

JavaScript es un lenguaje de programación interpretado de alto nivel que se encuentra bajo el estándar ECMA Script. Este lenguaje es comúnmente conocido por su uso en los scripts de las páginas web. Es un lenguaje tipado débil y dinámico. Está diseñado para ser utilizado en páginas Web, actualmente existen otras posibilidades de ejecutar *JavaScript* en todo tipo de aplicaciones haciendo uso de *Nodejs*.

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.

Este es el lenguaje en el que están programados los *drivers* de los robots implementados en *Kibotics*.

3.5. A-Frame

A-Frame es un entorno de código abierto destinado a crear experiencias de realidad virtual, siendo una de las comunidades creadoras de contenido para realidad virtual más grandes del mundo. Esto se debe a que es un entorno soportado por la mayoría de gafas de VR(*Virtual Reality*) como *OculusRift*, *HTC Vive* o *GearsVR*.

Esta creado a partir de *HTML* de forma que sea sencillo de leer y comprender. De esta manera es accesible para crear una gran comunidad. *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

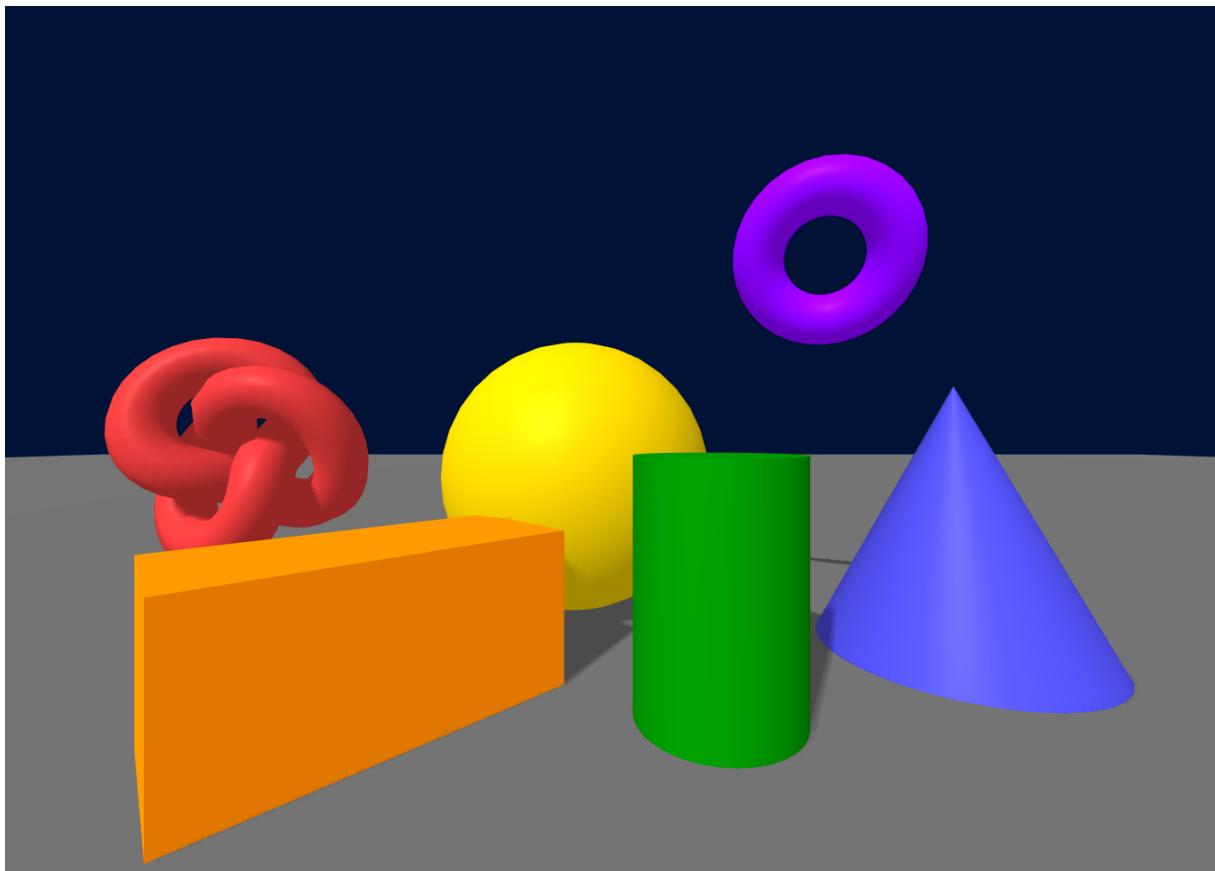


Figura 3.2: Algunos ejemplos de *A-Frame*

que agrega modifica sus datos

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.

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

Existe un almacén web desde donde es posible descargarse una gran variedad de modelos

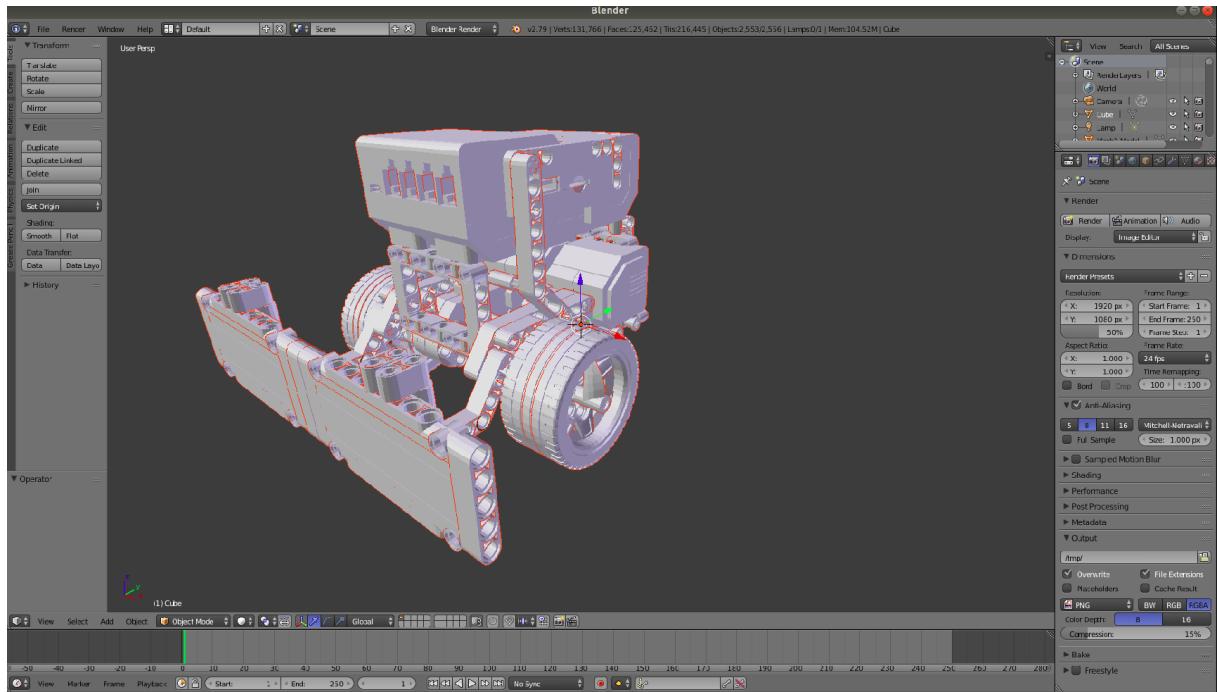


Figura 3.3: Interfaz gráfica de *Blender*

y escenarios, además de desarrollar los propios. Una vez desarrollado el modelo o escenario, estos editores exportan el modelo o escenario en formato ”gltf”, que es el formato soportado por *A-Frame* y estaría listo para insertarlo en el entorno simulado.

Los escenarios y modelos generados por estos editores son creados mediante la intersección de líneas, generando los distintos tipos de objetos. Es posible adjuntar una textura o color a cada cara que forma el objeto. Además, el editor *Blender*, al ser más complejo, permite la introducción de iluminación y trabajar con formas geométricas entre dimensiones directamente.

Este es el programa que usaremos para crear los modelos del robot simulado.

3.7. Simulador WebSim

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

3.7.1. Diseño

El simulador hace uso del entorno *A-Frame* y su diseño permite conectar un editor de texto o un editor de bloques para 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. En la figura 3.4 se puede ver el diseño de *WebSim*.

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 diferentes lenguajes de programación *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*.
- 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. 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.

