



GRADO EN INGENIERÍA TELEMÁTICA

Curso Académico 2020/2021

Trabajo Fin de Grado

Integración del Robot Lego Ev3 a la plataforma de Kibotics

Autor : Daniel Pulido Millanes

Tutor : Dr. José María Cañas Plaza

Índice general

Lista de figuras	5
Lista de tablas	7
1. Introducción	1
1.1. Robótica	1
1.1.1. Aplicaciones robóticas	3
1.1.2. Software en robótica	6
1.2. Robótica educativa	7
2. Objetivos	11
2.1. Objetivos del TFG	11
2.2. Metodología	12
2.3. Requisitos	13
2.4. Plan de trabajo	14
3. Soporte Simulado	15
3.1. Características del Lego Ev3	15
3.2. Soporte de sensores	18
3.2.1. Sensor de color	18
3.2.2. Sensor de ultrasonido	21
3.2.3. Sensor de contacto	23
3.2.4. Girosensor	25
3.3. Nuevos ejercicios	25
3.3.1. Sigue-líneas	25

3.3.2. Choca-gira	26
3.3.3. Atraviesa-bosque	28

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 gráfica de <i>Scratch</i>	8
1.8.	Interfaz de <i>LEGO WeDo</i>	9
1.9.	Interfaz gráfica de <i>LEGO Ev3</i>	9
1.10.	Kit de piezas y sensores de <i>LEGO Ev3</i>	10
2.1.	Metodología de modelo iterativo	13
3.1.	Conjunto <i>Ev3</i>	16
3.2.	Robot con sensor de color	17
3.3.	Robot con sensor tactil	17
3.4.	Robot con sensor de ultrasonidos	17
3.5.	Sensor color	18
3.6.	Sensor de ultrasonidos	21
3.7.	Raycaster	22
3.8.	Sensor tactil	24
3.9.	girosensor	25
3.10.	Escenario para el ejercicio <i>LEGO EV3 sigue-líneas</i>	26
3.11.	Solución en <i>Scratch</i> para el ejercicio <i>sigue-líneas</i>	26

3.12. Solución en <i>Scratch</i> para el ejercicio choca-gira	27
3.13. Escenario para el ejercicio atraviesa bosque	29
3.14. Solución en <i>JavaScript</i> para el ejercicio atraviesa bosque	29

Índice de cuadros

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.

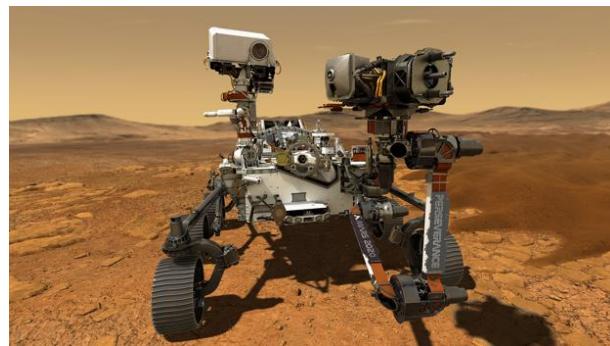


Figura 1.6: Robot *Perseverance* de la *NASA*

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 maquina tenía un software específico con sensores y actuadores únicos para ese robot y esa tarea a desarrollar. Esto hacia, 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++
- **JdeRobot**[?] Plataforma de desarrollo robótico, en la que se basa este proyecto. Tiene varios nodos programados con varios lenguajes de programación, con compatibilidad con otros *middleware*.

1.2. 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”[?].

Una de las mayores partes de la robótica tiene que ver con la programación, que además 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 más 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 fácil que es de entender el pensamiento computacional debido a su sencilla interfaz gráfica y la implementación de sus bloques.



Figura 1.7: 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 está diseñado el software.

