ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADO EN INGENIERÍA INFORMÁTICA

Modelado y Simulación de un Robot LEGO Mindstorms EV3 mediante V-REP y Matlab

Modeling and Simulation of a LEGO Mindstorms EV3 Robot with V-REP and Matlab

Realizado por
Alberto Martín Domínguez
Tutorizado por
Dra. Ana María Cruz Martín
Dr. Juan Antonio Fernández Madrigal
Departamento
Ingeniería de Sistemas y Automática

UNIVERSIDAD DE MÁLAGA
MÁLAGA, Diciembre de 2016

Fecha defensa:
El Secretario del Tribunal

RESUMEN

En algunas titulaciones de la Universidad de Málaga, concretamente en el Grado de Ingeniería Informática de la ETSI Informática, se imparten asignaturas de Robótica como, por ejemplo, *Programación de Robots*, las cuales incluyen en su temario prácticas con kits LEGO Mindstorms en el laboratorio. Alumnos y docentes, normalmente, no disponen de estos kits en casa; no pudiendo así preparar, previamente, las distintas prácticas para superar la asignatura solo teniendo disponibles las horas asignadas en el laboratorio.

Por ello en el presente trabajo se propone el entorno de simulación V-REP con un robot móvil LEGO Mindstorms EV3 modelado para dicho entorno, manejado mediante una toolbox de Matlab basada, en su mayoría, en el lenguaje NXC. Nuestro modelo de robot posee una configuración de ruedas diferencial mediante dos motores, que serán los actuadores, y una rueda arrastrada; en el tema de sensores, para controlar a nuestro robot, disponemos de: sensor táctil, ultrasónico, de luz/color, giroscópico y de rotación. Para modelar los botones, la pantalla y las luces de estado de nuestro robot, nos hemos servido de la realización de una interfaz gráfica. En términos de comunicación con el entorno y Matlab, nos apoyaremos en la API remota que proporciona V-REP, concretamente, en su función genérica que nos permitirá implementar un conjunto de funciones en el lado del simulador, para, luego, ir llamándolas desde Matlab, mediante funciones basadas, en su mayoría, en NXC.

En cuanto a los resultados, se han desarrollado pruebas de uso de la toolbox y dos prácticas de la asignatura *Programación de Robots* (seguimiento de líneas con un algoritmo PID y localización offline mediante odometría), todas ellas realizadas satisfactoriamente.

Palabras clave

Docencia, Simulación, Modelado 3D, Robótica educativa, Matlab, V-REP, LEGO Minsdtorms.

ABSTRACT

Some degree programs of the Universidad de Málaga, specifically the Bachelor's Degree in Computer Engineering from ETSI Informatica, offer subjects about Robotics as, for example, *Robots Programming*, which includes in it syllabus practices with LEGO Mindstorms kits in the laboratory. Students and teachers, usually, do not have these kits at home; thus they could not prepare the different practices only using the assigned laboratory hours.

In this Bachelor Thesis, we propose the V-REP simulation environment with a LEGO Mindstorms EV3 mobile robot modelled for this environment and driven by a Matlab toolbox based mostly on NXC language. Our robot model has a differential wheel configuration with two motors and a free wheel; regarding the sensors for controlling the robot we use: touch, ultrasonic, light/colour, gyroscopic and rotational sensors. For modelling the robot's buttons, screen and status lights, we have created a graphical user interface. In terms of the communication with Matlab and the simulator, we use the V-REP remote API, specifically, its generic function that allows to implement a function set in the simulator side, that can be called from Matlab, through NXC based functions.

We have deployed several toolbox tests, and we have developed two practices of the *Robots Programming* subject (line following using a PID algorithm and odometry based offline localization). These results performed satisfactorily.

Keywords

Teaching, Simulation, 3D Modelling, Educational Robotics, Matlab, V-REP, LEGO Minsdtorms

ÍNDICE DE FIGURAS

ÍNDICE DE CONTENIDOS

CAPÍTULO 1.
INTRODUCCIÓN

La Robótica puede definirse como “una combinación de la ingeniería mecánica, ingeniería eléctrica, ingeniería electrónica y ciencias de la computación que se ocupa del diseño, construcción, operación, disposición estructural, manufactura y aplicación de los robots” [1]. Actualmente, sus aplicaciones se extienden con rapidez y es posible encontrarlas en ámbitos cotidianos; tanto es así que ya se proponen soluciones aplicadas a la docencia, o de manera autodidacta, para la enseñanza de esta disciplina. Una de estas es la que nos presenta la compañía de juguetes LEGO [2] con su rama LEGO Mindstorms [3], en la cual oferta kits relativamente económicos para aprender robótica básica de la manera menos compleja posible.

En algunas titulaciones de ingeniería de la Universidad de Málaga en las que se imparten asignaturas de Robótica, como, por ejemplo, *Programación de Robots* en el Grado de Ingeniería Informática de le ETSI Informática, ya se incluyen en el temario ejercicios y prácticas con los mencionados kits. Este trabajo fin de grado proporciona un simulador del robot LEGO EV3 para la preparación de distintas prácticas, facilitando el trabajo de profesores y alumnos al no ser necesario disponer de un robot físico.

1.1. Objetivos

En este trabajo, se ha realizado el modelado de un robot LEGO Mindstorms EV3 en el entorno de simulación V-REP [4], y la construcción de una toolbox de Matlab [5] basada en las funciones del lenguaje NXC [6] que permite controlarlo en dicho entorno (Figura 1). Estos objetivos se detallan a continuación:

- Modelado en tres dimensiones de un robot móvil LEGO Mindstorms EV3 con una herramienta de diseño CAD; en este caso, se ha usado LeoCAD [7].

- Programación de los sensores y actuadores del simulador conforme a las especificaciones reales del robot mediante las herramientas de V-REP y su API nativa en el lenguaje Lua [8].

- Comunicación de los sensores y actuadores del robot modelado en el simulador con Matlab.

- Programación de una toolbox de Matlab para la comunicación transparente con el simulador.

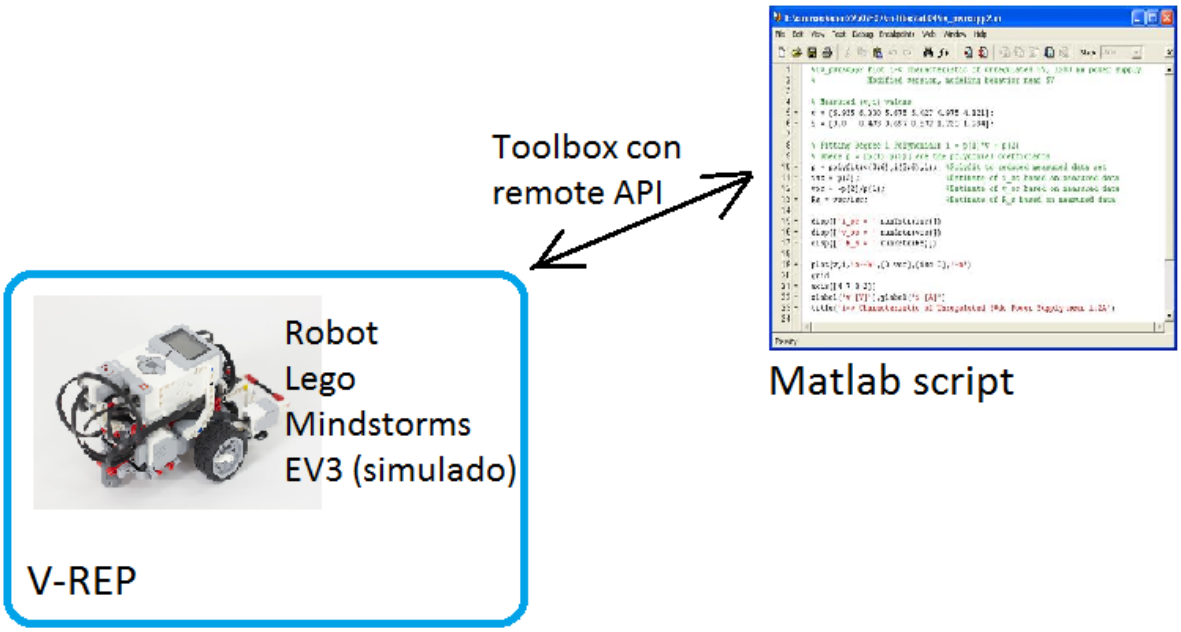

Figura 1. Diagrama funcional del trabajo

1.2. Métodos

La metodología software que se ha usado en este trabajo ha sido el modelo en cascada [9] (Figura 2), ya que este trabajo necesita ser dividido en etapas e irlas finalizando en su respectivo orden; además, esta metodología favorece la investigación y el diseño del trabajo previo a su desarrollo.

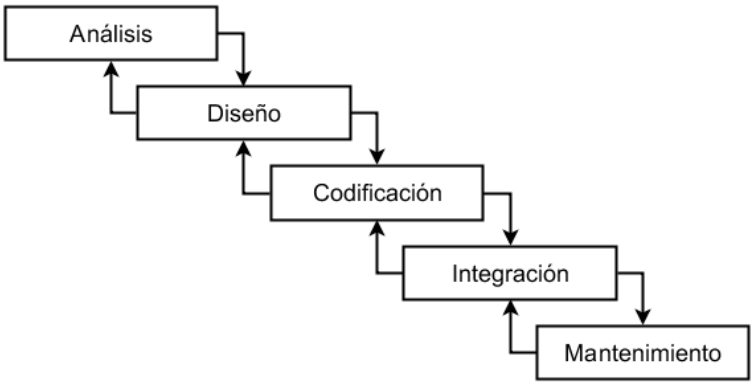

Figura 2. Ciclo de vida de la metodología en cascada

Para el control de versiones del código escrito en Matlab se han usado Git [10] y los repositorios privados de GitHub [11] en la nube, ya que nos ofrecen una herramienta de control de versiones muy utilizada en el ámbito de la ingeniería del software, y que nos permitirá seguir de cerca las fases del proyecto, sus copias de seguridad y la capacidad de poder clonar el proyecto a distintos equipos.

1.3. Fases del trabajo

Se ha seguido la siguiente planificación aproximada para la realización del trabajo fin de grado:

1. Estudio del simulador V-REP comprendiendo sus funcionalidades, de la herramienta de control de versiones Git y de LeoCAD (20 horas).
2. Instalación de V-REP y Matlab en un ordenador (1 hora).
3. Modelado del robot LEGO EV3 en 3 dimensiones (15 horas).
4. Programación y configuración de sensores y actuadores en el simulador (30 horas).
5. Pruebas de sensores y actuadores en el simulador (10 horas).
6. Estudio de la API remota [12] de V-REP para Matlab (10 horas).
7. Configurar V-REP para la API remota de Matlab (10 horas).
8. Programación de la toolbox de Matlab para la comunicación con el simulador (65 horas).
9. Pruebas de la toolbox para la comunicación con el simulador (10 horas).
10. Realización de dos ejemplos prácticos (30 horas).
11. Elaboración de la documentación del software desarrollado y la memoria escrita del trabajo (95 horas).

1.4. Medios Materiales

Con respecto al hardware, se han usado dos PC: uno con sistema operativo Linux [13] y otro con Windows 10 [14]. En Linux, se usará una distribución Linux mint 17 Xfce [15]. La elección de ambos sistemas operativos viene dada por las siguientes ventajas:

- Portabilidad del simulador entre sistemas Linux y Windows, en el caso de que tengamos que reinstalar el sistema operativo, por la existencia de versiones para ambos.

- Automatización de tareas sistemáticas y de copia de seguridad del proyecto mediante alias, shell scripts o Git.

- El código implementado en Matlab es multiplataforma, al igual que el modelo del robot realizado para V-REP y el propio simulador, por lo que el proyecto previsiblemente funcionará tanto en Linux como Windows y MacOS.

- Al ser la distribución de Linux que hemos elegido más ligera, el simulador se ejecutará de manera más fluida.

Además, se dispone de un disco duro externo y el repositorio privado de GitHub para las respectivas copias de seguridad.

Por parte del software, dispondremos de:

- Matlab R2016a versión estudiantes.
- LeoCAD versión 0.82.2.
- V-REP PRO EDU V3.3.1 (rev. 1).
- Git 2.10.0 (ambas versiones para Windows y Linux).
- Software de tratamiento de textos para la realización de la memoria.

1.5. Estructura del documento

La memoria del presente trabajo fin de grado se organiza como sigue:

- **Capítulo 1.** Introducción del trabajo: objetivos, metodología seguida, planificación y medios materiales.

- **Capítulo 2.** Estado del arte y descripción de las herramientas utilizadas para el desarrollo del trabajo: entorno de simulación V-REP, API remota para Matlab del simulador y LeoCAD como programa de diseño del robot.

- **Capítulo 3.** Descripción del modelo del robot en V-REP, incluyendo: su mecánica, la composición de los sensores y actuadores, su construcción, la interfaz gráfica y el conjunto de funciones que dispondrá para su control desde Matlab.

- **Capítulo 4.** Documentación de la toolbox para la programación desde Matlab.

- **Capítulo 5.** Resultados, en los que se mostrará un par de prácticas que se realizan en la asignatura *Programación de Robots*, así como unos tests que prueban las funcionalidades que estas prácticas no cubren.

- **Capítulo 6.** Conclusiones y futuros trabajos.

- **Anexos**. Estos son: manual de usuario y manual del desarrollador.

- **Referencias.** Usada para la realización de la presente memoria.

Todos los ficheros correspondientes al trabajo (el código desarrollado, muestras recogidas de experimentos, videos, etc.) se encuentran en el CD-ROM adjunto con la memoria o en la siguiente carpeta de Google Drive:
https://drive.google.com/folderview?id=0B2mnuwTFyL-7ZTJOekZmWkFISUU&usp=sharing

También, se pueden descargar todos los archivos, exceptuando los vídeos, en formato Zip:
https://drive.google.com/open?id=0B2mnuwTFyL-7MXRWM05lcGhlOWc

Posteriormente, a lo largo de la memoria, se harán referencias a los archivos almacenados según su directorio, por si se requiere su consulta.

Adicionalmente, los archivos referentes al paquete de instalación del trabajo, código fuente de Matlab, modelos y manuales, se encuentran en el repositorio público de GitHub:
https://github.com/albmardom/EV-R3P

CAPÍTULO 2.
ESTADO DEL ARTE Y HERRAMIENTAS SELECCIONADAS

En este capítulo se describirá el estado del arte en LEGO Mindstorms y entornos de simulación de Robótica, como se mencionó en el capítulo anterior. Además, se mostrarán con detenimiento los programas y herramientas utilizados para el desarrollo del TFG, que son:

- LEGO Mindstorms.

- V-REP, el simulador donde se va a proceder el modelado del robot y la posterior simulación.

- API remota del simulador V-REP para Matlab; esta herramienta nos ha sido muy útil para la comunicación entre el simulador y Matlab para el desarrollo de nuestra toolbox.

- LeoCAD, procedente del estándar LDraw [16], es un programa de diseño gráfico de bloques de LEGO, donde, hemos podido construir el robot de una manera rápida, sin necesitar de conocimientos avanzados de diseño gráfico.

Para la integración del simulador V-REP con Matlab se ha optado por implementar una arquitectura de capas, ya que “la ventaja principal de este estilo es que el desarrollo se puede llevar a cabo en varios niveles y, en caso de que sobrevenga algún cambio, solo se ataca al nivel requerido sin tener que revisar entre código mezclado” [17]. En la Figura 3 se muestra dicha arquitectura aplicada a nuestro trabajo, donde cada componente de cada capa se explicará a lo largo de los capítulos 3, 4 y este mismo.

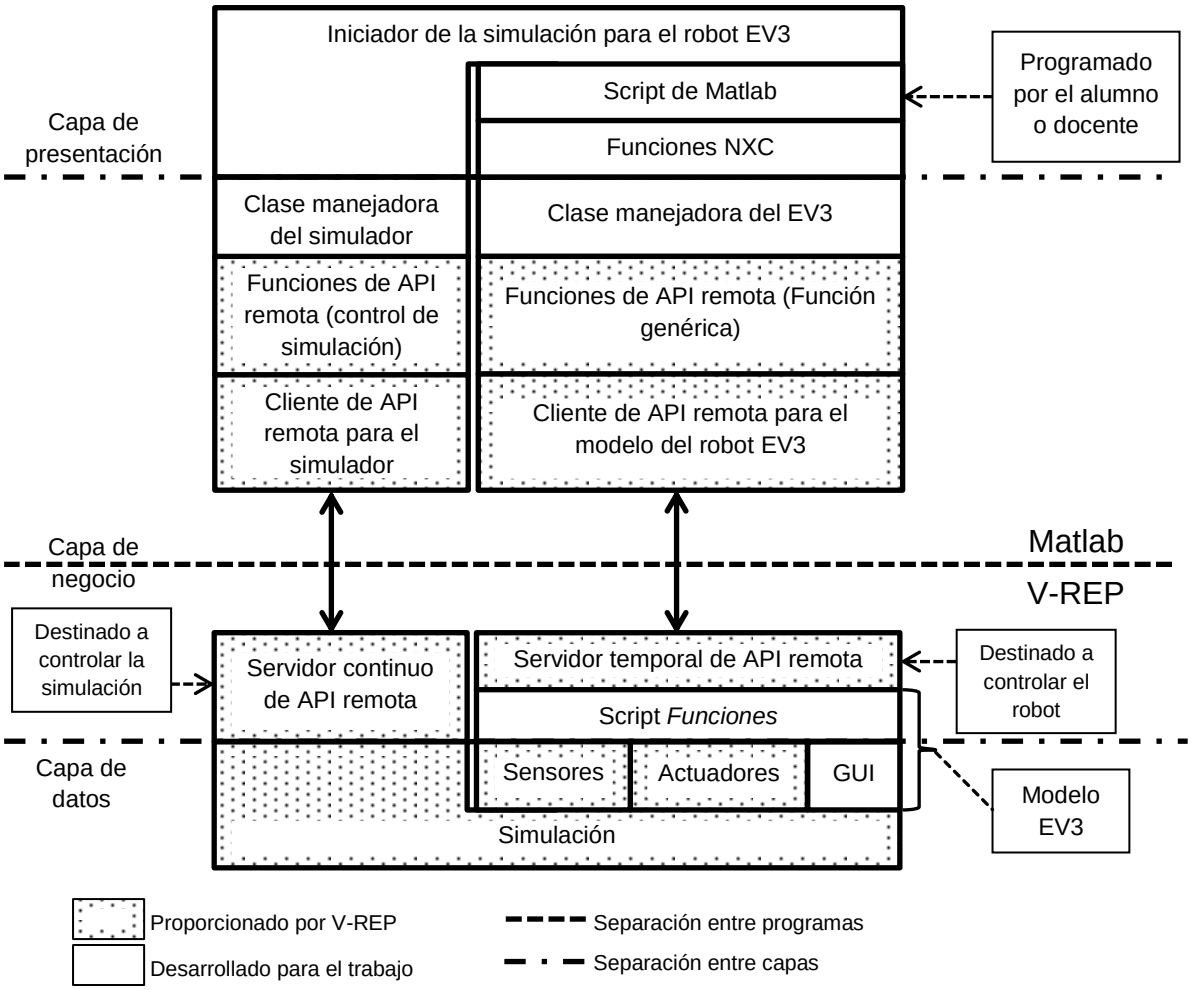

Figura 3. Esquema en profundidad del trabajo, siguiendo una arquitectura de capas

2.1. LEGO Mindstorms

Antes de la creación de LEGO Mindstorms en 1998, la empresa tuvo un largo recorrido de investigaciones y logros sobre el campo de la robótica didáctica. Para comenzar, LEGO en colaboración con el MIT (Instituto Tecnológico de Massachusetts) [18] desarrollaron el "MIT Programable Brick" (Ladrillo Programable), cuyo proyecto fue coordinado por el doctor Seymour Papert, uno de los creadores del lenguaje de programación didáctico Logo [19]. Este "bloque" podía ser programado en un lenguaje muy similar al mencionado anteriormente (Brick Logo). Además, el ladrillo permitía la conexión de sensores y motores para su programación, en la cual la información del programa se transmitía a través de infrarrojos.

En 1998 LEGO lanzó al mercado la primera generación de bloques inteligentes, con el nombre LEGO Mindstorms RCX (*Robotics Command eXplorer*). Se programaba en un lenguaje gráfico muy intuitivo, y posteriormente se añadieron muchos más lenguajes. El bloque RCX poseía 3 puertos para sensores y 3 puertos para actuadores, cuyas comunicaciones, en un principio, se realizaban mediante un

puerto serie y, posteriormente, por infrarrojos con un periférico USB. En cuanto a las especificaciones hardware, poseía un microcontrolador Renesas de 8-bit H8/300, una ROM de 16 Kb y una RAM de 32 Kb.

La segunda generación de Mindstorms y la más popular, lanzada en 2006, fue NXT; esta generación se compone de dos versiones: la educativa y la 2.0. La comunicación con el bloque es mediante cable USB y Bluetooth que, además, permite la comunicación con otros bloques y dispositivos móviles. Con respecto al hardware del controlador principal, posee un microcontrolador ATMEL ARM7 de 32 bits a 48 MHz, una memoria flash de 256 KB y una RAM de 64 KB; esto supone una sustancial mejora del hardware con respecto a su predecesor. En cuanto al controlador para los motores, encontramos un microcontrolador AVR de 8-bit, una memoria flash de 4 Kb y una RAM de 512 bytes. Con respecto a puertos, este modelo presenta tres puertos para actuadores y cuatro puertos para sensores.

NXT goza de una gran comunidad. Además, dispone de un gran surtido de sensores oficiales y no oficiales, como cámaras, acelerómetros y brújulas, además de los sensores de luz, ultrasonido, táctil, etc., que contienen los packs convencionales. Asimismo, cuenta con lenguajes de programación variados aparte de la programación gráfica oficial que proporciona LEGO con NXT-G [20]; algunos de estos lenguajes son: NXC, leJOS [21], Enchanting [22], ROBOTC [23] , una toolbox de Matlab aplicada a NXT [24], etc.

La generación actual es EV3, lanzada en 2013; su principal cambio se basa en aunar los últimos avances tecnológicos en el bloque y proporcionar más precisión en sensores y actuadores. El bloque posee cuatro entradas para actuadores y cuatro entradas para sensores, un puerto USB para agregar un conector wifi o realizar conexiones en cadena, un puerto de tarjetas Micro SD para ampliar la memoria del EV3, un altavoz integrado, un receptor de señales infrarrojas y un receptor Bluetooth. Respecto al hardware, podemos destacar su microcontrolador ARM926EJ-S de 300 MHz, una RAM de 64 MB y 16 MB de memoria Flash. Hoy en día, la comunidad dedicada a la programación de la robótica de LEGO todavía está desarrollando soluciones para la programación del bloque, aun así ya hay algunos lenguajes, en los cuales ya se puede programar: LabVIEW [25], leJOS, Monobrick [26], una toolbox de Matlab aplicada a EV3 [27], etc. Este modelo es al que está enfocado el presente trabajo fin de grado.

2.2. Entornos de simulación robótica

La finalidad de los simuladores de aplicaciones robóticas es múltiple; con ellos se pueden anticipar problemas de diseño, plantear mejoras en la morfología del robot, mejoras a nivel de algoritmo, evitar un desembolso en adquirir un robot real, etc. En

el mercado de simuladores de Robótica disponemos de múltiples opciones; algunos de estos son:

- **Player, Stage y Gazebo** [28]. Son tres aplicaciones creadas por el laboratorio de Robótica de investigación de la USC ("University of Southern California") [29]. Player, es una interfaz o librería para C [30] y C++ [31] de distintos dispositivos robóticos que hacen de driver para los dispositivos robóticos reales. Además, dispone de varios simuladores que emulan un robot en entornos de dos y tres dimensiones, estos son: Stage (2D) y Gazebo (3D).

  Estas 3 aplicaciones se pueden descargar de manera gratuita en la página del proyecto y permiten crear entornos para poder crear algoritmos antes de utilizar un robot real. Las principales dificultades que suponen usar este entorno es saber programar C o C++ y tener conocimientos de Linux para instalar las librerías y simuladores ya que estos tres programas necesitan de paquetes Linux adicionales para poder funcionar, por lo que en Windows esta plataforma no funcionaría. También, estas librerías y aplicaciones no han sido actualizadas recientemente.

- **Microsoft Robotics Developer Studio** [32]. Además de ser un conjunto aplicaciones para desarrollar código en robots reales mediante lenguajes propios de Microsoft y la programación gráfica que posee (VPL), contiene un simulador en tres dimensiones donde se pueden crear y editar escenarios reales para robots.

  La principal ventaja de este entorno es que posee una licencia gratuita con propósitos docentes (para profesores o alumnos) o para aficionados a la Robótica y, además, se pueden desarrollar algoritmos para robots usando la programación gráfica que proporciona, la cual sirve de ayuda para comenzar a programar robots. El inconveniente es que el tiempo que supone aprender dicho entorno es muy lento, ya que contiene una gran cantidad de módulos y programas que pueden suponer un gasto de tiempo a la hora de aprender a controlarlos correctamente; además, al ser una herramienta de Microsoft solo se pueden programar robots con los lenguajes de programación de este (C# [33], Visual Basic [34], etc.), no permitiendo utilizar otros lenguajes.

- **Marilou Robotics Studio** [35]. Es un entorno que permite la simulación de múltiples tipos de robots (brazos manipuladores, robots móviles, humanoides, etc.) en tres dimensiones. Incluye multitud de librerías que permiten la programación del robot modelado para este simulador en múltiples lenguajes (C, C++, C#, etc), además de ser compatible con lenguajes como Matlab o Java. También, permite el modelado de robots en su plataforma en un

entorno gráfico, pudiendo calcular parámetros que son importantes en la Robótica como son masa, centro de gravedad, etc.

Las ventajas que se destacan son que se permite el modelado y la simulación mediante una interfaz gráfica que, hipotéticamente, permite ahorrar tiempo y, además, soporta una gran multitud de lenguajes de programación para programar robots; adicionalmente, está disponible para multitud de plataformas (Windows y distribuciones Linux). El principal inconveniente de este entorno es que se requiere de adquirir una licencia para obtenerlo, incluso para la versión de estudiantes, no pudiendo estar al alcance de todos.

- **V-REP**. Este es el entorno de simulación que hemos elegido para modelar y simular el robot del presente trabajo. Toda la información relativa a este simulador se puede encontrar con más detalle en la sección 2.3. Las principales razones por la que se ha seleccionado este simulador son:

  - Todas las versiones de dicho entorno están disponibles en todos los sistemas operativos de hoy día (Windows, distribuciones Linux y MacOS), por lo que el modelo desarrollado para esta plataforma es reutilizable.

  - Actualizaciones de software constantes.

  - Documentación extensa y completa [36].

  - Disponible una versión completa educacional gratuita, por lo que no implica un desembolso económico para docentes y alumnos.

  - Permite la importación de modelos procedentes de programas de modelado 3D, ya que facilitan la tarea de la realización del modelo en tres dimensiones.

  - Integración con múltiples lenguajes de programación, entre ellos Matlab, lo cual nos interesa para la interacción entre el simulador y dicho lenguaje.

2.3. Simulador V-REP

V-REP, según sus creadores, es el simulador de robótica con más funciones, recursos, o API, más elaborado. Algunas de sus especificaciones son:

- Posee más de 400 funciones diferentes en su API nativa.
- Cuatro motores físicos para la simulación.

- Cálculo de cinemática.
- Detección de obstáculos.
- Cálculo de distancia mínima.
- Sensores de visión, usados para los sensores de luz y color de este trabajo.
- Sensores de proximidad, usado para el modelado del sensor de ultrasonidos del robot EV3.
- Cálculo de trayectorias.
- Interfaces de usuario integradas y personalizadas.
- Simulación de corte superficial; esta característica resulta muy útil para simular robots industriales donde se requiera el corte de una pieza.
- Registros de datos y visualización, ya sea en gráficos de tiempo, gráficos X/Y o curvas 3D.
- Navegador de modelos con función de arrastrar y soltar, incluso durante la simulación.
- Otras características, como función de grabación de vídeo, simulación de pintura, etc.

Como se ha comentado en las especificaciones, este entorno de simulación dispone de una gama de sensores y actuadores que nos han permitido modelar el robot; su manejo y funcionalidad se describe en las siguientes subsecciones.

2.3.1. Interfaz de V-REP

Toda la descripción de la interfaz gráfica del entorno de simulación V-REP (Figura 4) se encuentra en la documentación referenciada [37]. No obstante, se van a describir algunas características, ya que a lo largo de la memoria se hace uso de la terminología de algunas de estas herramientas; dichas herramientas son:

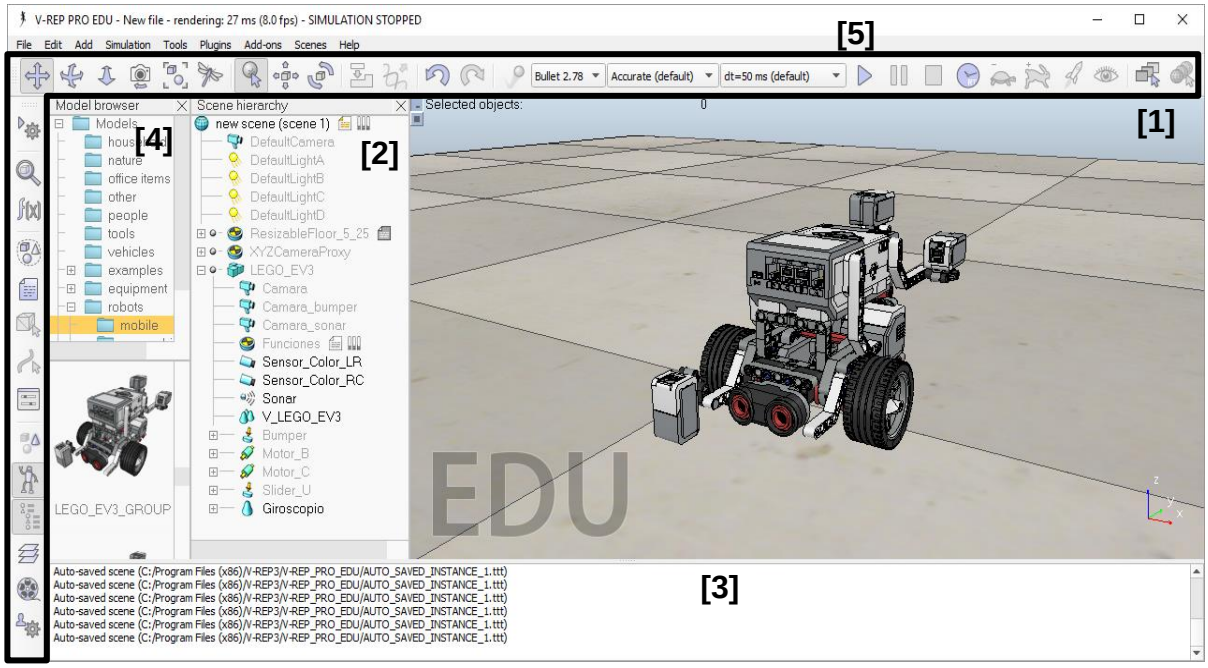

Figura 4. Entorno de simulación V-REP

[1]. Escena. En esta zona es donde se va a realizar el modelado del robot a simular y donde se va a producir el transcurso de la simulación.

[2]. Jerarquía de la escena. Este lugar muestra el contenido de una escena, es decir, los objetos que la componen. Los objetos pueden disponerse en forma de árbol jerárquico para construir un robot, permitiendo que los objetos que lo componen no se suelten o simplemente para agrupar objetos.

[3]. Barra de estado. Muestra información de operaciones realizadas, comandos y mensajes de error del intérprete del lenguaje Lua. También puede mostrar mensajes procedentes de scripts desarrollados mediante la función de la API nativa de V-REP "simAddStatusBarMessage".

[4]. Explorador de modelos. Aquí es donde se encuentran modelos ya creados que ofrece V-REP para su uso, ya sean robots, sensores o simples modelos (sillas, mesas, edificios, etc.). Muestra en la parte superior una estructura de carpetas y en su parte inferior las miniaturas de los modelos que se encuentran en la carpeta seleccionada; mediante una operación de arrastrar y soltar ("drag and drop") pueden irse cargando los modelos en la escena.

[5]. Barras de herramientas. En este lugar se encuentran las funcionalidades que normalmente se usan en el simulador (deshacer, rehacer, iniciar una simulación, manipulación de objetos, etc.).

2.3.2. Actuadores

En primer lugar, vamos a hablar de los actuadores, en los que se han utilizado solo articulaciones o, según la nomenclatura del programa para explicarlas, “joints”. Una articulación en V-REP tiene cinco modos de operación: modo pasivo (“pasive mode”), modo de cinemática inversa (“inverse kinematics mode”), modo dependiente (“dependent mode”), modo de movimiento (“motion mode”) y modo de torque o fuerza (“torque or force mode”). En el modelado de nuestro robot se ha utilizado el modo de torque o fuerza, ya que nos permite un control total sobre las articulaciones mediante la programación; es decir, con el uso de la API que proporciona el simulador, por ejemplo, podemos darle unos valores de velocidad a la articulación y ésta se moverá o rotará (dependiendo del tipo de articulación) a la velocidad indicada. Según el tipo de articulación distinguimos tres tipos:

- **Prismáticas**. Muy usadas en el modelado de robots manipuladores o brazos robóticos. Estas se mueven de manera lineal según la orientación que le hayamos dado.

- **Rotacionales**. Estas realizan movimientos rotacionales. Se utilizan, por ejemplo, para modelar los motores en robots móviles que usan ruedas para desplazarse, un ejemplo es el robot modelado para este trabajo. También se utilizan para simular brazos manipuladores.

- **Esféricas**. Surgen de la combinación de tres articulaciones rotacionales, por lo que tienen 3 grados de libertad en una sola articulación. Están configuradas para usar los ángulos de Euler (alfa, beta y gamma) y consiguen una gran abstracción a la hora de programarlas.

2.3.3. Sensores

En la parte de sensores, como se ha mencionado anteriormente, este entorno de simulación proporciona varios sensores de los cuales se han utilizado los siguientes:

- **Sensores de visión (“Vision sensors”)**. Estos sensores captan imágenes para su posterior tratamiento, como si fuesen una cámara, pero con más utilidades; por ejemplo: se puede obtener la media de luminosidad de este sensor mediante su API, pudiendo así implementar un algoritmo de seguimiento de líneas. Existen dos tipos de estos sensores: ortográficos y de perspectiva.

- **Sensores de proximidad (“Proximity sensors”)**. Este entorno de simulación ofrece la posibilidad de añadir sensores de proximidad, los cuales nos ayudan a detectar objetos y medir distancias según el rango de detección

y el tipo que le hayamos especificado. En cuanto a los tipos que hay, podemos destacar: de rayo, de pirámide, de cilindro, de disco y de cono.

- **Sensores de fuerza (“Force sensors”)**. Los sensores de este tipo poseen múltiples funcionalidades, como detectar con qué fuerza se está presionando un objeto. En nuestro modelo, los usaremos para modelar el sensor táctil y para que la parte trasera del robot pueda moverse.

2.3.4. Formas

El simulador V-REP dispone de formas para realizar modelos de robots. Entre ellas destacamos:

- **Forma aleatoria simple (“Simple random shape”)**. Puede representar cualquier modelo. Se caracteriza por tener un color y un conjunto de atributos visuales (textura, geometría, muestreo de bordes, etc.). No está recomendada para simular colisiones, ya que pueden hacer que la simulación sea inestable y muy lenta.

- **Forma aleatoria compuesta (“Compound random shape”)**. Puede representar cualquier conjunto de modelos. Puede tener múltiples colores y conjuntos de atributos visuales. Tampoco está recomendada para simular colisiones.

- **Forma convexa simple (“Simple convex shape”)**. Representa un modelo convexo con un color y un conjunto de atributos visuales. Estas formas se optimizan para las colisiones, pero siempre es mejor usar formas puras, las cuales se explicarán más adelante.

- **Forma convexa compuesta (“Compound convex shape”)**. Representa un conjunto de modelos convexos con múltiples colores y conjuntos de propiedades visuales. Al ser un conjunto compuesto por formas convexas, también están optimizadas para la simulación de colisiones.

- **Forma pura simple (“Pure simple shape”)**. Representa una forma primitiva, estas son: cuboide, cilindro, esfera, plano y disco. Estas formas son la opción que recomiendan los creadores de V-REP para la simulación de colisiones y físicas, ya que al ser diseñadas para este fin, la simulación de colisiones se realiza de manera rápida y estable.

- **Forma pura compuesta (“Pure compound shape”)**. Representa un grupo de cuerpos primitivos. Al estar compuesto de formas puras, también es la mejor opción para las colisiones.

- **Terreno (“Heightfield shape”)**. Puede representar un terreno como una cuadrícula regular, donde solo la altura de este puede ser cambiada. Son consideradas también como cuerpos simples puros, ya que están optimizadas para el cálculo de colisiones. Esta forma no se ha utilizado en ningún momento para el desarrollo del trabajo, porque no ha sido necesario el modelado de terrenos.

Para poder modelar un robot en V-REP, no se suele utilizar este entorno para su diseño, sino que se usa una herramienta externa de diseño 3D para el modelado, como por ejemplo: AutoCAD [38], Blender [39], 3D Studio Max [40], etc. Los modelos construidos con estos programas proporcionan más detalle, pero, a la hora de simularlos y utilizarlos para las colisiones, hacen que la simulación en V-REP vaya lenta al reconocerlas como formas aleatorias. Para esto, se utilizan las formas puras para la simulación de colisiones, ya que están diseñadas expresamente para se realice de la manera más eficiente posible. Esto se explicará con más profundidad posteriormente, concretamente en la sección 3.2.

2.3.5. Scripts

En el ámbito de la programación, V-REP dispone de muchos enfoques, como: programación mediante scripts incrustados, plugins, nodos ROS [41], API remotas, etc.

En este trabajo, hemos usado una combinación de API remotas y scripts incrustados; la decisión de usar scripts incrustados reside en que podemos implementar un conjunto de funciones en lenguaje Lua en el modelo de V-REP para el comportamiento del robot y adaptar los valores de sensores y actuadores del simulador a los del robot real (“script *Funciones*” en la Figura 3), con esto conseguimos que el robot tenga su repertorio de funciones independientemente de la implementación en Matlab (u otro lenguaje de programación) para ejecutar dichas funciones, además, esto provoca que el código en el simulador sea reutilizable de cara a la utilización de la API remota. Estas funciones serán llamadas por la función genérica de la API remota en Matlab, la cual se explicará en la subsección 2.4.4.