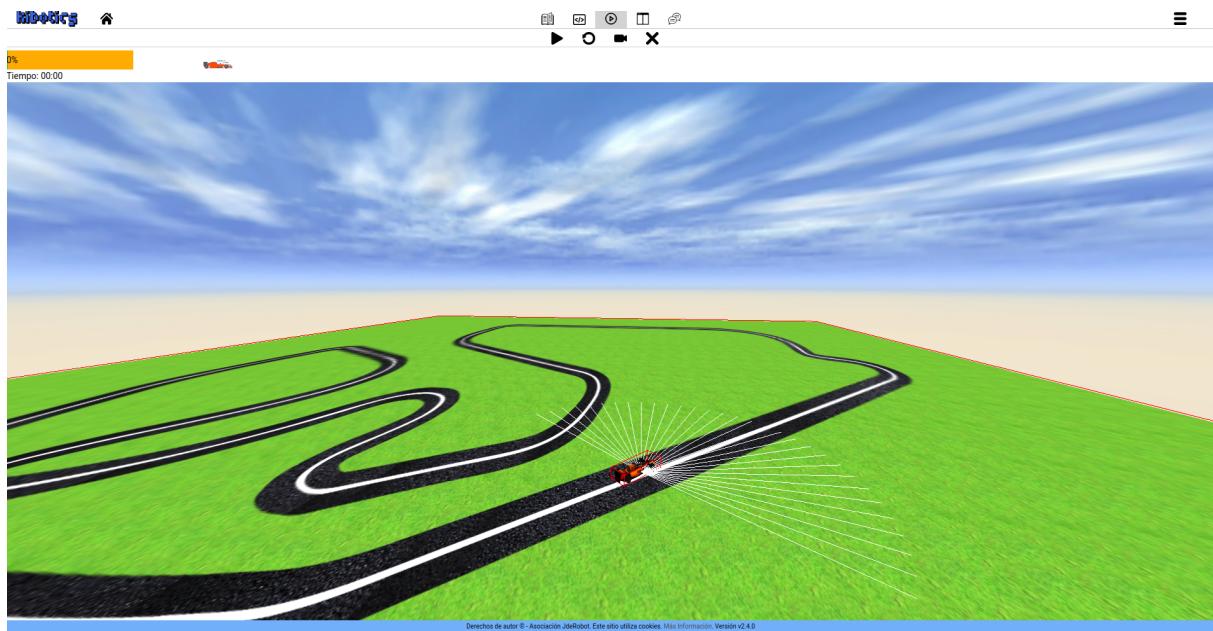


Figura 3.4: Interfaz gráfica de *WebSim*

3.7.2. Drivers de sensores

El robot consta de sensores simulados con *A-Frame*. Los drivers permiten que el usuario, vía *JavaScript*, pueda acceder a estos y obtener su información. En este entorno disponemos de los siguientes 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 elemento detectado por la cámara del color pasado por parámetro	{center:[x,y], area: int}
.getObjectColorRGB(valorBajo,valorAlto)	Devuelve un objeto con la posición del elemento detectado por la cámara con los valores pasados por parámetro	{center:[x,y], area: int}

3.7.3. 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 órdenes constantemente, si no que es suficiente con mandar la instrucción una vez y el robot seguirá ejecutándola hasta que reciba una nueva.

En la tabla 4.1 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.

Estos seran los *drivers* que tenga que agregar a la funcionalidad tanto en el robot simulado, programado en *JavaScript* y el real que tendrá unos equivalentes en *Python*.

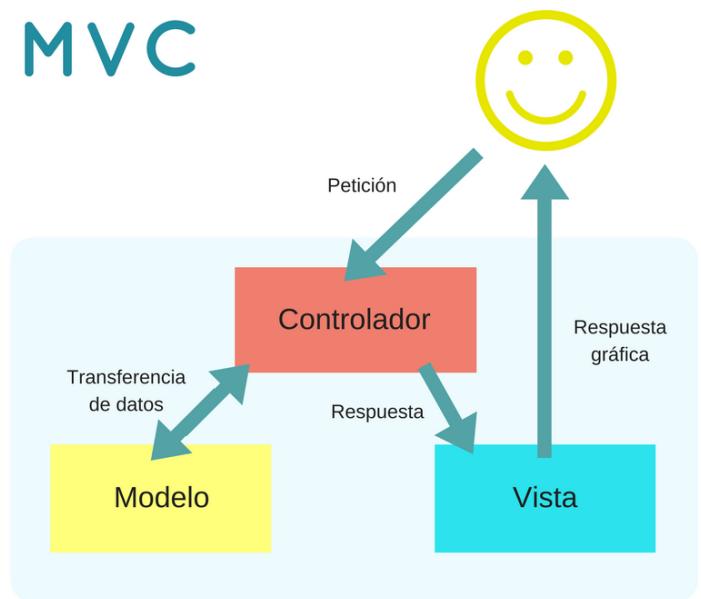
3.8. Django

Django es un framework de código abierto escrito en Python, creado en 2005, el cual permite desarrollar un entorno web complejo de una forma rápida y estructurada. Actualmente es mantenido por Django Software Foundation y se encuentra en la versión 3.0 (Novapros, 2020).

Sigue una arquitectura MVC (Modelo – Vista - Controlador), como se observa en la figura(Holovaty y Kaplan-Moss, 2007).

- *Modelo*. Los modelos de datos creados están mapeados directamente a las tablas de la base de datos, permitiendo aislar el código de la aplicación de la base de datos.
- *Vista*. Corresponde con la capa de presentación, y está basada en plantillas HTML.
- *Controlador* (*en Django llamado “views”*). Responsable de seleccionar la plantilla a mostrar. Atiende a una petición y, según el mapeo de la URL, (Uniform Resource Locator) redirige a una vista u otra.

La meta fundamental de Django es facilitar la creación de sitios web complejos. Django pone énfasis en el re-uso, la conectividad y extensibilidad de componentes, el desarrollo rápido y el principio No te repitas (DRY, del inglés Don't Repeat Yourself). Python es usado en todas las



[coding or not](http://codingornot.com)

Figura 3.5: Arquitectura Django

partes del framework, incluso en configuraciones, archivos, y en los modelos de datos. Ofrece una serie de características bastante interesantes como es su excelente capa de seguridad (p ej. permite una protección contra los ataques maliciosos “Cross-site request forgery”), dispone de un sistema de administrador “por defecto,” sin necesidad de realizar ningún tipo de configuración. También proporciona una interfaz para el acceso a la base de datos, facilitando las consultas (Novapros, 2020).

3.9. Flask

Flask, lanzado en abril de 2010 es, junto con Django, uno de los framework webs más famosos escritos en Python.

Está enfocado en proporcionar lo mínimo necesario para poner en funcionamiento una aplicación, por eso se le considera un framework “minimalista”. Por otro lado, no requiere de otras dependencias para realizar acciones básicas, de manera que la curva de aprendizaje para su comprensión es muy baja. Debido a su sencillez en la estructura, posee una velocidad mayor que Django y es bastante útil para iniciarse en el aprendizaje de desarrollo web, permitiendo

crear servidores de una forma rápida. Debido a ello, Flask está orientado al sector servicios, los cuales implican muchas visitas y una carga grande de peticiones, y también para proyectos personalizados o sencillos (Rodríguez, 2019).



Figura 3.6: Logo Flask

Capítulo 4

Soporte Simulado

Ahora que tenemos definidos cuáles van a ser nuestros objetivos y la infraestructura que vamos a utilizar para llevarlos acabo, en este capítulo se va a hablar del proceso de integración en la plataforma del robot simulado. Primero se describe cómo se ha materializado el *LEGO EV3* en el simulador *WebSim* de la plataforma *Kibotics*. Después se describe la implementación realizada de la interfaz de programación a través de la cual el estudiante recoge las lecturas de los sensores simulados u ordena movimiento a los motores simulados. Finalmente , a modo de validación experimental, se presentan los tres ejercicios desarrollados que utilizan esa interfaz de programación del robot simulado.. Pero antes de eso se darán unas nociones básicas de las características de este robot, para poder entender, el proceso de desarrollo.

4.1. Modelo 3D

Para simular el robot, el primer paso es crear un modelo tridimensional que puedas incluir en el entorno de *A-Frame*. Para ello se ha utilizado la aplicación de *Blender*, primero, busqué en el almacén Web (propio de la plataforma) un modelo del *Bloque Ev3* al que le pudiera insertar, los modelos de los brazos delanteros, sensores y piezas sueltas necesarias para completar el modelo de *Robot Lego Base*.