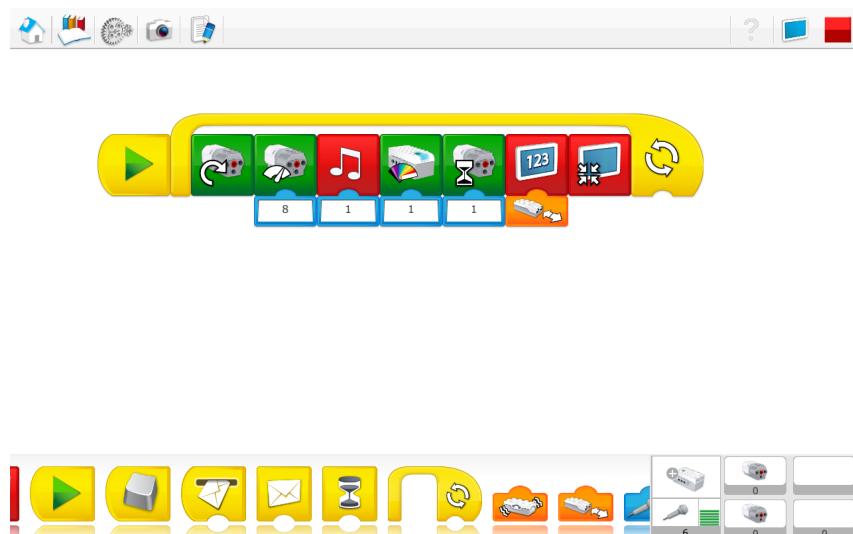


Figura 1.8: Interfaz de LEGO WeDo

Por ejemplo en la figura 1.8 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.

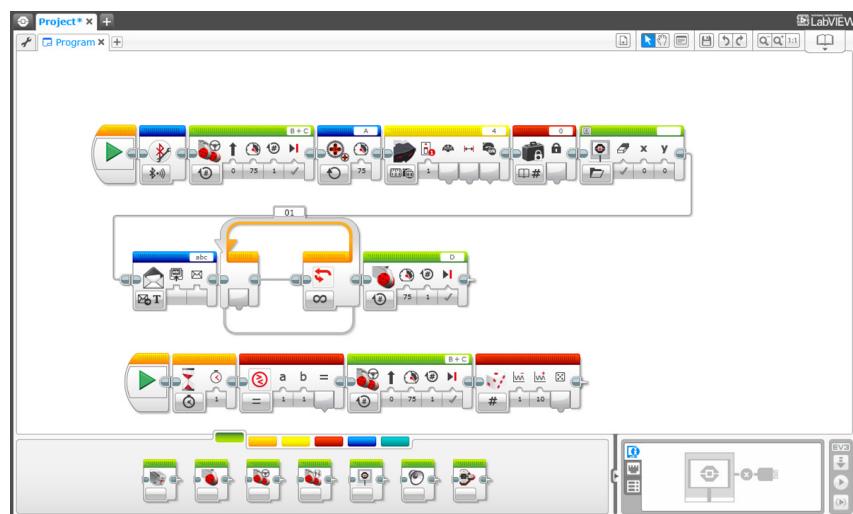


Figura 1.9: Interfaz gráfica de LEGO Ev3

En el caso del software para el **LEGO Ev3**, añade un grado de complejidad, incluyendo

apartados para realizar operaciones matemáticas, envío de archivos entre robots, y añade más 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 1.10: Kit de piezas y sensores de LEGO Ev3

En el siguiente capítulo, 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.

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* de varios modelos de robots, al menos uno por cada tipo de sensor que pueda llevar. Y que tenga sentido físico dentro de la simulación.
- Añadir la infraestructura necesaria como *drivers*, y funciones al *Robot API* para que el robot se programable en cualquier lenguaje soportado por la plataforma, y funcione en el robot real
- Crear un conjunto de ejercicios para que haya un temario fácil de seguir por el estudiante, y con una curva de dificultad moderada

¹<https://kibotics.org/>

Soporte Real

- Instalar una imagen de un sistema operativo en el *Lego ev3* en este caso, una distribución basada en *Debian Linux*.
- Instalar 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.