Antes de describir el funcionamiento de los scripts incrustados, para que estos scripts y la simulación funcionen correctamente, cada escena en V-REP posee un script principal que se encarga de gestionar la simulación.

El script principal es llamado por cada paso de simulación ("time step") (Figura 1), cuando ésta va a comenzar, y cuando la simulación termina.

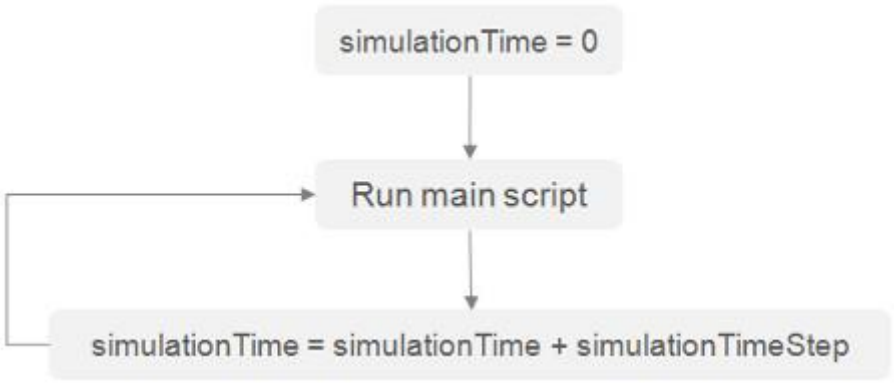

Figura 5. Ejecución del bucle de simulación (figura tomada de la documentación de V-REP)

Este script principal se divide en 3 partes:

- **Parte de inicialización**. Esta parte se ejecuta al comienzo de la simulación. Se encarga de cargar todos los elementos en la simulación.

- **Parte sistemática.** Esta parte se ejecutará en cada paso de simulación. Se encarga de manejar todas las funcionalidades de V-REP (detección de colisiones, dinámicas, sensores de proximidad, cálculo de la distancia, etc.). En esta parte, según sus creadores, dos comandos son muy importantes: "simLaunchThreadedChildScripts" y "simHandleChildScripts". El primero de los mencionados se encarga de lanzar los scripts hijo hilados, mientras que el segundo de estos dos se encarga de lanzar los scripts hijo no-hilados; ambos tipos de scripts se explicarán posteriormente. Sin estos comandos, los scripts hijo no se ejecutarán, y si algún elemento de la escena tiene alguna funcionalidad implementada con este tipo de scripts, provocaría que estos no operasen. La parte sistemática del script principal está dividida en:

  - **Actuación ("actuation")**. Se encarga de llamar a la fase de actuación en los scripts hijo no-hilados y de activar los actuadores del simulador (articulaciones, fresadoras, gráficos, etc.).

  - **Sondeo ("sensing")**. Se encarga de llamar a la fase de sondeo en los scripts hijo no-hilados y de activar los sensores del simulador (sensores de visión, de fuerza, de proximidad, etc.) para poder obtener valores actualizados.

  - **Exposición ("display")**. Se encarga de mostrar el nuevo estado de la simulación en la escena.

El orden de ejecución de estas fases es la que aparece en la Figura 6.

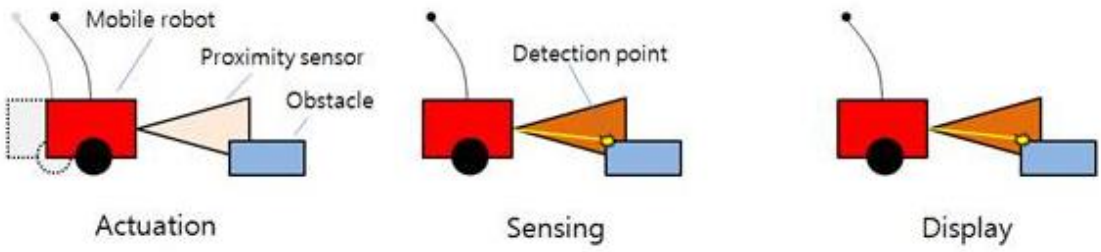

Figura 6. Fases de la parte sistemática del script principal (figura tomada de la documentación de V-REP).

- **Parte de restauración**. Esta parte se ejecutará una vez justo antes de que la simulación finalice. Se encarga de restaurar la configuración inicial de los objetos, la limpieza de estados de los sensores, los estados de colisión, etc.

La estructura de script principal de cada escena se puede modificar, pero según la documentación de V-REP, no es recomendable, ya que podría hacer la simulación inestable o que no funcionase correctamente.

En la programación de los scripts incrustados, se asocia un script a un objeto de la escena, a estos se les conoce como scripts hijo ("child scripts"), y se programan utilizando el lenguaje Lua.

Pasamos ahora a describir los scripts hijo. Como se mencionó anteriormente, pueden ser de dos tipos:

- **Scripts hijo no-hilados ("non-threaded child scripts")**. Estos scripts son del tipo paso a través ("pass-through"). Esto significa que cada vez que son llamados por el script principal de la escena deben realizar alguna tarea y luego devolver el control de la simulación al script que los ha llamado. Si no se devuelve el control, debido a, por ejemplo, un bucle infinito, la simulación no responderá. Estos scripts operan como funciones, y son llamadas dos veces por cada paso de simulación: durante la fase de actuación y durante la fase de sondeo. Este tipo de scripts, según la documentación del simulador V-REP, deben ser elegidos preferentemente sobre los scripts hilados. Un script de estas características, se divide en 4 partes o fases:

  - **Parte de inicialización ("initialization part")**. Esta parte se ejecutará una vez. Esta fase ocurre al principio de la simulación en los objetos con estos scripts cargados previamente en la escena, o cuando se agregan en el transcurso de ésta. La documentación recomienda que las referencias a objetos de la escena, se declaren en esta fase como variables globales del script.

  - **Parte de actuación (“actuation part”)**. Se ejecutará por cada paso de simulación durante la fase de actuación.

  - **Parte de sondeo (“sensing part”)**. Se ejecutará por cada paso de simulación durante la fase de sondeo.

  - **Parte de restauración (“restoration part”)**. Se ejecutará solo una vez cuando la simulación haya acabado, o antes de que el script sea destruido en mitad de esta.

- **Scripts hijo hilados (“threaded child scripts”)**. Estos scripts son lanzados en un hilo de ejecución aparte del hilo de simulación. Cuando un script hilado está ejecutándose, no podrá ejecutarse de nuevo; cuando estos finalizan, pueden ser relanzados. Estos scripts no han sido utilizados para el desarrollo del trabajo, ya que, según sus creadores, consumen muchos más recursos, gastan tiempo de procesamiento, y son menos sensibles a una orden de parada de la simulación, por lo que la simulación no pararía en el momento que se desee.

V-REP también dispone de formas de comunicación entre scripts, las cuales se explican en la documentación referenciada [42]; entre ellas se han usado las señales (“signals”), ya que es una manera muy sencilla de pasar datos entre scripts.

En cuanto a la programación con la API remota de V-REP para llamar funciones en estos scripts, se explicará en las sección 2.4.

2.3.6. Otros objetos

En V-REP, también existen otros objetos para modelar robots, modificar la escena añadiendo obstáculos, gráficos, etc. Algunos de estos que se han utilizado para el modelado de nuestro robot, son:

- **“Dummy”**. Es el objeto más simple que dispone el simulador. Son puntos con orientación, por lo que pueden ser vistos como un sistema de coordenadas. Los “dummies” sirven para el modelado de brazos manipuladores a la hora de implementar modelos cinemáticos; estos son utilizados para representar el sistema de coordenadas del efector final en los mencionados brazos. También pueden ser usados como objetos de apoyo para múltiples fines; en nuestro caso, se ha usado un “dummy” para adjuntar el script hijo que contiene todas las funciones de control del robot.

- **Cámara (“camera”)**. Las cámaras son objetos con los que se pueden obtener vistas de la escena. V-REP permite crear todas las cámaras que se necesiten, en la que cada una ofrece una vista diferente de la escena. Por ejemplo, es posible una cámara para obtener una vista aérea de la escena y, otra para conseguir una vista en primera persona del robot que se está modelando. Para poder visualizar la vista de la cámara creada, se pueden crear vistas flotantes o páginas asociadas a dichas cámaras [43].

Existen otros objetos como: espejos, luces, gráficos, fresadoras, etc., que no se han descrito ya que quedan fuera de los objetivos del presente trabajo.

2.4. API remota de V-REP

Como se ha mencionado anteriormente, este trabajo se ha realizado mediante la combinación de scripts incrustados y el uso de la API remota. El uso de esta API remota reside en que es una solución bastante sencilla para programar y no tendríamos que instalar más software aparte de Matlab y V-REP como pasaría con la programación con los nodos ROS. Adicionalmente, la API remota, al estar implementada mediante sockets, puede controlar la simulación desde otro PC.

La API remota del simulador está compuesta por una serie de comandos en forma de funciones que permiten controlar la simulación y obtener datos de esta, las cuales pueden ser llamadas desde otro lenguaje de programación, como: Matlab, C, C++, Lua, Python [44], Java [45], Octave [46] o Urbi [47]. La comunicación con el programa desarrollado en uno de los lenguajes mencionados anteriormente y el simulador es síncrona o asíncrona; por defecto, la API usa la comunicación asíncrona. La API remota de V-REP implementa una arquitectura cliente-servidor [48], donde la aplicación es el cliente y el simulador es el servidor (Figura 7).

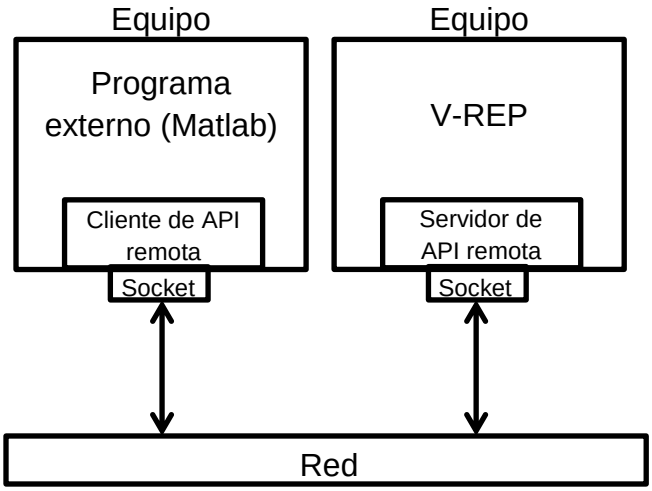

Figura 7. Arquitectura cliente-servidor aplicada a la API remota de V-REP

En resumen, las API remotas de V-REP establecen un canal de comunicación entre el simulador y un programa externo con el que se controla la simulación mediante llamadas a procedimiento remoto [49] desde dicho programa. En el caso de nuestro trabajo nos permite comunicar Matlab con V-REP.

2.4.1. Servidor

El servidor de API remota se encarga de recibir y ejecutar los comandos enviados por el cliente o enviar información a éste, como, por ejemplo, valores de sensores. La implementación del servidor se localiza en un plugin dentro del directorio de instalación de V-REP; éste ejecuta el comando enviado por el cliente antes de que el script principal sea ejecutado en cada paso de simulación. Existen dos tipos de servidores en el simulador:

- **Servidor continuo de API remota ("continuous remote API server")**. Es el servidor que siempre se inicia al arrancar V-REP. Siempre es iniciado con la configuración del fichero situado en el directorio de instalación del simulador *remoteApiConnections.txt*; es posible modificar las propiedades de dicho fichero, como, por ejemplo, el puerto al que se va a conectar el servidor. Con este tipo de servidor, obtenemos el control del entorno de simulación desde la aplicación cliente, sin necesidad de iniciar la simulación. Este servidor es al que hemos recurrido para controlar el simulador desde Matlab. Existen otras maneras de iniciar el servidor continuo de API remota, pero no entraremos en detalles al quedar fuera del ámbito del TFG.

- **Servidor temporal de API remota ("temporary remote API server")**. Este tipo de servidor es iniciado desde un script en V-REP. El servidor temporal de API remota es el preferido según la documentación, ya que el usuario es el que tiene el control del servidor para iniciarlo o pararlo. Cuando un servidor temporal es iniciado desde un script de simulación, como, por ejemplo, un script hijo, normalmente el servidor se detiene automáticamente cuando la simulación ha finalizado. Este tipo de servidores, se pueden iniciar o parar con dos funciones de la API nativa (no confundir con API remota) del simulador: "simExtRemoteApiStart" y "simExtRemote-ApiStop". Esta última es la que hemos usado para controlar el robot porque, en el momento destruir (o parar) el servidor, detiene las funciones que se ejecutan en el modo de transmisión de datos (explicación dada en el apartado 2.4.3).

2.4.2. Cliente

El cliente de API remota se encarga de enviar los comandos al servidor en V-REP desde un programa ajeno al simulador (en nuestro caso Matlab) y, en algunos casos,

recibir las respuestas de estos. Este cliente y su librería de funciones para cada lenguaje de programación poseen su manera de importarlo. Este trabajo solo se ha centrado en la API remota de Matlab, por lo que vamos a explicar su importación. Para ello, necesitaremos los siguientes archivos:

- *remoteApiProto.m*
- *remApi.m*
- *remoteApi.dll* (Windows), *remoteApi.dylib* (MacOS) o *remoteApi.so* (Linux).

Todos estos archivos se encuentran en el directorio de instalación del simulador, concretamente en la ruta *programming/remoteApiBindings/Matlab* separados según el tipo de procesador que ejecuta el sistema operativo (x86 o x64).

Una vez copiados estos archivos al directorio donde estemos programando, escribiremos en la consola de Matlab *vrep=remApi('remoteApi')* para crear el objeto que contienen las funciones y constantes. Para iniciar el cliente llamaremos a la función *vrep.simxStart*, la cual nos devolverá un valor de conexión, que llamaremos "clientID".

Para conocer con detalle qué funciones se pueden usar en Matlab con la API remota, en la sección de referencias se encuentra un enlace a la documentación correspondiente [12].

Las funciones de la API remota se llaman casi de igual manera que las funciones de la API nativa que posee el simulador en lenguaje Lua, si bien existen dos diferencias claves:

- Una gran cantidad de funciones para esta API requieren de dos argumentos extra:
  - **Modo de operación**. En este argumento se le indica al servidor cómo se va a ejecutar el comando en el simulador; los modos de operación se explicarán posteriormente en el apartado 2.4.3.
  - **Parámetro "clientID"**. Este parámetro nos es devuelto después de llamar a la función "simxStart" en la aplicación que ejecuta el cliente (en nuestro caso Matlab), y es requerido para cada llamada a las funciones de la API remota.
- La mayoría de estas funciones devuelven 1 byte que es el valor que indica como se ha ejecutado la función. Estos valores representados por constantes son:

- **“simx_return_ok” (0)**. La función se ha ejecutado correctamente.

- **“simx_return_novalue_flag” (1)**. No existe una respuesta en el buffer de entrada. Esto puede ser tratado como un error dependiendo del modo de operación, que explicaremos en la siguiente subsección.

- **“simx_return_timeout_flag” (2)**. Se ha esperado demasiado tiempo en la respuesta, esto puede ser debido a que la red esté caída o saturada.

- **“simx_return_illegal_opmode_flag” (4)**. El modo de operación especificado no es soportado por la función.

- **“simx_return_remote_error_flag” (8)**. La función ha causado un error en el lado del servidor.

- **“simx_return_split_progress_flag” (16)**. El hilo de ejecución de las comunicaciones todavía está procesando un comando del mismo tipo.

- **“simx_return_local_error_flag” (32)**. Se ha producido un error en el lado del cliente, es decir, en nuestra aplicación.

- **“simx_return_initialize_error_flag” (64)**. No se ha llamado a la función “simxStart”, esto significa que no se ha iniciado el canal de comunicación entre el cliente y el servidor de API remota.

2.4.3. Mecanismos de comunicación

En esta subsección explicaremos la necesidad de un valor de retorno y un modo de operación. Esto se debe a que la comunicación es realizada mediante sockets; para ello se necesitan parámetros para saber de qué manera se van a ejecutar los comandos enviados al servidor y para comprobar que esto se realice en V-REP de manera correcta. Para que el cliente se comunique con el servidor de esta forma, la API remota dispone de los siguientes mecanismos:

- **Llamadas a funciones no bloqueantes (“Non-blocking function calls”)**. Este mecanismo está destinado a aquellas funciones de la API remota que simplemente envían datos a V-REP sin necesitar una respuesta del servidor, como, por ejemplo, enviar una velocidad angular a una articulación rotacional, evitando latencia provocada por la comunicación entre las dos entidades (cliente y servidor). Este mecanismo se activa con el modo de operación

"simx_opmode_oneshot" y devuelve el valor de retorno "simx_return_no-value_flag" si no se ha producido ningún error. En la Figura 8 se muestra un diagrama de secuencia de una ejecución de una llamada a función no bloqueante.

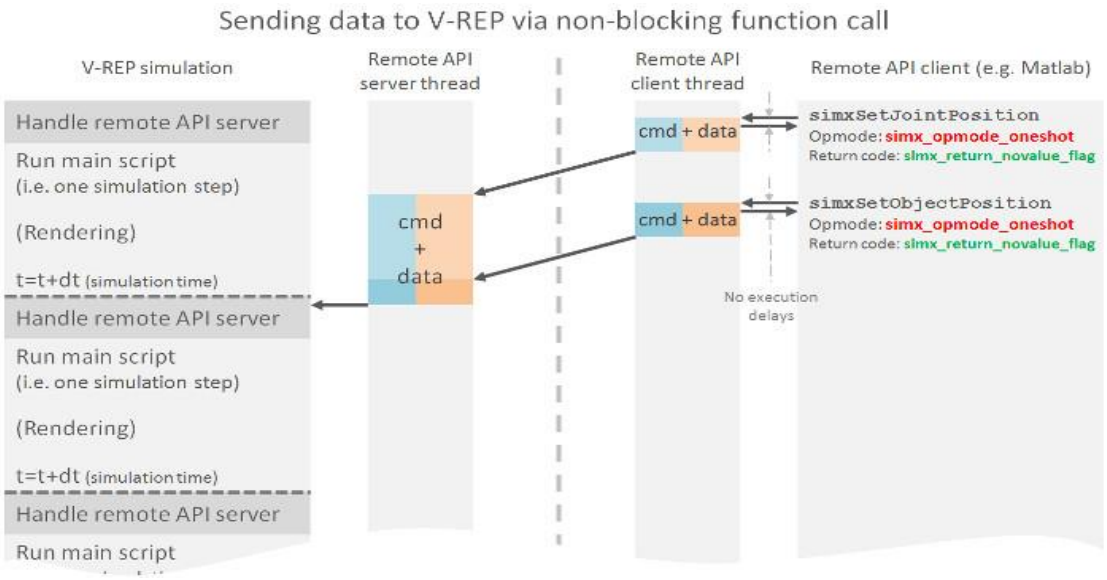

Figura 8. Ejecución de una función no-bloqueante (figura tomada de la documentación de V-REP).

- **Llamadas a funciones bloqueantes ("Blocking function calls")**. Este modo, al contrario de las llamadas a funciones no bloqueantes, está pensado para aquellas funciones de la API remota que necesitan expresamente una respuesta del servidor; un ejemplo es obtener la referencia de un objeto de la escena del simulador. Este mecanismo se activa con el modo de operación "simx_opmode_blocking"; el valor de retorno de este modo de operación será "simx_return_ok" si la función se ha ejecutado correctamente. Con más detalle, en la Figura 9 disponemos del diagrama de secuencia de una llamada a función bloqueante.

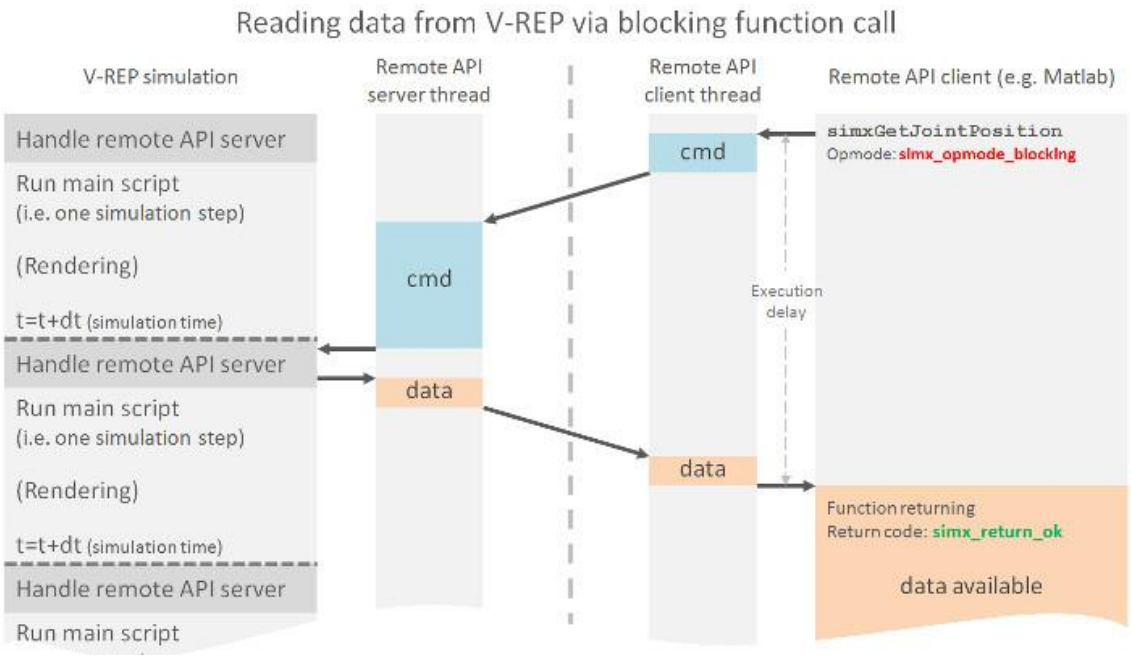

Figura 9. Ejecución de una función bloqueante (figura tomada de la documentación de V-REP).

- **Transmisión de datos (“Data streaming”)**. Está indicado para que el servidor pueda anticipar el comando que el cliente requiere y, posteriormente, enviar los datos por cada paso de simulación almacenándolos en un buffer. La ventaja de usar este modo de operación radica en que, al anticipar los datos, no tenemos que hacer llamadas bloqueantes en el flujo de ejecución de la aplicación que ejecuta el cliente (en nuestro caso Matlab), por lo que la ejecución de un algoritmo que requiera recoger muchos datos de sensores se producirá sin apenas latencia. Para utilizar este mecanismo de comunicación, el cliente tiene que indicar al servidor, en orden, los siguientes modos de operación:

  - **“simx_opmode_streaming”**. Se encarga de avisar al servidor que el comando que lleva este modo de operación como argumento, se va a ejecutar para transmitirlo por cada paso de simulación y, posteriormente, recoger los datos depositados en el buffer. Este modo de operación, además, devuelve el valor de retorno “simx_return_novalue_flag”.

  - **“simx_opmode_buffer”**. Se encarga de recoger los datos transmitidos del buffer, por lo que ningún comando es enviado al servidor. Este modo puede utilizarse todas las veces que se requiera para extraer datos, habiendo llamado, previamente, al modo descrito en el punto anterior. Si no ha ocurrido ningún error, la función devolverá el valor de retorno “simx_return_ok”.

- **“simx_opmode_discontinue”**. Se encarga de avisar al servidor de que el comando que lleva este modo de operación como argumento, el cual se está transmitiendo al cliente, pare de hacerlo. En este modo, por cada llamada se devuelve el valor de retorno “simx_return_no-value_flag”.

- **“simx_opmode_remove”**. Se encarga de limpiar el buffer del cliente del comando que lleva este modo de operación como argumento. Cuando el comando es ejecutado con este modo, no se devuelven valores a excepción del valor de retorno, el cual es “simx_return_no-value_flag”.

El diagrama de secuencia de la Figura 10 muestra el funcionamiento este mecanismo de comunicación.

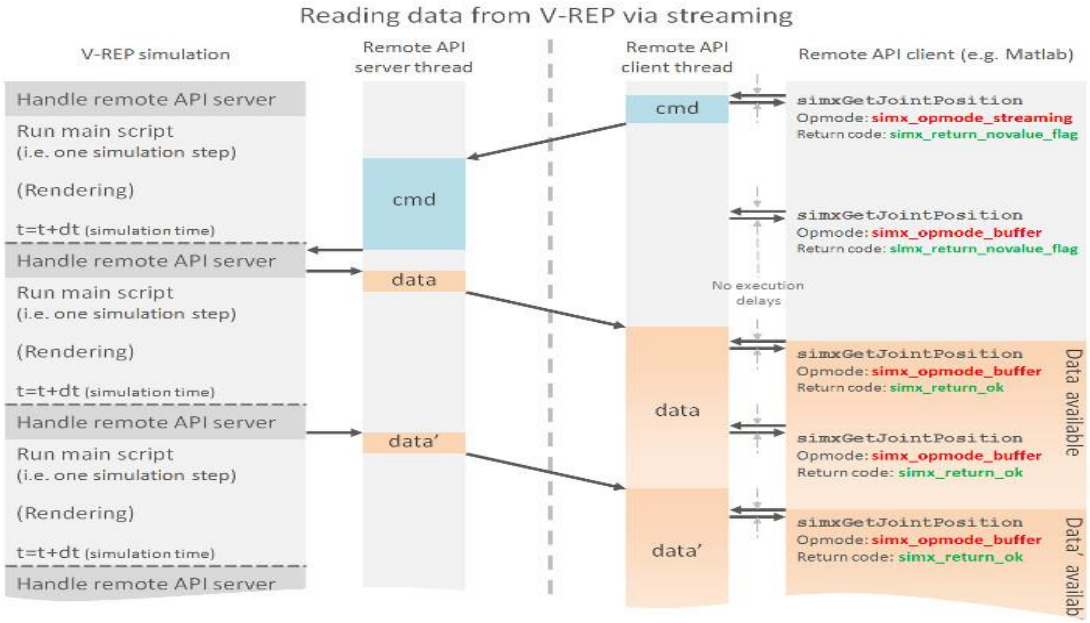

Figura 10. Ejecución del modo de transmisión de datos (figura tomada de la documentación de V-REP).

- **Operaciones síncronas (“Synchronous operations”)**. Algunas veces la aplicación necesita estar sincronizada con el simulador, y esto se consigue controlándola desde el cliente. Para esto existe la función de API remota “simxSynchronous”, donde se establece que la simulación debe ser sincronizada por la aplicación; esta función sirve también para deshabilitar este modo. Para indicar el siguiente paso de simulación se debe usar “simxSynchronousTrigger”. En la Figura 11 se muestra un ejemplo de ejecución de este modo, aunque no ha sido necesario para la elaboración de la toolbox de Matlab.

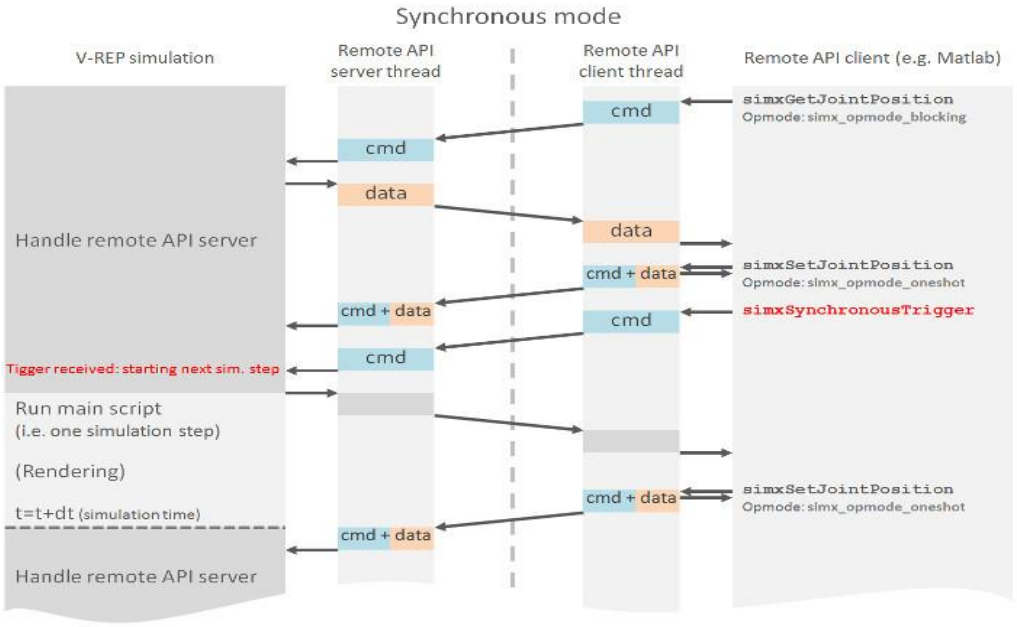

Figura 11. Ejemplo de ejecución del modo síncrono (figura tomada de la documentación de V-REP).

2.4.4. La función genérica

Esta función procedente de la librería de la API remota para el programa que ejecuta el cliente (en nuestro caso Matlab) permite lanzar funciones que están declaradas en un script dentro del simulador; con ello se consigue extender la funcionalidad de API remota sin tener que modificar sus archivos a bajo nivel. Esta función de la API remota para Matlab es la que hemos empleado para comunicarnos con el simulador para transmitir y recibir datos del robot, ya que nos proporciona las siguientes ventajas:

- Nos permite la abstracción de funciones para llamar desde Matlab, es decir, sin saber cómo está implementada dicha función en el simulador.

- Permite escalabilidad al crear más funciones en el script del robot.

- Puede evitarse el uso de referencias de objetos de V-REP en el programa que ejecuta el cliente, haciendo que el control en este tenga menos llamadas al servidor, además de eliminar variables que dificultan la abstracción a la hora de programar la aplicación o, en nuestro caso, la toolbox para controlar el robot.

- Permite usarse a modo de interfaz del robot, es decir, un conjunto de funciones que implementan toda la funcionalidad de este.

Los parámetros de entrada y salida se pasan mediante arrays, donde cada uno de ellos representa una colección de datos de cada tipo de datos. Para poder utilizar la función genérica de la API remota para Matlab para llamar a funciones creadas por el programador en el simulador, se tiene que declarar una función en lenguaje Lua dentro del script con los atributos de entrada en el siguiente orden:

1) **"inInts"**: Array de datos de tipo entero.
2) **"inFloats"**: Array de datos de tipo flotante.
3) **"inStrings"**: Array de datos de tipo cadena de caracteres.
4) **"inBuffer"**: Buffer en formato cadena de caracteres, este parámetro no ha sido usado, ya que no ha sido necesario para transmitir datos.

En cuanto a los parámetros de salida, se van devolviendo en el mismo orden que los de entrada. En la Figura 12 se muestra el formato para declarar una función en un script de V-REP para que pueda ser llamada desde, por ejemplo, Matlab mediante la función genérica.

```
funcionGenerica = function(inInts,inFloats,inStrings,inBuffer)

    -- CUERPO DE LA FUNCION

    return outInts, outFloats, outStrings, outBuffer
end
```

Figura 12. Formato de función en V-REP, que puede ser llamada por Matlab mediante la función genérica

Para poder llamar a dichas funciones desde Matlab, se tiene que llamar a la función "simxCallScriptFunction" (función genérica) cuyo formato aparece en la Figura 13; los parámetros se describen en la documentación de las funciones de la API remota, citada previamente en la subsección 2.4.2.

```
[returnCode, outInts, outFloats, outStrings, outBuffer] = ...
    simxCallScriptFunction(...
                           clientID,...
                           scriptDescription,...
                           scriptHandleOrType,...
                           functionName,...
                           inInts,...
                           inFloats,...
                           inStrings,...
                           inBuffer,...
                           operationMode);
```

Figura 13. Formato de la función genérica en Matlab

2.5. Herramienta de modelado LeoCAD

Para la realización de este trabajo no solo se han utilizado V-REP y Matlab. Previamente, se ha tenido que construir el modelo físico del robot (Figura 14); para ello, hemos usado un software llamado LeoCAD.

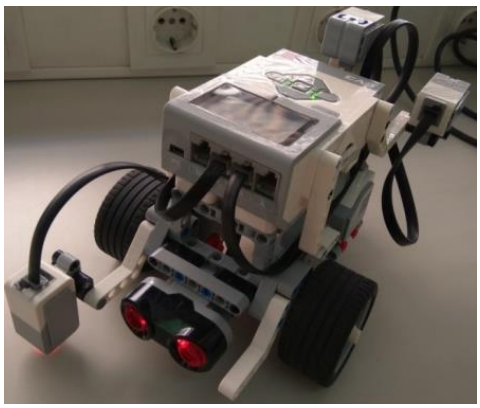

Figura 14. Modelo real del robot diseñado en LeoCAD

LeoCAD, como su propio nombre indica, es una herramienta CAD para crear modelos virtuales de LEGO. El porqué de la elección de este software para el diseño en tres dimensiones del robot radica en que su interfaz es muy intuitiva para usar, ya que no se necesitan conocimientos muy avanzados de diseño gráfico para poder construir un modelo de LEGO. La interfaz se muestra en la Figura 15 y, como podemos observar, ofrece distintas secciones:

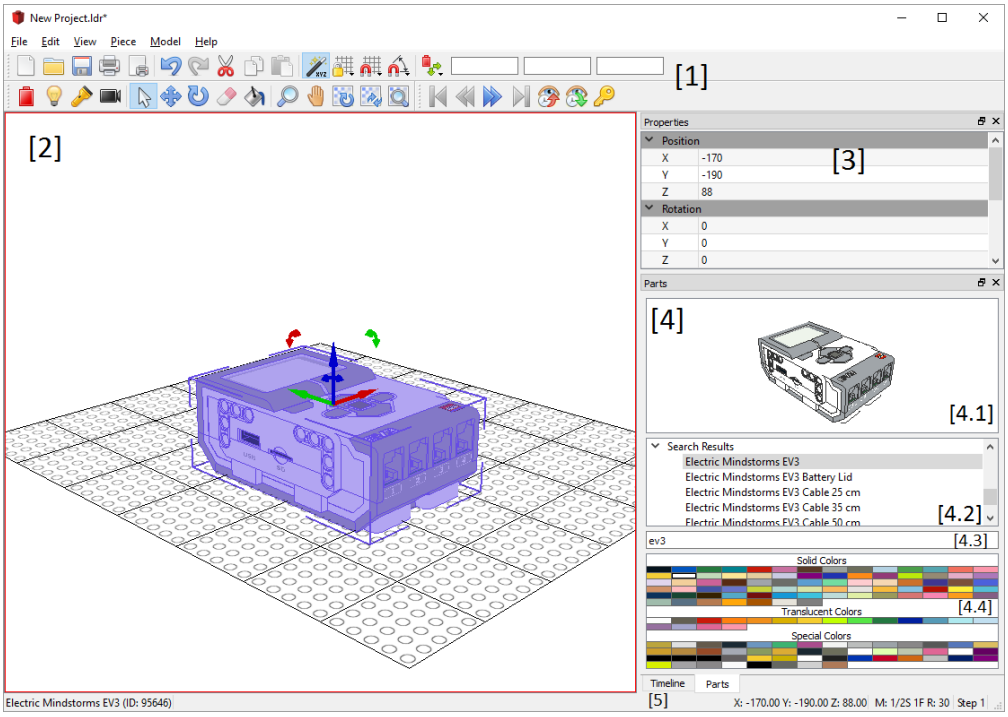

Figura 15. Interfaz de LeoCAD.

[1]. Barra de herramientas. Aquí se encuentran las principales herramientas para modelar. Algunas de las que hemos usado son: seleccionar, mover, rotar, borrar y pintar una pieza, hacer zoom, ajustar, rotar y rodar la cámara, abrir, guardar, etc.

[2]. Vista previa o escena. En esta zona es donde se va a realizar el modelado de la construcción de LEGO deseada. Se pueden añadir más ventanas de vista previa para ayudar en el modelado; en la Figura 16 se observa cómo se realiza esto, seleccionando “Split Horizontal” o “Split Vertical”.

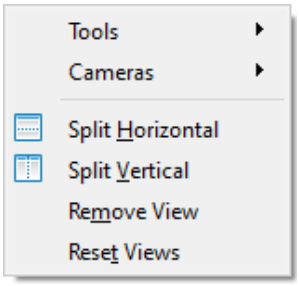

Figura 16. Menú contextual de la vista previa

[3]. Propiedades de la pieza. En esta sección se pueden modificar las propiedades de la pieza colocada en la vista previa, como posición, rotación, visibilidad y apariencia. Estas propiedades pueden cambiarse antes de colocar la pieza o con la pieza colocada en la vista previa.

[4]. Menú de piezas. Esta pestaña es la que nos permite seleccionar las distintas piezas que contiene el programa. Para ello, se selecciona la pieza deseada, se arrastra a la vista previa y se suelta. Este apartado puede subdividirse en:

[4.1]. Vista preliminar de la pieza. Aquí, puede verse la pieza seleccionada antes de ser puesta en la vista previa. También puede rotarse para verla con más detalle.

[4.2]. Listado de piezas. En este recuadro se encuentra el listado con las piezas de LEGO, de donde se seleccionan para añadirlas a la vista previa.

[4.3]. Búsqueda de piezas. En esta entrada de texto, es donde se pueden buscar piezas, ya sea por su nombre o por su código de librería en LDraw. Este último método de búsqueda nos ha servido para localizar todas las piezas del modelo del EV3 junto con

[4.4]. Gama de colores. En esta parte se selecciona el color que deseamos darle a la pieza, antes de soltarla en la vista previa.

[5]. Línea de tiempo. Esta pestaña está destinada a la realización de manuales de montaje con LEGO. Este apartado no ha sido usado para la realización del modelo.

Este programa posee, según su sitio web [7], más de 6000 piezas usando el estándar LDraw, incluyendo todas las necesarias para la construcción del robot. LDraw, es un estándar abierto para los programas destinados al diseño CAD de LEGO que permiten al usuario crear modelos y escenas. Se puede utilizar para crear instrucciones de montaje físico, para realizar animaciones y renderizar imágenes en 3D.

El Proceso de montaje se ha centrado en introducir el código de librería en LDraw de cada pieza en la búsqueda de piezas del programa, obteniendo dicho código mediante las bases de datos de piezas de los sitios web de Brickset [50] y Rebrickable [51], los cuales contienen el código de librería de cada pieza, facilitando su búsqueda. Teniendo la pieza en la vista previa, nos serviremos del eje de coordenadas que aparece en la Figura 17 para trasladarla o rotarla (según los ejes X, Y y Z) consiguiendo encajarla en lugar indicado en las instrucciones de montaje [52] creadas por el usuario Laurens en el sitio web de Robot Square [53].