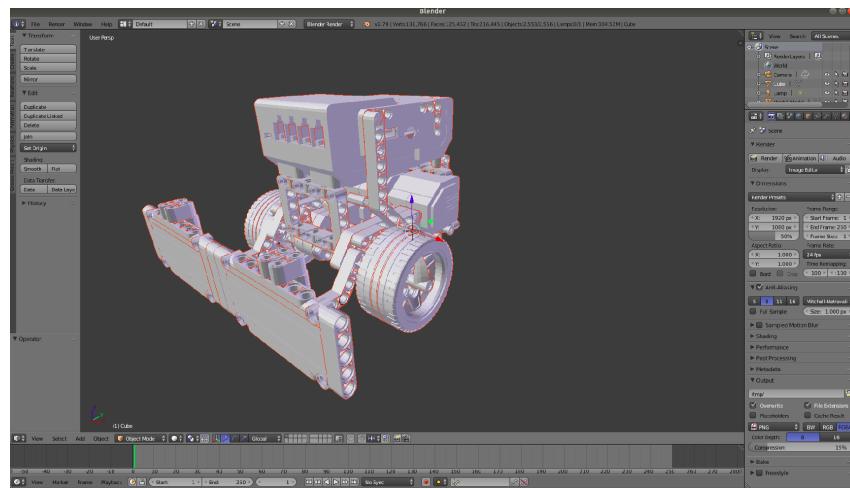


Figura 4.1: Robot base

Una vez creado este modelo, el número de aristas era demasiado alto al estar formado por muchos pequeños modelos de piezas, y hace que el modelo completo, pese demasiado, y los tiempos de carga sean demasiado largos. Además de que en *Blender* existen focos de luz, que crean reflejos en el modelo, que no son necesarios para la implementación en *A-Frame*.

Para solucionar esto y mejorar el modelo se realizan las siguiente modificaciones:

- Reducción del número de polinomios para reducir el número de aristas.
- Cambiar las texturas reflectantes por texturas mates.
- Dar color al robot imitando los colores del *LEGO EV3* real.

Una vez realizados estos cambios se procede a insertar en el modelo tridimensional simulando los tres tipos de sensores que incluye el robot *LEGO EV3* y los cambios en el modelo que sean necesarios para que tenga sentido la instalación del sensor.



Figura 4.2: Robot con sensor de color

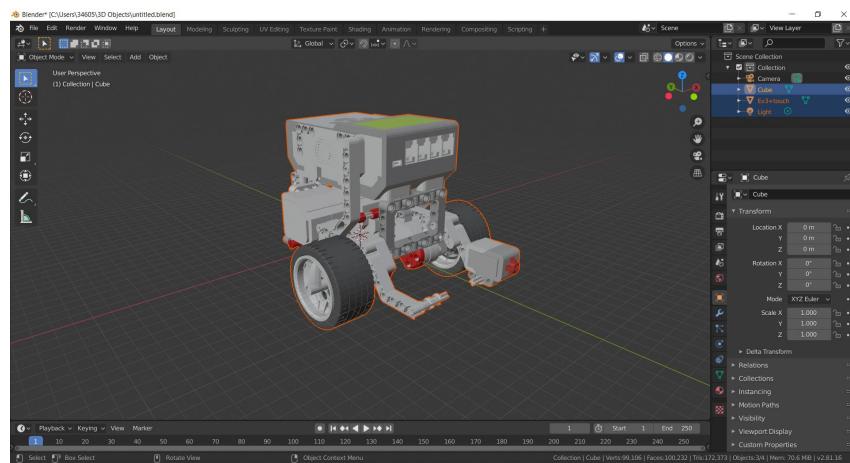


Figura 4.3: Robot con sensor tactil

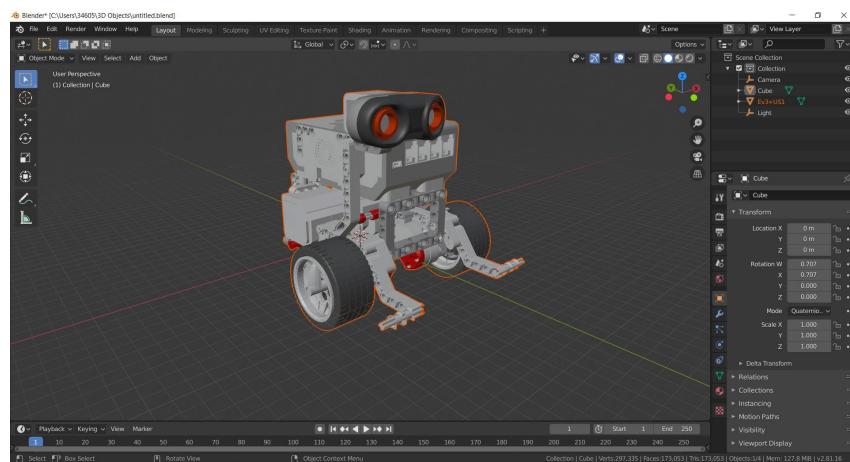


Figura 4.4: Robot con sensor de ultrasonidos

4.2. Interfaz de programación de sensores y actuadores

4.2.1. Soporte de actuadores

Los motores son los actuadores del *LEGO EV3*, son los que, a la orden del controlador, realizan el movimiento para completar su tarea asignada. En el kit de *LEGO* vienen incluidos dos tipos de motores, motor grande y motor mediano.

Los motores grandes, que son los que vamos a utilizar en el simulador, se suelen utilizar en pareja, para mover ambas ruedas del robot. A esto le llamamos Tramo Diferencial y se implementan las funciones pensando en este principio. Por ejemplo, si ejecutamos la función *GirarDerecha()*, lo que va a ejecutar internamente es frenar el motor derecho, acelerar el motor izquierdo, y así hacer que el robot gire a la derecha. Este pensamiento nos ayuda a implementar el *HAL API* de *Kibotics* en este nuevo robot.



Figura 4.5: Motores

Actuadores que vienen incluidos

Cuadro 4.1: 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.

Estas son las funciones que tiene el *HAL API* y que ofrece el *LEGO* simulado y que han sido programadas en *JavaScript*.

4.2.2. Soporte de sensores

Se han analizado por separado los sensores para saber que funciones hay que añadir en los *drivers*. Estos *drivers* están programados en *JavaScript* permiten al usuario acceder a los sensores y obtener información del robot simulado.

Sensor de color

El Sensor de color es un sensor digital que puede detectar el color o la intensidad de la luz que ingresa por la pequeña ventana de la cara del sensor. Este sensor puede utilizarse en tres modos diferentes: Modo color, Modo intensidad de la luz reflejada y Modo intensidad de la luz ambiental.

La tasa de muestreo del sensor de color es de 1 kHz.



Modo color



Modo intensidad de la luz reflejada



Modo intensidad de la luz ambiental

- **En Modo color**, el Sensor de color reconoce siete colores: negro, azul, verde, amarillo, rojo, blanco y marrón, además de Sin color. Esta capacidad de diferenciar los colores significa que su robot puede estar programado para clasificar pelotas o bloques de colores, y realizar acciones diferentes con cada color detectado.

Este tipo de acciones, están ya contempladas en la funcionalidad del *HAL API* de *Kibotics*. Aunque en este caso utiliza una cámara simulada para comprobar qué color está viendo.

getImage(cameraID): Método que devuelve *robot*.

```

1   function getImage (cameraID) {
2     /**
3      * Returns a screenshot from the robot camera
4      */
5     if (!cameraID || (this.camerasData.length === 1) ||
6         (cameraID > this.camerasData.length - 1)) {
7       return this.camerasData[0]['image'];
8     } else {
9       return this.camerasData[cameraID]['image'];

```

Figura 4.6: Sensor color

Sensor de color en sus 3 usos

4.2. INTERFAZ DE PROGRAMACIÓN DE SENSOR RÁPICTA Y ASORTE SIMULADO

```
10      }
11
12 }
```

El *LEGO EV3* no tiene una cámara instalada, pero para el robot simulado, es lo más sencillo de implementar, ya que puede analizar la imagen simulada y sacar el color RGB, para posteriormente dar nombre al color que ve.

getColorRGB(): Método que devuelve *RGB* en tres valores.

```
1 function getObjectColorRGB(lowval, highval) {
2 /**
3 * This function filters an object in the scene with a given color, uses OpenCVjs to
4     filter
5 * by color and calculates the center of the object.
6 *
7 * Returns center: CenterX (cx), CenterY (cy) and the area of the object detected in the
8     image.
9 */
10
11 if (lowval.length === 3) {
12     lowval.push(0);
13 }
14 if (highval.length === 3) {
15     highval.push(255);
16 }
17 var image = this.getImage();
18 var binImg = new cv.Mat();
19 var M = cv.Mat.ones(5, 5, cv.CV_8U);
20 var anchor = new cv.Point(-1, -1);
21 var lowThresh = new cv.Mat(image.rows, image.cols, image.type(), lowval);
22 var highThresh = new cv.Mat(image.rows, image.cols, image.type(), highval);
23 var contours = new cv.MatVector();
24 var hierarchy = new cv.Mat();
25
26 cv.morphologyEx(image, image, cv.MORPH_OPEN, M, anchor, 2,
27     cv.BORDER_CONSTANT, cv.morphologyDefaultBorderValue()); // Erosion followed by
28     dilation
29
30 cv.inRange(image, lowThresh, highThresh, binImg);
31 cv.findContours(binImg, contours, hierarchy, cv.RETR_CCOMP, cv.CHAIN_APPROX_SIMPLE);
32 if (contours.size() > 0) {
33
34     let stored = contours.get(0);
35     var objArea = cv.contourArea(stored, false);
```

```

33
34     let moments = cv.moments(stored, false);
35     var cx = moments.m10 / moments.m00;
36     var cy = moments.m01 / moments.m00;
37
38 }
39 return {center: [parseInt(cx), parseInt(cy)], area: parseInt(objArea)};
40 }

```

Una vez realizado este paso podemos ponerle un nombre al color, como hace el *LEGO EV3* real.