2.2. 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.

²<https://slack.com/>

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

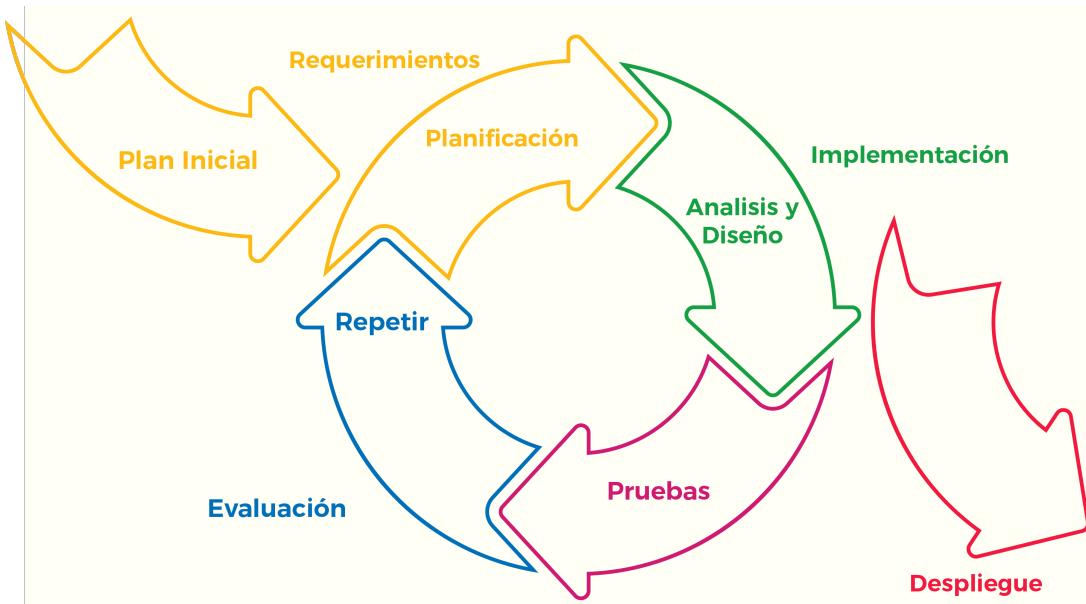


Figura 2.1: Metodología de modelo iterativo

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 directamente en el repositorio

2.3. 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* , en este caso he optado por la imagen creada por *ev3dev*, que es una distribución basada en el sistema operativo *Debian Linux*, de la cual entrare en mas detalle en el Capítulo 5: *Soporte Lego ev3 físico*.
- El resultado final debe ser lo más sencillo posible, no debe requerir configuraciones adicionales. Este software tiene que estar diseñado para estudiantes.

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 capitulo
- *Paso 2:* Comienzo de la creación del robot simulado. Comenzaremos por la creación de varios modelos 3D para usarlos en el entorno simulado, son varios diferentes ya que lo original que tiene *LEGO* es poder construir tu robot en base a la tarea que vaya a realizar, con las piezas que incluye el kit .
- *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.

Capítulo 3

Soporte Simulado

Ahora que tenemos definidos cuales 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 de la parte simulada del robot. Pero antes de eso se darán unas nociones básicas de las características de este robot, para poder entender, el proceso de desarrollo.