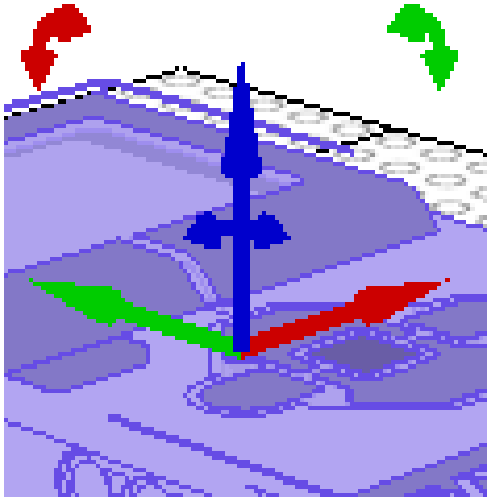

Figura 17. Eje de coordenadas para trasladar o rotar cada pieza

LeoCAD contiene una característica que ha sido crucial para su elección y es la opción de exportar el modelo a múltiples formatos de archivo muy comunes en el mundo del diseño gráfico, y que V-REP puede importar. Para realizar esto, en la barra de herramientas seleccionamos “File”, “Export” y, por último, en el caso de exportar a un modelo válido para V-REP, seleccionamos “Wavefront…”. Con esto, el modelo está guardado y listo para cargarlo en el simulador. En la Figura 18 se muestra el robot diseñado en este software. También en el directorio de los archivos del trabajo */Videos* se dispone de un vídeo con la construcción a cámara rápida del

modelo; este vídeo tiene el nombre *modelado.mp4*. Además, se dispone del modelo realizado en LeoCAD (“.ldr”) y el exportado a formato “.obj”, junto con el archivo “.mlt” que contiene los colores, en el directorio de los archivos de la memoria */Simulador/Modelos*.

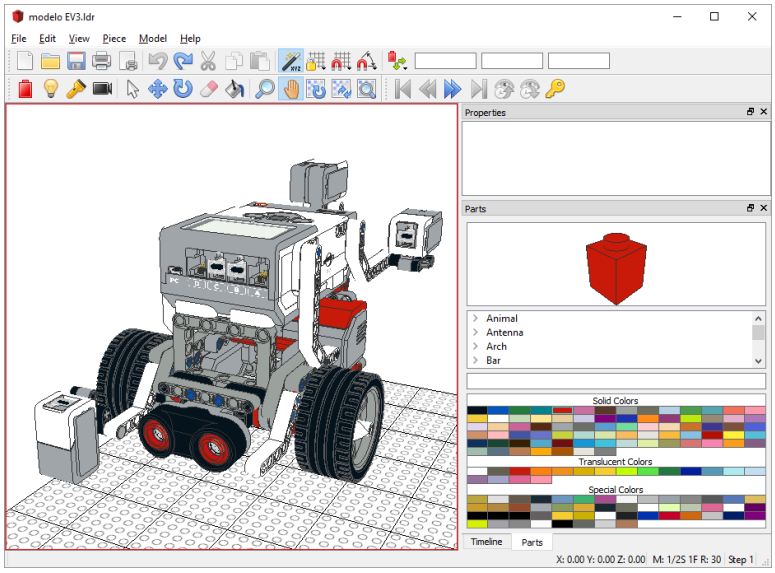

Figura 18. Modelo del robot en LeoCAD

CAPÍTULO 3.
MODELADO DEL ROBOT EN V-REP

En este capítulo, describiremos el proceso de diseño del robot en V-REP a partir del modelado en LeoCAD. Para ello se desarrollarán las siguientes secciones:

- **Mecánica**. Se comentará la estructura mecánica que tiene el robot diseñado y su repertorio de movimientos.

- **Adaptación del modelo al simulador**. Se explicará el proceso de la configuración inicial para el robot modelado en LeoCAD en el simulador sin entrar en el modelado de sensores y actuadores: importación, escalado, partes dinámicas, simulación de colisiones, etc.

- **Actuadores.** Descripción de los actuadores que componen el robot y su diseño en V-REP.

- **Sensores.** Esta sección tiene una estructura muy parecida al de actuadores, pero aplicada a los sensores, es decir, se describirán las especificaciones y el diseño de estos. Los sensores son los siguientes: sensor táctil, de ultrasonidos, de luz/color, giroscópico y de rotación del motor.

- **Interfaz gráfica.** Desarrollo de una interfaz gráfica para representar la pantalla, los botones y los LEDs de estado del robot.

- **Programación de funciones.** Descripción de todas las funciones desarrolladas en el simulador, así como sus fases.

3.1. Mecánica

El robot que se ha diseñado es un robot móvil con dos ruedas delanteras (tractoras) conectadas a su respectivo motor, y una trasera (loca) que en este caso es una bola (Figura 19). A este tipo de disposición se le conoce como configuración diferencial; en ella, los movimientos se realizan dando determinados sentidos de rotación a los motores. Como el robot no puede mantenerse por sí solo con las dos ruedas delanteras, se le añade la “rueda” trasera para mantener la horizontalidad sin que esté asociada a ningún motor.

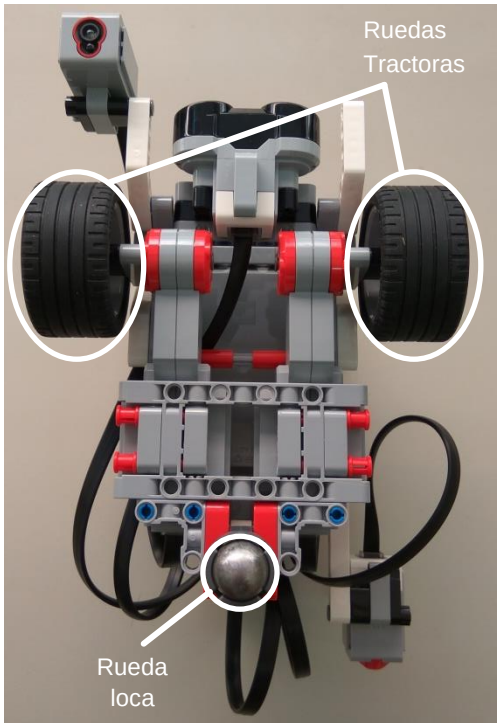

Figura 19. Configuración de las ruedas del modelo

Con la configuración ya definida, se puede diseñar el repertorio de movimientos que tendrá el robot. Para ello, definiremos los motores como habitualmente están conectados a los puertos del bloque del EV3, es decir, motor B (MB) al que le pertenece la rueda que está a la derecha en la Figura 19, y motor C (MC) a la que está a la izquierda. Cada motor dispone de tres estados, que corresponden con el sentido de giro de las ruedas; estos son, hacia adelante (↑), hacia atrás (↓) y parada (─). La siguiente tabla muestra el repertorio de movimientos del robot a partir de los 3 estados posibles de cada rueda:

MB	MC	MOVIMIENTO
↑	↑	Avanzar
↓	↓	Retroceder
↑	↓	Giro derecha sobre su eje
↓	↑	Giro Izquierda sobre su eje
↑	─	Giro derecha adelante
↓	─	Giro derecha atrás
─	↑	Giro izquierda adelante
─	↓	Giro izquierda atrás
─	─	Parada

Tabla 1. Repertorio de movimientos del robot diseñado

En la Figura 20 se presenta todo el repertorio de movimientos de manera gráfica, indicando tanto el movimiento final del robot como el sentido de giro de cada motor.

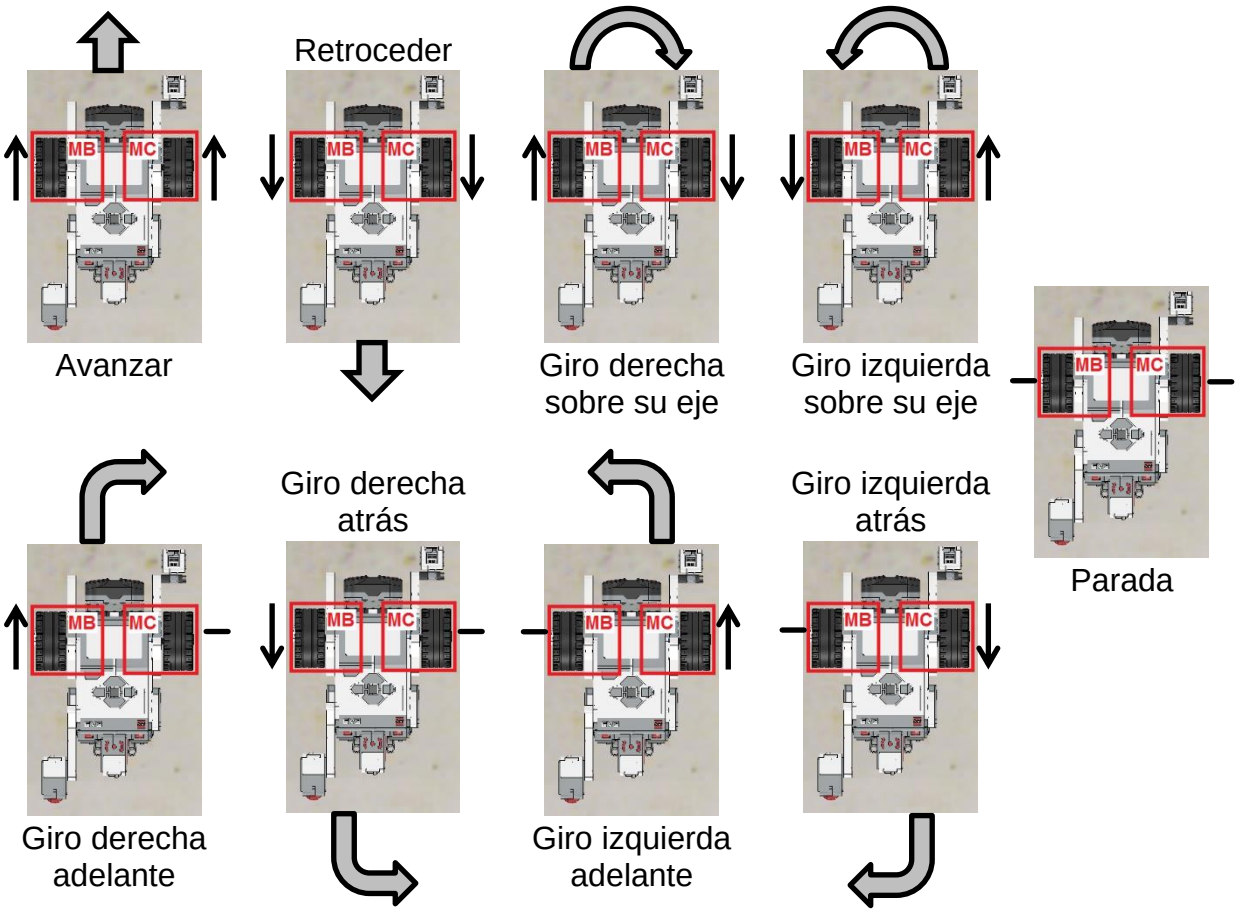

Figura 20. Repertorio de movimientos del robot

3.2. Adaptación del modelo de LeoCAD al simulador V-REP

Como se ha mencionado en el apartado 2.5, se puede exportar el modelo de LeoCAD a un formato que el simulador V-REP puede importar perfectamente. En nuestro caso hemos elegido el formato de importación ".obj", ya que puede importar el modelo con los colores originales del EV3.

El modelo de V-REP con todas sus características se ha guardado en un fichero ".ttm". Para que V-REP reconozca a nuestro robot como un modelo, se han seguido las consideraciones que aparecen en la documentación [54] y se ha guardado en el menú "File → Save model as…".

3.2.1. Importación y escalado

Antes de realizar la importación, el estándar LDraw utiliza una unidad de medida para la longitud en sus modelos, esta es el LDU ("LDraw Unit") [55]. Cada LDU

representa 0.04 cm; esto hay que tenerlo en cuenta a la hora de importar el modelo para aplicar un factor de escala que corrija las medidas.

Para poder importar el modelo en V-REP, nos vamos a “File”, luego a “Import” y, por último, “Mesh…”, donde aparecerá una ventana para cargar el modelo. Cuando se haya seleccionado, el modelo comenzará a cargarse y, cuando finalice la carga, nos aparecerá un cuadro de dialogo con las opciones de importación. En estas opciones se nos indica que seleccionemos una escala y la orientación del modelo, en nuestro caso, utilizaremos que una unidad representa un milímetro, que nos facilita aplicar el escalado del modelo, y que el vector Z representa la orientación hacia arriba.

A continuación, debemos aplicar el factor de escala para corregir las medidas. Para ello, como habíamos indicado a la hora de importar el robot que cada unidad de longitud del modelo exportado por LeoCAD (LDU) equivale a 1 mm y como 1 LDU son 0,4 mm, el factor de escala que hay que aplicar es de 0,4 para obtener las medidas reales. Para añadirlo, se seleccionan todas las formas que componen el modelo, abrimos las propiedades del objeto que se encuentran en la barra lateral izquierda del entorno de simulación, abrimos la pestaña “Common” (Figura 21) y, por último, pulsamos sobre el botón “Scaling”; ahí aplicamos el factor de escala (Figura 22). En la Figura 23 podemos ver la diferencia de tamaño del robot escalado y sin escalar.

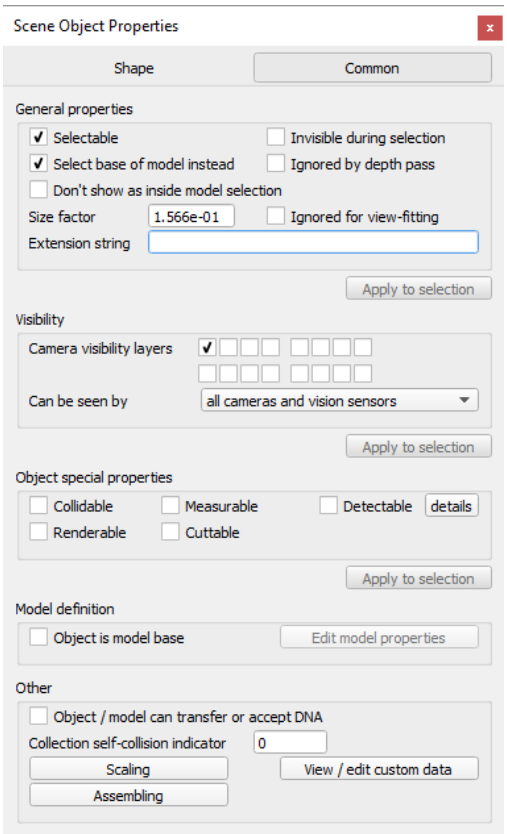

Figura 21. menú de propiedades del objeto (pestaña “Commons”)

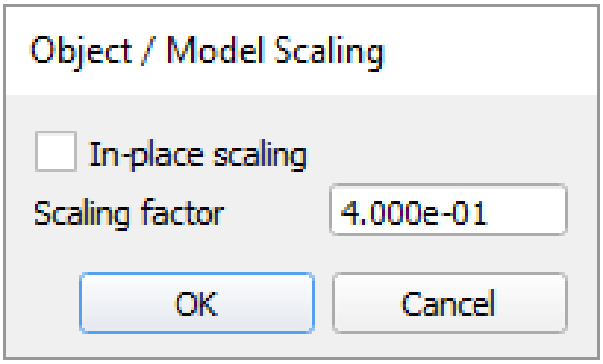

Figura 22. aplicación del factor del escala

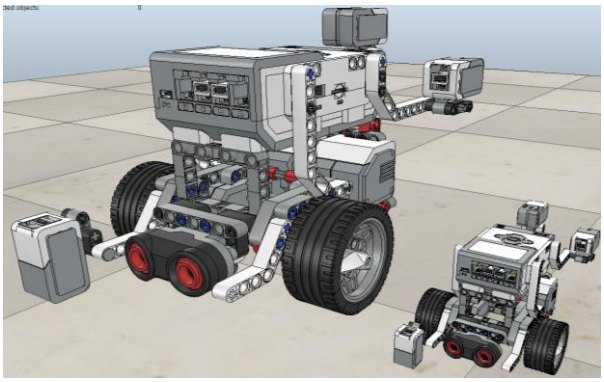

Figura 23. De izquierda a derecha: modelo sin escalar y escalado

Hecho esto, ya tenemos el robot escalado con las medidas reales. El siguiente paso es extraer formas puras del modelo, ya que, como se mencionó en la subsección 2.3.4, los modelos diseñados con programas externos a V-REP proporcionan más detalle, pero al simularlo con propiedades físicas, el motor físico de la simulación comenzará a ir lento o, peor aún, deje de funcionar. Para ello, se utilizarán las mencionadas formas puras que el simulador puede usar en su motor físico de manera eficiente. Estas formas han sido colocadas en las zonas donde el robot es más probable que choque con algún obstáculo, para darle una superficie a las ruedas, a la rueda trasera y para el sensor táctil.

3.2.2. Creación de formas puras

Para generar las formas puras en las diferentes piezas, se han utilizado los modos de edición de formas (“Shape edit mode”) y, dentro de éste, el de edición de triángulos (“Triangle edit mode”), donde se han ido seleccionando algunos triángulos de cada pieza y se han ido extrayendo formas puras, ya sean, cuboides, cilindros o esferas. Para más información de este modo en V-REP, consultar la documentación referenciada [56] [57].

Para el cuerpo del robot se han generado las formas puras presentadas en las Figuras 24 y 25; los cuerpos usados para ese fin han sido cuboides y algunos cilindros.

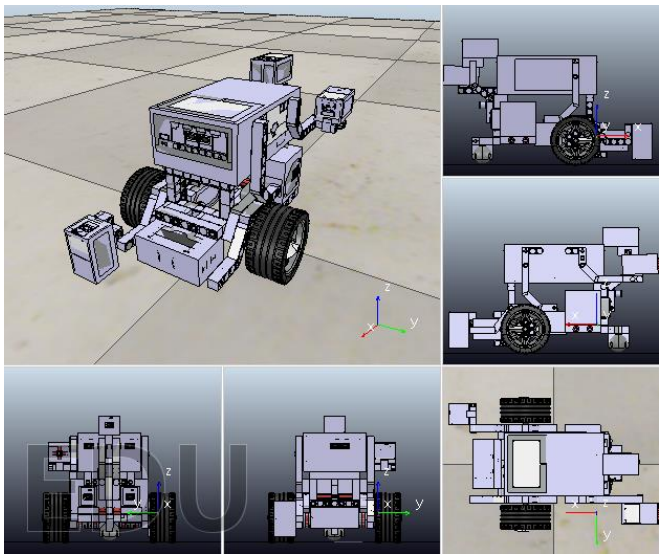

Figura 24. Formas puras mezcladas con el diseño del robot

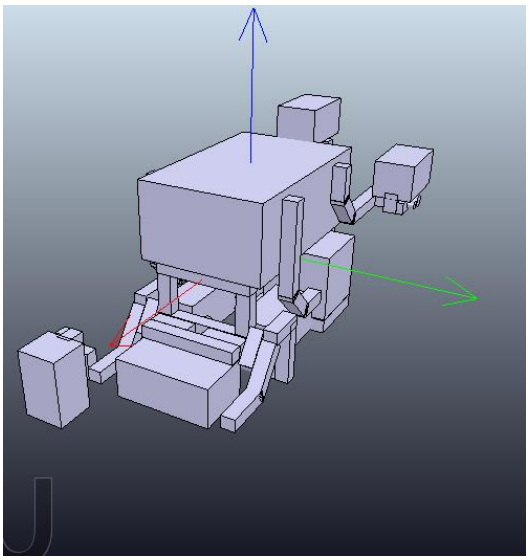

Figura 25. Formas puras del cuerpo del robot

Para las formas puras de las ruedas (Figura 26), se han utilizado, para cada una, un cilindro y un cuboide. El cilindro es para el contorno de la rueda, y el cuboide, para la pestaña de fijación de la rueda, ya que esta sobresale un poco. En cuanto a la rueda trasera, se ha utilizado una esfera con el mismo diámetro.

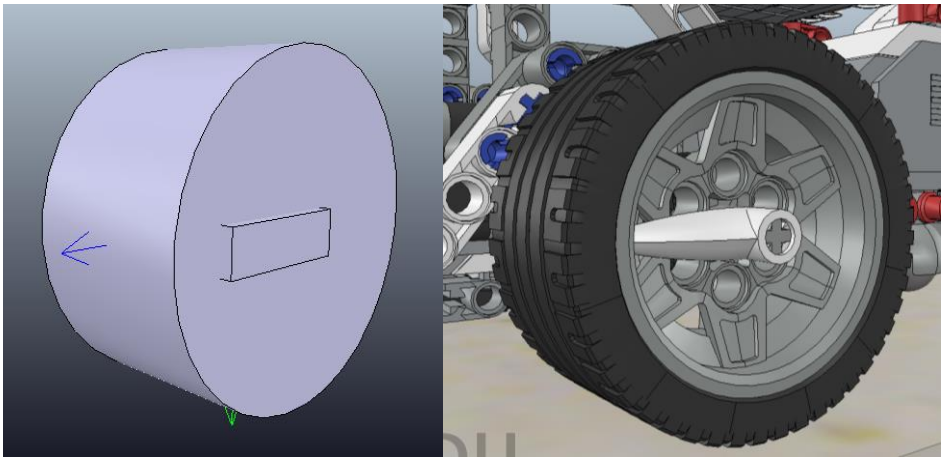

Figura 26. Formas puras de las ruedas y rueda del modelo

Por último, para el pulsador del sensor táctil, también se han utilizado cuboides y cilindros imitando la forma de este (Figura 27).

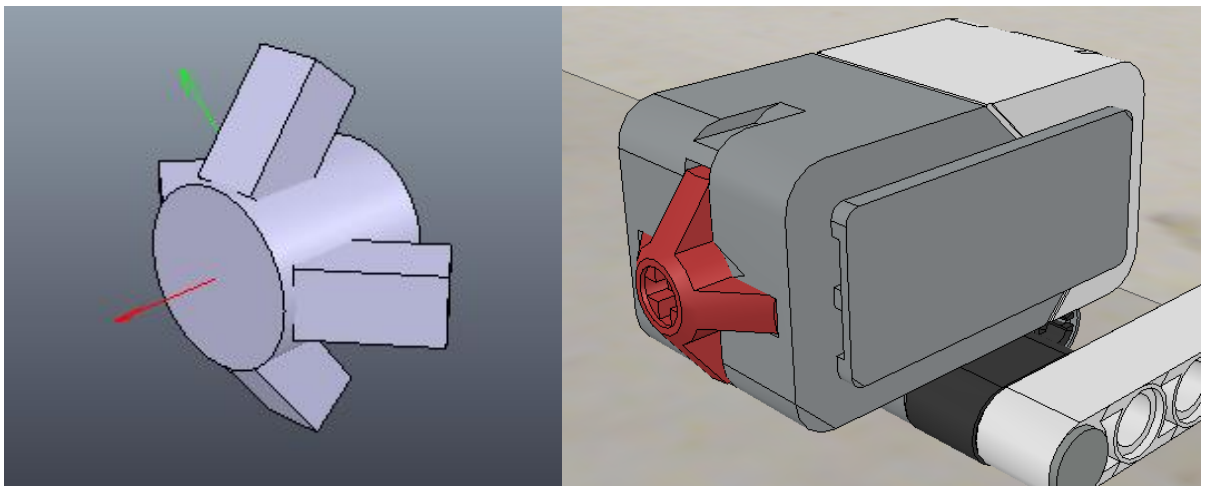

Figura 27. Formas puras del pulsador y modelo del sensor táctil

Las formas puras simples diseñadas y las del modelo de LeoCAD, se unirán para crear formas compuestas. Para ello, vamos a diferenciar cuatro grupos de formas para el modelo de LeoCAD (formas aleatorias simples) y cuatro grupos para para las formas diseñadas anteriormente; para ambas, estas son el cuerpo del robot, la rueda izquierda, la rueda derecha y el sensor táctil. Para agrupar todas las piezas, hay dos maneras de hacerlo, cada una con sus ventajas e inconvenientes:

- **Mediante mezcla (“merge”)**. Con este método, conseguimos unir todas las piezas perfectamente, pero se convierte en una forma simple, por lo que perdería todos los colores de las piezas y se quedaría de un solo color. Por otra parte, la simulación se ejecutaría de una manera más eficiente ya que, al ser una forma simple, solo tiene que renderizar una forma, por ello, la mezcla de formas es adecuada para ordenadores con un hardware poco avanzado o muy antiguos.

- **Mediante agrupación (“group”)**. La agrupación persigue el mismo fin que la mezcla y agrupa todos los modelos en un conjunto, por lo que se consigue que este conserve los colores de cada parte del robot. Sin embargo, no se logra mayor rendimiento que con la mezcla, ya que el simulador tiene que renderizar todas y cada una de las piezas que componen la agrupación.

Para unir las formas simples en nuestro modelo del simulador siempre utilizaremos la agrupación, ya que si usamos la mezcla estas formas pasarán a ser tratadas como una forma aleatoria simple y perderían las optimizaciones para simular las propiedades físicas. Por otro lado, para el modelo importado de LeoCAD podemos utilizar los dos métodos de unión de formas; por lo que, en este trabajo, se van a proporcionar 2 modelos del robot para V-REP: uno utilizando la unión por mezcla en las piezas del modelo importado y otro usando la unión por agrupación (Figura 28).

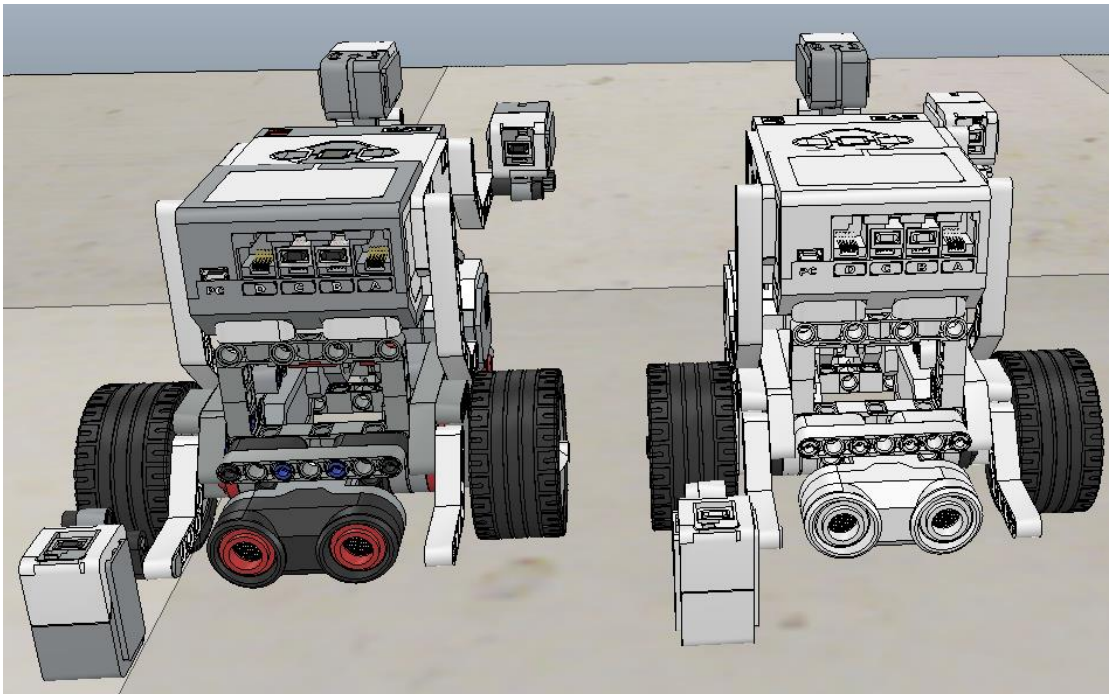

Figura 28. Modelo con agrupación de formas y modelo con mezcla de formas

Las Figuras 29, 30 y 31 muestran todas y cada una de las partes que se han utilizado para la unión por agrupación o mezcla de formas del modelo. No aparecen las uniones del pulsador, ni la de la bola, dado que estas piezas son una sola y no ha hecho falta mezclar o agrupar.

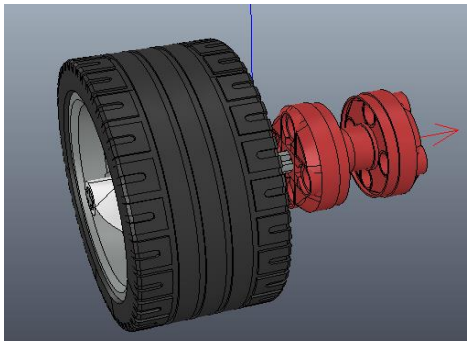

Figura 29. Agrupación de la rueda izquierda

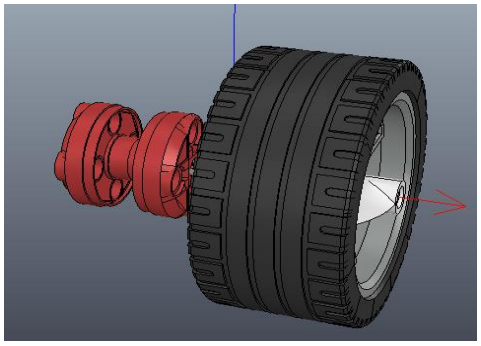

Figura 30. Agrupación de la rueda derecha

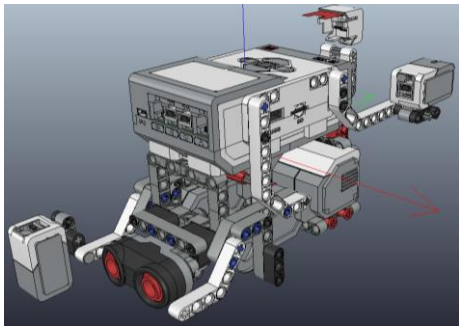

Figura 31. Agrupación del cuerpo del robot

3.2.3. Propiedades dinámicas de las formas puras

A continuación, se agregan las propiedades físicas al modelo, concretamente a las formas puras, para que el robot pueda moverse y chocar. Para ello, abrimos las propiedades de alguna forma pura ya agrupada y vamos a sus propiedades dinámicas (Figura 32). Aquí, indicamos que el cuerpo es dinámico y que puede simular colisiones. En la configuración de la simulación de colisiones, el material con el que están compuestos las ruedas y el cuerpo del robot, será el material por defecto.

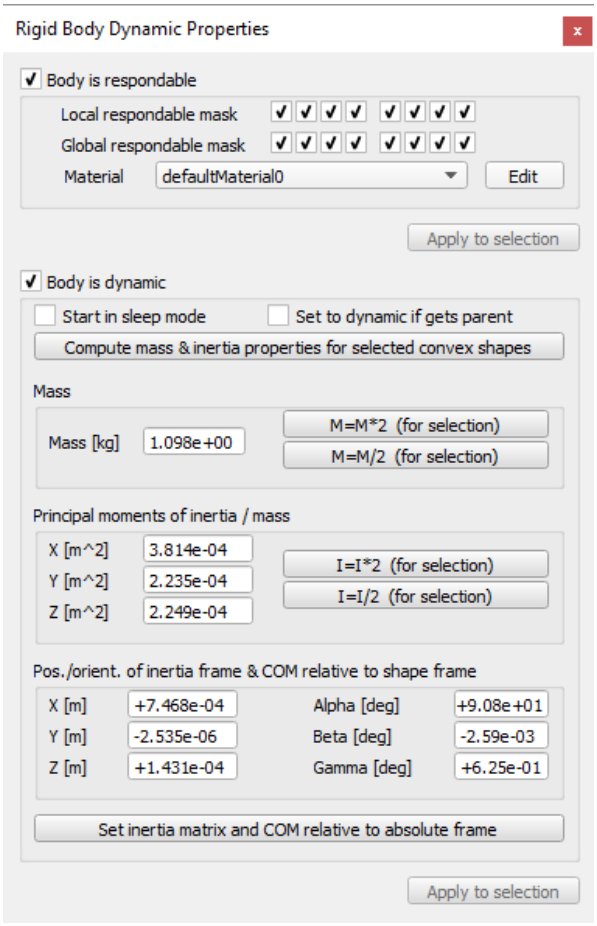

Figura 32. Propiedades dinámicas de una forma

Otro aspecto a tener en cuenta son las máscaras de simulación de colisiones, que indican cuándo se puede generar una colisión con otro cuerpo que también pueda simular colisiones. La máscara está compuesta por dos valores de 8 bits (2 bytes): global y local. Si dos cuerpos colisionan y comparten mismo padre (en uno o más niveles) en la jerarquía de la escena, se usará la máscara local; en otro caso, la global. Estas máscaras son definidas por el usuario mediante el marcado de casillas en las propiedades dinámicas del objeto, donde cada casilla marcada representa un bit activo (1).

Cuando dos cuerpos chocan, se realiza una operación “AND” bit a bit con su máscara (global o local) y, si el resultado de la máscara es distinto de 0, se genera una colisión; en otro caso, los cuerpos se atravesarán. En la Figura 33 se explica el funcionamiento de estas máscaras mediante 2 casos.

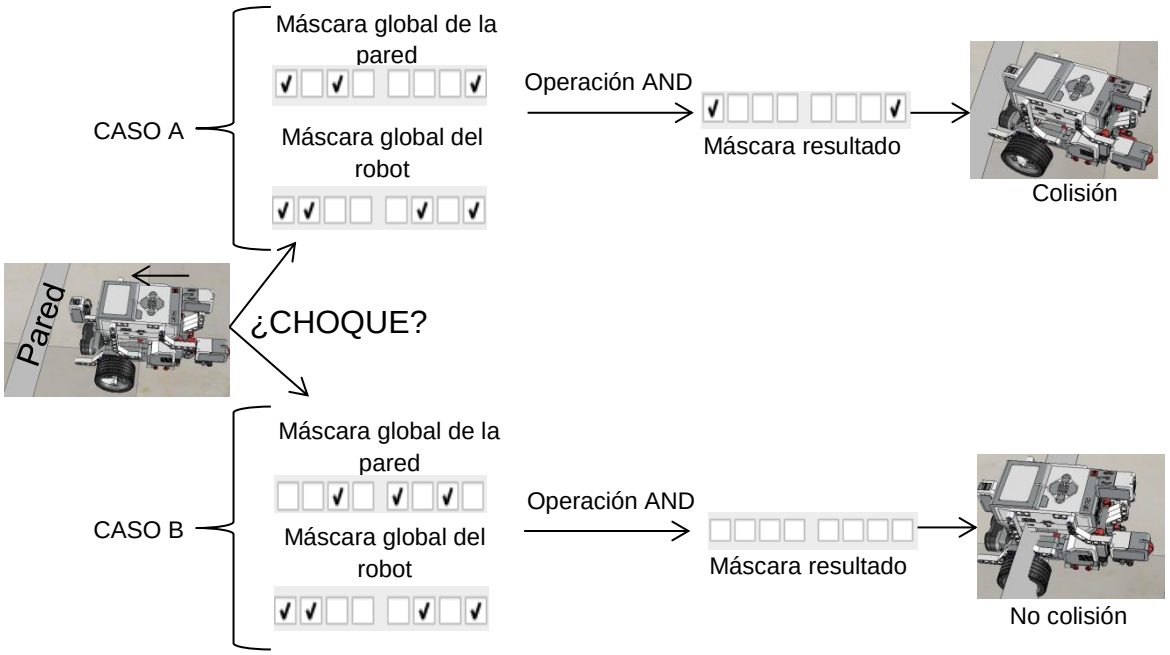

Figura 33. Funcionamiento de las máscaras de colisiones

En el caso de nuestro diseño, en la Figura 34 se puede ver cómo hemos dispuesto las máscaras de colisiones para las formas puras. Para las máscaras globales, se han marcado todas las cajas debido a que el robot tiene que chocar con los objetos de la escena; en cuanto a las máscaras locales, se han desactivado algunas cajas de la bola y del cuerpo del robot, ya que éstas tienen formas puras que chocan entre sí (Figura 35).

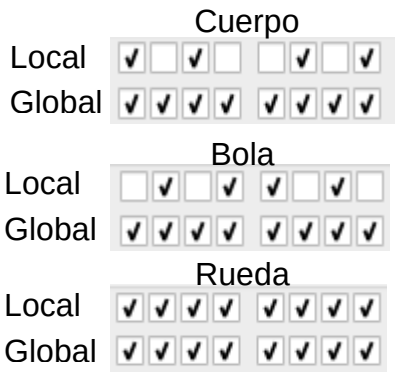

Figura 34. Máscaras de colisión para las formas puras

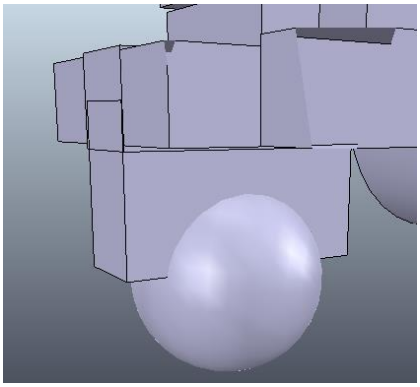

Figura 35. Estado de las formas puras del cuerpo del robot y la bola

3.2.4. Disposición en la jerarquía y visualización en la escena

Es conveniente que las formas puras no se visualicen en la escena, si no el robot aparecerá como en la Figura 36, esto es debido a que las formas puras por defecto aparecen en la escena. Para ocultar estas formas, se cambia la capa de visibilidad de la pieza para las cámaras de la escena; para ello, abrimos las propiedades de la

pieza y seleccionamos la pestaña “Common”, desmarcamos la caja que viene seleccionada por defecto y marcamos la caja que está justamente debajo, en la zona de opciones de visibilidad (Figura 37). Hecho esto, podremos ver el modelo como en la Figura 38.