- **En Modo intensidad de la luz reflejada**, el Sensor de color mide la intensidad de la luz que se refleja desde una lámpara emisora de luz color rojo. El sensor utiliza una escala de 0 (muy oscuro) a 100 (muy luminoso). Esto significa que su robot puede estar programado para moverse sobre una superficie blanca hasta detectar una línea negra o para interpretar una tarjeta de identificación con código de color. Esto en el robot simulado, es diferente, ya que no podemos ver como una magnitud física como es la luz se refleja un objeto, pero este efecto depende del color que se esté viendo en la imagen, la luminosidad del color se puede calcular con esta función:

getLightness(valueMin, valueMax): Método que devuelve *Luminosidad*.

```

1   function getLightness(valueMin, valueMax) {
2
3     let image = this.getObjectColorRGB(valueMin, valueMax);
4     let L = ((image.center[0]- image.center[1])/2)*100/255;
5
6     /**
7      * Returns lightness with a percent
8     */
9
10    return L;
11
12 }

```

Esta función puede resultar útil, para, por ejemplo, poder seguir una línea, cuando haya colores muy similares, o cuando se esté utilizando el robot en una mesa o superficie alta, detectar antes, donde está el borde.

- **En Modo intensidad de la luz ambiental**, el Sensor de color mide la intensidad de la luz que ingresa en la ventana desde su entorno, como la luz del sol o el haz de una linterna.

4.2. INTERFAZ DE PROGRAMACIÓN DE SENSOR RÁPIDO Y ASORTE SIMULADO

El sensor utiliza una escala de 0 (muy oscuro) a 100 (muy luminoso). Esta funcionalidad, no puede ser implementada en la plataforma, ya que no tenemos un foco de luz que se puede analizar, ni tampoco una magnitud dentro del entorno que represente la luz.

Sensor de ultrasonido

El Sensor ultrasónico es un sensor digital que puede medir la distancia a un objeto que se encuentra frente a él. Para hacerlo, envía ondas de sonido de alta frecuencia y mide cuánto tarda el sonido en reflejarse de vuelta al sensor. La frecuencia de sonido es demasiado alta para el oído humano. La distancia a un objeto puede medirse en pulgadas o centímetros.

Esto le permite programar su robot para que se detenga a una distancia determinada de una pared. Al utilizar unidades en centímetros, la distancia detectable es entre 3 y 250 centímetros (con una exactitud de +/- 1 centímetro). Un valor de 255 centímetros significa que el sensor no puede detectar ningún objeto frente a él.

En *WebSim* la implementación que hay para las distancias es usar un *RayCaster*. Que es equivalente a poner láseres apuntando hacia todas direcciones por delante del robot de esta forma:



Figura 4.7: Sensor de ultrasonidos

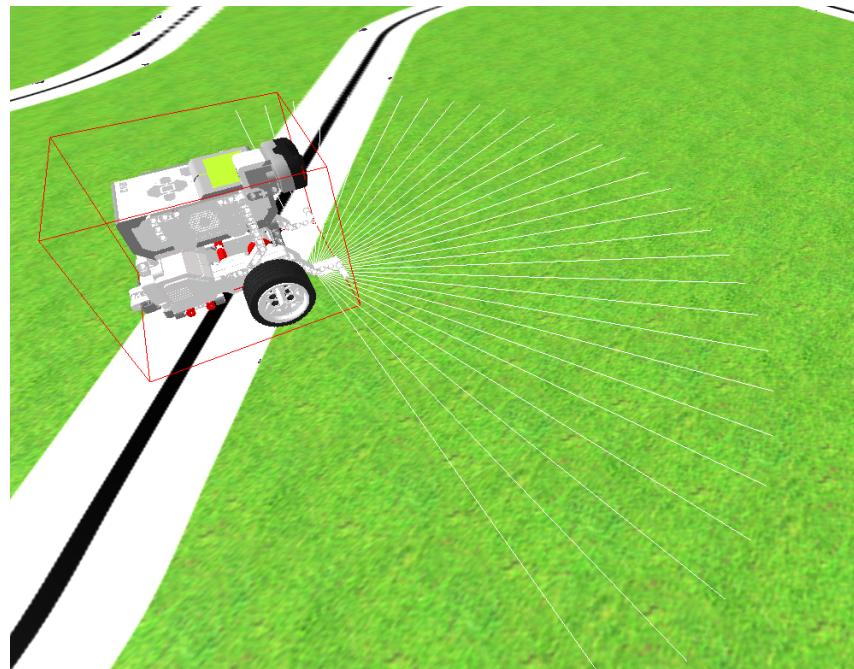


Figura 4.8: Raycaster

Lo cual devuelve un *array* de valores con las distancias a las que rebotaçada láser. Esto tratándose de un sensor ultrasonidos que solo devuelve un único valor con el obstáculo más cercano, es un poco irreal. Por lo que creare la función *getDistance* que devuelve el valor más cercano de todos los que hay en el *array* de *RayCaster*

Funciones que calculan la distancia: Método que devuelve un único valor con la distancia más corta.

```
1 function getDistance() {
2
3     /**
4      * This function returns the distance for the raycaster in the center of the arc of rays.
5      */
6
7     var distances = this.getDistances();
8
9
10    if (distances[13] !== 10 || distances[14] !== 10 || distances[15] !== 10 || distances[16]
11        !== 10 || distances[17] !== 10) {
12
13        let distance0 = 100;
14
15        let distance1 = 100;
16
17        let distance2 = 100;
18
19        let distance3 = 100;
20
21        let distance4 = 100;
22
23        if (distances[13] !== 10) {
24
25            distance0 = distances[13];
26
27        }
28
29        if (distances[14] !== 10) {
```

4.2. INTERFAZ DE PROGRAMACIÓN DE SENSOR RÁPIDO Y ALGORITMO SIMULADO

```
17         distance1 = distances[14];
18     }
19     if (distances[15] !== 10) {
20         distance2 = distances[15];
21     }
22     if (distances[16] !== 10) {
23         distance3 = distances[16];
24     }
25     if (distances[17] !== 10) {
26         distance4 = distances[17];
27     }
28     let min_distances = [distance0, distance1, distance2, distance3, distance4];
29     Array.min = function (array) {
30         return Math.min.apply(Math, array);
31     };
32     return Array.min(min_distances);
33 } else {
34     return 10;
35 }
36 }
37
38 function getDistances() {
39     /**
40      * This function returns an array with all the distances detected by the rays.
41      */
42     var distances = [];
43     for (var i = 0; i <= 31; i++) {
44         distances.push(10);
45     }
46     var groups = ["center", "right", "left"];
47     for (i = 0; i < groups.length; i++) {
48         this.distanceArray[groups[i]].forEach((obj) => {
49             if (typeof obj.d != "undefined") {
50                 distances[obj.id] = obj.d;
51             }
52         });
53     }
54     return distances;
55 }
```

Sensor de contacto

El Sensor táctil es un sensor analógico que puede detectar el momento en el que se presiona y se lanza el botón rojo del sensor. Esto significa que el Sensor táctil puede programarse para actuar según tres condiciones: presionado, lanzado o en contacto (tanto presionado como lanza-

do). Con la información del Sensor táctil se puede programar un robot para ver el mundo como lo haría una persona no vidente, es decir, extendiendo un brazo y respondiendo cuando toca algo (presionado).