3.1. Características del 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

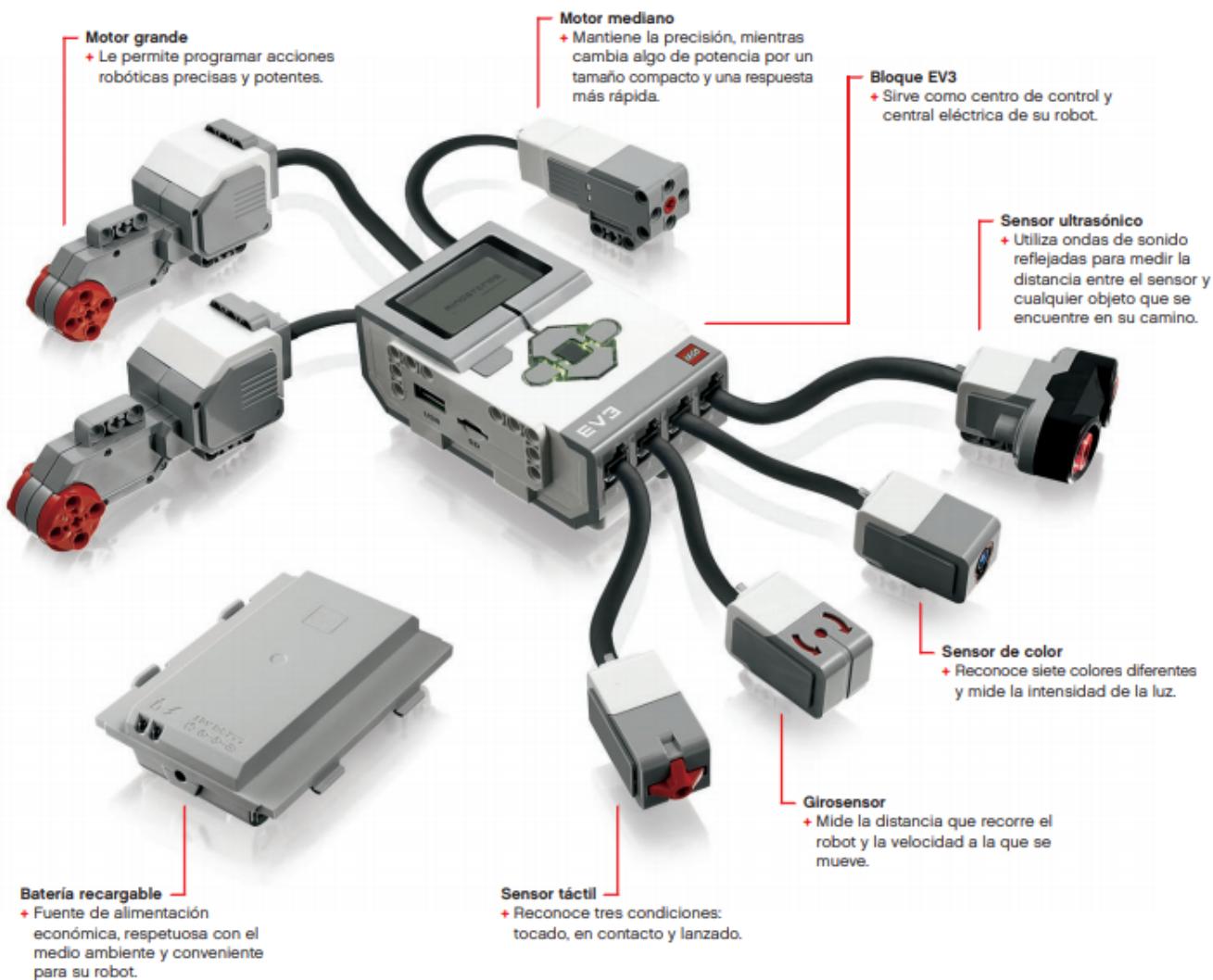


Figura 3.1: Conjunto Ev3

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.