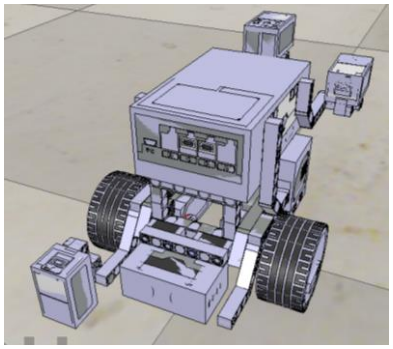

Figura 36. Formas sin ocultar a la cámara de la escena

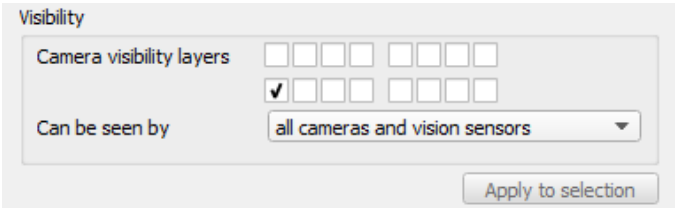

Figura 37. Opciones de visibilidad (pestaña “Commons” en las propiedades del objeto)

Figura 38. Formas ocultadas a la cámara de la escena

Cuando queramos ver las formas puras para, por ejemplo, cambiar su morfología o situación de manera visual, podemos usar las herramientas de capas de cámara, situado en la barra lateral izquierda del programa; ahí, marcaremos las cajas de las capas de cámara que queramos ver.

Hecho esto, comenzamos a colocar las piezas (agrupadas o mezcladas) en una estructura de árbol, para que el robot esté modelado en el simulador, aunque todavía falte por modelar sensores y actuadores. Si no hacemos esto, cada vez que comencemos la simulación, el robot se desmontará. Nos encontramos en el estado en el que las piezas tienen la disposición de la Figura 39; como necesitamos que sus partes estén “soldadas”, tomaremos como padre la forma pura compuesta del cuerpo del robot. Este proceso se realiza pinchando la pieza que se desea colocar como hija, y soltando encima de la pieza que deseamos que sea padre. De esta manera, se llega a la disposición de la Figura 40.

Figura 39. Partes del robot sin jerarquizar

Figura 40. Partes del robot jerarquizadas

Aun así, si ponemos en marcha la simulación, las partes se siguen desmontando, ya que necesitamos los sensores y/o actuadores para poder unirlas correctamente.

3.3. Actuadores

Como hemos mencionado en la sección 3.1, este modelo tiene una configuración de ruedas diferencial y, para ello, se necesitan dos motores de EV3 (Figura 41); la solución que nos sugiere V-REP en su simulación es usar articulaciones rotacionales y añadírselas a la jerarquía de las ruedas. Se han tenido que consultar sus especificaciones reales [58], consiguiendo que estos se parezcan lo máximo posible a su homólogo real. Estas son:

- Velocidad máxima entre 160 rpm y 170 rpm. Estos valores han sido utilizados para la parte de la programación de la toolbox, concretamente, las respectivas funciones que hacen que los motores se muevan.

- Torque de rotación de 20 Ncm y de 40 Ncm en parada. Estas dos medidas nos servirán para definir los torques máximos en el modelo e iremos cambiándolos dependiendo de si el motor está moviéndose o está parado.

- Resolución de 1º en el sensor de rotación.

Figura 41. Motor de LEGO EV3 (imagen tomada de la guía de uso del EV3)

En primer lugar, en lo relativo a la configuración de los motores, tenemos que añadir una articulación rotacional y definir sus especificaciones; para ello, con la articulación rotacional ya creada, nos iremos a las propiedades del objeto (Figura 42).

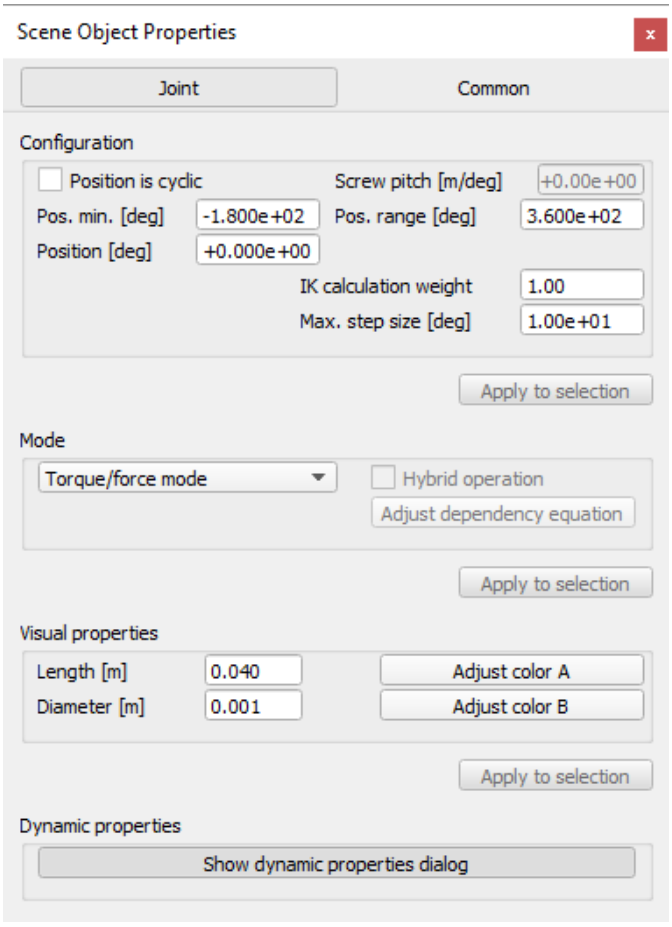

Figura 42. Propiedades de una articulación rotacional

En estas propiedades, para emular el sensor rotacional del motor con la API de V-REP, desmarcamos la caja “Position is cyclic” y nos aseguramos de que la posición mínima (“Pos. min. [deg]”) es -180º, el rango de posiciones (“Pos. range [deg]”) es 360º y la posición (“Position [deg]”) es 0º. Como usaremos el modo de operación “Torque/force mode”, según la documentación, los campos de “IK

calculation weight" y "Max. step size [det]" se dejaran los valores por defecto. También se pueden cambiar los valores visuales de la articulación, como: colores, longitud y diámetro; estos valores son opcionales, pero se han modificado en este caso para que las articulaciones parezcan de mayor tamaño.

A continuación, en las propiedades dinámicas (Figura 43), marcamos el cuadro "Motor enabled" para activar el motor y que responda a los comandos que se le administren vía programación; después, se rellenan los campos de velocidad a 0 y el torque máximo con el valor real del torque en parada del robot LEGO, ya que siempre cuando iniciemos la simulación comenzaremos con el robot en reposo.

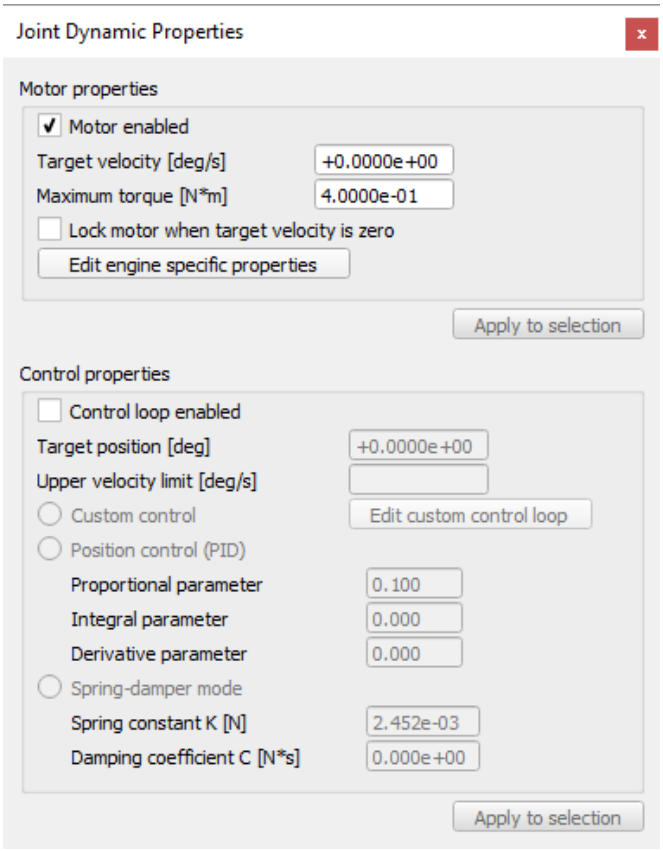

Figura 43. Propiedades dinámicas de una articulación rotacional

Con esto, ya tenemos las propiedades del motor añadidas, y solo faltaría colocar correctamente las articulaciones a cada eje de cada una de las ruedas. Para hacer esto, primero, tenemos que duplicar la articulación ya creada para tener así dos iguales; para conseguir una articulación idéntica, basta con copiar (Ctrl+c) y pegar (Ctrl+v), después, se renombrarán con los nombres que se le han asignado en el apartado 3.1, es decir, "Motor_A" y "Motor_B"

Ahora se colocan las articulaciones en los correspondientes ejes de las ruedas; para realizar esta tarea, situaremos las articulaciones en la misma posición que las formas puras de cada rueda, ya que el centro de dichas formas coincide con el eje

de los motores. Para posicionar cada articulación, primero, seleccionamos el motor y la forma pura de la rueda que le corresponde en ese orden. Después, nos vamos a al menú de posición de objeto (Figura 44); dentro de este modo, en el apartado "Object / item position", pulsamos el botón "Apply to selection", el resultado será el mismo que aparece en la Figura 45.

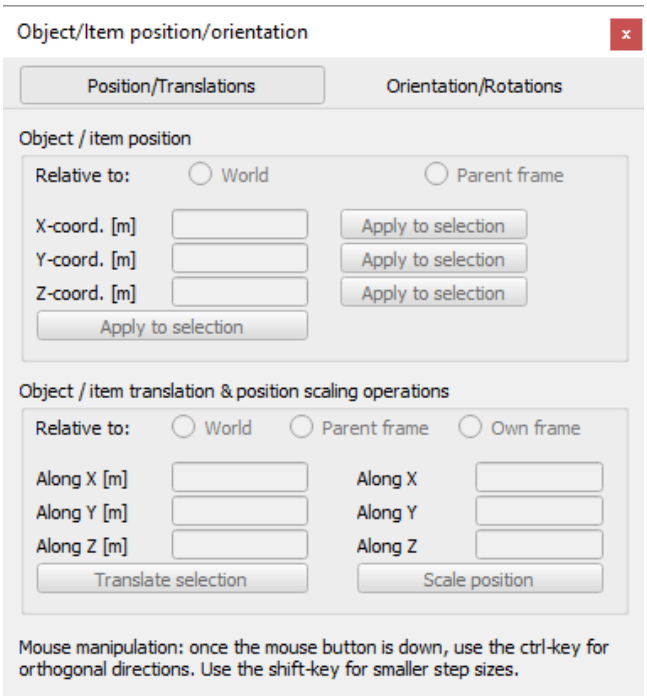

Figura 44. Menú de posición del objeto

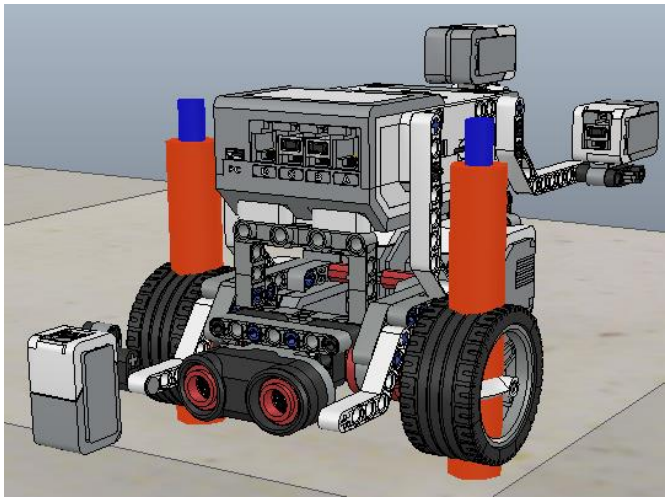

Figura 45. Articulaciones colocadas en la misma posición que la forma pura de su respectiva rueda

Finalmente, giraremos -90º la articulación con respecto al eje Y de la escena para orientarla con el eje de las ruedas (Figura 46). Para realizar esta tarea, iremos a la pestaña "Orientation/rotations" en el mismo menú de posición del objeto (Figura 47) y añadimos -90º al ángulo Beta en el aparatado "Object / item orientation". Las articulaciones se orientan de esta manera porque las articulaciones rotacionales en

V-REP giran en sentido antihorario indicándole velocidad positiva, por lo que al hacer esto conseguimos que al proporcionar una velocidad positiva a la articulación se logra que el robot avance hacia adelante y no hacia atrás.

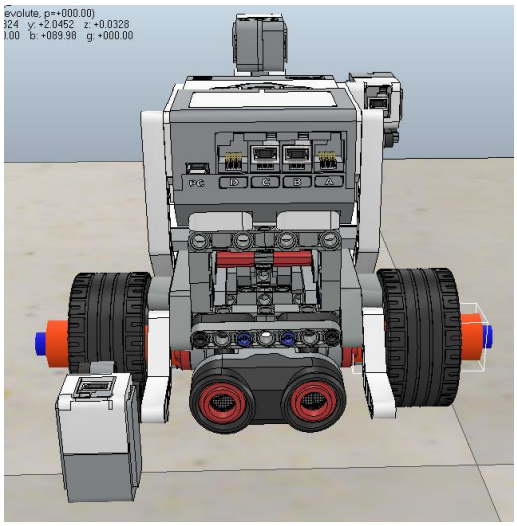

Figura 46. Articulaciones orientadas respecto al eje de las ruedas

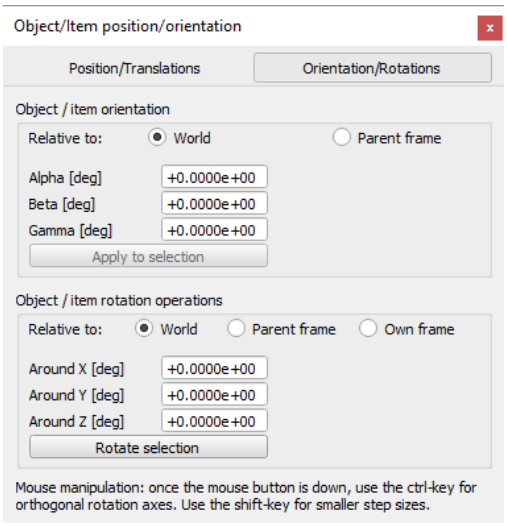

Figura 47. Menú de orientación del objeto

Para finalizar la colocación de las articulaciones, las añadimos al árbol de la jerarquía de la escena como en la Figura 48, para que queden unidas al cuerpo del robot junto con las ruedas y, además, haremos los mismo que se ha hecho con las formas puras, es decir, cambiar las capas de cámara para que no sean visibles en la escena.

Figura 48. Colocación de las articulaciones

3.4. Sensores

En el robot modelado contamos con 5 tipos de sensores: táctil, giroscópico, de luz/color, de ultrasonidos y rotacionales en los motores. Todos los sensores tienen una frecuencia de muestreo de 1 kHz, por tanto pueden simularse estableciendo un paso de simulación de 1 ms; esto produce que la simulación sea lenta debido a la

cantidad de veces que V-REP tiene que ejecutar el script principal antes de renderizar el comportamiento de la escena.

Para que la simulación sea rápida, el simulador recomienda un paso de simulación de 50 ms, o sea, es como si tuviésemos los sensores funcionando a una frecuencia de muestreo de 20 Hz. La solución que se propone es que si, por ejemplo, se programa un algoritmo para el robot que requiera valores más actualizados de los sensores, se utilice un paso de simulación más pequeño, haciendo que la simulación sea más lenta pero lo más fiel a los sensores reales. En cambio, si los valores de los sensores no suponen una prioridad, se puede utilizar un paso de simulación más alto, logrando que la simulación se realice más rápidamente pero perdiendo a cambio frecuencia de muestreo en los sensores. A continuación, se describirá el desarrollo y el diseño de los mencionados sensores.

3.4.1. Sensor táctil

El sensor táctil de LEGO (Figura 49) es un sensor que detecta si el botón está presionado o no. Este sensor no contiene especificaciones especiales ya que es un simple pulsador, lo único que ha tenido que utilizarse para modelarlo es la forma pura descrita para el pulsador (subsecciones 3.2.2 y 3.2.3) y un sensor de fuerza.

Figura 49. Sensor táctil de LEGO

Para el sensor de fuerza, la configuración que tenemos se muestra en la Figura 50. Se ha escogido un valor medio de 5 muestras para que tengamos algo de precisión al recoger el dato de la fuerza ejercida en el sensor y no suponga un coste computacional alto, también, lo redimensionaremos para que sea más pequeño, esto es debido a que, a la hora de posicionar el sensor, nos sea visualmente fácil de situarlo.

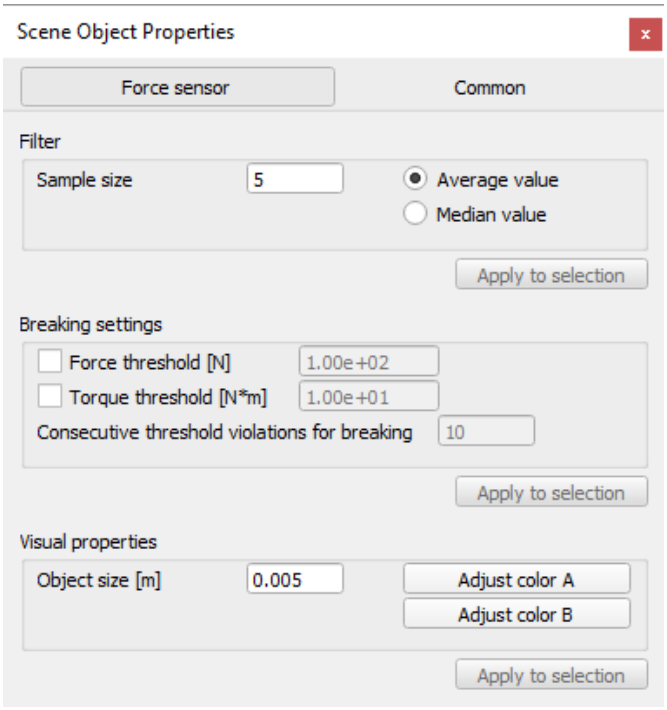

Figura 50. Propiedades del sensor de fuerza para el sensor táctil

En cuanto a la posición del sensor en el modelo del robot, se ha colocado en la parte interna del sensor táctil más próxima al pulsador y girado con la misma orientación que este (Figura 51); esta posición nos ayudará a la hora de programar el sensor extrayendo la fuerza proveniente del eje Z de este. En la jerarquía de la escena, el sensor se ha colocado según se indica en la Figura 52.

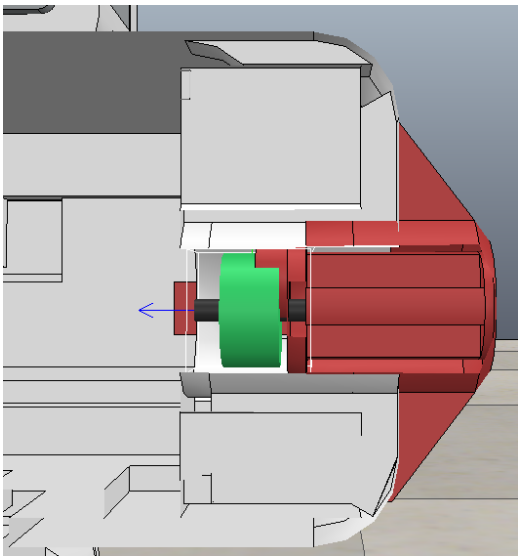

Figura 51. Posición del sensor de fuerza en el modelo en la escena

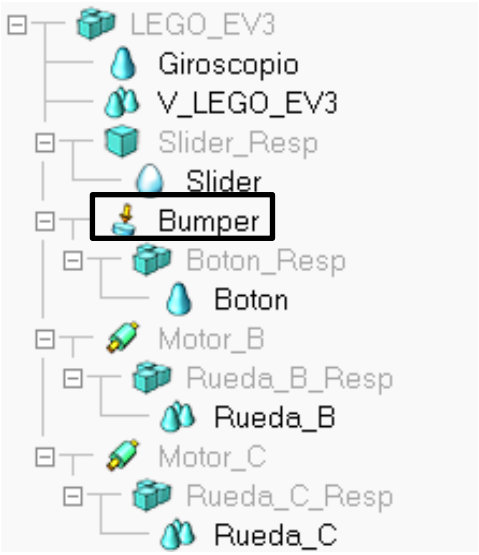

Figura 52: Lugar del sensor de fuerza para el sensor táctil en la jerarquía

3.4.2. Sensor ultrasónico

Este sensor (Figura 53), puede medir la distancia que existe entre el sensor y un objeto. Esto se realiza emitiendo un sonido de alta frecuencia y midiendo cuánto tarda la onda en volver al receptor del sensor. La distancia calculada se puede medir tanto en centímetros como en pulgadas; en nuestro modelo solo usaremos centímetros. La distancia mínima que puede llegar a medir el sensor es 3 cm y la máxima es 250 cm, con una precisión de 1 cm. Si el sensor no es capaz de medir, debido a que el objeto está muy cerca, muy lejos o por cualquier otra causa, el sensor devolverá el valor de 255. También, es capaz de detectar otros sensores ultrasónicos en su modo de presencia, pero este modo no se ha modelado debido a que queda fuera de los objetivos de este trabajo ya que no se va a simular más de un robot EV3.

Figura 53. Sensor ultrasónico de LEGO

Para modelar el sensor de ultrasonidos, usaremos un sensor de proximidad que proporciona V-REP. Su configuración es la que aparece en las Figuras 54, 55 y 56, tomando como referencia la página de Afrel [59]. Se ha optado por un modelo de sensor de cono, ya que las ondas sonoras del sensor se propagan de esta forma [60].

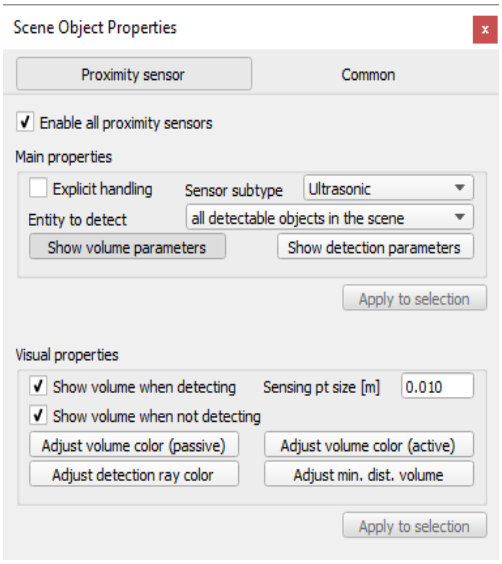

Figura 54. Propiedades del sensor de proximidad

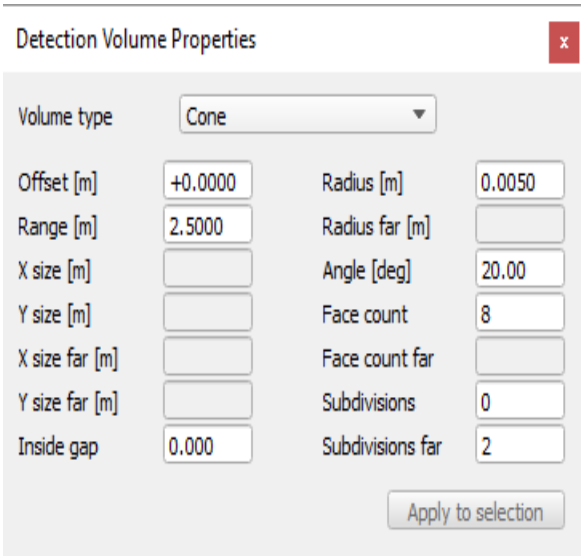

Figura 55. Propiedades del volumen de detección

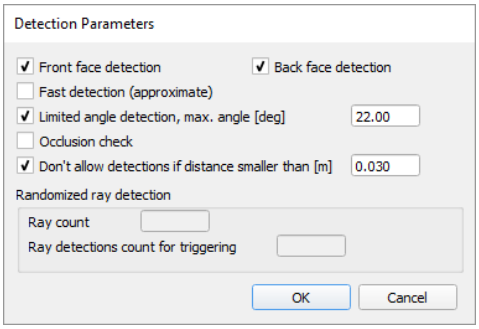

Figura 56. Parámetros de detección del sensor

En cuanto a los parámetros de detección, como el sensor de proximidad que se ha modelado es un sensor ultrasónico, se ha añadido un ángulo máximo de detección para los obstáculos, esto significa que el ángulo que se genera entre el vector del rayo de detección y el vector normal del objeto a detectar no debe de ser mayor que el ángulo que hemos especificado (Figura 57). Esto permite simular el sensor de ultrasonidos real cuando tenemos un objeto que se encuentra en una posición demasiado oblicua frente al sensor provocando que el receptor no encuentre la señal enviada por el emisor.

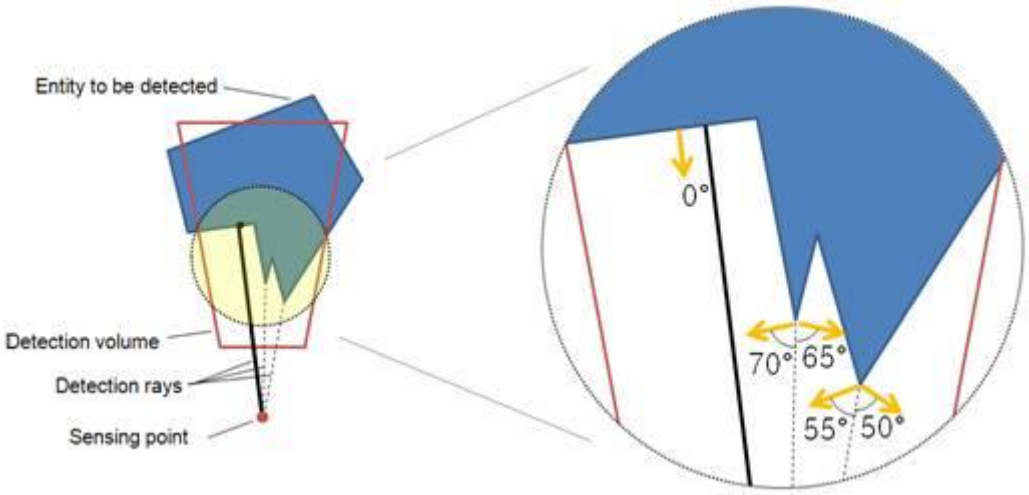

Figura 57. Principio del ángulo máximo de detección

Hecho esto, colocaremos el sensor de proximidad en mitad del sensor ultrasónico del modelo, ya que es el punto medio entre el emisor y el receptor del sensor (Figura 58). Finalmente, añadimos el sensor a la jerarquía de la escena como se muestra en la Figura 59.

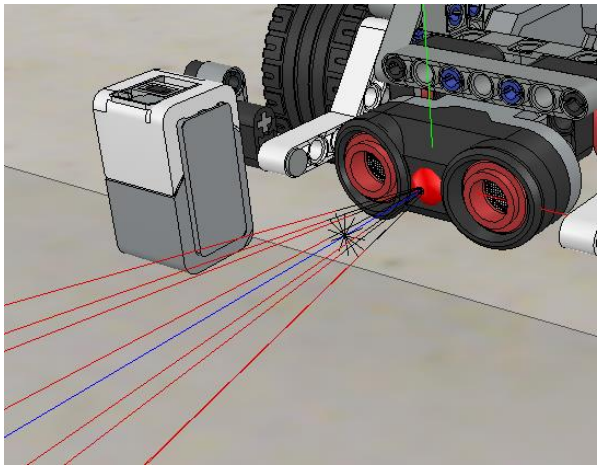

Figura 58. Posición del sensor de proximidad

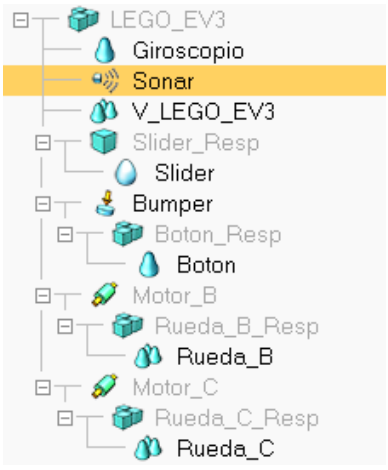

Figura 59. Posición del sensor de proximidad en la jerarquía

3.4.3. Sensor de luz/color

El sensor de color de LEGO EV3 (Figura 60) es un sensor que puede detectar colores e intensidad de luz. Dispone de 3 modos de operación:

Figura 60. Sensor de color de LEGO

- **Modo color**. El sensor puede reconocer hasta siete colores en este modo: negro, azul, verde, amarillo, rojo, blanco y marrón (Figura 61); además, admite una variante sin color, es decir, que no se ha detectado o no se ha sabido determinar qué color es.

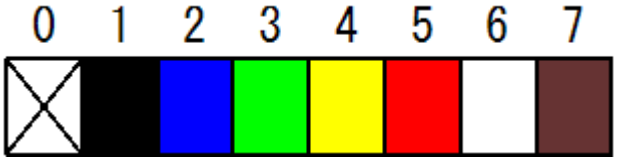

Figura 61. Gama de colores que detecta el sensor de color de EV3, donde "0" es el valor "sin color"

- **Modo luz reflejada**. Este modo consiste en medir la intensidad de luz que se refleja desde una luz roja que emite el sensor. En este modo, el sensor utiliza una escala de 0 a 100, donde el valor más bajo es el más oscuro y el más alto

es el más luminoso. Normalmente, estos valores máximos no se alcanzan, por lo que a la hora de programarlos en el script *Funciones* habría que generar una escala con menor rango, en este caso, de 7 a 71, que es lo que aproximadamente oscila el sensor entre los valores de negro y blanco, dependiendo de factores externos como, por ejemplo, la luminosidad de la sala en la que se encuentra el sensor. En ciertos casos, como al tapar el sensor con la mano o poner una lámpara directamente sobre este, se consiguen valores superiores o inferiores del rango definido, pero como en el simulador no vamos a poder usar una lámpara, ni vamos a poder poner la mano, se pueden prescindir de estos valores.

El rango de valores ha sido establecido mediante un experimento con el sensor de luz real en el laboratorio 2.1.5 de la ETS Ingeniería informática, obteniendo 6 muestras de 30 valores cada una tanto para el valor negro como el blanco mediante la toolbox de EV3 para Matlab. Con estos datos se ha realizado la media aritmética obteniendo el valor medio de ambos colores; en cuanto a las desviaciones típicas de cada color se han obtenido calculando las desviaciones de cada muestra y luego se ha obtenido la media aritmética de la desviación de las 6 muestras en cada caso (estos datos se usarán para la generación del error en el sensor simulado). En la Figura 62 se muestra el lugar y la zona aproximada de donde se han obtenido las muestras. En los ficheros de Matlab *muestras_luz_blanco.mat* y *muestras_luz_negro.mat* se encuentran los datos obtenidos para cada experimento y *muestras_luz.m* con el algoritmo que ha recogido las muestras, esto se encuentra en el directorio de los archivos del trabajo */Programas/Experimiento del sensor de luz*.

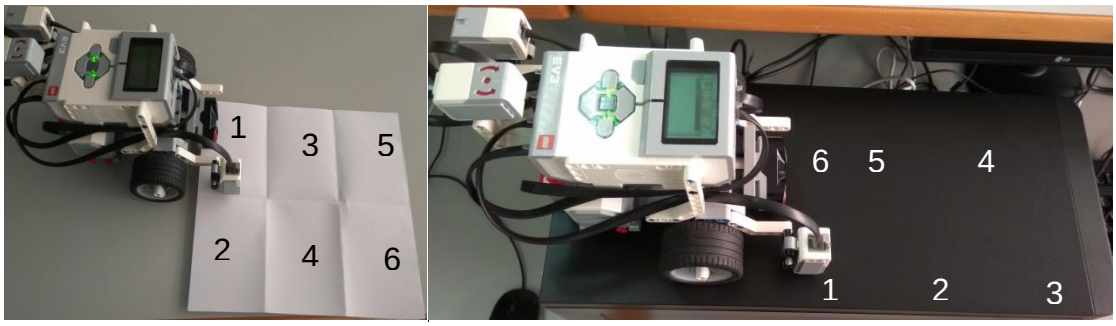

Figura 62. Lugares de obtención de las muestras de negro y blanco

- **Modo luz ambiental**. Es similar al modo de luz reflejada, pero en este modo mide la intensidad de la luz que entra por la cara del sensor a través del entorno, ya sea procedente de una lámpara o la luz del sol. El sensor utiliza la misma escala que el modo de luz reflejada. Este modo no ha sido desarrollado ni en el simulador ni en la toolbox, debido a que en la configuración física del robot, el sensor apunta hacia el suelo, por lo que no podría recibir correctamente la luz del ambiente.

En el simulador hemos utilizado dos sensores de visión; uno ha sido modelado para el modo color y otro para el modo luz reflejada, ya que estos modos tienen diferentes rangos de visión. Las especificaciones de los sensores de luz y color están en las Figuras 63 y 64; también han sido tomadas de la página de Afrel.

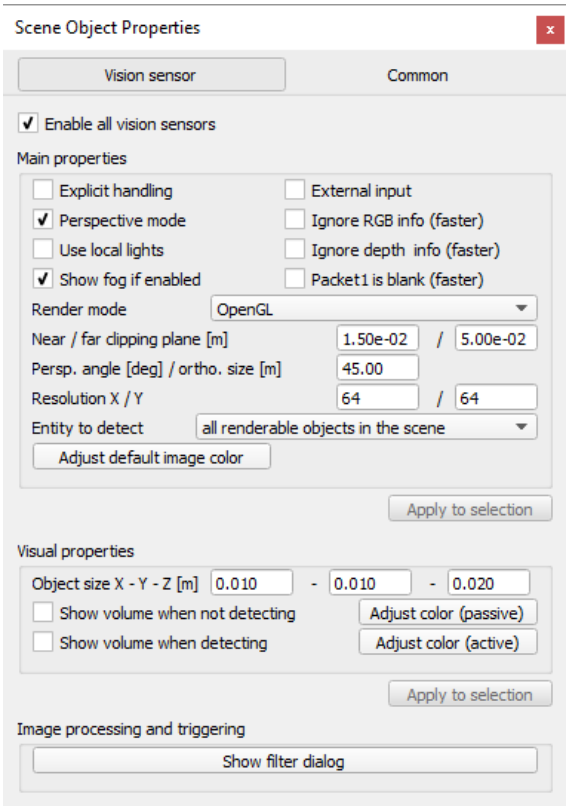

Figura 63. Especificaciones del sensor de visión para el modo luz reflejada

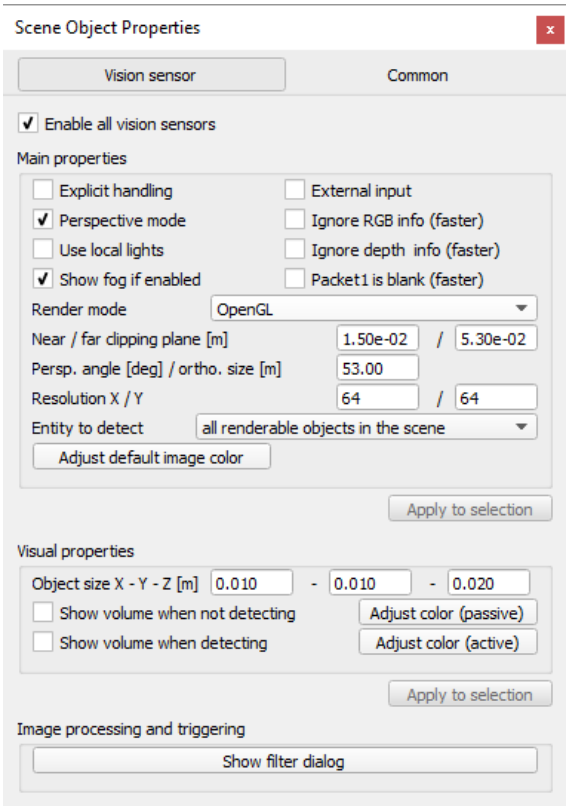

Figura 64. Especificaciones del sensor de visión para el modo color

A estos dos sensores de visión se les ha proporcionado una resolución de 64x64 debido a que es una resolución baja y puede dar valores medios de los sensores sin mucho error.

También, según el usuario “Dr. E´s Lab” en YouTube [61], el sensor de color de EV3 posee una superficie de detección cónica en la que la parte visible es el área de la base del mencionado cono. Para modelar esto en el sensor de visión, abrimos el menú de filtros para el procesamiento de imágenes (Figuras 63 o 64 en el botón inferior “Show filter dialog”). Ahí, añadiremos un recorte circular del área de visión del sensor (Figura 65), que hará que el sensor solo recoja las imágenes que aparezcan en el área circular (Figura 66).

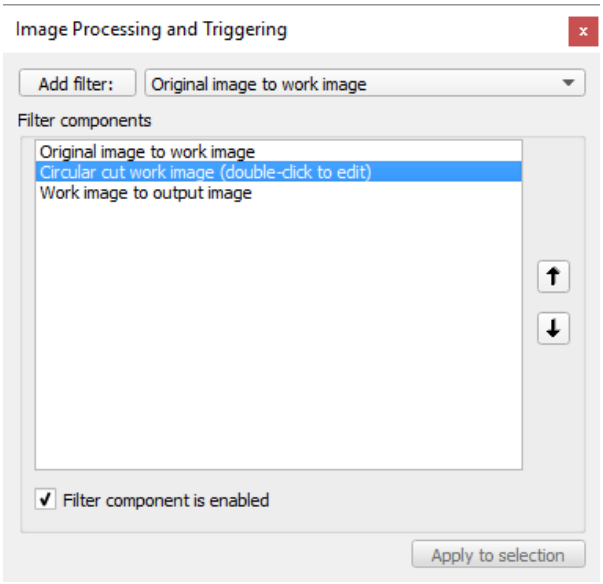

Figura 65. Filtro de recorte circular añadido

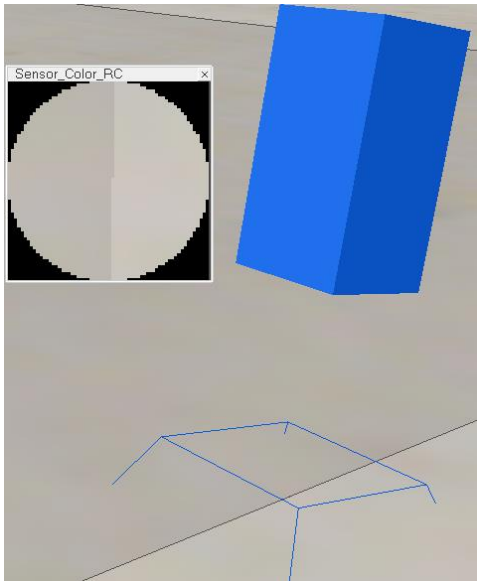

Figura 66. Imagen del área detectada por el sensor

Finalmente, los sensores serán colocados en el modelo (Figura 67), en la zona donde está situada la fotorresistencia en el sensor real. En la Figura 68 se muestra el lugar de estos sensores en la jerarquía de la escena.