Se puede construir un robot con un Sensor táctil presionado contra la superficie. Luego, puede programar el robot para que responda (se detenga) cuando esté a punto de pasar el borde de la mesa (cuando el sensor se lanza). Un robot de pelea puede programarse para continuar empujando hacia adelante en dirección a su oponente hasta que este se retire. Ese par de acciones, presionado y lanzado, constituyen el estado En contacto.

En la plataforma no había ninguna función similar que implementara un sensor de contacto. Así que, se han creado un par de funciones, que aprovechándose del *array* de distancias, cogerán la distancia central y cuando sea mínima, darán por hecho que el robot está tocando la superficie

```

1     getCenterDistance () {
2
3         if (this.distanceArray["center"] [0] != null) {
4             return this.distanceArray["center"] [0].d;
5         } else {
6             return 10;
7         }
8     }
9     isTouching () {
10        return (this.getCenterDistance () < 3);
11    }

```



Figura 4.9: Sensor táctil

Esta función al ser totalmente nueva, habrá que añadirla en el *HAL API*, el cual está programado en *JavaScript* y tendrá que tener su equivalente en *Python* y en *Blockly*, esto significa que hay que traducirla en diferentes idiomas y añadirla a la lista de bloques que son visibles en el

editor visual. Esta implementación la mostrare en la siguiente parte con la prueba de este sensor en un ejercicio.

Girosensor

El Girosensor es un sensor digital que detecta el movimiento de rotación en un eje simple. Si rota el Girosensor en la dirección que indican las flechas que se encuentran en la caja del sensor, este puede detectar la razón de rotación en grados por segundo. (El sensor puede medir una razón de giro máxima de 440 grados por segundo.) Entonces, puede utilizar la razón de rotación para detectar, por ejemplo, si gira una parte del robot o si el robot se cae.



Figura 4.10: girosensor

Además, el Girosensor registra el ángulo de rotación total en grados. Puede utilizar este ángulo de rotación para detectar, por ejemplo, cuánto ha girado su robot. Esta función le permite programar giros (sobre el eje que está midiendo el Girosensor) con una exactitud de +/- 3 grados en un giro de 90 grados.

Este sensor no requiere una implementación dentro de la plataforma ya que el único uso que puede darse dentro de la plataforma, es medir un giro de unos grados que pasas como argumentos. Y eso ya está implementado en el *HAL API*. El otro uso real que se le puede dar a este sensor es para que un robot mantenga el equilibrio dinámico, mientras se mueve. Pero los tres montajes elegidos de *LEGO Ev3* para implementar en *Kibotics* no tienen un centro de gravedad en el mantenerse.

4.3. Validación experimental con ejercicios

Se han añadido ejercicios para comprobar la funcionalidad de cada uno de los sensores anteriormente explicados

4.3.1. Sigue-líneas

Este ejercicio consiste en seguir una línea negra en el suelo sobre fondo blanco haciendo uso de la cámara del *robot*, que recoge las imágenes y las filtra para poder seguirla.

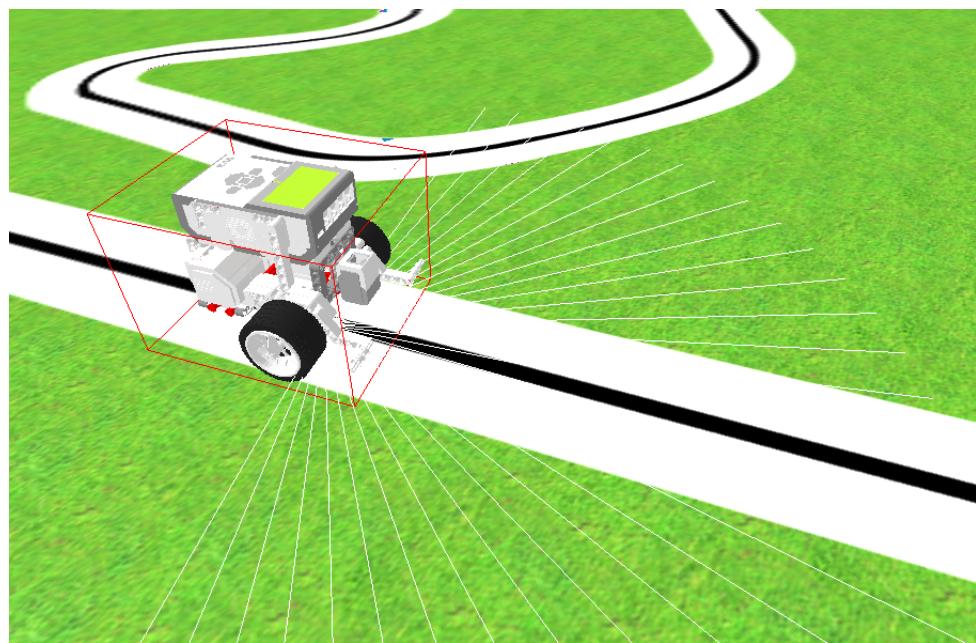


Figura 4.11: Escenario para el ejercicio *LEGO EV3 sigue-líneas*

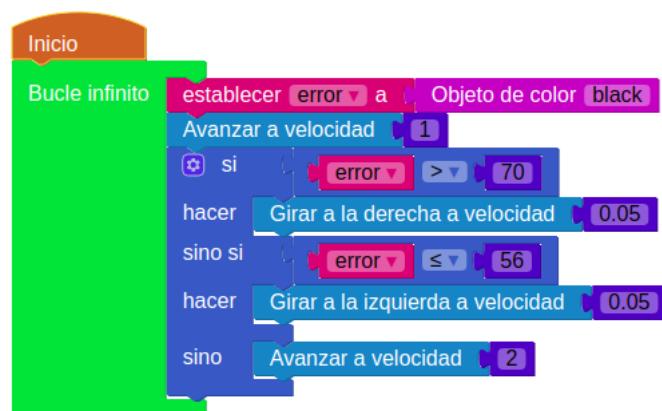


Figura 4.12: Solución en *Scratch* para el ejercicio *sigue-líneas*

En la solución se establece un error con el color negro, de modo que dependiendo del lado por el que me pase, corrijo girando hacia al lado contrario. En este ejemplo utilizo una actuali-

zación de funciones que ya existían, por lo que puedo programarlas en *Blockly*, habiendo hecho solo cambios en el *HAL API* que está programado en *JavaScript* para el robot *LEGO Ev3*.

4.3.2. Choca-gira

En este ejercicio hay programar al *robot* para que avance recto mientras no haya obstáculos haciendo uso del sensor de ultra-sonidos. Si encuentra un obstáculo tiene que detenerse, retroceder un poco, girar a la derecha y seguir adelante.

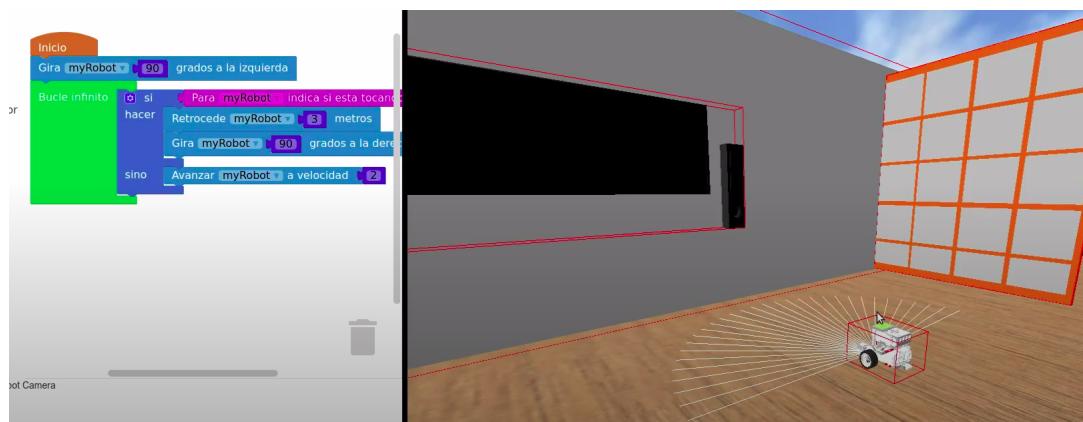


Figura 4.13: Solución en *Scratch* para el ejercicio choca-gira

En esta solución¹ se prueba el bloque anteriormente mencionada que equivale a la función *IsTouching* en el lenguaje *Scratch*. Cabe destacar que aquí estamos utilizando la nueva función que se ha creado para este sensor. En este caso, estamos utilizando el bloque "Para **myRobot** indica si está tocando" que es el bloque de *Blockly* equivalente a la función *IsTouching* en *JavaScript* del *HAL API*

4.3.3. Atraviesa-bosque

Ejercicio basado en atravesar un pasillo con diversos objetos que hay que esquivar. El sensor necesario es el ultrasonidos para detectar en qué posición se encuentra el siguiente obstáculo. Este ejercicio solo se puede hacer simuladamente, ya que el *LEGO EV3* con el sensor ultrasónico no puede saber en qué posición se encuentra el obstáculo, solo la distancia. Aún así, este

¹<https://youtu.be/figTbXKEXD4>

ejercicio tiene un gran interés educativo, por lo que he decidido añadirlo, y ademas crear una nueva función para que sea más sencillo e intuitivo de completar para el estudiante.