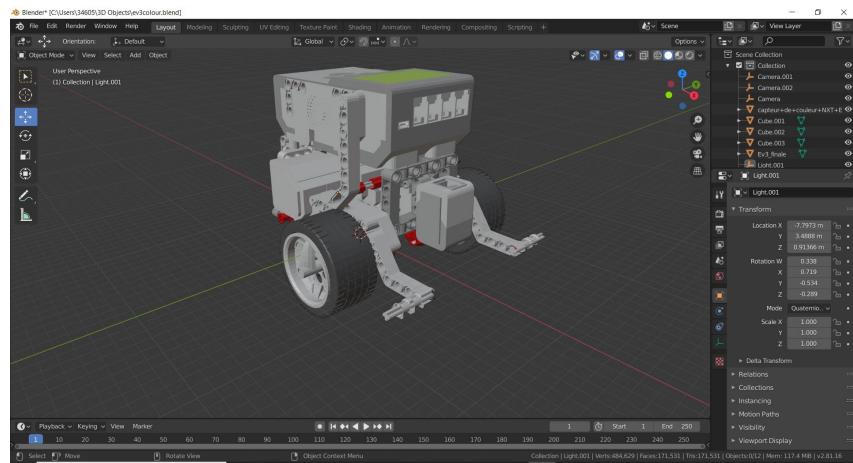


Figura 3.2: Robot con sensor de color



Figura 3.3: Robot con sensor tactil

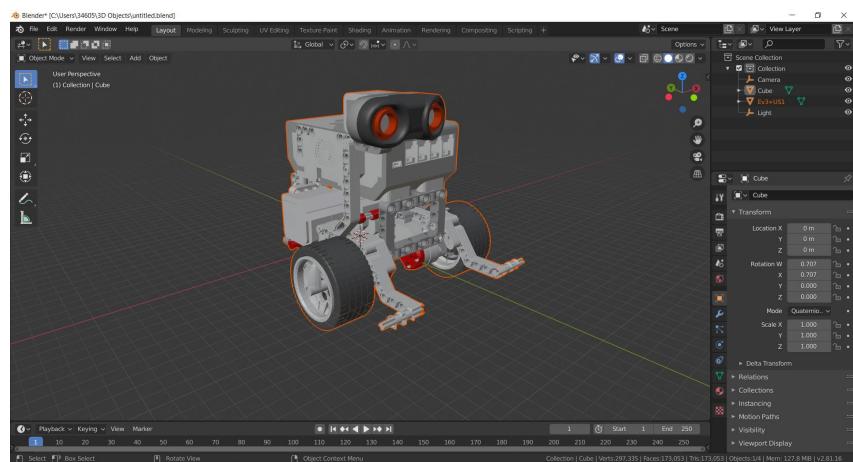


Figura 3.4: Robot con sensor de ultrasonidos

Para la tarea del modelaje 3D, he usado el programa de *Blender*.

Ahora que tenemos los tres modelos, voy a dividir el soporte del robot simulado en los diferentes sensores que implementar.

3.2. Soporte de sensores

3.2.1. Sensor de color

El primer sensor que vamos a analizar es el 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.



- **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 que color está viendo.

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



Modo intensidad de la luz ambiental

Figura 3.5: Sensor color

Sensor de color en sus 3 usos

```

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'];

```

```

10      }
11
12  }
```

El *LEGO EV3* no tiene una camara instalada, pero para el robot simulado, es lo mas 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      /**
4       * This function filters an object in the scene with a given color, uses OpenCVjs to
5       * filter
6       * by color and calculates the center of the object.
7       *
8       * Returns center: CenterX (cx), CenterY (cy) and the area of the object detected in the
9       * image.
10
11  */
12
13  if (lowval.length === 3) {
14      lowval.push(0);
15  }
16  if (highval.length === 3) {
17      highval.push(255);
18  }
19  var image = this.getImage();
20  var binImg = new cv.Mat();
21  var M = cv.Mat.ones(5, 5, cv.CV_8U);
22  var anchor = new cv.Point(-1, -1);
23  var lowThresh = new cv.Mat(image.rows, image.cols, image.type(), lowval);
24  var highThresh = new cv.Mat(image.rows, image.cols, image.type(), highval);
25  var contours = new cv.MatVector();
26  var hierarchy = new cv.Mat();
27
28  cv.morphologyEx(image, image, cv.MORPH_OPEN, M, anchor, 2,
29      cv.BORDER_CONSTANT, cv.morphologyDefaultBorderValue()); // Erosion followed by
30          dilation
31
32  cv.inRange(image, lowThresh, highThresh, binImg);
33  cv.findContours(binImg, contours, hierarchy, cv.RETR_CCOMP, cv.CHAIN_APPROX_SIMPLE);
34  if (contours.size() > 0) {
35
36      let stored = contours.get(0);
37      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 este 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.