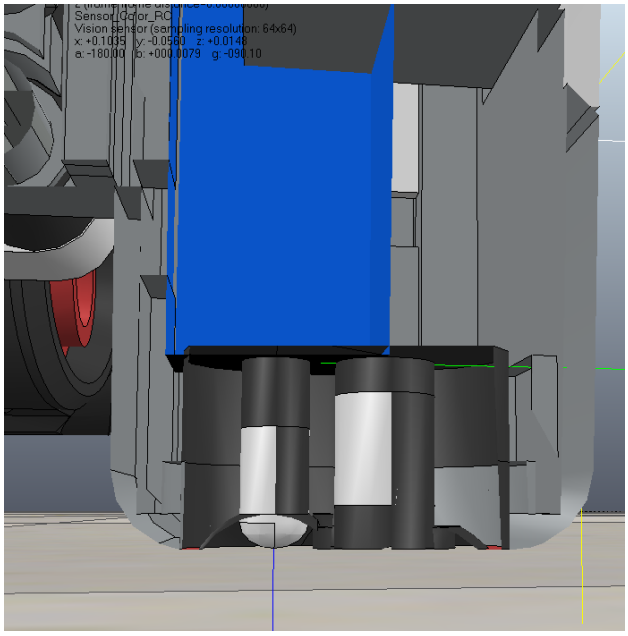

Figura 67. Posición de los sensores de visión en el modelo

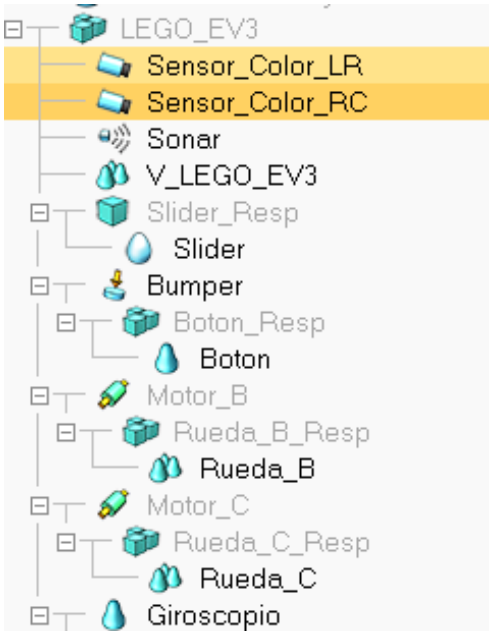

Figura 68. Posición de los sensores de visión en la jerarquía

3.4.4. Sensor giroscópico

El sensor giroscópico de LEGO (Figura 69), es un sensor que mide la rotación sobre un eje de coordenadas. Si el sensor gira en el sentido de las agujas del reloj proporcionará valores positivos, ya sea en grados o en grados/segundos, y si gira en sentido contrario, dará valores negativos (Figura 70).

Figura 69. Sensor giroscópico de LEGO

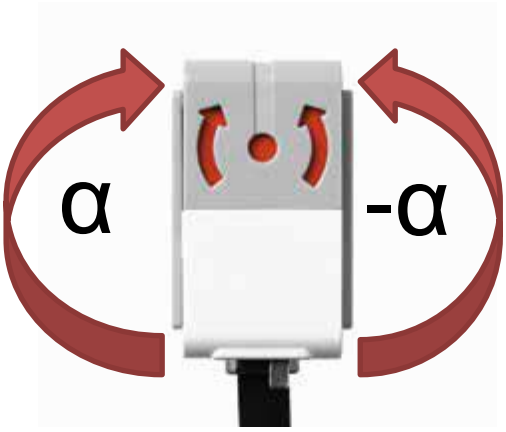

Figura 70. Sentido de giro del sensor

Este sensor posee dos modos de operación que se han desarrollado a partir de las modificaciones de un sensor giroscópico programado en lenguaje Lua ofrecido por V-REP, y que mide la velocidad angular de este. Estos modos de operación se describen a continuación:

- **Modo de detección de giro**. En este modo, el sensor recoge el ángulo de giro a partir de su posición base. Esta posición es la que inmediatamente se asigna cuando el sensor se inicia o cuando se ha llamado al comando o función para resetear el sensor. Este sensor tiene una precisión de ±3º para un giro de 90º.

  Para generar valores de giro con error a partir de la precisión especificada para un giro de 90º se propone generar valores aleatorios de una distribución normal ya que estos oscilan aproximadamente entre una cota superior y otra inferior acercándose lo máximo posible al valor real; como V-REP no dispone de una función para generar datos de dicha distribución en su API nativa en Lua se ha optado por utilizar el método de Box-Muller que enuncia que "si tienes dos números aleatorios U y V uniformemente distribuidos en [0, 1], (por ejemplo, la salida de un generador de números aleatorios), entonces X e Y son dos variables aleatorias estándar normalmente distribuidas" [62] cuyas fórmulas aparecen en la Ecuación 1; en este caso, solo vamos a utilizar la primera ecuación ya que genera valores de una distribución normal estándar.

$$X = \sqrt{-2\ln U}\cos(2\pi V)$$
$$Y = \sqrt{-2\ln U}\sin(2\pi V)$$

Ecuación 1. Ecuaciones del método de Box-Muller

  Para poder generar valores con error mediante el método descrito, necesitaremos una media (μ), que será el ángulo medido en el simulador, y una desviación típica (σ), la cual estimaremos con los datos de precisión descritos en las especificaciones. Para la estimación de la desviación típica se

ha optado por diseñar un algoritmo voraz [63] en Matlab, mediante sucesivas pruebas de Montecarlo [64], que estime mediante aproximación la varianza para que, en el caso de un giro de 90 grados, los valores devueltos por el método de Box-Muller estén aproximadamente comprendidos entre 93 y 87 con un porcentaje de aciertos entre el 95 y 95,3 % en dicho rango; la razón de esto es que los algoritmos voraces son una opción rápida y segura para obtener valores deseados mediante ensayo y error. Como valor de varianza inicial se ha optado por $\sqrt{3}$ debido a que en una prueba de Montecarlo de 1000000 de valores generados se ha obteniendo un porcentaje de aciertos de aproximadamente el 91 %. Todo este algoritmo se realiza siguiendo el siguiente esquema:

1. Declaración de variables auxiliares, asignación de la primera desviación típica (σ) y media de aciertos al 0% como primera entrada del algoritmo.

2. Si la media de aciertos está comprendida entre 95 y 95,3, se finaliza el algoritmo; en otro caso realizar una iteración:

   2.1. Si la media de aciertos es superior al 95,3 % sumar 0,001 a la desviación estimada; si es inferior a 95 % restar 0,001.

   2.2. Se realizan 30 pruebas de Montecarlo; cada iteración consiste en:

      2.2.1. Se genera una muestra de 1000000 de valores aleatorios procedentes de una distribución normal de media 90 y desviación típica σ;

      2.2.2. Se calcula el porcentaje de aciertos a los valores comprendidos entre 93 y 87.

      2.2.3. Se guarda el porcentaje en un array.

   2.3. Realizar la media del porcentaje de aciertos de cada prueba.

La ejecución del algoritmo nos ha calculado una desviación típica de 1,5301 con un porcentaje de aciertos de aproximadamente el 95,001 % para un giro de 90 grados realizando 202 iteraciones; este script de Matlab que implementa el algoritmo se puede encontrar en el directorio de los archivos del trabajo */Programas/Error en giro* con el nombre de *montecarlo_giro.m*. Con este valor podemos conseguir desviaciones típicas mediante proporcionalidad directa (solución hipotética) mostrada en la Ecuación 2.

$$\sigma = \frac{|\alpha| * 1{,}5301}{90}$$

Ecuación 2. Obtención de la desviación típica para el ángulo medido por el sensor

El código de la implementación de este modo de operación del sensor en lenguaje Lua del sensor giroscópico con su error, se puede encontrar en el directorio de los archivos del trabajo (CD-ROM o Drive) */Fuentes/ V-REP/GyroSensor_G.lua* o en el simulador en el script hijo adjunto al objeto "GyroSensor_G".

- **Modo de velocidad angular**. Este modo se encarga de medir la velocidad rotacional del sensor en grados/segundo. El sensor puede llegar a medir hasta un máximo de ±440º/s. Para poder programar el sensor también se ha tenido que modificar el código del sensor giroscópico que proporciona V-REP, consiguiendo que solo mida la velocidad angular en el eje Z del sensor y aplique el máximo de velocidad angular de las especificaciones.

  El código de este modo de operación se encuentra en el directorio de los archivos de la memoria */Fuentes/V-REP/GyroSensor_VA.lua* o en el script hijo adjunto al objeto "GiroSen-sor_VA" en el simulador.

Para finalizar, en el simulador ambos modos se han colocado dentro del sensor giroscópico del modelo (Figura 71) y su posición en la jerarquía de la escena en una forma del modelo del EV3 (Figura 72) para que estos queden unidos al modelo y no se desmonten.

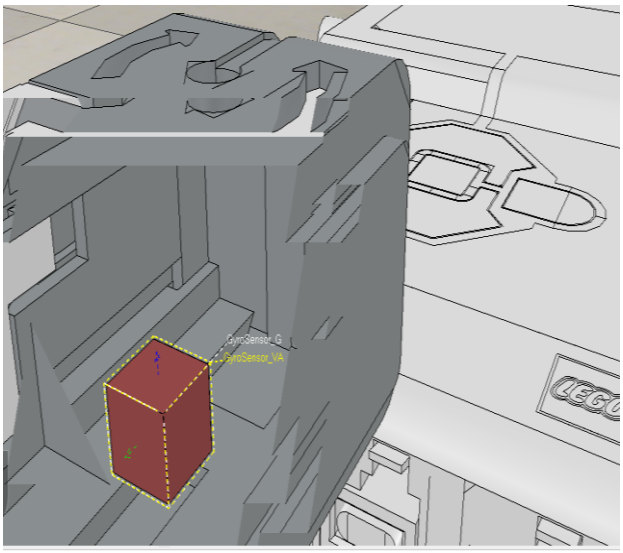

Figura 71. colocación de los modos del sensor giroscópico en la escena

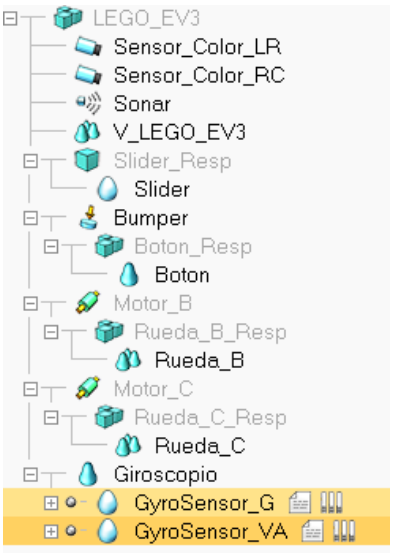

Figura 72: Posición de modos del sensor en la jerarquía

3.4.5. Sensor de rotación

Como ya se ha mencionado en la sección 3.3, los motores de LEGO EV3 poseen un sensor de rotación en cada uno de ellos, los cuales miden el ángulo de giro de estos en grados con una resolución de 1º. Como en la configuración de las articulaciones rotacionales hemos especificado que la posición de esta no es cíclica, mediante la API de V-REP podemos extraer el ángulo de giro de la articulación. En este sensor, no se han simulado errores ya que no se han encontrado suficiente información sobre este sensor para realizar un experimento que determine la generación de un error. Posteriormente, en la subsección 3.7.3, se comentará como se han desarrollado las funciones de reinicio del sensor de rotación y la extracción de los valores.

3.5. Otras consideraciones para el modelado

Esta sección está dedicada a comentar los extras que se han añadido al modelo del robot, para que pueda funcionar correctamente o pueda facilitar la tarea del testeo o realización de las prácticas.

3.5.1. Creación de la rueda loca

En nuestro modelo, como ya se ha explicado en la sección 3.1, la configuración de las ruedas del robot es de 2 ruedas tractoras y una bola que hace las veces de rueda loca. Para modelar el funcionamiento de dicha bola, se ha seguido el método que recomienda la documentación de V-REP en el tutorial del robot BubbleRob [65]. Para ello, se ha modificado el material de la forma pura de la bola en sus propiedades dinámicas, donde se ha establecido que la bola sea un material sin fricción, que solo nos va a servir de apoyo para las dos ruedas delanteras. Hecho esto, para que esté unida al resto del robot y no se desmonte, añadiremos un sensor de fuerza que será la unión entre la bola y el resto del robot. Colocaremos el sensor en dicha unión (Figura 73) así como su posición en la jerarquía (Figura 74).

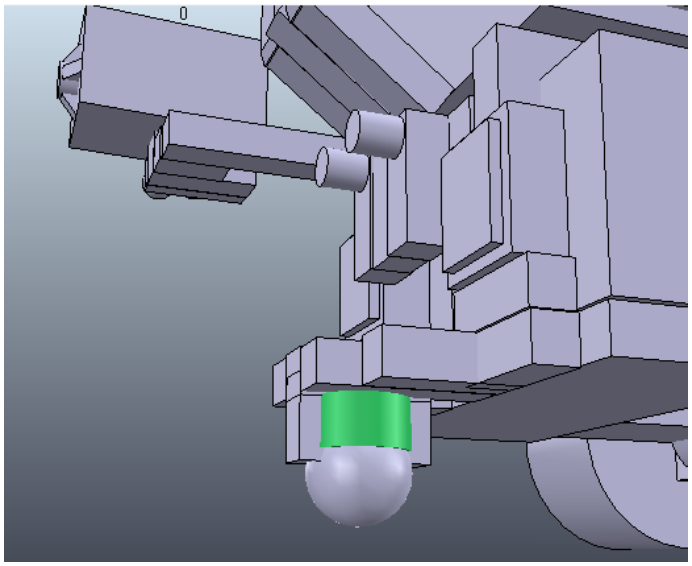

Figura 73. Colocación del sensor de fuerza para la rueda loca

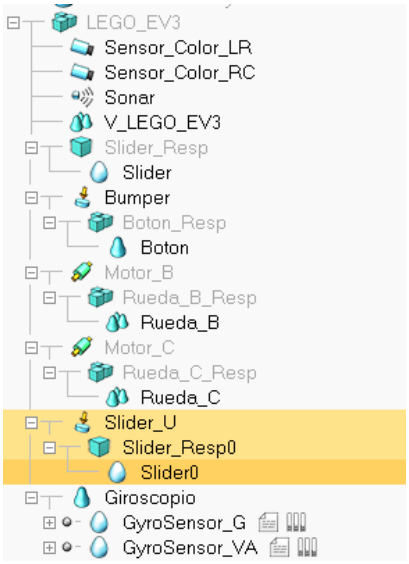

Figura 74. Posición del slider en la jerarquía de la escena

Hecho esto, al iniciar la simulación, se comprueba que el robot se mueve solo como si este temblara. La solución que se propone es redistribuir el peso de las ruedas (Figura 75) para que queden fijas en el suelo. Se ha utilizado un peso hipotético de 2,5 kg, debido a que se carece de una báscula para pesar el robot real, y asignando 0,6667 kg para cada una de las ruedas y la bola consiguiendo un reparto de peso para que el modelo no esté desbalanceado en las ruedas, el resto se lo añadiremos al cuerpo del robot; con esto corregiremos el movimiento del robot parado y, así tendremos la configuración de las ruedas de manera estable.

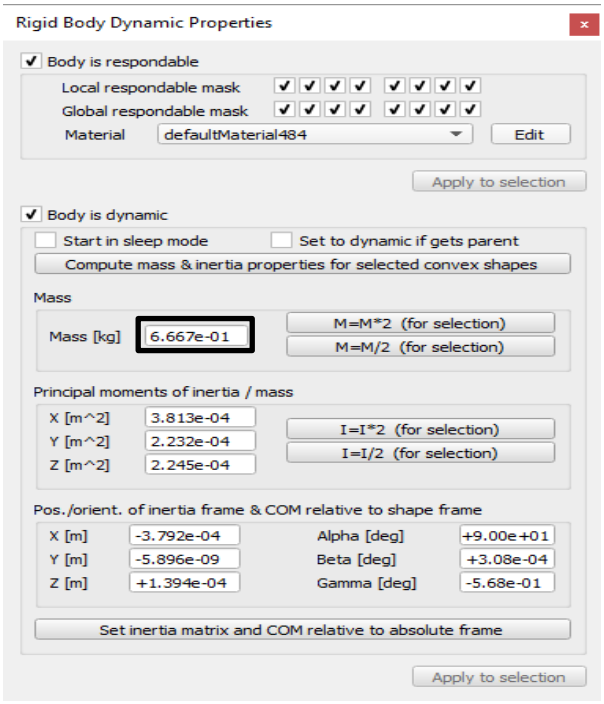

Figura 75. Propiedades dinámicas de las ruedas; marcado en negro, el campo de la masa del objeto

3.5.2. Cámaras y vistas

Para las pruebas o el apoyo a la resolución de las prácticas, se han colocado cámaras en sensores y el cuerpo del robot. Las cámaras, además de su posición en la jerarquía de la escena (Figura 76), se han colocado concretamente en los siguientes lugares:

- En la parte alta del ladrillo (Figura 77).
- En el campo de "visión" del sensor ultrasónico (Figura 78).
- En el botón del sensor táctil (Figura 79).

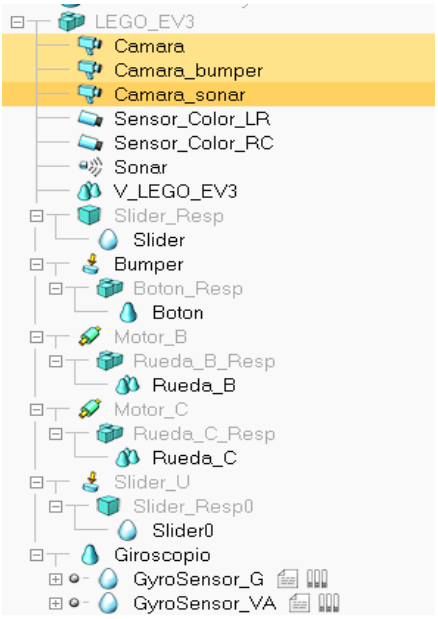

Figura 76. Colocación de las cámaras en la jerarquía

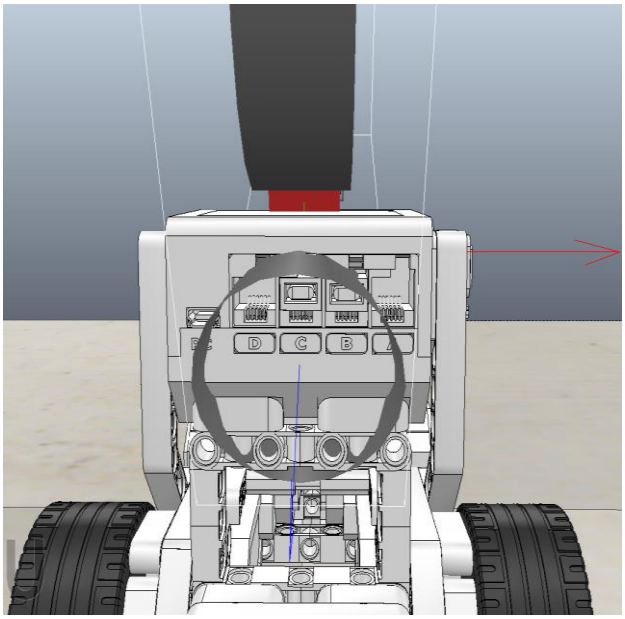

Figura 77. Cámara en la parte alta del LEGO EV3

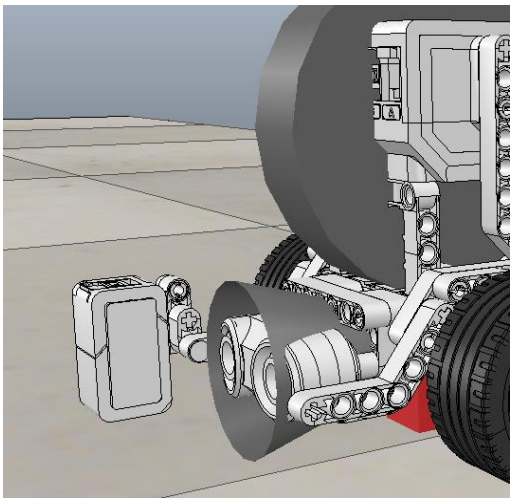

Figura 78. Cámara en el campo de visión del sensor ultrasónico

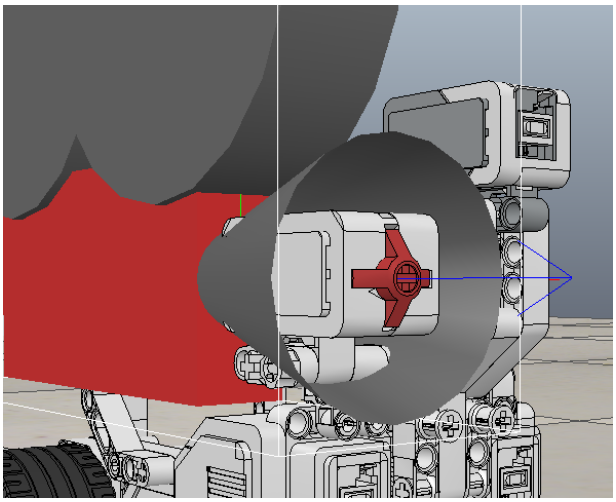

Figura 79. Cámara en el botón del sensor táctil

Con las cámaras ya colocadas, se pueden obtener imágenes de ellas si añadimos vistas flotantes; esto se explicará en el Anexo A, sección A.5.

3.6. Interfaz gráfica

Nuestro robot LEGO EV3 posee una pantalla en la que podemos visualizar datos (Figura 80). Además, incluye un conjunto de botones para interactuar de forma sencilla con él, así como luces de estado que nos indican su estado.

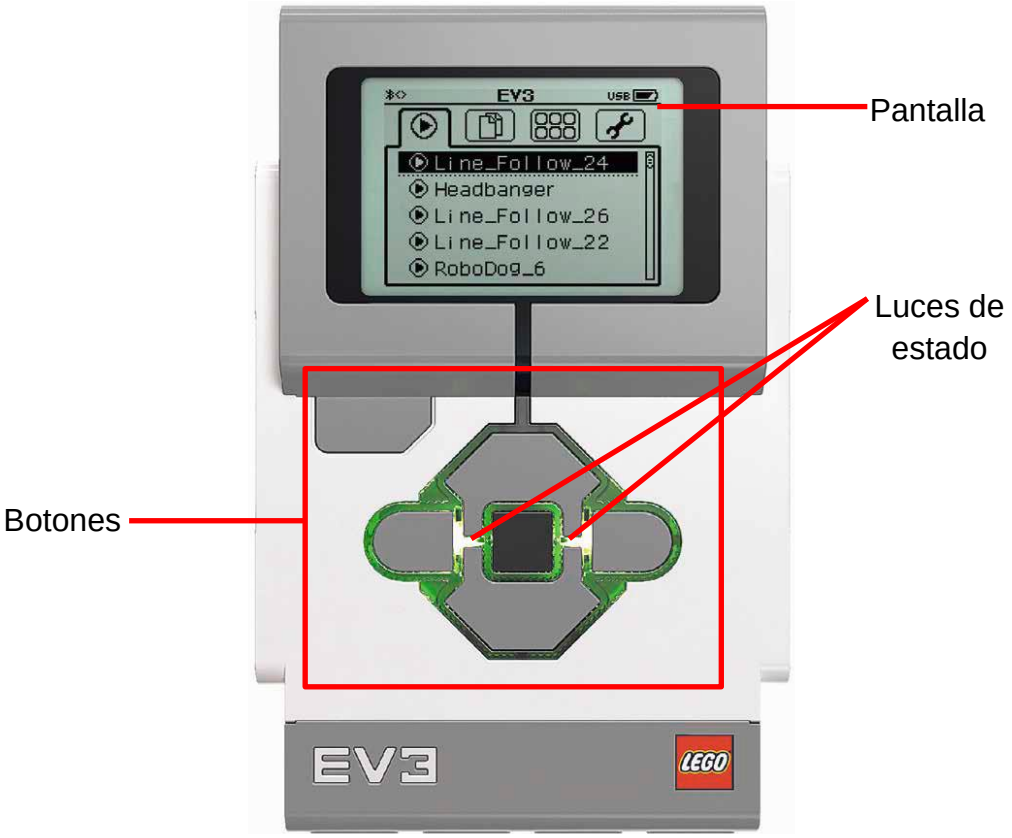

Figura 80. Bloque central de LEGO EV3

Para poder imitar el comportamiento del bloque, se ha optado por diseñar una interfaz gráfica en V-REP, ya que puede simular con bastante similitud la pantalla del EV3 y, al ser una interfaz gráfica de usuario, su uso es más intuitivo. Para dicha interfaz se ha usado el modo de edición de GUIs basado en OpenGL ("OpenGL-based custom UI") [66]. En sección de referencias se puede encontrar un enlace a la documentación del simulador donde se explica este modo.

Brevemente explicado, la creación de interfaces gráficas basadas en OpenGL de V-REP se basa en el diseño a partir de una matriz de rectángulos o cuadrados predefinidos por el usuario. Para conseguir un tamaño de interfaz aceptable para su visualización y más cercano a la resolución de la pantalla, se han definido cuadrados de 14 x 14 para generar una interfaz de 12 x 20 cuadrados. En estos cuadrados se pueden crear: botones, etiquetas de texto, cuadros de texto y sliders. También se puede seleccionar más de un cuadrado para crear elementos de mayor tamaño, además de poder agregar texturas a estos elementos. Mediante esta técnica, hemos

construido la interfaz gráfica imitando al bloque del robot LEGO EV3; en la Figura 81 podemos ver la interfaz completa.

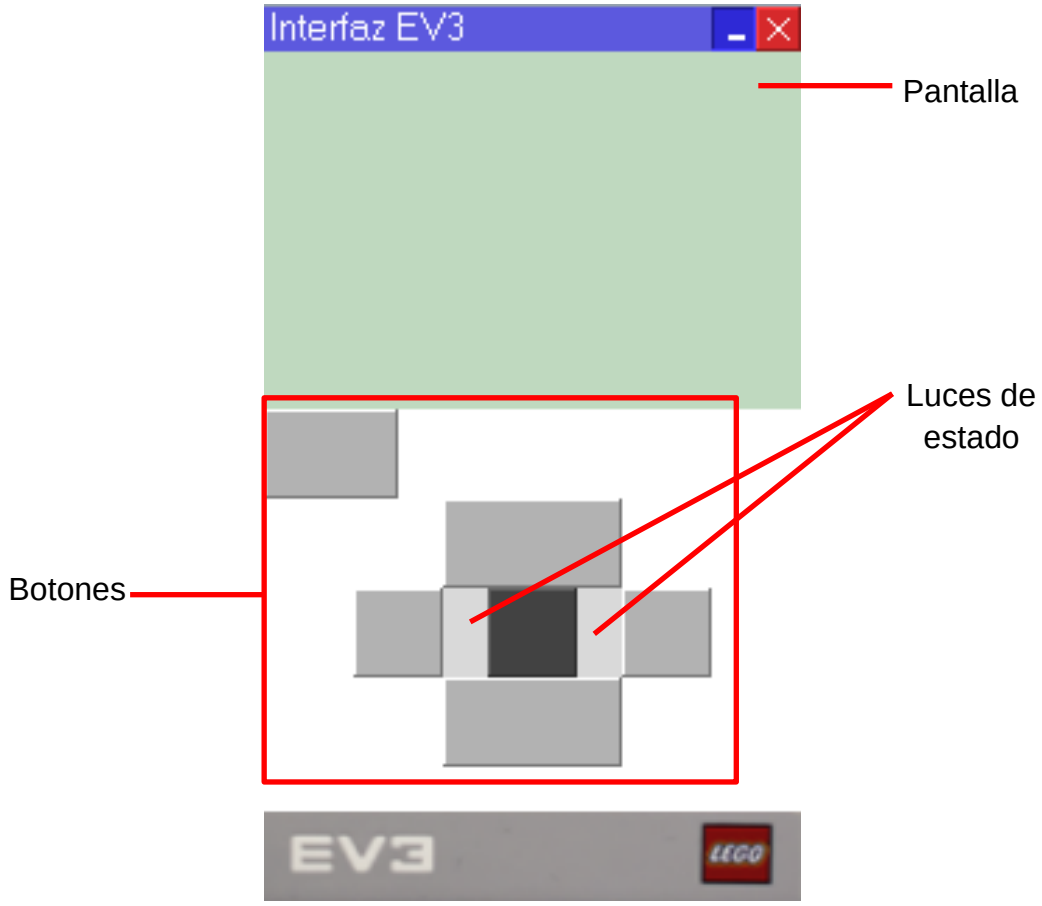

Figura 81. Interfaz del bloque del EV3

A continuación, detallaremos el desarrollo de cada una de las partes:

- **Pantalla.** La pantalla del robot LEGO EV3 real es de una resolución de 178 x 128 píxeles. Para diseñarla lo más fielmente posible, como el tamaño de los cuadrados es de 14 x 14 puntos, 12 x 8 cuadrados en la parte superior de la interfaz se destinan a la pantalla; por lo que la pantalla simulada sería de unos 168 x 112 puntos. No se ha podido conseguir un tamaño más grande de pantalla debido a que V-REP solo permite cuadrados o rectángulos menores que 14 x 14 puntos. Para la simulación de las líneas de texto en la pantalla, se han utilizado ocho etiquetas de texto (Figura 82).

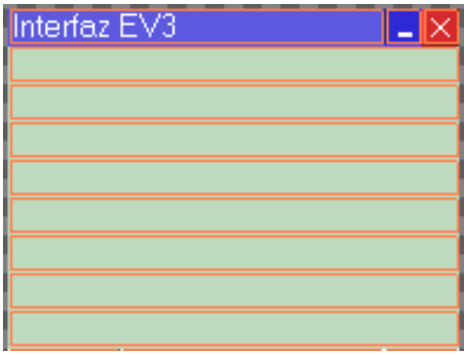

Figura 82. Líneas de texto de la interfaz de nuestro robot LEGO EV3

- **Botones**. Los botones de nuestro robot se han diseñado para responder a eventos de pulsar y soltar, es decir, cada vez que el botón es pulsado se crea un evento de pulsación para el identificador del botón indicado y, cuando este botón se deja de presionar, se produce otro. Estos eventos se pueden capturar mediante la API nativa de V-REP para los scripts.

- **Luces de estado**. Se han utilizado dos etiquetas de texto las cuales cambiarán al color que se les indique mediante las funciones de la API del simulador. Los colores que utilizará la interfaz gráfica para las luces de estado serán las mismas que las del robot real: roja, ámbar y verde (Figura 83).

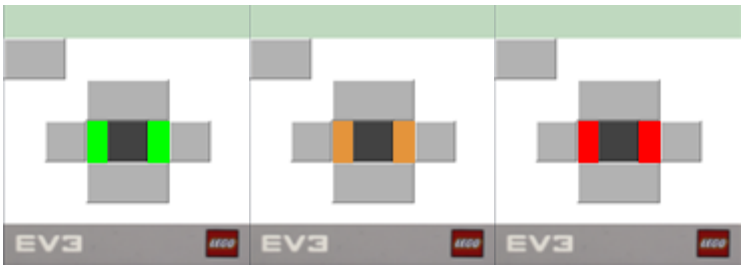

Figura 83. Colores de las luces de estado de la interfaz

Para terminar el diseño, asociamos la interfaz al objeto “LEGO_EV3” (Forma pura del cuerpo del robot). Esto es debido a que, como vamos a guardar un modelo para V-REP, la interfaz se pueda guardar junto con el modelo; el procedimiento es abrir el desplegable con el nombre “UI is associated with” y seleccionar el objeto. También estableceremos que la interfaz solo sea visible cuando la simulación esté ejecutándose para que, durante la creación de escenas con el robot o la preparación de un escenario para las prácticas, no aparezca. Además, estableceremos que la GUI siempre sea visible en la simulación y se pueda mover. Toda esta configuración viene reflejada en la Figura 84.

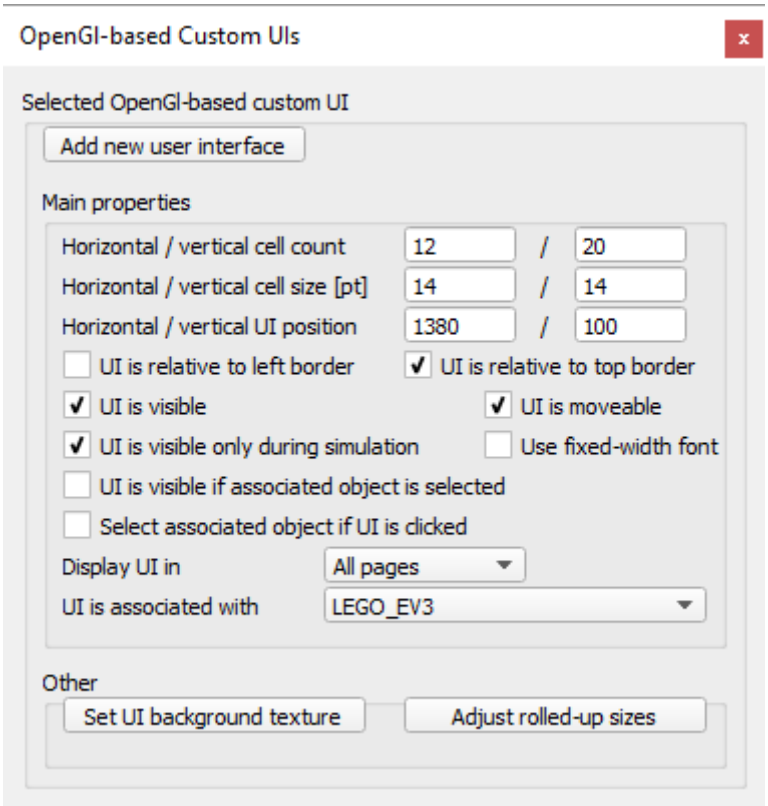

Figura 84. Configuración de la interfaz gráfica

Posteriormente, en la subsección 3.7.4, se explicará cómo se han ido capturando los eventos de los botones, la impresión de texto en la pantalla del interfaz y el comportamiento de las luces de estado.

3.7. Programación del script *Funciones* en V-REP

En el simulador, como se ha mencionado en la subsección 2.3.5, el enfoque de programación que se ha utilizado para controlar el robot en el simulador ha sido el uso de scripts hijo no-hilados; por ello, se ha creado un elemento “dummy”, denominado *Funciones* (Figura 85), que tendrá asociado el script hijo correspondiente para contener todas las funciones necesarias para poder ser llamadas mediante API remota desde Matlab, aparte de sus respectivas fases en la simulación. Además se ha añadido un parámetro para especificar el puerto donde se encontrará el servidor temporal de API remota que inicia este script para el control del robot (“PUERTO_API_REMOTA”).

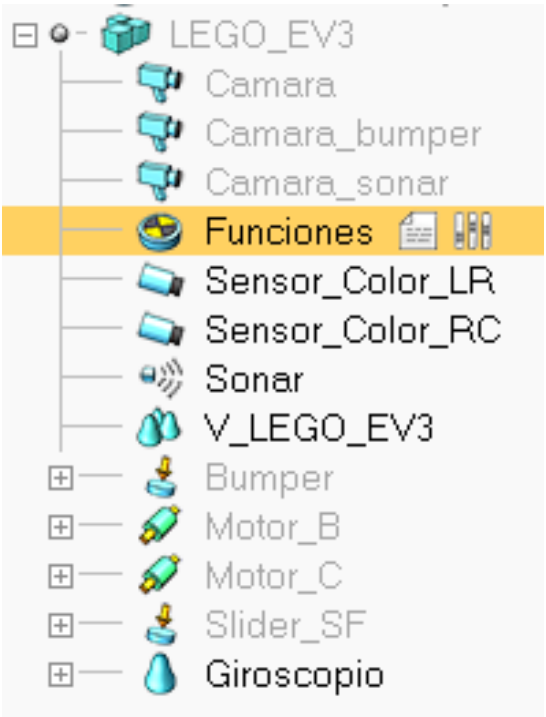

Figura 85. Situación del "dummy" Funciones en la jerarquía

El script se divide en dos secciones: las funciones; para ser ejecutadas mediante API remota desde Matlab, y las fases del script; para realizar algunas funcionalidades que presenta el robot. En estas funciones podemos comprobar que tienen el formato para ser ejecutadas desde Matlab con la función genérica de la API remota, ya que se llamarán mediante este procedimiento; dichas funciones en el script *Funciones* se agrupan en: de actuadores, de sensores, de interfaz gráfica y otras que no entran en estos grupos.

En el directorio de los archivos del trabajo (CD-ROM o google drive) */Fuentes/V-REP* puede encontrarse el archivo *Funciones.lua* donde se encuentra el código en lenguaje Lua de este script; este código también puede visualizarse en V-REP seleccionando el script hijo asociado al "dummy" Funciones. En las siguientes subsecciones se explicarán todas las funciones desarrolladas en V-REP y las partes del script implementadas.

3.7.1. Fases del script

Como se ha explicado en la subsección 2.3.5, los scripts hijo no-hilados se dividen en 4 partes, en las que se han declarado variables y constantes globales, funcionalidades que necesitan actualizarse en cada paso de simulación, etc. A continuación, se describirá la funcionalidad que desempeñará cada parte:

- **Parte de inicialización**. En esta parte, se han declarado todas las variables globales y constantes necesarias para la simulación, así como variables de

referencias a objetos de la escena y a la interfaz gráfica. Además, se iniciará el servidor temporal de API remota para ejecutar las funciones del script que son llamadas por el cliente en Matlab. Todo esto ha sido implementado en esta parte debido a que solo se ejecuta cada vez que se inicia la simulación, lo que es idóneo para declarar todas estas variables y constantes.

- **Parte de actuación**. Esta parte no implementa nada.

- **Parte de sondeo**. En esta parte se ha implementado el controlador de la interfaz gráfica. La razón por la ha desarrollado en la parte de sondeo es porque se ha interpretado que la captura de eventos son un tipo de sensor, pero no supondría ninguna diferencia implementarlo en la fase de actuación. En la subsección 3.7.4 se comentará la implementación de las funciones para la interfaz gráfica.