GetMinorDistance: Método que devuelve en que dirección se encuentra el obstáculo más cercano.

```
1  getMinorDistance() {  
2      /*  
3          This function returns the minor distance of an array with all distances  
4      */  
5  
6      var distances = []  
7      let ray = 32;  
8      let distance=5;  
9      for (var i = 0; i <= 31; i++) {  
10          distances.push(10);  
11      }  
12      var groups = ["center", "right", "left"];  
13      for (var i = 0; i < groups.length; i++) {  
14          this.distanceArray[groups[i]].forEach((obj) => {  
15              if (typeof obj.d != "undefined") {  
16                  distances[obj.id] = obj.d;  
17              }  
18          });  
19      }  
20      for (var i=0; i<distances.length; i++) {  
21          if (distances[i]<distance){  
22              distance=distance[i]  
23              ray=i;  
24          }  
25      }  
26      if (ray > 15 && ray < 32) {  
27          return groups[1];  
28      }  
29      else if(ray == 15){  
30          return groups[0];  
31      }  
32      else if (ray < 15) {  
33          return groups[2];  
34      }  
35      else {  
36          return "unhindered";  
37      }  
38  }
```

Con esta función se puede resolver el ejercicio con un simple bucle que distingue entre las tres opciones que devuelve.

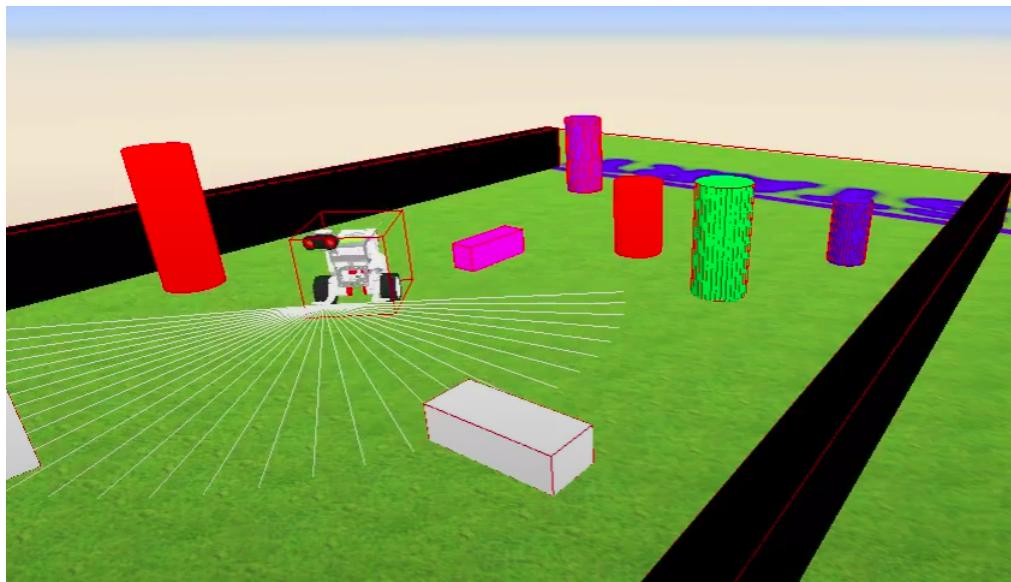


Figura 4.14: Escenario para el ejercicio atravesía bosque

```

myRobot.move(1, 0, 0);
console.log('Executing')
while(true){

    way= await myRobot.getMinorDistance()
    console.log(way);
    if (way=="left"){
        myRobot.turnUpTo(-13)
    }
    if(way == "right"){
        myRobot.turnUpTo(13)
    }
    if(way=="center"){
        myRobot.turnUpTo(13)
    }
}

```

Figura 4.15: Solución en *JavaScript* para el ejercicio atravesía bosque

En esta solución² se obtienen todos los valores que devuelve el sensor de ultra-sonidos y, según donde detecte el obstáculo, gira en un sentido u otro. Este último ejemplo está programado en *JavaScript* ya que esta función no está implementada en el *HAL API* y no tiene equivalentes en *Blockly* ni *Python*

²<https://youtu.be/ZSYWMSSRkS8>

Capítulo 5

Soporte Fisico

Ahora nos centraremos en cual ha sido el desarrollo seguido para que el bloque *Ev3* reciba código desde un cliente exterior y pueda ejecutarlo en local. Y luego como adaptarlo para su posterior implementación en el server de *Kibotics*

5.1. Interfaz de programación de sensores y actuadores

Para crear los *drivers* de *Kibotics* del robot *LEGO EV3* real se ha utilizado la librería `ev3dev2` donde se encuentran las funciones necesarias para el control del robot en Python

5.1.1. Soporte de actuadores

Los motores son los actuadores como ya he explicado en el anterior capítulo, y en la forma de uso de *Kibotics* es la de *Tracción diferencial*, en el cual los motores se mueven a la par. Por lo tanto para crear las funciones del *HAL API* en Python, Voy a utilizar la clase *MoveTank* dentro de la librería de `ev3dev2`¹.

```
1
2 class MoveTank(MotorSet):
3     """
4     Controls a pair of motors simultaneously, via individual speed setpoints for each motor.
5
6     Example:
7
8     .. code:: python
```

¹<https://github.com/ev3dev/ev3dev-lang-python>

5.1. INTERFAZ DE PROGRAMACIÓN DE SENSORES & CÁRTEA DE REPORTE FÍSICO

```
9
10     tank_drive = MoveTank(OUTPUT_A, OUTPUT_B)
11     # drive in a turn for 10 rotations of the outer motor
12     tank_drive.on_for_rotations(50, 75, 10)
13
14     """
15     def __init__(self, left_motor_port, right_motor_port, desc=None, motor_class=LargeMotor):
16         motor_specs = {
17             left_motor_port: motor_class,
18             right_motor_port: motor_class,
19         }
20
21         MotorSet.__init__(self, motor_specs, desc)
22         self.left_motor = self.motors[left_motor_port]
23         self.right_motor = self.motors[right_motor_port]
24         self.max_speed = self.left_motor.max_speed
25         self._cs = None
26         self._gyro = None
```

En esta clase se encuentran todas los métodos con los que construiré mi *Driver*

```
1
2
3 import ev3dev2.motor
4 import time
5
6
7 class Ev3Wrapper:
8
9     def __init__(self,
10                 left_motor_port=OUTPUT_D,
11                 right_motor_port=OUTPUT_A):
12
13
14     tank_drive = MoveTank(left_motor_port, right_motor_port)
15
16
17     def TurnDegrees(self, degrees, speed):
18         # get the starting position of each motor
19         tank_drive.turn_degrees(self, speed, degrees)
20
21     def avanzar(self, val):
22         tank_drive.on(self, val, val)
23
24     def retroceder(self, val):
25         tank_drive.on(self, -val, -val)
26
27     def girar_derecha(self, w):
```

```
29         tank_drive.turn_right(self, 360, 2*3,1416/w)
30
31     def girar_izquierda(self, w):
32
33         tank_drive.turn_left(self, 360, 2*3,1416/w)
34
35     def avanzar_hasta(self, distance):
36         dist_mm = distance * 1000
37
38         # the number of degrees each wheel needs to turn
39         tank_drive.on_for_distance(self, 20, distance_mm, brake=True, block=True)
40
41     def retroceder_hasta(self, distance):
42         self.avanzar_hasta(-distance)
43
44     def girar_derecha_hasta(self, degrees):
45         self.TurnDegrees(degrees, 100)
46
47     def girar_izquierda_hasta(self, degrees):
48         self.TurnDegrees(-degrees, 100)
49
50     def parar(self):
51         tank_drive.odometry_stop()
52
53
54     def target_reached(self, left_target_degrees, right_target_degrees):
55         tolerance = 5
56
57         min_left_target = left_target_degrees - tolerance
58         max_left_target = left_target_degrees + tolerance
59         min_right_target = right_target_degrees - tolerance
60         max_right_target = right_target_degrees + tolerance
61
62         current_left_position = left_motor_port.position()
63         current_right_position = right_motor_port.position()
64
65         if current_left_position > min_left_target and \
66             current_left_position < max_left_target and \
67             current_right_position > min_right_target and \
68             current_right_position < max_right_target:
69             return True
70         else:
71             return False
```

Listing 5.1: Drivers

5.1.2. Soporte de sensores

Dentro de los sensores soportados por el *HAL API* de *Kibotics* hay algunos, que no se pueden aplicar al robot *LEGO EV3* ya que no dispone de sensores, como la cámara o el sensor de Infrarrojos. Por lo que los *drivers* que se incluirán en este apartado son:

Cuadro 5.1: Métodos (HAL API) soportados por Ev3.