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.

3.2.2. 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). Al utilizar unidades en pulgadas, la distancia detectable es entre 1 y 99 pulgadas (con una exactitud de +/- 0,394 pulgadas). Un valor de 255 centímetros o 100 pulgadas significa que el sensor no puede detectar ningún objeto frente a él.

En *Kibotics* 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 3.6: Sensor de ultrasonidos

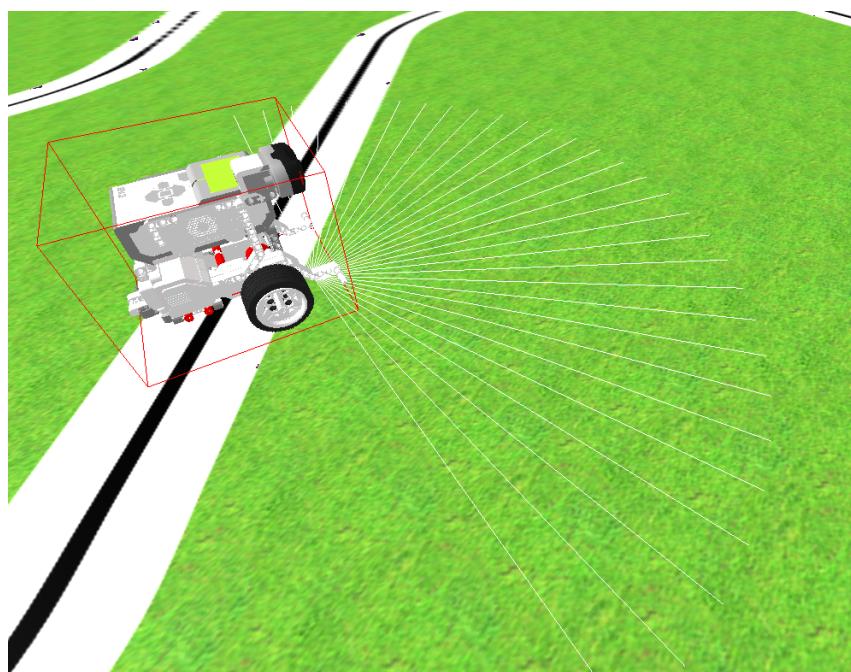


Figura 3.7: Raycaster

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 mas cercano, es un poco irreal. Por lo que creare la función *getDistance* que devuelve el valor mas 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) {
```