- **Parte de restauración**. En esta parte, se restaurarán las luces de estado en el caso de que tengan un color, ya que si no se hiciera esto, el color asignado permanecería en la siguiente simulación. Además, se detendrá el servidor temporal de API remota para este robot, dejando libre el puerto asociado.

3.7.2. Funciones de actuadores

Estas funciones se encargan del movimiento de los motores del robot. Como hemos mencionado en la sección 3.3, los motores del robot LEGO EV3 poseen una velocidad máxima entre 160 y 170 rpm. Para utilizar las funciones de movimiento de articulaciones en V-REP necesitamos la velocidad en radianes por segundo (rad/s); por lo que se ha utilizado el valor medio de la velocidad máxima de los motores, es decir 175 rpm, y se ha realizado la conversión a rad/s, dando como resultado 18.326 rad/s que será nuestra velocidad máxima para los motores del robot tanto positiva como negativa (hacia adelante o hacia atrás). Además, los motores presentan dos tipos de torque: el de parada y el de movimiento. Estos dos torques irán variando si el motor está apagado o encendido; en el caso de que el motor esté apagado, se aplicará el torque de parada y en el caso de que el motor esté encendido se usará el de movimiento. Los torques se irán asignando con las funciones correspondientes de modificación de torque máximo (“simSetJointForce”, para más información de esta función, consultar la documentación sobre funciones de la API de V-REP).

Cabe destacar que para hacer funcionar los motores hace falta indicar a qué motor o motores se le va proporcionar la velocidad angular; para ello, se ha optado por unos identificadores mediante números enteros, escogidos sin ningún criterio en particular, que nos permiten, en una misma función, utilizar todas las posibles combinaciones con los siguientes valores:

- **1**: motor B.
- **2**: motor C.
- **3**: ambos motores.

Para estas funciones, se han representado los movimientos básicos que definimos en la sección 3.1 (↑, ↓ y ─); estas funciones son las siguientes:

- **"On" (↑ y ↓)**. En esta función como entrada se necesitan dos valores enteros: el identificador del motor y la velocidad en radianes por segundo. La función consiste en asignar una velocidad angular en el motor o motores indicados, pudiendo ir hacia adelante o hacia atrás aplicando el torque de movimiento y una restricción para la velocidad máxima. En el caso de que se proporcione una velocidad igual a 0 se conservará el torque en movimiento.

- **"Off" (─)**. En esta función solo se necesita el identificador del motor; su funcionamiento consiste en detener el motor o motores indicados y aplicar el torque de parada.

3.7.3. Funciones de sensores

Estas funciones se encargarán de extraer y manipular valores de los sensores modelados en el simulador para que estos imiten, lo más fielmente posible, los valores del robot real; dichas funciones son:

- **"MotorRotationCountB" y "MotorRotationCountC"**. Estas funciones devuelven el ángulo de giro del motor en grados desde la posición inicial establecida, ya sea desde la posición de la rueda al inicio de la simulación o mediante la función "ResetRotationCount", la cual explicaremos en el siguiente punto. Para la obtención de estos datos se ha utilizado la función de la API de V-REP "simGetJointPosition", que devuelve la posición intrínseca de una articulación; en el caso de una articulación rotacional, nos devolverá el valor en radianes de la cantidad girada realizando, posteriormente, la conversión a grados. La razón por la que se han desarrollado una función para cada sensor rotacional de los motores, en vez de una única función que controle los dos, es debido a que como vamos a usar normalmente desde el cliente el modo de transmisión de datos para obtener los valores del sensor, no se permite llamar a la misma función con argumentos distintos.

- **"ResetRotationCount"**. Esta función tiene como entrada el identificador del motor, es decir, solo se permite el motor B o el C, pero no ambos motores. El funcionamiento consiste en guardar la posición de la articulación en el instante en que la función es llamada, para, posteriormente, restar esa

posición a la obtenida en la funciones “MotorRotationCountB” y “MotorRotationCountC” obteniendo así el reseteo de los sensores de rotación.

- **“SensorTouch”**. Esta función está destinada a devolver el estado de pulsación del sensor táctil; estos valores son: 0, para el botón sin pulsar y 1, en el caso de que este pulsado. Para ello, utilizaremos “simReadForceSensor” de la API del simulador para extraer los valores de estado (si se ha detectado fuerza en algún eje) y los Newton (N) de fuerza en el eje de coordenadas Z del sensor de fuerza “Bumper”, como se mencionó en la subsección 3.4.1. Para evitar que el sensor de lecturas erróneas debido a razones de inercia o por alguna otra causa, se ha establecido un umbral de 1 N en el eje Z para indicar que dicho botón ha sido pulsado.

- **“SensorSonar”**. Devuelve el valor medido por el sensor de proximidad del simulador; como se ha explicado en la subsección 3.4.2, el sensor ultrasónico devuelve un valor en cm. de la distancia medida con un decimal. Para calcular el error, se ha utilizado una hipotética desviación estándar de 0,5 para la oscilación de los valores decimales ya que no se ha dispuesto del tiempo suficiente para estimar este valor. En el caso de que el sensor no detecte nada, es decir, que el objeto esté fuera del rango de detección, muy cerca del sensor o demasiado lejos, la función devolverá el valor 255.

- **“SensorLight”**. Recupera el número entero del modo de intensidad de luz del sensor de visión “Sensor_Color_LR” mediante la función “simReadVisionSensor” devolviéndonos el valor medio de intensidad de color (valores entre 0 y 1, siendo 0 el color negro y 1 el blanco en escala de grises). Con ese valor, aplicando la Ecuación 3, se procede a la conversión entre los valores 7 y 71 y añadiendo un error aleatorio con media 0 y desviación típica entre 0,2905 para el valor de negro y 0,6102 para el valor de blanco (obtenido en el experimento explicado en subsección 3.4.3 para el modo color del sensor de luz reflejada) mediante el método de Box-Muller, cuya desviación se consigue calculándola con la Ecuación 4; donde I es el valor obtenido del sensor e I' el valor calculado a partir del anterior.

$$I' = 7 + 64I + X$$

Ecuación 3. Escalado del valor obtenido del sensor

$$\sigma = 0{,}2905 + 0{,}3197I$$

Ecuación 4. Cálculo de la desviación típica mediante el valor obtenido del sensor

- **“SensorColor”**. Devuelve los valores medios de rojo, verde y azul y la distancia del objeto más cercano al sensor de visión “Sensor_Color_RC” con la función “simReadVisionSensor”. No se ha realizado el reconocimiento de

colores en el simulador debido a que se carece de conocimientos de visión por computador para realizar esta tarea.

- **"SensorGryroVA"**. Retorna, en grados, la velocidad angular del sensor giroscópico, descrito en la subsección 3.4.4, mediante la señal "gyroZ_velocity".

- **"SensorGyroA"**. Recupera el valor del ángulo girado del sensor giroscópico en grados/s a partir de la posición inicial (al inicio de la simulación) o desde que fue llamada la función "ResetGyroA" (explicada en el siguiente punto), mediante la señal "gyroZ_angle".

- **"ResetGyroA"**. Vuelve el ángulo girado del sensor giroscópico a 0 mediante la señal "gyroZ_angle_reset".

3.7.4. Funciones de la interfaz

Para la interfaz gráfica, se ha intentado seguir el patrón MVC [67], donde cada uno de sus componentes es:

- **Modelo**: todas las funciones en Lua de la API de V-REP que nos permiten obtener datos de las interfaces gráficas.

- **Vista**: la interfaz gráfica diseñada.

- **Controlador**: toda la funcionalidad implementada en la fase de sondeo, que permite controlar los eventos de los botones y el control del parpadeo en la luz de estado; parte de esta funcionalidad está implementada en las funciones, ya que no ha sido necesario obtener o modificar datos por cada paso de simulación.

Las funciones que controlan y extraen datos de la interfaz gráfica son:

- **"TextOut"**. Se encarga de escribir en la pantalla del EV3 una cadena de caracteres en la línea de la pantalla especificada mediante un entero del 1 al 8. Para escribir en la pantalla del interfaz, se ha usado la función "simSetUIButtonLabel" donde se especifica la referencia de la etiqueta en la interfaz, en este caso del 1 al 8 (Figura 86), la referencia a la interfaz y el texto a escribir.

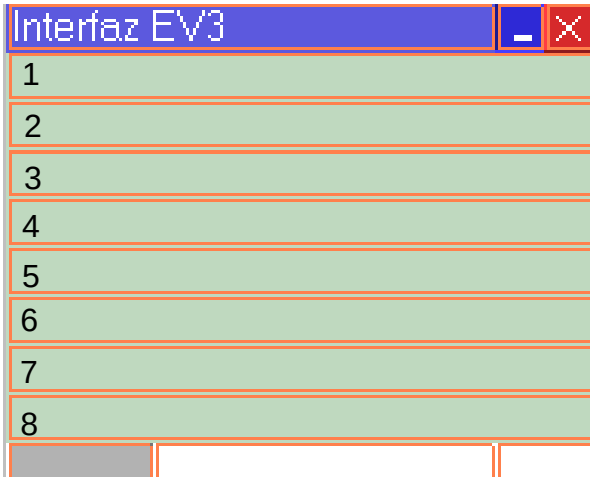

Figura 86. Referencias de las líneas de la pantalla

- **"ClearScreen"**. Esta función borra todas las líneas de la pantalla del EV3.

- **"StatusLight"**. Función para establecer o apagar las luces de estado, pudiendo hacer que parpadeen o se queden fijas. Para realizar el parpadeo, en la fase de sondeo se ha implementado una máquina de estados siguiendo el diagrama que aparece en la Figura 87, donde el estado 1 corresponde a las luces apagadas, y el estado 0 a las luces encendidas. El estado cambia pasados, aproximadamente, 0.5 segundos (variable "t", donde esta se vuelve a 0 cuando cambia de estado) o cuando la función de parpadeo se deshabilita (variable "a"). Los colores disponibles para la luz de estado son: verde (0), ámbar (1) y rojo (2); esta misma función sirve para apagar las luces (valor 4).

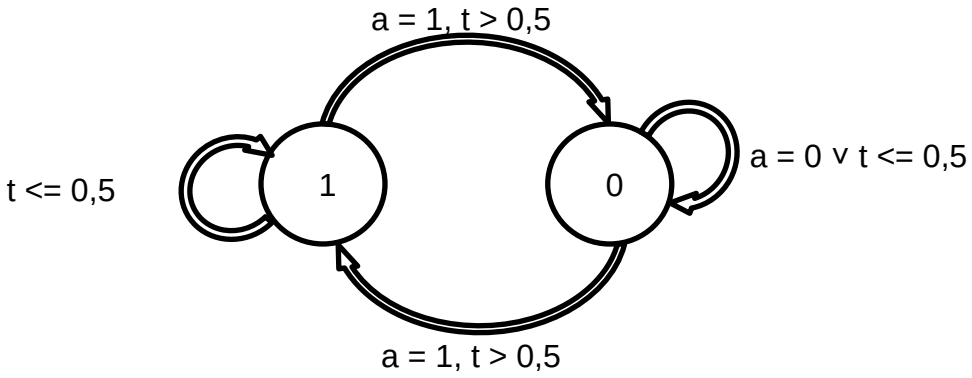

Figura 87. Diagrama de estados del parpadeo de la luz de estado.

- **"ButtonPressed"**. En esta función se devuelve el identificador del botón pulsado (Figura 88). En V-REP hemos diseñado estos botones para que respondan a un evento de pulsación y soltado de botón, por lo que en la fase de sondeo, se ha desarrollado un controlador que detecte la pulsación prolongada, siguiendo el diagrama de estados de la Figura 89; donde la variable "e" es el evento de pulsación (0 para cuando el botón es soltado y 1 cuando es pulsado), y los estados son:

- Estado “0”, que indica que no se ha pulsado ningún botón, y por tanto la función devolverá 0.

- Estado “1”, que indica que se ha pulsado algún botón y la función devolverá el identificador del botón.

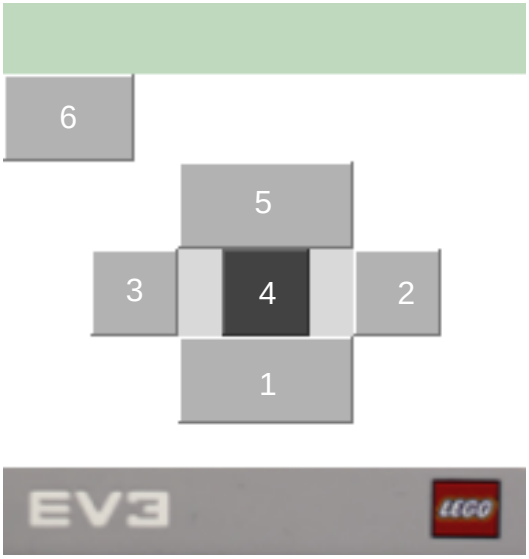

Figura 88. Identificador de cada botón

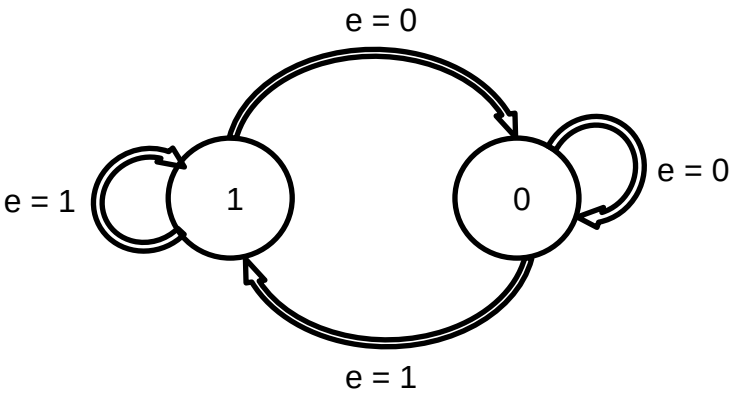

Figura 89. Diagrama de estados del controlador de los botones.

3.7.5. Otras funciones

Las funciones que se han implementado para extender la funcionalidad del robot; estas son:

- **“SimulationTimeStep”**. Devuelve el valor del paso de simulación, es decir, cada cuanto se ejecuta el script principal.

- **“emptyFunction”**. El fin de esta función es para realizar una funcionalidad de espera dado un tiempo (sección 4.3.7, función “wait”).

- **“CurrentTick”**. Devuelve un entero con el tiempo de simulación en segundos.

- **“Stop”**. Función destinada a terminar la ejecución de un programa para EV3. Pone variables globales a 0, borra la pantalla, apaga las luces de estado, limpia señales y activa una variable de control (“finish”) para que las funciones y las fases del script de actuación y sondeo no puedan ejecutarse.

CAPÍTULO 4.
TOOLBOX PARA LA PROGRAMACIÓN DESDE MATLAB

En este capítulo se documentarán todas las funciones y estructuras disponibles en la toolbox Matlab para simular el robot en V-REP. Como se ha mencionado en los objetivos de este trabajo (sección 1.1), el lenguaje de programación en el que se ha basado la sintaxis de las funciones de esta toolbox es NXC. La elección de este lenguaje es debida a que, actualmente, en la asignatura *Programación de Robots* se siguen usando robots LEGO NXT para el desarrollo de las prácticas, y para ayudar en la abstracción de la implementación del código en el robot real.

Además, se comentará el funcionamiento de cada función de Matlab y su interacción con el simulador. Todo el código desarrollado se encuentra en el directorio de los archivos de la trabajo (CD-ROM o drive) */Fuentes/Matlab*, separados por subcarpetas dependiendo del contenido tratado. Para todas las funciones, se ha habilitado el uso del comando “help”, que permite consultar toda la información necesaria para su uso. Este capítulo se dividirá en:

- **Clases manejadoras del cliente de API remota**. En esta sección se explicarán las clases de Matlab diseñadas para controlar la simulación y el robot en V-REP, proporcionando una capa de abstracción entre las funciones de NXC y el manejo del cliente de API remota y sus comandos.

- **Constantes de NXC y Scripts**. Se describirá el funcionamiento de los scripts desarrollados en Matlab para poder poner en marcha o detener una simulación, así como las constantes necesarias para el desarrollo de programas para el robot.

- **Funciones de NXC**. Se explicará el funcionamiento de todas las funciones desarrolladas (públicas y privadas) para el manejo del robot basadas en su mayoría en el lenguaje NXC; esta sección se divide en: funciones de actuadores, de inicialización de sensores, de uso de sensores, de interfaz, de manejo de ficheros, privadas y de otras que no se tratan en las demás subsecciones.

- **Iniciador de la simulación para EV3**. Se describirá la función de Matlab denominada “ejecutarCodigoNXC” que realiza todo lo necesario para ejecutar una simulación.

Todas estas implementaciones se han guardado en un archivo “.mltbx” para que Matlab pueda importar esta toolbox; para ello se ha utilizado la aplicación de empaquetado de toolbox de Matlab y el tutorial en la respectiva documentación [68].

4.1. Clases manejadoras del cliente de API remota

Como ya se ha mencionado en la subsección 2.4.2, V-REP dispone de la clase remApi que contiene los métodos y constantes necesarias para iniciar el cliente y comunicarse con el servidor de API remota. También posee ciertas funciones para la inicialización y finalización de la comunicación. Se ha optado por crear clases en Matlab proporcionando una capa de abstracción entre la clase remApi y las funciones de NXC debido a las siguientes razones:

- El formato de las funciones de los comandos de la API remota es muy verboso, provocando en muchas ocasiones la ilegibilidad del código, con esto se consigue ocultar argumentos innecesarios para la implementación de las funciones de NXC.

- Para inicio y finalización del cliente para la comunicación con la API remota se precisan de numerosos comandos para realizar estas tareas, por lo que creando constructores y destructores para estas clases que inicien la comunicación se lograría la automatización de dichas tareas.

- Crear una capa de abstracción entre la clase remApi (iniciador del cliente de API remota y su colección de comandos) y las funciones de NXC.

- Conseguir un control de la simulación más sencillo con métodos de inicio, pausa y parada.

- En el caso que se cree un conjunto de funciones para controlar al robot distinto del formato de funciones de NXC o simplemente para ampliar su funcionalidad, se usen estas clases como framework para extraer datos del robot (ventaja del modelo en capas).

- Permite la identificación de los valores de retorno mediante sus métodos pudiendo lanzar excepciones de Matlab.

El modelado de estas clases aparece en la Figura 90, cuyo código se encuentra en el directorio */Fuentes/Matlab/Clases manejadoras* junto con los archivos necesarios para su llamada (para arquitecturas x64).

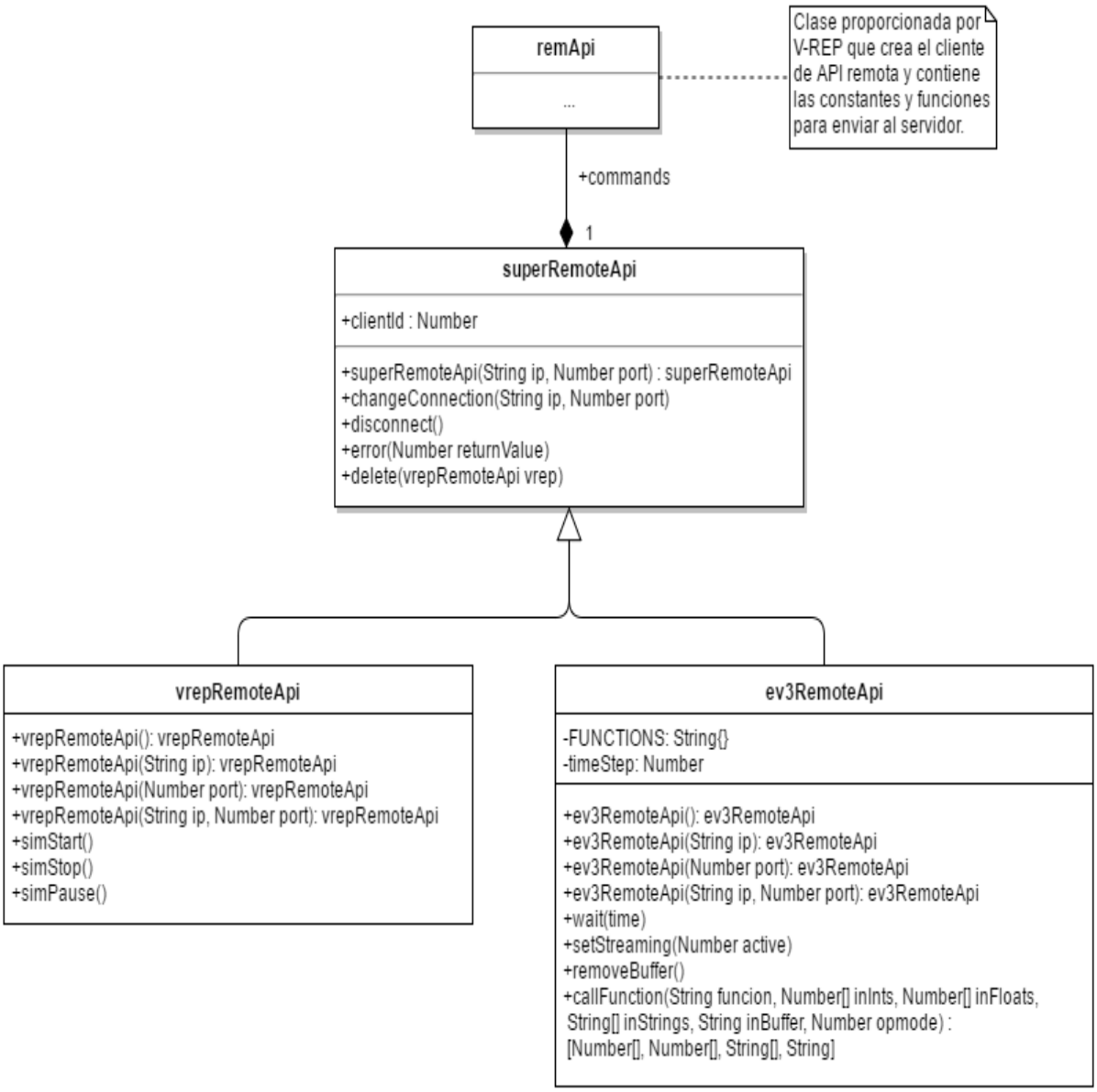

Figura 90. Diagrama de clases de los objetos manejadores del cliente

A continuación se describen con más detalle las 3 clases creadas:

- **"superRemoteApi"**. Esta es la clase padre de las posteriores que se van a describir; se encarga de crear el objeto y de conectarse, desconectarse y cambiar de servidor de API remota en V-REP. Además, contiene el destructor para finalizar la conexión y liberar memoria. Otro método importante es el método "error" que se encarga de interpretar los valores de retorno de cada función de la API remota y, si esta función produce un error, lanzar una excepción de Matlab. Sus atributos son: "commands", que contiene la instancia de la clase remApi, y "clientId" que tiene el valor de conexión con el servidor, necesario para llamar funciones de la API remota.

- **“vrepRemoteApi”**. Esta clase hereda de “superRemoteApi”, por lo que conserva los métodos y atributos de esta; está destinada al manejo de la simulación en V-REP conectándose al servidor continuo de API remota (subsección 2.4.1), es decir, se encarga de iniciar, detener y pausar la simulación (métodos “simStart”, “simStop y “simPause”). Sus constructores permiten conectarse al simulador mediante una ip, un puerto o ambos; en el caso de que se usen los constructores ”vrepRemoteApi()”, “vrepRemoteApi(String ip)” y “vrepRemoteApi(Number port)”, se usará la ip “localhost” y/o el puerto 19997, que es el puerto por defecto para el servidor continuo de API remota.

- **“ev3RemoteApi”**. Esta clase también hereda de “superRemoteApi”, pero en este caso está destinada al control del robot en el simulador. Para poder iniciar el cliente que va a controlar el robot, hay que iniciar la simulación primero; esto es debido a que, al usar un servidor temporal de API remota (subsección 2.4.1), el servidor se inicia desde el script *Funciones* cuando la simulación ha comenzado. El funcionamiento de los constructores es el mismo que en la clase “vrepRemoteApi”, pero el parámetro del puerto por defecto será el 20000, ya que la documentación de V-REP recomienda su uso a partir de este número, además de estar indicado en el parámetro del script hijo asociado al objeto “Funciones” en el simulador. También posee una constante privada llamada “FUNCTIONS” que contiene el nombre de todas las funciones del script *Funciones* que devuelven valores; su utilidad es tener una colección de nombres para aplicar los métodos “setStreaming” y “removeBuffer”, los cuales se explicarán posteriormente. Por último, esta clase tiene un atributo privado llamado “timeStep” que almacena el dato del tiempo entre pasos de simulación, muy útil para la función “wait”. Los métodos específicos de esta clase son:

  - **“wait”**. Este método se encarga de llamar a la función "EmptyFunction” del script hijo asociado al objeto “Funciones”, donde, como parámetro, se hace esperar un tiempo de simulación determinado. Se ha implementado calculando la cantidad de veces que tiene que llamarse a la función “EmptyFunction” desde Matlab con el modo de operación de llamadas bloqueantes. La razón por la que se ha implementado de esta manera es que, al usar el modo bloqueante, forzamos a la función a esperar una respuesta en el cliente por cada paso de simulación, esto produce que los retardos de respuesta sean, aproximadamente, de la misma duración que el tiempo entre pasos de simulación; por ejemplo, si tenemos que hacer una espera de 1 segundo y tenemos un paso de simulación de 0,05 segundos, tendremos que llamar a la función “EmptyFunction” 20 veces. El funcionamiento de “wait” aparece reflejado en la Figura 91.

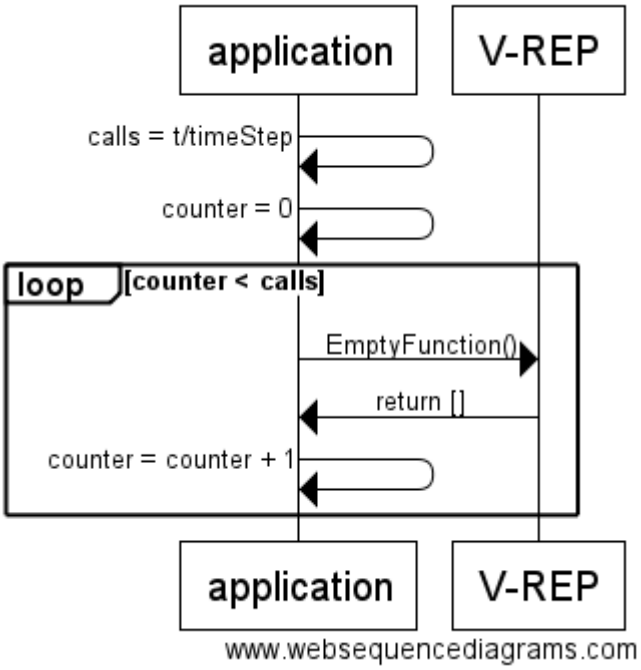

Figura 91. Diagrama de secuencia de la función "wait"

- **"callFunction"**. Este método se encarga de llamar a la función de la API remota "simxCallScriptFunction" (función genérica para la API remota de Matlab) para poder llamar a las funciones desarrolladas en el simulador que controlan y obtienen información del modelo. Los atributos de salida de esta función son los arrays de cada tipo de datos que devuelve dicha función. En cuanto a los atributos de entrada destacamos:

  - **"función"**: Nombre de la función a llamar.
  - **Atributos "in-"**: todos los atributos de entrada para la función genérica en forma de array para cada tipo.
  - **"opmode"**: modo de operación que se usará para llamar a la función; estos están representados mediante los enteros del 0 al 5 y estos, por orden, son: "simx_opmode_oneshot", "simx_opmode_buffer", "simx_opmode_blocking", "simx_opmode_streaming", "simx_opmode_discontinue" y "simx_opmode_remove".

  Para el modo de operación "simx_opmode_buffer", se puede tardar un tiempo en recibir los datos de entrada en el buffer; en el caso de que ocurra esto, si el valor de retorno es "simx_return_novalue_flag" con dicho modo de operación, entrará la función en un bucle hasta que el valor esté disponible, es decir, que el valor de retorno sea "simx_return_ok"; si pasadas 5000 iteraciones no se ha obtenido el valor de retorno correcto, se supondrá que la comunicación con el simulador se ha detenido y el método lanzará una excepción de Matlab.

- **“setStreaming”**. Activa o desactiva el modo de operación de transmisión de datos para las funciones que están almacenadas en la constante “FUNCTIONS”; estas son: “SensorTouch”, “SensorSonar”, “SensorLight”, “SensorColor”, “SensorGyroVA”, “SensorGyroA”, “CurrentTick”, “ButtonPressed”, “MotorRotationCountB” y “MotorRotationCountC”. Este método llama a “callFunction” con el modo de operación “simx_opmode_streaming”.

- **“removeBuffer”**. Elimina los datos de los buffers de entrada de las funciones que retornan valores del simulador y usan el modo de operación de transmisión de datos.

4.2. Constantes de NXC y scripts

El lenguaje NXC para el robot NXT posee multitud de constantes que referencian al hardware, ya sea para indicar que motor mover, el puerto de entrada donde está conectado cada sensor, botones del ladrillo, etc. Para imitar este lenguaje, hemos seleccionado un conjunto de constantes que emulan la sintaxis de estas; no se han añadido todas las constantes debido a que el conjunto de funciones que hemos usado es bastante reducido en comparación con el lenguaje real y solo se han recogido las imprescindibles. Además, en cada función que tome como parámetro una de estas constantes, se les ha hecho una comprobación mediante expresiones regulares para que la constante utilizada sea la correcta. A continuación se describen más detalladamente dichas constantes y su funcionalidad en la toolbox:

- **“LCD_LINE<N>”**. Estas constantes, basadas en la alineación de la pantalla en NXT, indica la línea de la pantalla en la interfaz del EV3. <N> está comprendido entre los naturales [1,8]. Los valores de estas constantes serán utilizadas para las funciones que se encargarán de escribir en la pantalla de la interfaz del robot.

- **“BTN<posición>”**. Son las constantes de botones para NXT, en este caso como el lenguaje es para ese modelo de Mindstorms, se ha tenido que utilizar su configuración de botones (Figura 92). <posición> puede ser: “EXIT”, “RIGHT”, “LEFT” o “CENTER”, que indican cada uno de los botones. Estas constantes toman los valores del 1 al 4, en el orden en que han sido mencionados. Cada una de ellas nos sirve para usarlas en la función que comprueba si un botón ha sido pulsado. En la Figura 93 se puede observar este mapeo de botones en la interfaz, donde los dos botones restantes no tendrán uso.

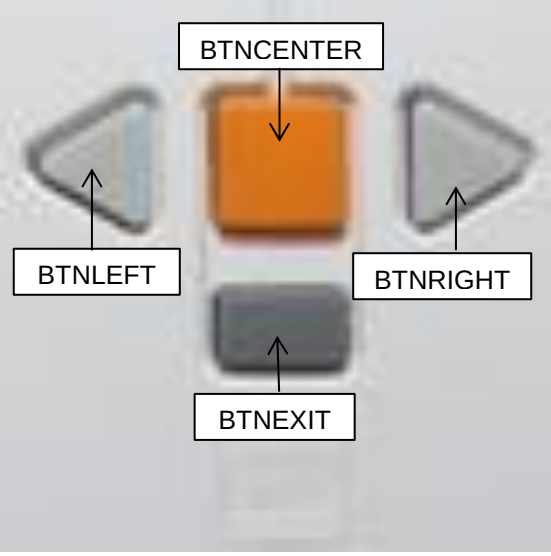

Figura 92. Configuración de botones de NXT y su correspondiente constante

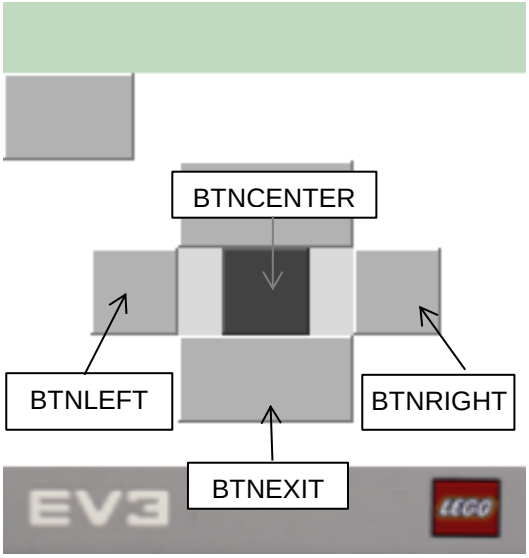

Figura 93. Mapeo de los botones en la interfaz

- **"IN_<N>"**. Estas variables sirven para referenciar el puerto de entrada de datos de sensores, donde <N> en el nombre de la constante es el número del puerto que se va a usar, comprendido entre los números naturales [1,4]. En NXC, inicialmente el robot no sabe a qué puerto está conectado cada sensor; para ello estas variables toman valores, mediante las funciones de inicio de sensores (subsección 4.3.2), para saber que sensor tiene conectado el puerto. Estos valores son:

  - -1: indica que no tiene ningún sensor conectado.
  - 0: sensor táctil conectado.
  - 1: sensor de luz conectado.
  - 2: sensor de color conectado (no usado).
  - 3: sensor giroscópico conectado.

  Por ejemplo, si en el puerto 4 tenemos conectado un sensor de luz, la constante de NXC "IN_4" deberá tener el valor 1 (asignado mediante una función de inicio de sensores).

- **"OUT_<puerto>"**. Estas constantes son de salida o de actuadores, en este caso se encargarán de indicar a qué motor irá la orden. <puerto> en el nombre de la constante indica a qué puerto irá la orden, los cuales son: A; orden para el puerto A (valor 1), C; orden para el puerto C (valor 2) y AC; orden para ambos puertos (valor 3). Como nos estamos basando en NXC, la definición de puertos a los que están conectados, son distintos a los que hay en el simulador, donde el puerto A es el puerto B y el puerto C sigue siendo igual que en EV3.

Para construir el espacio de trabajo y usar estas constantes, se han creado scripts que automatizan esta tarea; estos son:

- **"InitNXC"**. Se encarga de inicializar las constantes anteriormente descritas y crear las clases manejadoras de V-REP para arrancar la simulación y EV3. Estas variables cargadas en el espacio de trabajo de Matlab no deben modificarse debido a que, si llegaran a cambiarse, el programa para el robot podría no funcionar correctamente; por tanto se tienen que tomar como palabras reservadas.

- **"ClearNXC"**. Su función es detener la simulación, destruir los objetos manejadores de EV3 y V-REP y limpiar el espacio de trabajo de Matlab. Usar siempre este script para limpiar el espacio de trabajo si se está usando las funciones de la toolbox desarrollada en este trabajo, ya que, en el caso de que usemos la función "clear" de Matlab, los objetos se quedarían en memoria y no referenciados al eliminarles sus variables, por lo que ocurriría que no se podrían destruir; además, si la conexión con el simulador está iniciada, no se volvería a llamar, ya que el objeto eliminado está ocupando la conexión, por lo que habría que reiniciar Matlab para solucionar este problema.

Estos scripts se encuentran en el directorio de ficheros del trabajo */Fuentes/Matlab/Scripts*.

4.3. Funciones de NXC

En esta sección se explicará el funcionamiento de todas las funciones desarrolladas de la toolbox en Matlab para el manejo del robot. Como en el caso de las constantes (sección 4.2), se han recogido sólo un número determinado de funciones de NXC, debido a que son las imprescindibles para que el robot pueda funcionar correctamente y usar datos de sensores, utilizar actuadores y la interfaz gráfica, así como otras funciones que sirven de ayuda al desarrollo de programas para este. Todas estas funciones se encuentran en el directorio de los archivos del trabajo */Fuentes/Matlab/Funciones NXC* repartidos en subcarpetas dependiendo de la subsección.

4.3.1. Funciones de actuadores

Estas funciones se encargan del movimiento de los motores del robot. Las funciones de movimiento del motor en NXC, en lo referente a la velocidad, están escaladas de 0 a 100, donde 0 es el motor parado (no recomendado para parar el robot real ya que deja encendidos los motores y en nuestro robot simulado no cambia al torque en parada), y 100 es la máxima velocidad a la que pueden llegar estos, en nuestro

caso, 165 RPM. Como las funciones que se han desarrollado en V-REP únicamente aceptan valores de velocidad angular en rad/s, nuestra velocidad angular máxima es, aproximadamente, de 18.326 rad/s; por lo tanto, para calcularla a partir de su porcentaje, se ha usado la Ecuación 5, donde "velocity" es la velocidad proporcionada en la función (de 0 a 100), y "target" es la velocidad angular suministrada al motor.

$$target = \frac{18.326 * velocity}{100}$$

Ecuación 5. Calcular la velocidad del robot a partir de la escala de 0 a 100

Podemos destacar 3 funciones:

- **"OnFwd"**. Se encargará de enviarle al/los motor/motores un valor de velocidad de avance, pudiendo ir hacia adelante (↑) o hacia atrás (↓) dependiendo del signo de la velocidad. Esta función llama a "On" en el simulador mediante la clase manejadora del robot.

- **"OnRev"**. Similar a "OnFwd". También permite ir hacia delante o hacia atrás, pero en este caso en sentido inverso; por ejemplo, si queremos mover el motor A hacia atrás (↓), escribiremos "OnRev(OUT_A, 60)" que es equivalente a escribir "OnFwd(OUT_A, -60)". Esta función, también, llama a "On" en el simulador mediante la clase manejadora del robot.

- **"Off"**. Se encarga de detener el/los motor/motores (─); por ejemplo, "Off(OUT_AC)". Esta función llama a "Off" en el simulador mediante el objeto manejador del robot.

4.3.2. Funciones de inicio de sensores

En NXC, para extraer datos de un sensor, previamente hay que indicar a qué puerto está conectado. El robot realizado en V-REP acepta llamadas a las funciones del script *Funciones* para extraer datos de los sensores sin necesidad de especificar el puerto, pero las funciones de inicio de sensores tienen que ser utilizadas en el robot real para indicar en qué puerto el sensor está conectado; por lo que estas funciones se han utilizado para imitar este proceso e impedir que se utilicen funciones de sensores sin haber especificado el puerto.

Usaremos los puertos definidos en las constantes de puerto de sensores (IN_<N>) para almacenar el tipo de sensor que hemos conectado al puerto indicado en la función. Por último, no necesitaremos comunicarnos con V-REP para este proceso.