Método	Descripción	Salida
.getDistance()	Devuelve la distancia entre el robot y la intersección del raycaster en el centro	number(metros)
.IsTouching()	Devuelve si algo presiona el sensor	Boolean
.getObjectColor(color)	Devuelve un objeto con la posición del elemento detectado por la cámara del color pasado por parámetro	{center:[x.y], area: int}

Para este *driver*, utilizaré las funciones definidas en el API de `Ev3dev2.sensor`, las cuales incluyen todas los sensores que puede llevar el *LEGO EV3*. Pero solo utilizaremos las clases `TouchSensor()`, `ColorSensor()` y `Ultrasonic()` que son los tres sensores que usamos en este proyecto.

```

1 from ev3dev2.sensor import INPUT_1, INPUT_2, INPUT_3, INPUT_4
2 import ev3dev2.sensor.lego
3 import time
4
5
6 class Ev3Wrapper:
7
8     def __init__(self):
9         us= UltraSonic()
10        to= TouchSensor()
11        cs = ColorSensor()
12
13
14    def getObjectColor(self, color):
15        cs.raw(color)
16
17    def getDistance(self):
18        us.distance_centimeters_continuous()
19
20    def isTouching(self):
21        to.is_pressed()

```

Listing 5.2: Drivers

5.2. Descarga de la plataforma

5.2.1. Preparación ordenador a bordo

Lo primero para la recepción de mensajes es tener un dispositivo que pueda recibirlas, en este caso tendremos que instalar en el *LEGO EV3* un sistema operativo que nos permita desplegar un servidor. Comenzaré hablando del apartado técnico del robot, para entender la solución adoptada. El Bloque *Ev3* es la mejora de su predecesor el *LEGO NXT*, añadiéndole más potencia, memoria y expandibilidad. Cuenta con un procesador ARM9 a 300 MHz con memoria Flash de 16 MB, 64 MB de RAM y memoria ampliable con tarjetas mini SD hasta 32 GB. Además de eso dispone de conexiones *Bluetooth 2.1* y puerto USB.



Figura 5.1: Diseño hardware del Ev3



Figura 5.2: Ev3Dev Logo

Son las características de un PC en miniatura, por lo que la mejor opción era insertar una micro SD con una imagen de *Linux* que permitiera tener un sistema de ficheros, descargar librerías y ejecutar programas. Después de contemplar varias opciones me decante por la opción de *Ev3Dev*².

Ev3Dev es un sistema operativo basado en *Debian-Linux* diseñado especialmente para el *LEGO MINDSTORM EV3*, haciendo ingeniería inversa del blo-
que.

Una instalada la imagen en el robot para tener conexión a Internet hay dos opciones, la más simple es conectar un *Wifi Dongle* al puerto USB del Bloque *Ev3*, y se accede a la red *Wifi* como cualquier ordenador. La segunda opción, es conectar por cable el robot a tu ordenador, y conectarlo a la red como una extensión de tu red.

Una vez conectado a Internet, ya puedes conectarte al robot vía SSH.

Figura 5.3: Interfaz de la terminal de Ev3

Además crearon una plataforma de software para programar el robot. Esto incluye, soporte para muchos lenguajes de programación, entre ellos *Python*, *Java* o *C++*. Para este proyecto he elegido utilizar el lenguaje *Python*, ya que es uno de los lenguajes que comprende **Kibotics** y estoy más familiarizado con él.

5.2.2. Diseño

En la siguiente figura se muestra el diseño seguido para integrar el Ev3 real a la plataforma Kibotics. El proceso consta de 4 pasos y en las siguientes secciones se profundizará en cada una de las partes y en las interacciones necesarias entre el cliente, el servidor de Kibotics y el servidor *Flask* que lanzaremos en el *Robot Ev3*, lo que permitirá programarlo desde el navegador web usando *Kibotics*.

²<https://www.ev3dev.org/>

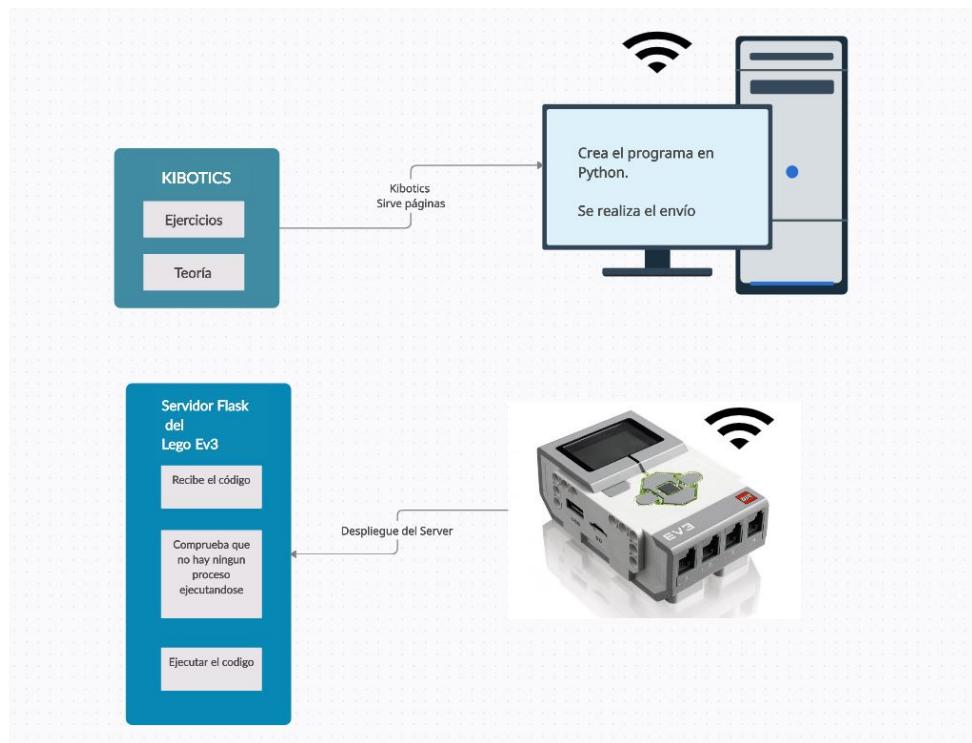


Figura 5.4: Diseño para la integración del Ev3

5.2.3. WebBrowser

Editor en el navegador web

El editor de texto se encontrara entre la parrilla de ejercicios de *Kibotics*, en este caso, *Kibotics* solo actua como plataforma que guarda los ejercicios que envió, y me proporciona la interfaz para poner el editor de texto que permita escribir su programa utilizando el lenguaje *Python*. En el editor se importa la librería “*Ev3dev2*” que proporciona una API para interactuar con los sensores y actuadores del robot. Pero en este caso, *Kibotics* no actúa en el envío, solo sirve la página para que podamos introducir el código.