```

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 }

```

3.2.3. 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).

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, creare un par de funciones, que aprovechándose del *array* de distancias, cogeré la distancia central y cuando sea mínima, daré por hecho que el robot esta 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 3.8: Sensor tactil

Esta función al ser totalmente nueva, habrá que añadirla en el *HAL API* y tendrá que tener su equivalente en *Python* y en *Blockly*, que mostrare en la siguiente parte con la implementación de este sensor en un ejercicio.

3.2.4. 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 3.9: 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 en *Kibotics*, no tienen un centro de gravedad en el mantenerse.

3.3. Nuevos ejercicios

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

3.3.1. 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.

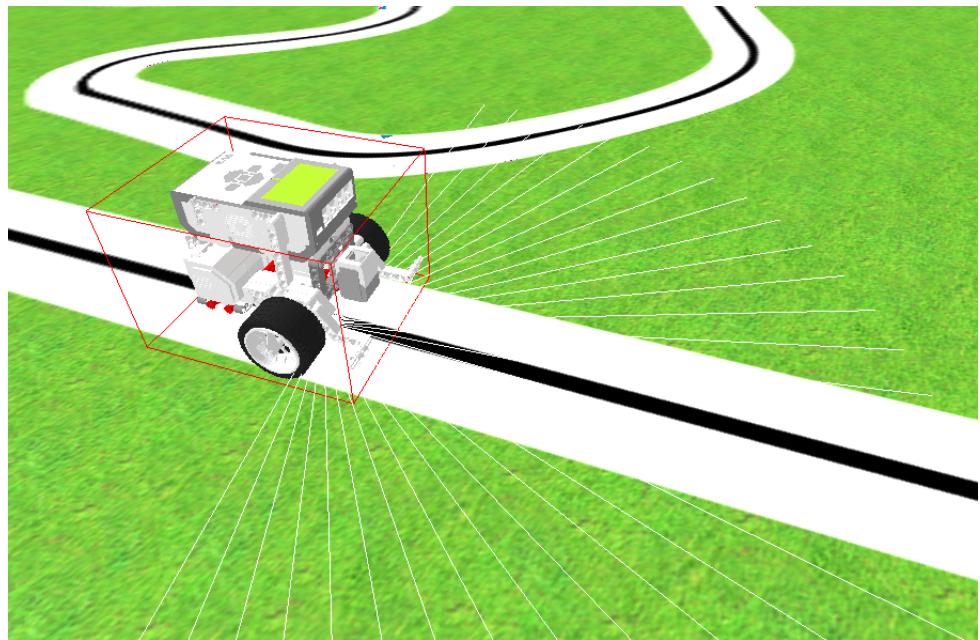


Figura 3.10: Escenario para el ejercicio *LEGO EV3 sigue-líneas*

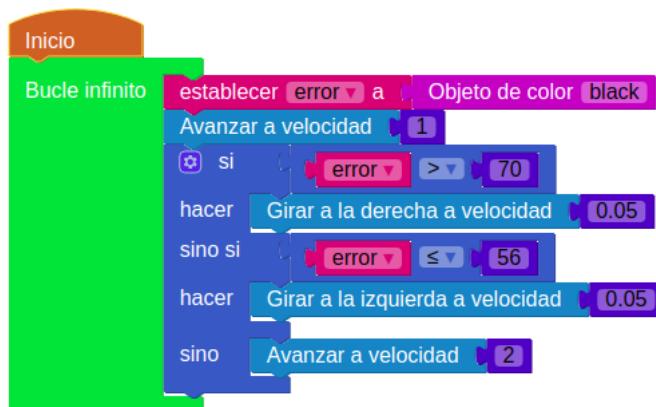


Figura 3.11: Solución en *Scratch* para el ejercicio *sigue-líneas*

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

3.3.2. 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.

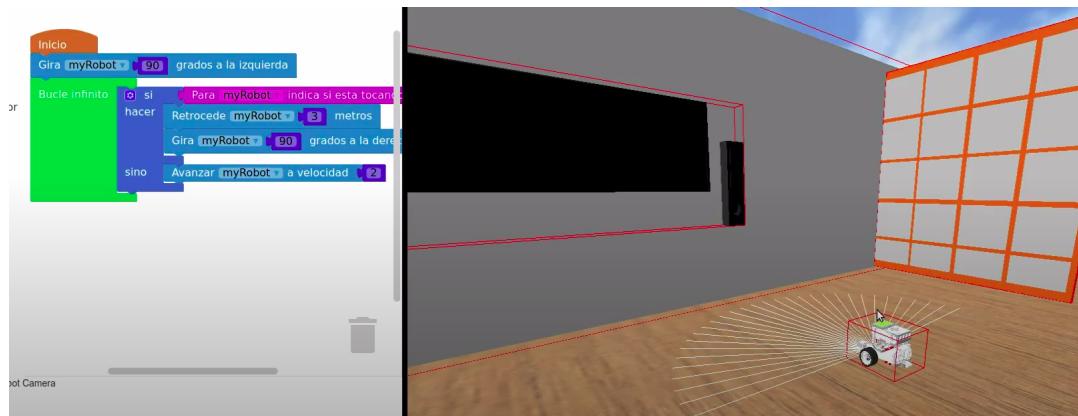


Figura 3.12: Solución en *Scratch* para el ejercicio choca-gira

En esta solución¹ se prueba el bloque anteriormente mencionada que equivale a la función *IsTouching* en el lenguaje *Scratch*.

3.3.3. Atraviesa-bosque

Ejercicio basado en atravesar un pasillo con diversos objetos que hay que esquivar. El sensor necesario es el infrarrojos 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 ultrasonido no puede saber en qué posición se encuentra el obstáculo, solo la distancia. Aún así, este ejercicio tiene un gran interés educativo, por lo que he decidido añadirlo, y además crear una nueva función para que sea más sencillo e intuitivo de completar para el estudiante.

GetMinorDistance: Método que devuelve en qué 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);