El nombre de todas las funciones de inicio de sensores es el siguiente:

- **“SetSensorTouch”**, para el sensor táctil.
- **“SetSensorUltrasonic”**, para el sensor ultrasónico.
- **“SetSensorLight”**, para el sensor de color en modo luz reflejada.
- **“SetSensorHtGyro”**, para el sensor giroscópico en modo velocidad angular.

Todos ellos están implementados de igual forma, la única diferencia es el valor que toma las variables IN_<N>.

Al ejecutar alguna de estas funciones en Matlab, la constante de sensores definida como parámetro en la función adquirirá el valor, según el tipo de función descrito anteriormente, del sensor conectado. Un ejemplo de una llamada correcta es la que aparece en la Figura 94, y una llamada en la cual se le ha pasado una constante distinta a las de entrada en la Figura 95.

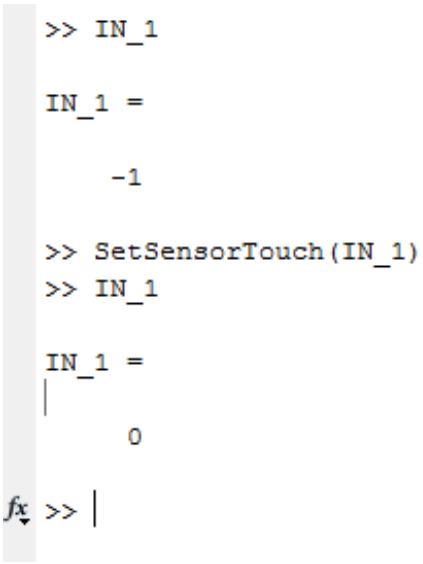

Figura 94. Ejemplo correcto de ejecución de la función “SetSensorTouch”.

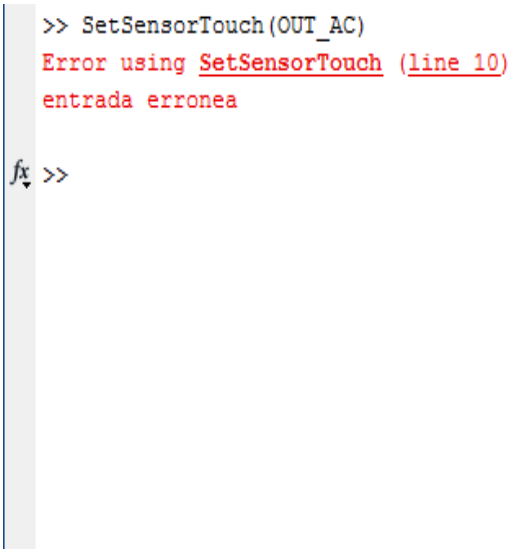

Figura 95. Ejemplo de entrada errónea en la ejecución de la función “SetSensorTouch”.

4.3.3. Funciones de uso de sensores

Estas funciones de nuestro repertorio de NXC están destinadas a la extracción de datos de los sensores, utilizando la respectiva constante de puerto de entrada, la cual corresponde a cada sensor. Todas estas funciones, en el caso de que se les ingrese una variable de entrada de sensores (IN_<N>), comprueban que el sensor asociado a dicha entrada corresponde con la función; por ejemplo, si se quiere obtener la distancia calculada del sensor ultrasónico y este está conectado en el puerto 3 de sensores, previamente se llama a la función “SetSensorUltra-sonic(IN_3)”, para indicar que el sensor está conectado a dicho puerto, para más

adelante llamar a “SensorUS(IN_3)” que nos devolverá el valor de distancia; estas funciones son:

- **“MotorRotationCount”**. Recoge el ángulo de giro del motor en grados usando la función con su mismo nombre en V-REP. Como entrada recibe la constante correspondiente del puerto de actuadores donde se van a obtener los datos, es decir, “OUT_A” y “OUT_C”; un ejemplo de llamada a esta función sería “rotationA = MotorRotationCount(OUT_A)”. En esta función no es necesario indicar a que puerto está conectado el motor, debido a que, en la realidad, los puertos de salida de tanto EV3 y NXT siempre van a tener conectados motores.

- **“ResetRotationCount”**. Reinicia el conteo de grados del ángulo de giro del motor, por ejemplo “ResetRotationCount(OUT_C)”. Para que el reinicio sea efectivo, si usamos un paso de simulación muy largo (por ejemplo, 50 ms), puede que el valor del sensor de rotación del motor no se haya actualizado y, por lo tanto, se tenga que realizar una espera hasta que el sensor devuelva el valor reseteado. Como parámetros de entrada, utilizará el mismo que la función del punto anterior.

- **“SENSOR_<N>”**. Según la guía del programador de NXC [69], extrae el valor de un sensor escalado donde <N> es el número de puerto de sensores (del 1 al 4), por ejemplo, “touch = SENSOR_2”; en nuestro caso, los sensores escalados que usamos en nuestro modelo son el modo de luz reflejada del sensor de luz (valores 0 a 100) y el sensor táctil (0 o 1). La implementación de esta especie de “constante” se ha realizado mediante el uso de una función, pero como Matlab acepta la notación de constante para la ejecución de funciones sin argumento, hemos podido implementarla de esta manera. Para la extracción de ambos sensores se ha usado una función privada llamada “privateSensor” descrita en la subsección 4.3.6.

- **“Sensor”**. Mismo funcionamiento que “SENSOR_<N>” pero pasando la constante de puerto de sensores como argumento, por ejemplo “sensor = Sensor(IN_4)”.

- **“SensorHtGyro”**. Para esta función, nos hemos basado en la función del sensor giroscópico de HiTechnic para NXT [70]; y consiste en devolver la velocidad angular del giroscopio en grados/segundo con un relleno dado el puerto de sensores y llamando a la función “SensorGyroVA” en el simulador. Un ejemplo de llamada a esta función de este sensor sería “rate = SensorHtGyro(IN_1, 0)”.

- **“SensorUS”**. Esta función se encarga de recoger el valor medido por el sensor ultrasónico del simulador, como se ha dicho en la subsección 3.4.2, devuelve un valor en cm de la distancia medida. Para extraer el dato del sensor del simulador, llama a la función “SensorSonar” del script *Funciones* en el simulador. Para llamar a esta función, un ejemplo sería “distance = SensorUS(IN_3)”.

4.3.4. Funciones de la interfaz gráfica

Este apartado se va a dedicar a las funciones basadas en el lenguaje NXC relativas a la interfaz del robot, es decir, a la pantalla y los botones; estas funciones son:

- **“ButtonPressed”**. Como parámetro de entrada se utiliza una constante de botones para comprobar si ese botón ha sido pulsado. Esta función solo detecta los botones que tienen constantes definidas (sección 4.2) y devuelve 0 o 1 dependiendo de si el botón que se ha pasado como parámetro ha sido pulsado (1 si ha sido pulsado; 0 si no lo ha sido), por ejemplo, “pressed = ButtonPressed(BTNCENTER)”. “ButtonPressed” se encarga de llamar a la función en V-REP que lleva el mismo nombre, para recoger el ID del botón que ha sido pulsado.

- **“TextOut”**. Escribe un texto en la pantalla del EV3 indicando las posiciones “tabulación” y “línea”, donde “tabulación” será el espacio que se creará antes de escribir el texto y “línea” será la constante de línea de pantalla. Un ejemplo de uso de esta función sería “TextOut(0,LCD_LINE5, ‘HELLO WORLD!!!!’)”.

- **“NumOut”**. Mismos argumentos de entrada que “TextOut”, sirve para mostrar números en la interfaz, por ejemplo, “NumOut(0, LCD_LINE2, 300)”.

- **“ClearScreen”**. Es una función sin argumentos que elimina todo el texto de la pantalla de la interfaz gráfica.

4.3.5. Funciones de manejo de ficheros

En la toolbox desarrollada también simulamos el manejo de ficheros de NXC usando, en Matlab, las funciones de gestión de ficheros (fopen, fclose, fwrite, etc.). Se ha utilizado un struct de Matlab que guarda los datos del valor de la referencia al fichero devuelto por la función “fopen”, los bytes del archivo y el nombre y ruta del archivo (Figura 96). Conforme al tamaño del archivo, si se ha abierto para su lectura, este valor es el tamaño total del archivo; pero si el fichero se ha creado, este valor será el tamaño máximo de bytes que se pueden escribir en él. Para realizar todas las operaciones de manejos de ficheros no ha sido necesaria la comunicación con el simulador. También, en algunas de las funciones que usan esta estructura como

parámetro, se ha creado la función privada “isFileNXC”, que comprueba características clave para saber si se trata de la estructura mencionada y se describe en la subsección 4.3.6.

```
file =

      handle: 3
      nombre: 'funciones/ejemplo.txt'
    tamTotal: 1000

fx >>
```

Figura 96. Ejemplo de contenido del “struct” de un fichero

Las funciones que usan esta estructura, devuelven el valor “returnCode”, que indica si se ha producido un error en algún momento. A continuación se describen las funciones de manejo de ficheros creadas:

- **“CreateFile”**. Crea un fichero en el directorio especificado y con un tamaño en bytes establecido, reescribiendo la variable que se le pasa como argumento de entrada para generar el “struct” con todos los datos del fichero (imitando el paso de parámetros por referencia de C); por lo que este parámetro siempre tiene que ser una variable. Un ejemplo para crear un fichero, teniendo la variable “file” creada, puede ser: “CreateFile(‘/directorio/fichero.txt’, 1000, file)”; esto significa que crea el fichero “fichero.txt” en el directorio “/directorio” con un tamaño de 1000 bytes y guarda la referencia del fichero en la variable “file”.

- **“OpenFileRead”**. Abre el un fichero para su lectura; esta función tiene los mismos parámetros que “CreateFile”, pero en este caso la variable que se encarga de asignar el tamaño del fichero devuelve el tamaño total de este pasando una variable por referencia. Un ejemplo de uso de esta función, teniendo las variables “tamTotal” y “file” ya creadas, es: “OpenFileRead (‘/directorio/fichero.txt’, tamTotal, file)”; esto significa que abre el fichero */directorio/fichero* en modo lectura y escribe el tamaño total en la variable “tamTotal” y el struct con los datos del fichero en la variable “file”.

- **“ReadLnString”**. Devuelve la línea del fichero referenciado (abierto previamente con OpenFileRead) en la variable pasada por referencia. Si el valor de retorno devuelto por la función es mayor que 0, el fichero tiene más líneas para leer, pero, si el valor es 0, se ha llegado al final del fichero. Un ejemplo de llamada a esta función es: “returnCode = ReadLnString(file, text)”; donde “file” es nuestra referencia al fichero, “text” es la línea leída de este y “returnCode” nos indicará si hay más líneas o se ha llegado al final del fichero.

- **“WriteLnString”**. Escribe una línea en el fichero referenciado (creado en modo escritura) y devuelve el tamaño de bytes escritos en este en la variable pasada por referencia. Para comprender mejor esto, se muestra el siguiente ejemplo: “WriteLnString(file, ‘Esto es una línea’, bytes); esto significa que se escribe la línea “Esto es una línea” en el fichero referenciado en “file”, devolviendo el número de bytes en la variable “bytes” que se han escrito en este, donde el parámetro “bytes” debe de ser una variable ya creada.

- **“CloseFile”**. Cierra el fichero con el manejador indicado como parámetro (por ejemplo “CloseFile(file)”).

- **“DeleteFile”**. Elimina un archivo con la ruta especificada (por ejemplo “DeleteFile(‘/directorio/fichero.txt’)”). En esta función no es necesaria la referencia al fichero, pero sí es recomendable que, si se ha usado el fichero con un manejador para leer o escribir, se cierre previamente.

4.3.6. Funciones privadas

En la presente subsección se describirán funciones que han servido para el desarrollo de las anteriores, pero no se podrán usar en la toolbox al ser privadas; estas son:

- **“privateSensor”**. Esta función está destinada a extraer los valores de los sensores de luz y táctil para las funciones: “SENSOR_1”, “SENSOR_2”, “SENSOR_3”, “SENSOR_4” y “Sensor”, indicándole el tipo de sensor (luz o táctil) y el manejador del EV3.

- **“isFileNXC”**. En algunas de las funciones que usan la estructura para referenciar a un fichero con las funciones de manejo, se usa esta función privada que comprueba características clave para saber si se trata de la estructura mencionada; la función comprueba: si es un struct, los nombres de cada campo y los tipos de datos que lo conforman.

4.3.7. Otras funciones

En esta subsección se recogen aquellas funciones que no se clasifican en las anteriores subsecciones y son de cierta utilidad para el desarrollo de programas para el robot simulado. A continuación se describen dichas funciones:

- **“CurrentTick”**. Devuelve el tiempo de ejecución de la simulación (tiempo de simulación) en milisegundos. Para poder llamar a esta función, se tiene que declarar de la siguiente forma: “time = CurrentTick()”.

- **“Wait”**. Detiene la ejecución del programa dado un tiempo en milisegundos. Esta función llama internamente a la función “wait” de la clase manejadora de EV3 en Matlab (sección 4.1). Un ejemplo de la función es “Wait(1000)”, esto quiere decir que el programa hará una espera de 1 segundo (en tiempo de simulación).

- **“FreeMemory”.** Libera memoria de una variable. Esta función se limita a asignar un array vacío a una variable, por lo que no es necesaria la comunicación con el simulador. Un ejemplo de uso de esta función es: “var = FreeMemory()”.

- **“Stop”**. Finaliza la ejecución del programa enviando al simulador el código de finalización llamando a la misma función con el mismo nombre (subsección 3.7.5).

4.3.8. Funciones que no pertenecen al formato NXC

En la presente subsección se propone para aquellas funciones que no siguen el formato de NXC debido a que no existen por diversas razones, las cuales se explicarán en cada punto, las cuales son:

- **“SensorAngle”**. Como no existe ningún sensor que devuelva el ángulo girado, NXC no pose ninguna función para este fin, por lo que “SensorAngle” devuelve el ángulo girado del sensor llamando a la función “SensorGyroA” en el simulador (subsección 3.7.3). Para hacer una llamada a la función, la sintaxis es: “angle = SensorAngle()”.

- **“ResetAngle”**. Resetea el contador de ángulos del sensor giroscópico; la función llama a “ResetGyroA” en V-REP (subsección 3.7.3). Para declarar la función simplemente hay que escribir: “ResetAngle()”.

- **“SensorColor”**. Al ser distinto el código de colores detectados del sensor RGB de NXT respecto al de EV3, se ha creado esta función que devuelve el color detectado; la parte de transformación de los datos se ha realizado en Matlab, ya que posee funciones de tratamiento de imágenes. Para realizar el reconocimiento, se han transformado los valores devueltos de la función del script *Funciones* en V-REP con el mismo nombre de RGB [71] a HSV [72], utilizando rangos de valores de matiz, saturación y brillo fijos para cada color. La razón por la que se ha usado esta conversión es debido a que los distintos colores con RGB se logran mediante la mezcla de valores de rojo, verde y azul, lo que supone un trabajo más costoso a la hora de ir seleccionando el rango de colores que queremos identificar; esto no ocurre con el HSV dado

que al ser el matiz del color uno de los valores (H) podemos reconocer el tipo de color que es y mediante los valores de saturación y brillo (SV) podemos excluir tonalidades, por lo que resulta más sencillo poner restricciones de rango para el reconocimiento del color; un ejemplo de este reconocimiento es el que aparece en la Figura 97, si el color estuviese fuera de estos rangos, el algoritmo devolverá el color 0 (color no reconocido). La sintaxis de esta función es: “color = SensorColor()”.

Figura 97. Ejemplo de reconocimiento de color nuestro algoritmo.

- **“StatusLight”**. Se encarga de encender las luces de estado de la interfaz gráfica, dado un color y si parpadean o no. El código de colores va del 0 al 3 y en orden son: verde, ámbar, rojo y gris (luz apagada); para indicar si se quiere que las luces parpadeen, hay que indicarlo en el segundo argumento: 0; para indicar que la luz no va a parpadear y 1; para que parpadee. Un ejemplo para de uso de la función sería: “StatusLight(0,1)”, que encendería una luz verde intermitente.

4.4. Iniciador de la simulación para EV3

Para poder usar toda esta toolbox sin necesidad de preocuparse por la inicialización de las constantes y la conexión con el simulador, se ha creado una función de Matlab que oculta este proceso y, además, permite el uso de logs para facilitar la labor de depuración. También se encarga de manejar las excepciones para que, en caso de error, la conexión con V-REP no se quede abierta y el espacio de trabajo quede limpio. El código se encuentra en el directorio */Fuentes/Matlab/Iniciador de la simulación*. Para poder usar esta función tenemos las siguientes llamadas:

- **“ejecutarCodigoNXC(String script)”**. Ejecuta el programa de NXC con el nombre “script”. También se permite la escritura como comando de Matlab: “ejecutarCodigoNXC script”.

- **“ejecutarCodigoNXC(String script, Bool log)”**. Ejecuta el programa EV3 con el nombre “script” y, si “log” es true o 1, guarda en un fichero de texto los mensajes escritos en la consola de Matlab con el nombre “script".

- **“ejecutarCodigoNXC(String script, Bool log, String fichero)”**. Igual que el anterior, solo que el archivo de log se guarda con el nombre “fichero”.

Para que se pueda ejecutar un programa correctamente, necesitamos tener V-REP abierto con el modelo cargado en la escena; en otro caso, podríamos tener errores. Un ejemplo de ejecución con esta función es la que aparecen en las Figuras 98 y 99, en el que se muestra un mensaje en la consola y no continúa hasta que pulsamos el botón central del robot en la interfaz; luego, se muestra un mensaje en la consola de Matlab y, finalmente, termina el programa. El código puede encontrarse en el directorio */Programas/Ejemplo* con el nombre *holaMundo.m*.

Figura 98. Pantalla del robot durante la ejecución del script “holaMundo”

```
>> ejecutarCodigoEV3 holaMundo
***Creando el cliente de api remota de V-REP...***
Note: always make sure you use the corresponding remoteApi library
(i.e. 32bit Matlab will not work with 64bit remoteApi, and vice-versa)
conexión establecida

***Iniciando la simulación...***

***Creando el cliente de api remota de EV3...***
Note: always make sure you use the corresponding remoteApi library
(i.e. 32bit Matlab will not work with 64bit remoteApi, and vice-versa)
conexión establecida

***Ejecutando el programa...***
--------CONSOLA DEL SCRIPT--------
programa finalizado
------FIN CONSOLA DEL SCRIPT------
***Parando la simulación...***

***Destruyendo el cliente de api remota de EV3...***
El objeto RemoteApi se ha eliminado correctamente

***Destruyendo el cliente de api remota de V-REP...***
El objeto RemoteApi se ha eliminado correctamente

***Limpiando memoria...***
fx >>
```

Figura 99. Traza de ejecución del script

CAPÍTULO 5.
RESULTADOS

En este apartado se van a mostrar ejemplos de programas donde se puede programar el robot, una de ellas son tests para probar el funcionamiento de todas las funciones de la toolbox. También se han desarrollado dos prácticas de la asignatura *Programación de Robots*: seguimiento de líneas basado en PID y localización offline mediante odometría.

5.1. Pruebas de la toolbox

En lo relativo a las pruebas de todas las funciones desarrolladas se han programado una serie de tests que usan todas ellas. Para poder realizarlas correctamente, es necesario cargar la escena llamada escenaTest.ttt en el simulador, situado en el directorio de los ficheros del trabajo */Simulador/Escenas*; estas pruebas son las siguientes:

- **“testInterfaz”**. Prueba los botones haciendo que se pulsen los cuatro botones en secuencia, muestra todos los tipos de luces de estado e, implícitamente, la pantalla.
- **“testBumper”**. Prueba el sensor táctil con las funciones SENSOR_<N>, para todos los puertos.
- **“testGiroscopio”**. Prueba los dos modos del giroscopio con todas sus funciones, haciendo girar el robot.
- **“testColorSensor”**. Prueba los dos modos del sensor de color utilizando sus funciones; para ello, se vale de las dos plantillas del suelo de la escena.
- **“testSonar”**. Prueba el sensor ultrasónico midiendo la distancia entre la pared de la escena y el robot.
- **“testEncoders”**. Prueba el sensor rotacional de los motores, haciendo girar ambas ruedas.
- **“testFicheros”**. Prueba el manejo de ficheros creando un fichero con 10 valores aleatorios; luego, se vuelve a leer el fichero con los 10 valores que se mostrarán en la pantalla del robot y, finalmente, se eliminará el archivo.
- **“testOtras”**. Prueba las funciones que no han sido utilizadas en los demás test, estas funciones son “CurrentTick” y “FreeMemory”.
- **“tests”**. Ejecuta todos los test mencionados anteriormente.

Todo el código de las pruebas se encuentra en el directorio */Programas/Tests*. Además, se añade un vídeo en la carpeta */Videos* en el archivo *tests.mp4* mostrando el funcionamiento de todas estas pruebas.

5.2. Localización offline mediante odometría

En esta práctica, originaria de la asignatura *Programación de Robots*, tendremos que realizar un programa en el que el robot EV3 recorra un cuadrado de manera continua, al menos una vez completa, y tomando muestras de los sensores rotacionales de los motores y del tiempo. Para describir el cuadrado hay que realizar dos tipos de movimiento:

- Para los tramos rectos, se inician los motores A y C con potencia igual a 50 durante un cierto tiempo.

- Para hacer los giros se apaga el motor C durante otro cierto tiempo (dejando el A encendido).

Para que este movimiento ocurra el alumno debe ajustar los valores de tiempo de tramo recto y de giro médiante prueba y error, hasta conseguir que este siga el cuadrado lo mejor posible. El tiempo que tomaremos entre muestra y muestra será de 30 ms, por lo que el paso de simulación escogido será de 25 ms para que tengamos el valor de los sensores rotacionales de los motores actualizados; cuando el programa finaliza, guarda todos los registros en un fichero. El código de este programa se encuentra en el directorio de los archivos del trabajo */Programas/Prácticas/Localización offline/*, concretamente, en el fichero *localizacion_muestras_offline.m*.

Una vez ejecutado el programa anterior, el alumno debe estimar offline en Matlab la posición cartesiana que ha tenido el robot en cada momento de muestreo, usando por ello los datos grabados en el fichero de salida y el modelo cinemático reflejado en la Ecuación 6.

$$\Delta x = \left(\frac{(\Delta rotr * radio + \Delta rotl * radio)}{2}\right) * \cos(\theta_0)$$
$$\Delta y = \left(\frac{(\Delta rotr * radio + \Delta rotl * radio)}{2}\right) * \sin(\theta_0)$$
$$\Delta\theta = \frac{(\Delta rotr * radio + \Delta rotl * radio)}{d}$$

Ecuación 6. Modelo cinemático usado para la localización odométrica.

Donde “rotr” es la rotación de la rueda derecha, “rotl” la rotación de la rueda izquierda, y “radio” es el radio de las ruedas del robot LEGO (su valor es de 0,028 m), x e y son las coordenadas en el plano cartesiano y θ su orientación. La implementación de este modelo cinemático, llamado como función “odometry”, se

encuentra en mismo directorio que *localizacion_muestras_offline.m* en el fichero *odometry.m*.

La función “localizacion_offline” de Matlab, llamada con una matriz “muestras” obtenida al cargar el fichero “.log” como argumento, llama a la función “odometry” con sus parámetros pertinentes para calcular la nueva pose del robot, y acto seguido la representa en un gráfico. La pose antigua se iguala a la nueva para la siguiente llamada a “odometry”. Esta función se encuentra en el fichero “localizacion_offline.m” en el mismo directorio que las anteriores. También, se adjunta un vídeo en el directorio de los archivos del trabajo */Videos/localización offline.mp4* que muestra todo este proceso; además, en el directorio de esta práctica se encuentra el fichero *“muestras_video.log”* con las muestras generadas por la ejecución del vídeo. Los resultados pueden verse en la Figura 100, donde los puntos azules son la posición del robot en instantes determinados y las líneas rojas la orientación del robot en ese instante.

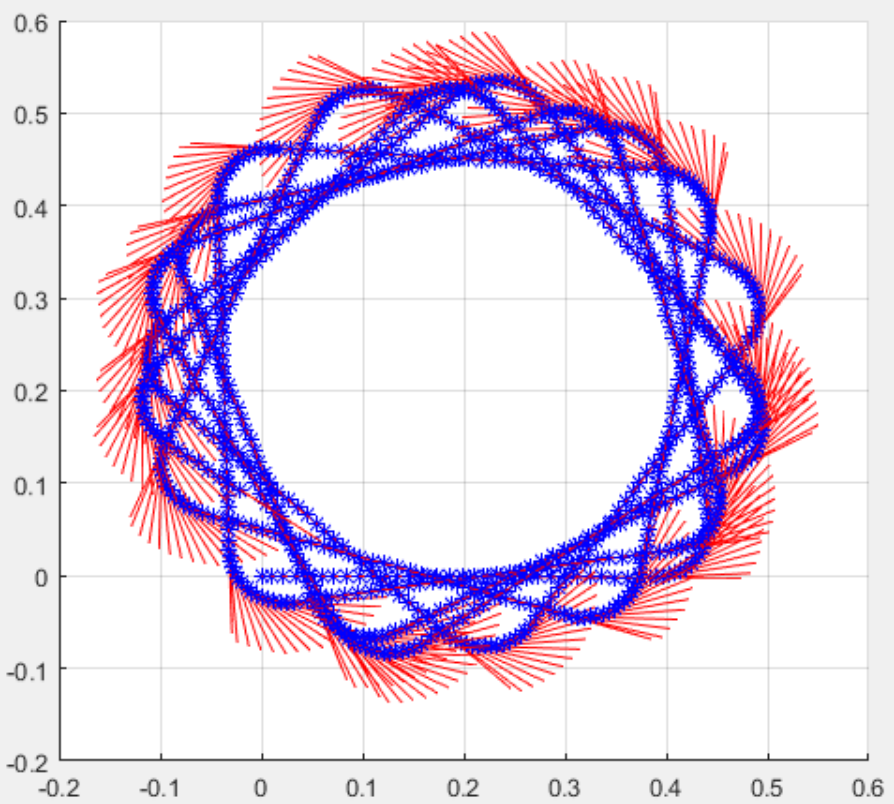

Figura 100. Resultado de las muestras tomadas con el programa "localizacion_muestras_offline" aplicando la localización por odometría

Las principales incidencias para lograr el objetivo de esta práctica fueron las siguientes:

- Coordinar las iteraciones del algoritmo de toma de muestras (“localización_muestras_offline”) para que el robot gire aproximadamente 90 grados,

cuya solución ha sido ajustarlo mediante ensayo y error y la aplicación de un paso de simulación óptimo.

- Buscar un paso de simulación óptimo que se ajuste al problema del punto anterior, que consiga que los sensores rotacionales se actualicen en el menor tiempo posible y que no ralentice demasiado la simulación. El paso de simulación que se ha llegado a establecer es de 25 ms ya que en nuestro algoritmo de recogida de muestras establecemos entre iteración una espera de 30 ms, lo que provoca que cada vez que recojamos datos de los sensores de rotación estos están actualizados.

- La creación del fichero con las muestras, debido al desconocimiento inicial de este tipo de escritura de ficheros (similar a C).

5.3. Seguimiento de líneas basado en PID

En esta práctica, proveniente de la misma asignatura que la anterior, vamos a desarrollar un algoritmo de seguimiento de líneas basado en el controlador PID [73]; este controlador es un mecanismo de control aplicado en bucle cerrado que calcula el error entre un valor medido y un valor deseado. En nuestro robot EV3 seguiremos este esquema en el código:

- Definición de constantes.
- Cuerpo del programa.
    - Declaración de variables.
    - Calibración de la luz del sensor.
    - Algoritmo de control en un bucle.
        - Leer el sensor.
        - Ajuste de la lectura a los niveles de luz del sensor.
        - Actualización de la actuación del controlador.
        - Ajuste de la actuación para prevenir el efecto WindUp, es decir, evitamos la saturación de los actuadores para que el robot no tarde en reaccionar.
        - Ajuste de la potencia de los motores con Saturación.

El código implementado y comentado, se encuentra en el directorio de los archivos de la memoria */Programas/Practicas/PID* en el fichero PID.m.

Para ejecutar este programa correctamente se necesita un camino que se puede construir en el simulador [74] sobre un plano de color blanco o cargando la escena sigueLineas.ttt (esta escena suele tardar en cargar) ajustando un paso de simulación de 25 ms, debido a que necesitamos una frecuencia de refresco de los sensores y

una respuesta a la actuación mucho más rápida; la escena se encuentra en el directorio de los ficheros del trabajo */Simulador/Escenas*. Además, se muestra un vídeo de su funcionamiento en la carpeta */Videos* con el nombre *PID.mp4*.

Las principales incidencias para la realización de la práctica fueron, principalmente:

- En un principio, el robot giraba en sentido contrario al que debería al detectar los valores de blanco y negro, esto se debió a que los valores constantes de blanco y negro estaban definidos erróneamente y, por consiguiente, la calibración no se realizaba de manera correcta, por lo que se procedió a intercambiar estos valores constantes.

- En algunos momentos, el robot no seguía correctamente la línea debido a que los valores del PID no estaban ajustados correctamente, para corregir este problema se han establecido los valores mediante ensayo y error hasta que el robot reaccionara correctamente.

- Debido a la misma incidencia explicada en la práctica anterior, había que buscar un paso de simulación para que el sensor de luz actualice el valor en el menor tiempo posible y la actuación de los motores sea rápida. En este caso se ha establecido un tiempo de simulación de 25 ms ya que en nuestro algoritmo de seguimiento de líneas, entre iteración e iteración, se realizado una espera de 30 ms que consigue que cada vez que se tome un dato del sensor de luz, estos sean siempre actualizados y tengamos una actuación acorde con el paso de simulación.

CAPÍTULO 6.
CONCLUSIONES Y TRABAJOS FUTUROS

Este capítulo consta de dos secciones:

- **Conclusiones**. Se explicaran los objetivos alcanzados propuestos en el capítulo 1 y las conclusiones que se han llegado durante el cumplimiento de los mismos.

- **Trabajos futuros**. Se enumerarán los posibles trabajos que se podrían realizar a partir de este dando una idea general de como ampliarlo.

6.1. Conclusiones

En primer lugar, vamos a recapitular los objetivos recogidos en el capítulo 1 y su cumplimiento:

- Se ha logrado el modelo en 3 dimensiones del robot con una herramienta CAD gracias al software LeoCAD de manera sencilla, su adaptación en V-REP añadiendo formas puras, su disposición en la jerarquía de la escena y creación de la interfaz gráfica de la pantalla, luces y botones del bloque central (Capítulo 3).

- En lo referente a la programación de los sensores y actuadores del simulador conforme a las especificaciones reales del robot (secciones 3.3, 3.4 y subsecciones 3.7.2 y 3.7.3), se ha conseguido que el simulador devuelva los valores en magnitudes del robot real (sensor ultrasónico en cm, sensor rotacional en grados, modo de luz reflejada del sensor de luz escalado de 0 a 100, etc.) con las herramientas de V-REP y su API nativa en el lenguaje Lua. No obstante, en lo referente a la generación de los errores de medición, no se ha logrado como se esperaba; a excepción del modo luz reflejada en el sensor de luz/color, los errores de medición del resto de sensores se han generado con valores hipotéticos de desviación típica para la generación de estos en base a las especificaciones, la razón de esta solución ha sido el desconocimiento o la ausencia de modelos experimentales para obtener dichas desviaciones.

- Se ha alcanzado el objetivo de la comunicación de los sensores y actuadores del robot modelado en el simulador con Matlab mediante la API remota de V-REP para dicho lenguaje implementando un conjunto de funciones en el simulador (script *Funciones*) para que, con las clases manejadoras diseñadas en este trabajo, se llamen desde Matlab.

- En cuanto a la programación de una toolbox de Matlab para la comunicación transparente con el simulador, se ha propuesto una implementación basada, en su mayoría, en funciones del lenguaje NXC apoyándose en las clases manejadoras creadas en este trabajo, cuyo objetivo se ha alcanzado con éxito.

Cumplidos en mayor medida estos objetivos llegamos a las siguientes conclusiones:

1. La simulación de robots no es un trabajo sencillo. Las razones se comentan a continuación:

    - Al principio del trabajo, el simulador no disponía de la función genérica para Matlab en su API remota (versiones anteriores a la 3.3.0 de V-REP) que nos permite implementar nuestras propias funciones en un script del simulador; esto provocaba la realización de versiones no completas utilizando señales y funciones propias de la API remota, lo que provocaba que se tuvieran que hacer sincronizaciones a bajo nivel de las señales y utilizar referencias a objetos del simulador en Matlab.

    - El problema de los pasos de simulación cada vez que desarrollemos un escenario de prácticas, ya que si establecemos un paso de simulación muy corto (de, por ejemplo, 1 ms), la simulación nos irá lenta, pero tendremos un periodo de muestreo de los sensores y una respuesta a la actuación más frecuente; el caso contrario nos ocurrirá si establecemos un paso de simulación más largo.

    - La adaptación del modelo al simulador, ya sean añadir el peso para corregir el temblor de este en simulación, el diseño de formas puras para una simulación más eficiente, etc.

    - La simulación de los errores de medición debido a que de manera nativa la simulación de sensores y actuadores no poseen error.

2. El uso de un programa de control de versiones (Git) y el almacenamiento en dispositivo físico y en la nube de un repositorio privado (GitHub) son herramientas de utilidad que han servido a la hora de realizar el trabajo en tareas de copia de seguridad y migración del proyecto en distintos equipos, por lo que el uso de estos programas son muy importantes en el desarrollo de proyectos, ya que lo protegen de los distintos problemas que puedan ocurrir durante el transcurso de este.

3. El descubrimiento de LeoCAD para el modelado en 3D del robot supuso un ahorro de tiempo considerable para la construcción de este, además de la

funcionalidad que posee al exportar al formato ".obj", frente a la primera opción que se propuso, la cual fue el diseño mediante AutoCAD que suponía un diseño de piezas desde cero, lo que suponía un gasto de tiempo excesivo para la tarea del modelado.

4. La elección del lenguaje NXC para la sintaxis de funciones que controlan al robot desde Matlab es debida al uso de dicho lenguaje en la realización de prácticas en la asignatura *Programación de Robots* consiguiendo su abstracción, por lo que el alumno o docente desde casa puede tener el algoritmo de la práctica diseñada mediante la solución propuesta en este trabajo y adaptar el programa a dicho lenguaje.

6.2. Trabajos futuros

A partir de este trabajo, se pueden añadir las siguientes mejoras o trabajos futuros:

- Usar técnicas de visión por computador para el reconocimiento de colores en el sensor de color.

- Obtener desviaciones típicas mediante experimentación de los sensores simulados para obtener errores de medición generados por distribución normal.

- Obtener el peso del robot real para añadirlo al modelo en V-REP.

- Creación de una nueva toolbox con funciones de lenguajes desarrollados para robots EV3 reales sustituyendo a las funciones de NXC.

- Adaptar el modelo ya desarrollado para poder simular múltiples robots LEGO EV3 y crear sistemas multi-robot utilizando sufijos numerados en V-REP para identificar cada uno de ellos.

- Adaptar la toolbox para crear hilos concurrentes en el robot utilizando "workers" de Matlab con la *Parallel Processing Toolbox*.

ANEXOS

Anexo A. Manual de usuario

En este manual se explica cómo utilizar el modelo de EV3 en el simulador V-REP y la toolbox de Matlab para controlarlo. En las primeras secciones se enumeran los pasos básicos para la instalación. A continuación, en las siguientes secciones, se describirá cómo poner en marcha una simulación, los modelos de EV3 y escenas para V-REP disponibles en el paquete de instalación, y como añadir vistas flotantes para las cámaras proporcionadas en el modelo.

A.1. Instalación de V-REP

Para poder simular nuestro robot, se utiliza el entorno de simulación V-REP. En primer lugar hemos de instalar V-REP en nuestro equipo. La versión de V-REP usada en nuestro trabajo es la 3.3.0 para Windows y Linux, ya que es la versión estable que los equipos han utilizado. Para ello basta con instalar los paquetes que aparecen en su web:

http://www.coppeliarobotics.com/downloads.html

Con ello, nos descargaremos el archivo de instalación al ordenador para posteriormente extraerlo (Linux) o instalarlo (Windows). Una vez concluida la instalación ya podemos empezar a utilizar el entorno, cuya apariencia se muestra en la Figura 101.

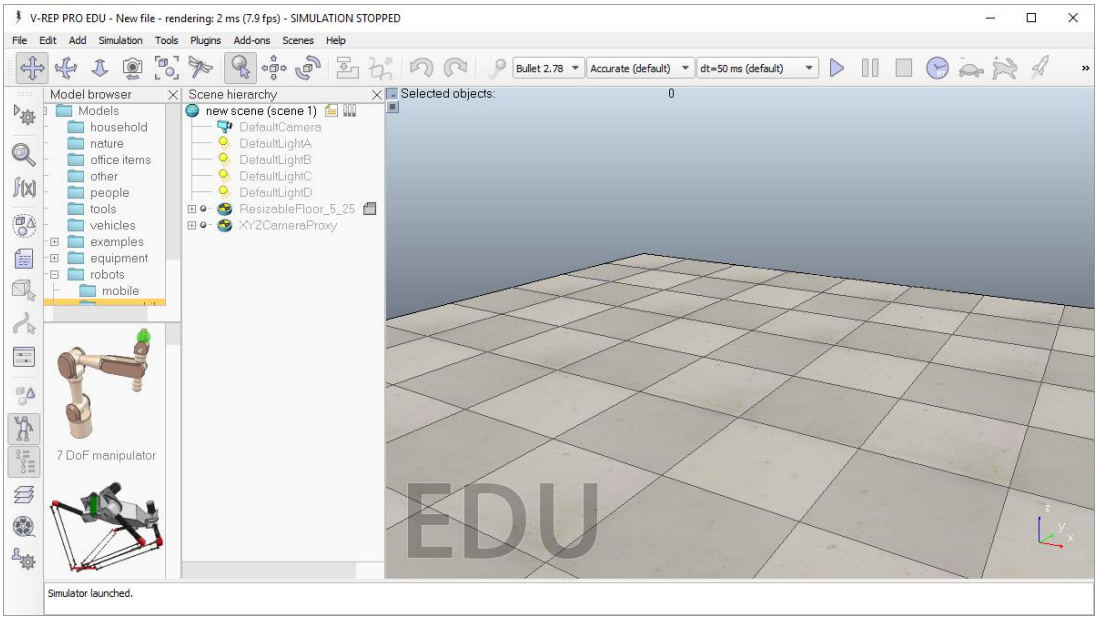

Figura 101. Interfaz del entorno de simulación de V-REP

A.2. Instalación de Matlab

La versión de Matlab que se ha usado para la realización del trabajo es la 2016a. Para poder obtener Matlab, hace falta adquirir una licencia, la cual se puede comprar desde su sitio web; no obstante, la Universidad de Málaga puede distribuir licencias para estudiantes y docentes de manera gratuita desde el Servicio Central de Informática en la siguiente dirección:

https://software.uma.es/index.php?option=com_content&view=article&id=412:matlab-estudiante&catid=198&Itemid=3070&showall=&limitstart=8

Para poder instalar Matlab, basta con seguir los pasos que aparecen en dicho enlace. Una vez instalado y con la licencia activada, ya podemos utilizar Matlab, cuya interfaz aparece en la Figura 102.