Envío del código al servidor Flask en el robot

Utilizamos el editor “*ace*” para que el usuario escriba su código en la página Web, y cuando le de al botón “*Enviar*”, comienza el proceso de envío desarrollado en el script siguiente:

```

1
2 <div class="text-center">
```

```

3   
5
6 <div class="text-center">
7   
10
11  <button type="button" id="send_mbot" class="btn btn-info" onclick="send_code_to_ev3()"><
12    span
13      glyphicon glyphicon-send"></span> Enviar
14 </button>
15 </div>

```

Listing 5.3: Función de envío del código al servidor captionpos

Tras pulsar el botón Enviar, este código que hemos escrito en el editor se envía como un *query parameter* de una petición POST. Desde el navegador se manda al servidor *Flask* levantado en el robot.

```

1 let responseOk;
2
3 function send_code_to_ev3() {
4   var editor = ace.edit("ace");
5   let code = editor.getValue();
6   console.log(code);
7   const message = {
8     method: "POST"
9   };
10  url = 'http://10.42.0.1:8001/run?python_code=' + JSON.stringify(code);
11  fetch(url, message)
12    .then(function(response) {
13      if(response.ok) {
14        responseOk = true
15      }else{
16        responseOk = false
17      }
18      return response.text();
19    })
20    .then(function(data) {
21      if(responseOk){
22        console.log("Ok")
23      }else{
24        console.log("Send Fail")

```

```

25         }
26
27     })
28     .catch(function(err) {
29       console.error(err);
30     });
31   }
32 }
```

Listing 5.4: Función de envío del código al servidor

El servidor Flask hará la recepción del código que escribimos en el editor, y realizará determinadas acciones para que se pueda ejecutar en el *LEGO EV3*

5.2.4. Back-End

El servidor que se va a crear, va a hacer uso de Flask que es un entorno creado por *Django* escrito en *Python*, por lo que voy a programarlo en *Python*. Va a estar creado para que reciba peticiones del navegador con el código que el usuario escribió en el editor

```

1 import subprocess
2 import os
3 import signal
4 import time
5
6 from flask import Flask, make_response, request
7 app = Flask(__name__)
8
9 exercice = None
10 @app.route('/run', methods=["POST"])
11 def run_program():
12     global exercice
13     my_json = request.data.decode('utf8').replace("\r", "\n")
14     data = json.loads(my_json)
15     code = json.dumps(data["code"])
16
17 #-----
18
19
20
21 if __name__ == "__main__":
22     app.run(host='0.0.0.0', port=8001, debug=True)
```

Listing 5.5: Extracción del código en el servidor Flask

Creación del proceso que ejecuta la aplicación robótica

Una vez ha recibido el código, el servidor ya sabe que el lenguaje en que viene es *Python*, pero viene con algunos simbolos añadidos de *HTML*, por lo que se realizan dos *replace* para dejarlo como estaba en origen.

Una vez hecho este proceso, se procede a escribir un ejecutable con el código, para ello hay creado un archivo llamado *ejercicio.py*. Una vez creado el archivo se comprueba que no hay otro proceso ejecutandose, si es así se envía una señal *kill* a dicho proceso. Tras ejecutar el programa en un nuevo proceso, el robot realizará las instrucciones que el usuario programó en el editor del navegador web.

```

1 if exercice:
2     try:
3         os.killpg(os.getpgid(exercice.pid), signal.SIGKILL)
4     except ProcessLookupError:
5         pass
6
7     time.sleep(2)
8
9     code = code[1:-1].split("\n")
10    print(code)
11    fdOut = open("./ejercicio.py", "w")
12    for line in code:
13        fdOut.write(line + "\n")
14
15    exercice = subprocess.Popen(["python", "ejercicio.py"], stdout=subprocess.PIPE, preexec_fn=os.setsid)
```

Listing 5.6: Creación del proceso con el programa para el robot

Respuesta del Servidor

Por ultimo, se contesta al cliente con el código *200 OK*

```

1
2 headers = {'Content-Type': 'text/plain', 'Access-Control-Allow-Origin': '*'}
3     return make_response('Ok', 200, headers)
```

Listing 5.7: Creación del proceso con el programa para el robot

5.3. Validación experimental con Ejercicios

5.3.1. Bump and go

Para comprobar que todo lo anterior funciona hice tres ejercicios enviados desde un navegador web y ejecutados en el robot real. El primero, el *Bump and Go*³ probando los el sensor de contacto.

```

1
2 #!/usr/bin/env python3
3
4 from time import sleep
5
6 from ev3dev2.motor import LargeMotor, OUTPUT_A, OUTPUT_B, SpeedPercent, MoveTank
7 from ev3dev2.sensor import INPUT_1
8 from ev3dev2.sensor.lego import TouchSensor
9 from ev3dev2.led import Leds
10
11 ts = TouchSensor()
12 leds = Leds()
13 tank_drive = MoveTank(OUTPUT_A, OUTPUT_B)
14
15 print("Bump and go!")
16
17 while True:
18     if ts.is_pressed:
19         tank_drive.on_for_rotations(SpeedPercent(-50), SpeedPercent(-75), 10)
20     else:
21         tank_drive.on(SpeedPercent(30), SpeedPercent(30))
22     # don't let this loop use 100% CPU
23     sleep(0.01)

```

Listing 5.8: Bump and Go

5.3.2. Siguelineas

El siguiente ejercicio que envié fue el *Siguelineas*⁴ para también probar la función *reflected-lightintensity* del sensor de color:

³https://www.youtube.com/watch?v=5dD707Qku6I&ab_channel=DanielPulidoMillanes

⁴https://www.youtube.com/watch?v=iMn7UsaB2vo&ab_channel=DanielPulidoMillanes

```
1
2 #!/usr/bin/env python3
3
4 from time import sleep
5
6 from ev3dev2.motor import LargeMotor, OUTPUT_A, OUTPUT_B, SpeedPercent, MoveTank
7 from ev3dev2.sensor import INPUT_1
8 from ev3dev2.sensor.lego import ColorSensor
9 from ev3dev2.led import Leds
10
11 cs = ColorSensor()
12 leds = Leds()
13 tank_drive = MoveTank(OUTPUT_A, OUTPUT_B)
14
15
16 print("SigueLineas!")
17
18 while True:
19     while cs.reflected_light_intensity>30:
20         tank_drive.on(SpeedPercent(5), SpeedPercent(20))
21         tank_drive.on(SpeedPercent(20), SpeedPercent(5))
22         # don't let this loop use 100% CPU
23         sleep(0.02)
```

Listing 5.9: SigueLineas

Capítulo 6

Conclusiones

En este capítulo se recogen las conclusiones a las que se han llegado una vez realizado el Trabajo de Fin de Grado, y voy a valorar los objetivos alcanzados, comparándolos con los previstos en un principio. Así como explicación sobre los conocimientos adquiridos y se presentarán una serie de mejoras para realizar en un futuro.

6.1. Conclusiones

El objetivo principal era el de integrar el robot *LEGO EV3* en la plataforma de *Kibotics* el cual ha sido alcanzo con exito , este objetivo lo dividimos en varios subobjetivos que vamos a ir analizando.

El primer subobjetivo consistía en dar soporte a la parte simulada del robot *LEGO EV3* en *WebSim*. En ell Capitulo 4 se explica cómo se ha llevado a cabo, aportando los modelos en 3D, *drivers* y bloques para las nuevas funcionalidades que añade el *LEGO EV3*. Entre ellos esta el nuevo sensor de contacto con su función *IsTouching* y actualización de la funciones en los *drivers* de los sensores de *Ultrasonidos* y *Sensor de IR*.

El segundo subobjetivo era dar soporte al robot real para aceptar código desde el navegador web y el *LEGO EV3* fuera capaz de recibir ese codigo, interpretarlo y ejecutarlo en local. Para ello se crearon unos *drivers* en Python que daban la funcionalidad al robot real, y conectaba con los *drivers* en JavaScript de los que dispone *Kibotics*. Para que el robot real pudiera ejecutar programas que le enviaban, necesito que se le instalara una imagen de distribución *Linux*, se ins-

taló los *drivers* al *LEGO* y se lanzó un servidor *Flask* que se encargaba de recibir los paquetes *HTTP* que enviaba el navegador Web con el código en *Python*. Una vez recibido se encargaba de guardarlo como ejecutable y lanzarlo en local. Todo esto viene con más detalle en el Capítulo 5

El tercer subobjetivo era incluir ejercicios que actuaran como validación experimental de los avances que he ido explicando anteriormente, y ademas para tener una bateria de ejercicios que tener como temario. Se han implementado: *Bump and Go*, *AtraviesaBosque*, *SigueLineasIR* y *CuadradoEv3* para el robot simulado, además de el *Bump and Go* y *SigueLineas* para el robot real en *Python*

6.2. Mejoras futuras

La implementación del *LEGO EV3* en *Kibotics* ha supuesto un avance para la plataforma, pero además ha dejado las puertas a futuros proyectos que pueden enriquecer más la plataforma, como por ejemplo:

- Añadir el *LEGO EV3* real al catálogo de robots que ofrece *Kibotics*, y que se pueda programar y lanzar desde la propia página web de *Kibotics* en vez de, desde una página externa
- Añadir más ejercicios a la plataforma con los sensores que ofrece *LEGO* aparte de los que vienen con el paquete *MINDSTORM* como pueden ser el micrófono, el sensor de temperatura, o el sensor de luz ambiental.
- En relación con el anterior apartado, añadir más variables físicas, como que el propio robot simulado tenga un centro de gravedad y pueda caer, para que el *girosensor* tenga más utilidad. O que haya un nivel de luz, o temperatura en el escenario, lo que daría mas posibilidades a la hora de crear ejercicios.
- Añadir mas diseños de modelados 3D de robots de *LEGO EV3* y con ellos crear más tipos diferentes de ejercicios.