```

¹<https://youtu.be/figTbXKEXD4>

```
11      }
12
13     var groups = ["center", "right", "left"];
14     for (var i = 0; i < groups.length; i++) {
15       this.distanceArray[groups[i]].forEach((obj) => {
16         if (typeof obj.d != "undefined") {
17           distances[obj.id] = obj.d;
18         }
19       });
20     }
21     for (var i=0; i<distances.length; i++) {
22       if (distances[i]<distance){
23         distance=distance[i]
24         ray=i;
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.

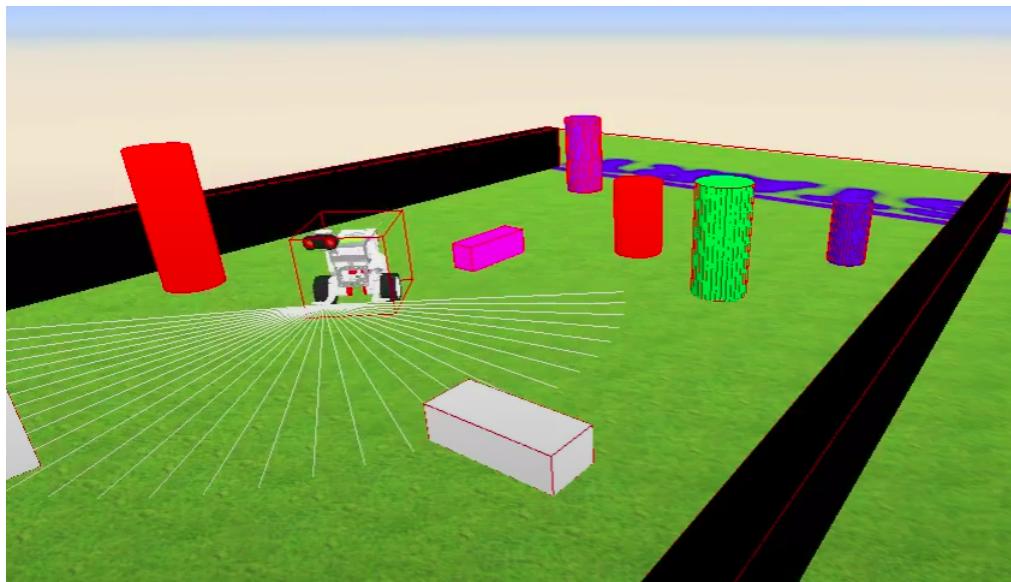


Figura 3.13: Escenario para el ejercicio atraviesa bosque

```
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 3.14: Solución en *JavaScript* para el ejercicio atraviesa 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.

²<https://youtu.be/ZSYWMSSRkS8>