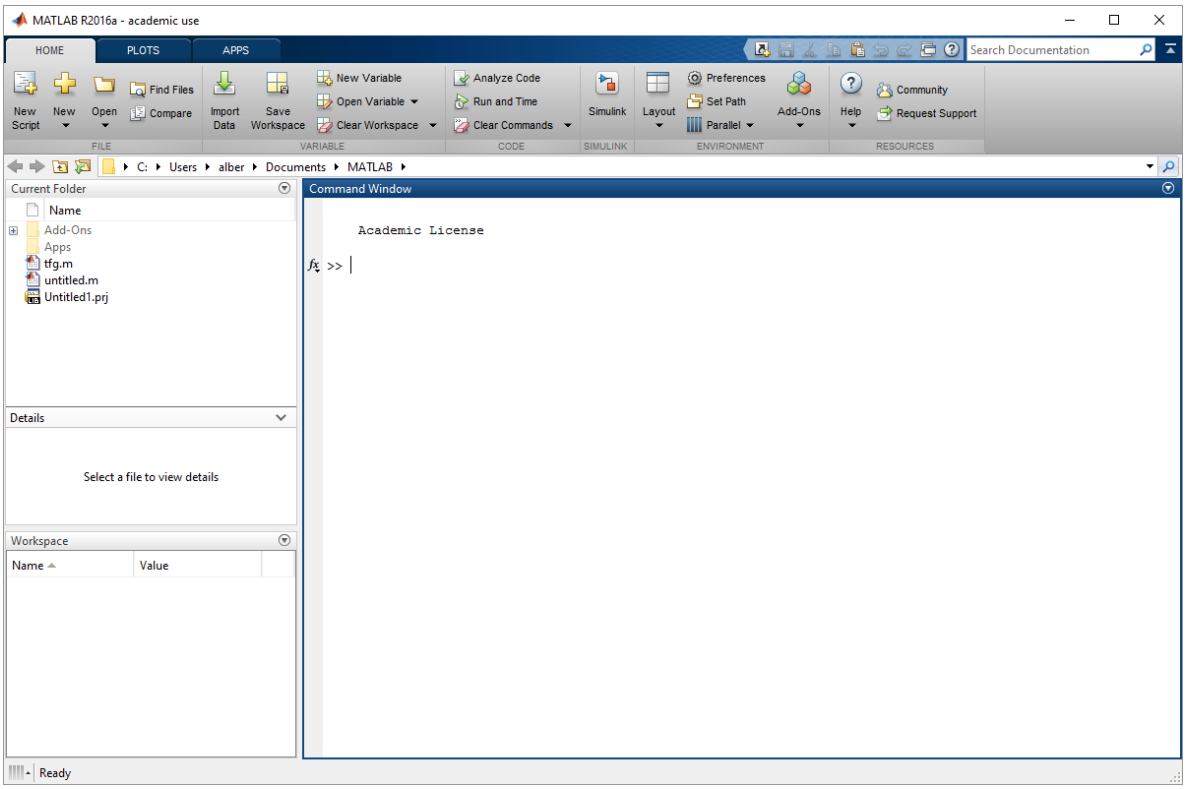

Figura 102. Interfaz de Matlab

A.3. Instalación de la toolbox en Matlab

Para poder instalar la toolbox desarrollada, para todos los sistemas operativos (Windows, Linux y, previsiblemente, MacOS), hay que seguir los siguientes pasos:

1. Descargar el archivo EV-R3P_1.3.1.zip del siguiente enlace: https://drive.google.com/open?id=0B2mnuwTFyL-7ZkNFa1JlcmpXeWM

O del repositorio de público d GitHub:
https://github.com/albmardom/EV-R3P
O en el directorio */Paquete de instalación* del CD adjunto con la presente memoria.

2. Al descomprimir el archivo, deben encontrarse los siguientes directorios (Figura 103):

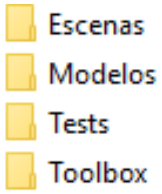

Figura 103. Directorios del paquete de instalación

3. Instalar la toolbox en Matlab; para ello, abriremos previamente Matlab y buscaremos el archivo *LEGO EV3 V-REP toolbox x64 (NXC versión).mltbx*, para procesadores x64, o *LEGO EV3 V-REP toolbox x86 (NXC versión).mltbx*, para procesadores x86, en el directorio descomprimido, concretamente, en la carpeta *Toolbox*. Al hacer doble clic nos aparecerá un cuadro de diálogo para confirmar que vamos a instalarla (Figura 104), y pulsaremos el botón “install”.

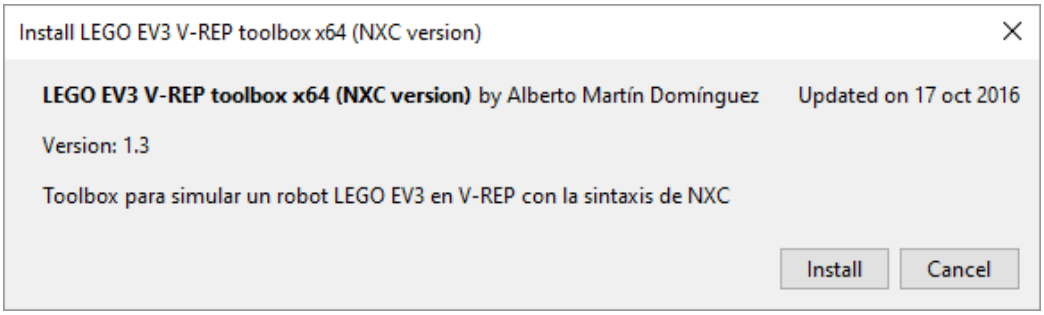

Figura 104. Cuadro de diálogo de instalación de toolbox

Para comprobar si se ha instalado correctamente la toolbox, escribir en la consola de Matlab un comando de ayuda con algunas de las funciones como argumento, como, por ejemplo: “help OnFwd” (Figura 105).

```
>> help OnFwd
 OnFwd proporciona una velocidad al motor indicado
    OnFwd(motor, velocity) activa el motor "motor" a una velocidad
    "velocity" indicada entre -100 y 100 hacia adelante (siendo el porcentaje de velocidad a
    partir de la velocidad máxima). Las constantes de motores que se pueden
    usar son las siguientes:

    * OUT_A: puerto A de actuadores (motor A)
    * OUT_C: puerto C de actuadores (motor C)
    * OUT_AC: puertos A y C de actuadores (ambos motores)

    See also OnRev, Off, MotorRotationCount, ResetRotationCount

fx >>
```

Figura 105. Comando "help" a la función "OnFwd"

En el caso de que no funcione, podemos abrir con un programa de extracción de archivos el fichero de toolbox (".mltbx"), extrayendo los códigos de este del directorio *fsroot*. Luego, añadiremos al path de Matlab el directorio donde hemos descomprimido los ficheros fuente, además de la subcarpeta Funciones, pero, sin añadir la subcarpeta *private* (Figura 106).

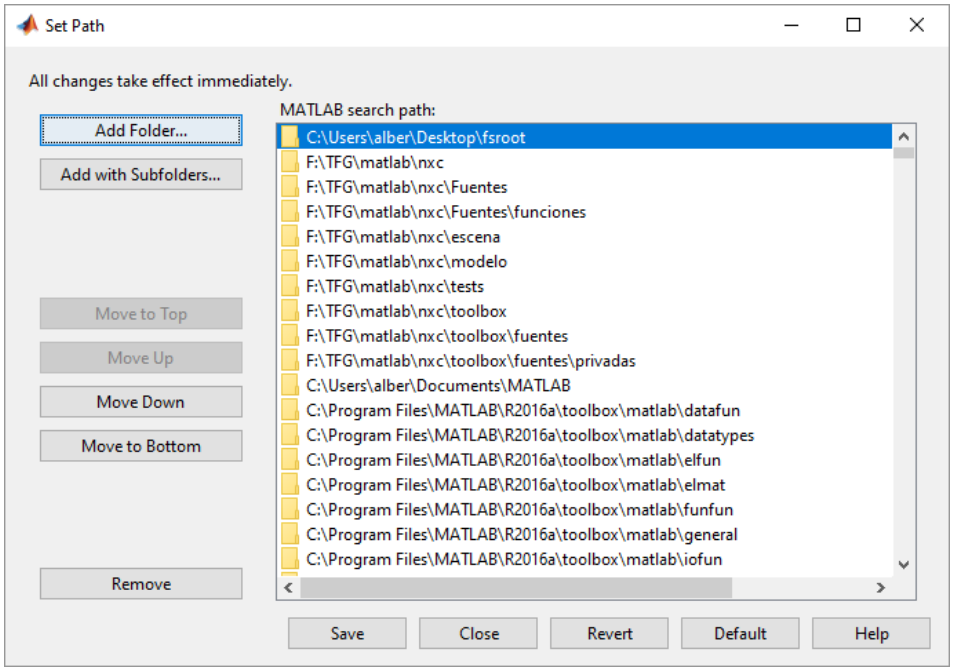

Figura 106. Path de Matlab con la carpeta importada de la toolbox

Con esto, ya solo falta probar el comando "help" mencionado anteriormente.

A.4. Carga de los modelos de EV3 en V-REP

En el simulador, para cargar el modelo solo tendremos que irnos, en la barra de menús, a "File → Load model…", y buscar, en el directorio donde se han extraído los ficheros descargados, en la carpeta *Modelo*, los dos modelos disponibles de EV3:

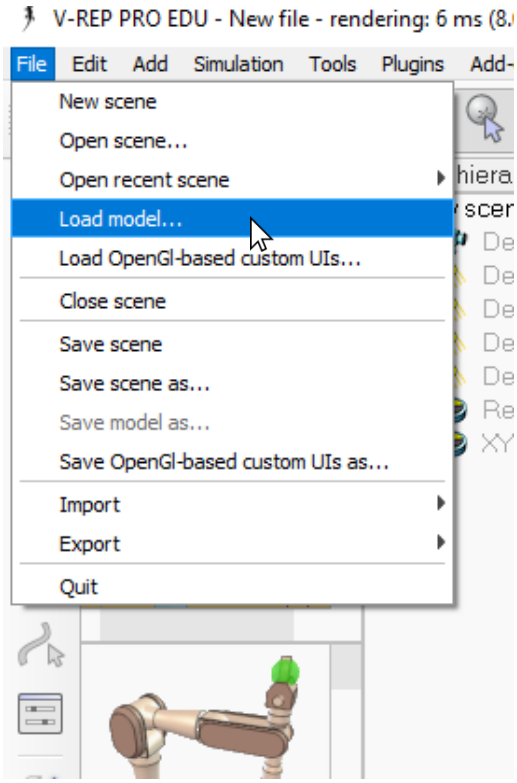

Figura 107. Situación de la opción "Load model…" en V-REP

- **"LEGO_EV3_GROUP.ttm"**. Este modelo es el más detallado, pero, por el contrario, es el más pesado para renderizar; no recomendable para equipos de bajo rendimiento.

- **"LEGO_EV3_MERGE.ttm"**. Este modelo es menos detallado que el anterior, pero en cambio es más eficiente en la renderización; recomendado para ordenadores que no puedan utilizar fluidamente el modelo anterior.

Si se quiere que el modelo cargado aparezca en el Explorador de Modelos del simulador, basta con guardar el modelo en "File → Save model as…" (teniendo el objeto "LEGO_EV3" seleccionado en la escena), en el siguiente directorio: *<DIRECTORIO_INSTALACIÓN_V-REP>/Models/robots/mobile*.

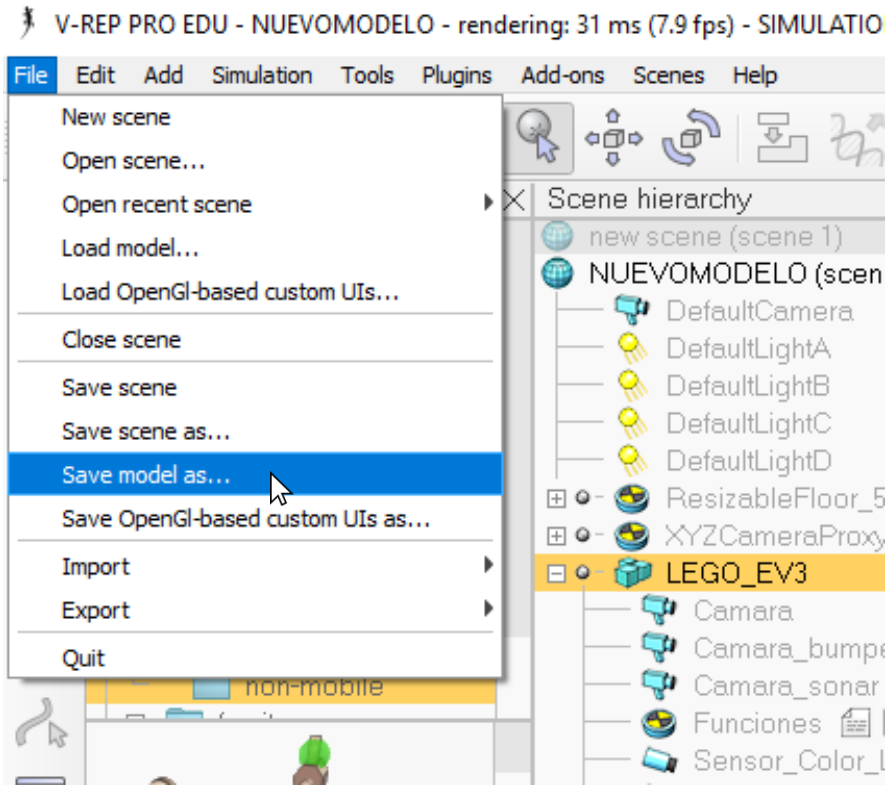

Figura 108. Situación de la opción "Save model as…" con el objeto "LEGO_EV3" seleccionado en V-REP

Hecho esto, los modelos deberían aparecer en el Explorador de Modelos (Figura 109), situado a la derecha de la pantalla.

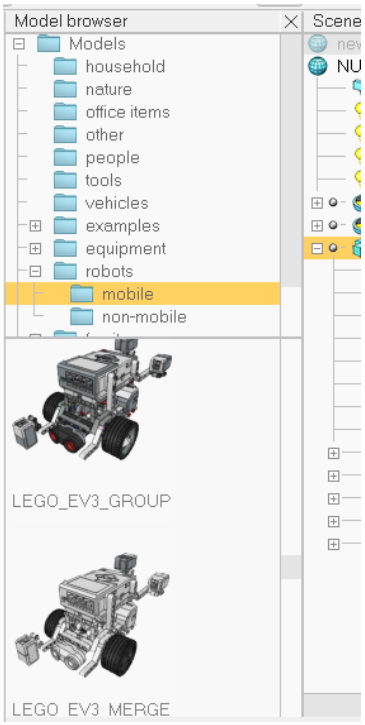

Figura 109. Explorador de modelos de V-REP con los dos tipos de LEGO EV3

Por último, siempre que se cargue el modelo (o una escena que lo contenga), se recomienda iniciar la simulación y luego detenerla, esto es debido a que la simulación tarda mucho al iniciarse la primera vez y, al usar las funciones y los objetos manejadores, se produciría errores en Matlab.

A.5. Vistas

A ambos modelos se les ha dotado de cámaras para ayudar al desarrollo de programas. Podemos obtener imágenes de ellas si añadimos vistas flotantes; estas se añaden haciendo clic derecho en la escena y luego "add → Floating view" (Figura 110).

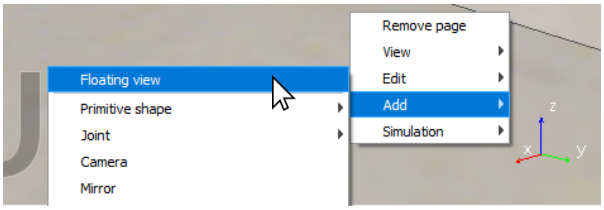

Figura 110. Lugar para añadir vistas flotantes en V-REP

Ahora, haciendo clic derecho dentro de la vista flotante, seleccionamos "view → View selector…" y ahí seleccionaremos la cámara que queramos ver.

Figura 111. Situación del menú "View selector…" en V-REP para las vistas flotantes

Este procedimiento también sirve para ver las imágenes del sensor de visión, como, por ejemplo, los modos del sensor de color del EV3; en la Figura 112 podemos ver todas las cámaras del robot como vistas flotantes.

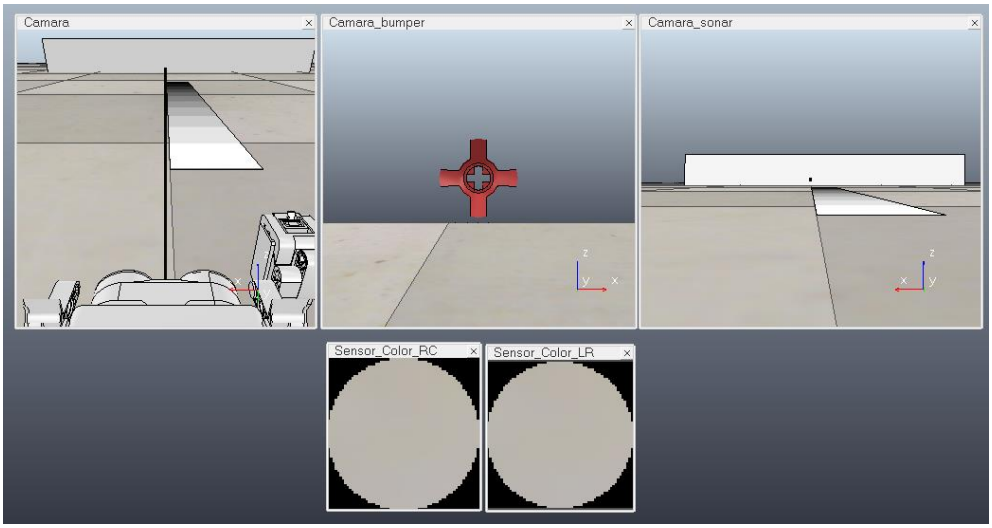

Figura 112. Vistas de las cámaras y sensores de visión en vistas flotantes

A.6. Escenas

Además, se proporcionan escenas, que pueden servir de apoyo para el alumno y personal docente a la hora de desarrollar las prácticas; estas son:

- **"escenaTest.ttt"**. Esta escena está indicada para probar toda la toolbox mediante los test que se encuentran en el directorio con su mismo nombre del paquete de instalación proporcionado.

- **"sigueLineas.ttt"**. Contiene caminos ya trazados para diseñar un algoritmo de seguimiento de líneas para el robot (esta escena suele tardar en cargar).

- **"localizacionOffline.ttt"**. En esta escena, el robot ya está posicionado para realizar prácticas de localización.

A.7. Ejecutar un script

Para ejecutar un script ya programado, se dispone de la función "ejecutarCodigoNXC" que se explicará en el anexo posterior (Anexo B) o aplicando el comando "help" en Matlab para dicha función.

Anexo B. Manual del desarrollador

En el presente anexo se listarán y clasificarán todas las funciones disponibles para la toolbox, así como los objetos manejadores, scripts y la función para iniciar una simulación. Para saber cómo se utiliza cada función, estas disponen de ayuda escribiendo el comando de Matlab "help" más la función, script o clase a utilizar; por ejemplo: "help OnFwd".

B.1. Funciones de control del robot

En esta sección del anexo se listarán, por categorías, todas las funciones que la toolbox posee para el control del robot; estas son:

- Funciones de actuadores:
  - Off
  - OnFwd
  - OnRev
- Funciones de inicio de sensores:
  - SetSensorHtGyro
  - SetSensorLight
  - SetSensorTouch
  - SetSensorUltrasonic
- Funciones de sensores (entre paréntesis la función de inicio de sensor que se tiene que utilizar previamente para definir el puerto de entrada que se va a utilizar para este):
  - MotorRotationCount
  - SENSOR_1 (SetSensorLight o SetSensorTouch)
  - SENSOR_2 (SetSensorLight o SetSensorTouch)
  - SENSOR_3 (SetSensorLight o SetSensorTouch)
  - SENSOR_4 (SetSensorLight o SetSensorTouch)
  - Sensor (SetSensorLight o SetSensorTouch)
  - SensorHtGyro (SetSensorHtGyro)
  - SensorUS (SetSensorUltrasonic)
  - ResetRotationCount
- Funciones de la interfaz gráfica:
  - ButtonPressed
  - ClearScreen
  - NumOut
  - TextOut
- Funciones de manejo de ficheros:
  - CloseFile
  - CreateFile
  - DeleteFile

  - OpenFileRead
  - ReadLnString
  - WriteLnString
- Otras funciones:
  - CurrentTick
  - FreeMremory
  - Stop
  - Wait
- Funciones que no pertenecen al repertorio de funciones de NXC (entre paréntesis una breve descripción de esta):
  - ResetAngle (reseteo del modo de conteo de ángulos en el sensor giroscópico)
  - SensorAngle (devuelve el ángulo del modo conteo de ángulos en el sensor giroscópico)
  - SensorColor (devuelve el código de color leído por el sensor de color)
  - StatusLight (apaga o enciende las luces de estado de la interfaz con el color especificado y si estas parpadean)
- Funciones privadas:
  - privateSensor
  - isFileNXC

B.2. Clases manejadoras del cliente

Estos objetos se encargan de la comunicación y la ejecución de funciones en V-REP para el robot y el control de la simulación; pueden ser usadas si se quiere desarrollar otra toolbox con otro tipo de estructura de funciones o ampliar la desarrollada. En la Figura 113 se detalla la estructura de las clases que implementan estos objetos y su contenido (atributos y métodos). Para más información de cada función, utilizar el comando “help” o generar toda la documentación del objeto con “doc”, por ejemplo: “doc ev3RemoteApi”.

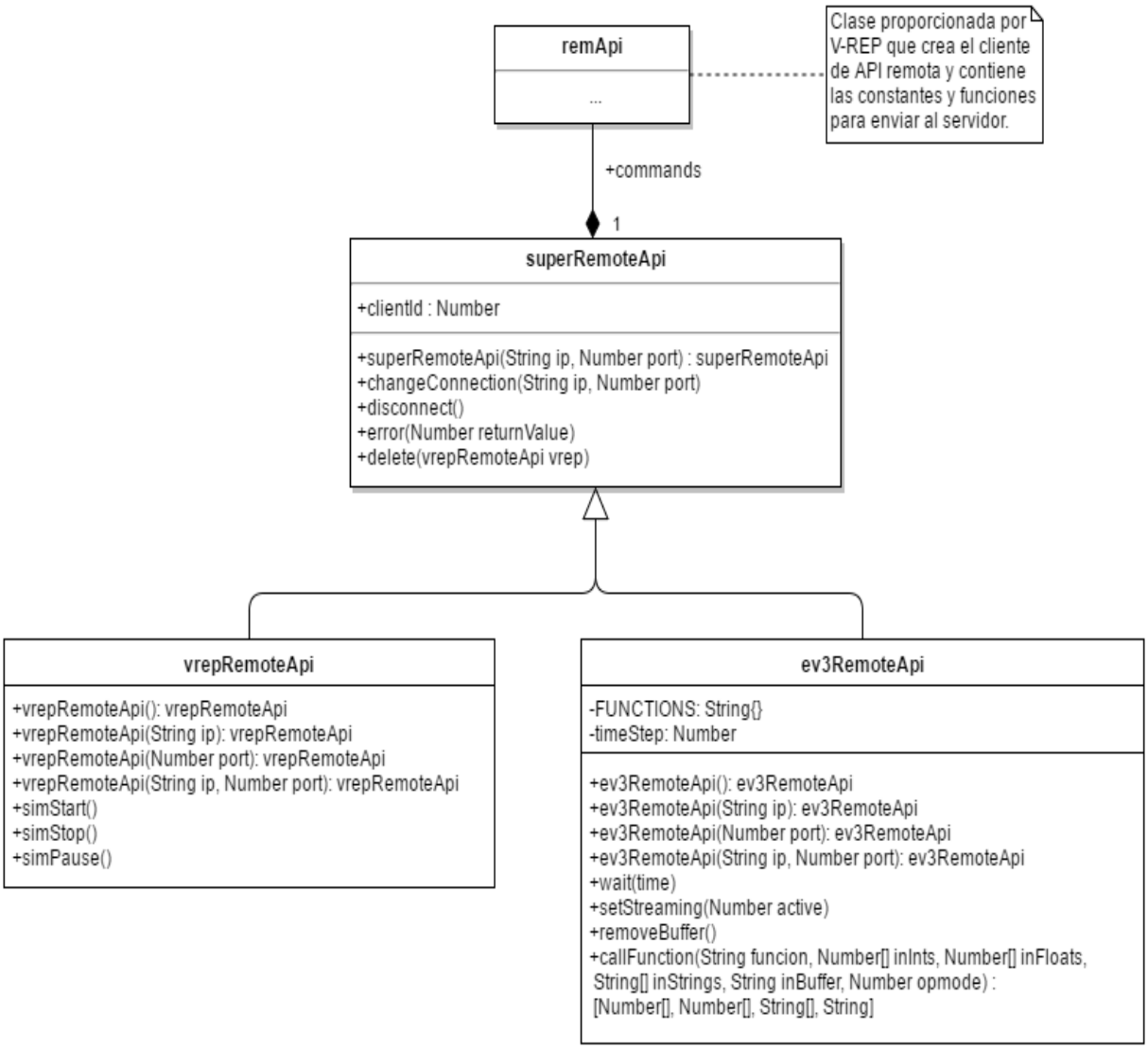

Figura 113. Diagrama de clases de los objetos manejadores de la toolbox

Si se quiere crear una instancia de “vrepRemoteApi”, el simulador debe de estar iniciado; por otro lado, si se desea crear una instancia de “ev3RemoteApi”, además de tener V-REP iniciado, el robot tiene que estar cargado en la escena y la simulación iniciada.

B.3. Función EjecutarCodigoNXC

Para poder usar toda esta toolbox sin necesidad de preocuparse por la inicialización de las constantes y la conexión con el simulador, se ha creado una función que oculta este proceso y, además, permite el uso de logs para facilitar la labor de depuración. También se encarga de manejar las excepciones para que, en caso de error, la conexión con V-REP no se quede abierta y el espacio de trabajo quede limpio. Para poder usar esta función tenemos las siguientes llamadas:

- **“ejecutarCodigoNXC(String script)”**. Ejecuta el programa de NXC con el nombre “script”. También, se permite la escritura como comando de Matlab: “ejecutarCodigoEV3 script”.

- **“ejecutarCodigoNXC(String script, Bool log)”**. Ejecuta el programa de NXC con el nombre “script” y, si “log” es true o 1, guarda en un fichero de texto los mensajes escritos en la consola de Matlab con el nombre “script".

- **“ejecutarCodigoNXC(String script, Bool log, String fichero)”**. Igual que el anterior, solo que, el archivo de log se guarda con el nombre “fichero”.

REFERENCIAS

1. **Wikipedia.** Robótica. [En línea] 1 de Junio de 2016. www.wikipedia.org.
2. **LEGO.** LEGO.com. [En línea] 2016. http://www.lego.com.
3. **LEGO Mindstorms.** Mindstorms LEGO.com. [En línea] 2016. http://mindstorms.lego.com.
4. **Coppelia Robotics V-REP.** Coppelia Robotics v-rep: Create. Compose. Simulate. Any Robot. [En línea] 2016. [Citado el: 8 de Junio de 2016.] http://www.coppeliarobotics.com/.
5. **Matworks Matlab.** MATLAB - El lenguaje del cálculo técnico - Mathworks españa. [En línea] 2016. [Citado el: 8 de Junio de 2016.] http://es.mathworks.com/products/matlab/.
6. **Bricxcc.** NBC - NeXT Byte Codes, Not eXactly C, and SuperPro C. [En línea] 2016. http://bricxcc.sourceforge.net/nbc/.
7. **LeoCAD.** LeoCAD. [En línea] 2016. [Citado el: 9 de Junio de 2016.] http://www.leocad.org.
8. **Lua.** The Programming Languaje Lua. [En línea] 2016. https://www.lua.org/.
9. **Wikipedia.** Desarrollo en cascada - Wikipedia, la enciclopedia libre. [En línea] 2016. https://es.wikipedia.org/wiki/Desarrollo_en_cascada.
10. **Git.** Git. [En línea] 2016. https://git-scm.com/.
11. **GitHub.** How people build software · GitHub. [En línea] 2016. https://github.com/.
12. **Coppelia Robotics V-REP.** Remote API Functions (Matlab). [En línea] 2016. http://www.coppeliarobotics.com/helpFiles/en/remoteApiFunctionsMatlab.htm.
13. **Wikipedia.** GNU/Linux - Wikipedia, la enciclopedia libre. [En línea] 2016. https://es.wikipedia.org/wiki/GNU/Linux.
14. **Microsoft.** Windows | Sitio oficial de sistemas operativos, portátiles, PCs, tabletas y mucho más para Microsoft Windows 10. [En línea] 2016. https://www.microsoft.com/es-es/windows.
15. **Linux Mint.** Main Page - Linux Mint. [En línea] 2016. https://www.linuxmint.com/index.php.
16. **LDraw.** LDraw.org - Home. [En línea] 2016. http://www.ldraw.org/.
17. **Wikipedia.** Programación por capas. [En línea] 2015. https://es.wikipedia.org/wiki/Programaci%C3%B3n_por_capas.
18. **MIT.** MIT - Massachusetts Institute of Technology. [En línea] 2016. http://web.mit.edu/.
19. —. Logo Foundation. [En línea] 2016. http://el.media.mit.edu/logo-foundation/.
20. **Mazzari, Vanessa.** NXT-G: the development environment supplied with Lego Mindstorms, NXT-G. *The blog Génération Robots.* [En línea] 26 de Septiembre de 2015. http://www.generationrobots.com/blog/en/2015/09/nxt-g-the-development-environment-supplied-with-lego-mindstorms-nxt-g/.

21. **LeJOS.** LeJOS, Java for Lego Mindstorms. [En línea] 2016. http://www.lejos.org/index.php.
22. **Enchanting.** Enchanting : Enchanting : Enchanting. [En línea] 2016. http://enchanting.robotclub.ab.ca/tiki-index.php.
23. **ROBOTC.** ROBOTC.net :: Home of the best robot programming language for Educational Robotics. Made for NXT programming and VEX programming. [En línea] 2016. http://www.robotc.net/.
24. **Mathworks.** LEGO MINDSTORMS NXT Support from MATLAB - Hardware Support - MathWorks España. [En línea] http://es.mathworks.com/hardware-support/lego-mindstorms-matlab.html.
25. **National instruments.** Módulo de LabVIEW para LEGO® MINDSTORMS® - National Instruments. [En línea] 2016. http://sine.ni.com/nips/cds/view/p/lang/es/nid/212785.
26. **MonoBrick.** MonoBrick.DK | Home of MonoBrick. [En línea] 2016. http://sine.ni.com/nips/cds/view/p/lang/es/nid/212785.
27. **MathWorks.** LEGO MINDSTORMS EV3 Support from MATLAB - Hardware Support - MathWorks España. [En línea] 2016. http://es.mathworks.com/hardware-support/lego-mindstorms-ev3-matlab.html.
28. **The Player Project.** Player Project. [En línea] 2014. http://playerstage.sourceforge.net/.
29. **University of Southern California.** [En línea] 2016. http://www.usc.edu/.
30. **Wikipedia.** C (lenguaje de programación). [En línea] 2016. https://es.wikipedia.org/wiki/C_(lenguaje_de_programaci%C3%B3n).
31. —. C++. [En línea] https://es.wikipedia.org/wiki/C%2B%2B.
32. **Microsoft.** Welcome to Robotics Developer Studio. [En línea] 2012. https://msdn.microsoft.com/en-us/library/bb648760.aspx.
33. —. C#. [En línea] 2016. https://msdn.microsoft.com/es-es/library/kx37x362.aspx.
34. —. Visual Basic. [En línea] 2016. https://msdn.microsoft.com/es-es/library/2x7h1hfk.aspx.
35. **anyKode.** Marilou: the universal mechatronic software. [En línea] 2016. http://www.anykode.com/marilou.php.
36. **Coppelia Robotics V-REP.** V-REP User Manual. [En línea] 2016. http://www.coppeliarobotics.com/helpFiles/.
37. —. User interface. [En línea] 2016. http://www.coppeliarobotics.com/helpFiles/en/userInterface.htm.
38. **Autodesk.** AutoCAD para Mac y Windows | AutoCAD | Autodesk. [En línea] 2016. http://www.autodesk.es/products/autocad/overview.
39. **Blender.** blender.org - Home of the Blender project - Free and Open 3D Creation Software. [En línea] 2016. https://www.blender.org/.
40. **Autodesk.** 3ds Max | Software de modelado y renderización en 3D | Autodesk. [En línea] 2016. http://www.autodesk.es/products/3ds-max/overview.

41. **Open Source Robotics Fundation.** ROS.org | Powering the world's robots. [En línea] 2016. http://www.ros.org/.
42. **Coppelia Robotics V-REP.** Means of communications in and arround V-REP. [En línea] 2016. http://www.coppeliarobotics.com/helpFiles/en/meansOfCommunication.htm.
43. **coppelia robotics V-REP.** Pages and views. [En línea] 2016. http://www.coppeliarobotics.com/helpFiles/en/pagesAndViews.htm.
44. **Python Software Foundation.** Welcome to Python.org. [En línea] 2016. https://www.python.org/.
45. **ORACLE.** java.com: Java y Tú. [En línea] 2016. https://www.java.com/es/.
46. **Eaton, John W.** GNU Octave. [En línea] 2016. https://www.gnu.org/software/octave/.
47. **Gostai.** Urbi Open Source. [En línea] 2016. http://www.gostai.com/products/urbi/.
48. **Wikipedia.** Cliente-servidor. [En línea] 2016. https://es.wikipedia.org/wiki/Cliente-servidor.
49. —. Llamada a procedimiento remoto. [En línea] 2016. https://es.wikipedia.org/wiki/Llamada_a_procedimiento_remoto.
50. **Brickset.** Brickset home page | Brickset: LEGO set guide and database. [En línea] 2016. http://brickset.com/.
51. **Rebrickable.** Rebrickable - What Can You Build? [En línea] 2016. https://rebrickable.com/.
52. **Laurens.** Educator Vehicle (with attachments). *Roboscuare.* [En línea] 1 de 10 de 2013. http://robotsquare.com/wp-content/uploads/2013/10/45544_educator.pdf.
53. **Robot Scuare.** Robot Square - (Mindstorms) Robot design and development. [En línea] 2016. http://robotsquare.com/.
54. **Coppelia Robotics V-REP.** Models. [En línea] 1016. http://www.coppeliarobotics.com/helpFiles/en/models.htm.
55. **Brickwiki.** LDraw unit. [En línea] 2016. http://www.brickwiki.info/wiki/LDraw_unit.
56. **Coppelia Robotics V-REP.** Shape edit modes. [En línea] 2016. http://www.coppeliarobotics.com/helpFiles/en/shapeEditModes.htm.
57. —. Triangle edit modes. [En línea] 2016. http://www.coppeliarobotics.com/helpFiles/en/triangleEditMode.htm.
58. **LEGO Mindstorms.** *Lego Mindstorms EV3. Guía de uso.* 2013.
59. **Afrel.** 教育版レゴ マインドストーム 正規代理店（株）アフレル (Agencia Regular Educativa de Lego Mindstorms Co.). [En línea] 2016. http://www.afrel.co.jp/en/.
60. **Wikipedia.** Sonar. [En línea] 2016. https://es.wikipedia.org/wiki/Sonar.
61. **Lab, Dr. E´s.** EV3 Color Sensor (Part 2) - Youtube. [En línea] 25 de Mayo de 2014. https://www.youtube.com/watch?v=IplkMchOabc.
62. **Wikipedia.** Distribución normal. [En línea] 2016. https://es.wikipedia.org/wiki/Distribuci%C3%B3n_normal#Generaci.C3.B3n_de_valores_para_una_variable_aleatoria_normal.

63. —. Algoritmo voraz. [En línea] 2015.
https://es.wikipedia.org/wiki/Algoritmo_voraz.
64. —. Método de Montecarlo. [En línea] 2016.
https://es.wikipedia.org/wiki/M%C3%A9todo_de_Montecarlo.
65. **Coppelia Robotics V-REP.** BubbleRob tutorial. [En línea] 2016.
http://www.coppeliarobotics.com/helpFiles/en/bubbleRobTutorial.htm.
66. **OpenGL.** OpenGL - The Industry Standard for High Performance Graphics. [En línea] 2016.
67. **Wikipedia.** Modelo–vista–controlador. [En línea] 2015.
https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador.
68. **Mathworks.** Create and Share Toolboxes. [En línea] 2016.
https://es.mathworks.com/help/matlab/matlab_prog/create-and-share-custom-matlab-toolboxes.html.
69. **NXC.** NXC Programmer's Guide. [En línea]
http://bricxcc.sourceforge.net/nbc/nxcdoc/nxcapi/.
70. **HiTechnic.** NXT Gyro Sensor. [En línea] 2016. https://www.hitechnic.com/cgi-bin/commerce.cgi?preadd=action&key=NGY1044.
71. **Wikipedia.** RGB. [En línea] 2016. https://es.wikipedia.org/wiki/RGB.
72. —. Modelo de color HSV. [En línea] 2016.
https://es.wikipedia.org/wiki/Modelo_de_color_HSV.
73. **Sluka, J.** PID Controller For Lego Mindstroms Robots. [En línea] 2016.
http://www.inpharmix.com/jps/PID_Controller_For_Lego_Mindstorms_Robots.html.
74. **Coppelia Robotics V-REP.** Paths. [En línea] 2016.
http://www.coppeliarobotics.com/helpFiles/en/paths.htm.