UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Universitat Politècnica de València

TRABAJO DE FIN DE GRADO

Grado en Ingeniería Electrónica Industrial y Automática

Modelado y Simulación de un robot de tracción diferencial basado en LEGO Mindstorms NXT mediante CoppeliaSim y Matlab

Realizado por: Luis Ortiz Enguix

Tutor: Raúl Simarro Fernández

Cotutora: Clara Furió Novejarque

Valencia, 30 de Junio de 2021

Resumen

En este trabajo se realiza un modelado 3D de un robot móvil NXT Mindstorms de LEGO con tracción diferencial, cuya implementación se realiza en el simulador CoppeliaSim. En este simulador se configura e implementa la dinámica de los distintos elementos que conforman el robot. Además también se realiza la programación de los diferentes sensores y actuadores de los que dispone. Para poder manejarlo, se hace uso de una API remota de MATLAB y de las funciones desarrolladas basándose en el lenguaje RobotC, para tener una interfaz común. Con todo esto se logra tener un simulador muy versátil, en el cual poder implementar nuestro código antes de probarlo en el robot real, teniendo en consideración las variaciones que nos encontramos en el proceso real.

Palabras clave: **CoppeliaSim NXT Mindstorms API Matlab Modelado 3D Simulación RobotC**

Abstract

In this work, a 3D modelling of a LEGO NXT Mindstorms mobile robot with diferential traction is carried out, the implementation of the model is performed in the CoppeliaSim simulator. In this simulator, the dynamics of the different elements that make up the robot are configured and implemented. In addition, the programming of the diffenrent sensors and actuators is also carried out. In order to handle it, a remote MATLAB API and the functions developed based on the RobotC language are used to have a common interface. With all this we achieve to have a very versatile simulator, in which we can implement our code before testing it on the real robot, taking into consideration the variations that we find in real process.

Keywords: **CoppeliaSim NXT Mindstorms API Matlab 3D modelling Simulation RobotC**

Resum

En aquest treball es realitza un modelatge 3D d'un robot mòbil NXT Mindstorms de LEGO amb tracció diferencial, la implementació de la qual es realitza en el simulador CoppeliaSim. En aquest simulador es configura i implementa la dinàmica dels diferents elements que conformen el robot. A més també es realitza la programació dels diferents sensors i actuadors dels quals disposa. Per a poder manejar-ho, es fa ús d'una API remota de MATLAB i de les funcions desenvolupades basant-se en el llenguatge RobotC, per a tindre una interfície comú. Amb tot això s'aconsegueix tindre un simulador molt versàtil, en el qual poder implementar el nostre codi abans de provar-lo en el robot real, tenint en consideració les variacions que ens trobem en el procés real.

Paraules clau: **CoppeliaSim NXT Mindstorms API Matlab modelatge 3D Simulació RobotC**

Índice general

Índice de figuras

Índice de tablas

Parte I

Memoria

Capítulo 1

Introducción

1. Objeto

Este documento tiene como objetivo especificar las condiciones técnicas y económicas adoptadas para el desarrollo del modelado 3D y la simulación mediante CoppeliaSim y MATLAB, del robot LEGO Mindstorms NXT. La finalidad de este trabajo es la de crear un robot simulado, con las mismas características que el robot real, para que ser utilizado en las prácticas de asignaturas de robótica móvil.

Las condiciones para el desarrollo de la simulación son: que contenga las mismas piezas que incluye el modelo real, que la experiencia de simulación sea fluida y la utilización API de de CoppeliaSim para poder trabajar con MATLAB, así como emular algunas de las funciones del lenguaje RobotC.

2. Antecedentes

2.1. Robótica

La robótica es una tecnología multidisciplinar, formada por diferentes ramas: la automática, la electrónica, la informática, la mecánica, la ciencia de materiales, etc. Aunque, en los últimos años están ganando terreno las ramas de ciencias cognitivas y la inteligencia artificial.

Los antiguos egipcios ya utilizaban robots situados en estatuas de dioses, operados

por los sacerdotes, para fascinar a los fieles. Con la misma finalidad los griegos también automatizaron sus templos, basándose en la hidráulica, lo que les permitía aumentar el misticismo de sus templos [7].

En el siglo XV el genio Leonardo da Vinci ya diseñó el primer robot humanoide, conocido como caballero mecánico. En los siglos XVIII y XIX se empieza a utilizar la robótica para fines industriales, Joseph Jacquard [8] inventó el primer telar programable, que permitía tejer diferentes patrones sin casi intervención humana mediante el uso de tarjetas perforadas.

En el siglo XX comienza la época dorada de los robots, dado que se produce un gran avance en el campo de la robótica. Se diseña el primer robot programable, y rápidamente se abren laboratorios de investigación en las universidades. Los japoneses se ponen a la cabeza de esta tecnología fundando nuevas empresas y desarrollando nuevos robots industriales. Todos estos avances dieron lugar a la tercera revolución industrial [9].

En el siglo XXI la industria de la robótica está enfocada a conseguir la autonomía total de los robots mediante el uso de la inteligencia artificial, así como de los nuevos sensores que se desarrollan. Implementar el *machine learning* en los robots, podría implicar una mejora en cualquier trabajo que realicen gracias a una mayor optimización.

2.2. LEGO

Lego es una compañía juguetera danesa, que lleva fabricando juguetes desde 1932, cuyo nombre proviene del danés *leg godt* (jugar bien). Originariamente la empresa se dedicaba a la producción de juguetes de madera, pero debido a problemas de suministro de este material, comenzaron a fabricar juguetes por inyección de plástico. La empresa siempre ha estado adaptándose a las nuevas tecnologías e incluso adelantándose a su tiempo, a la vez que produce juguetes que ya han ayudado a millones de niños a desarrollarse jugando.

En este deseo de adaptarse y adelantarse comenzaron el desarrollo de su línea de sets LEGO TC LOGO en 1986, en colaboración con el MIT. Este sistema ya contaba con luces, sensores y motores. Eran productos destinados a las aulas, con los que enseñar STEAM (Ciencia, Tecnología, Ingeniería, Artes y Matemáticas). Esta primera versión tuvo poca acogida debido al alto coste y a la poca extensión de los ordenadores en la

época.

El siguiente modelo que lanzaron en la misma línea fue el primer bloque RCX, la que sería la primera versión de los actuales Mindstorms. Esta versión ya contaba con una pantalla LCD y unos puertos determinados para entrada y salida, siendo los de entrada 1 , 2 y 3, y los de salida A, B y C. En esta versión la comunicación con el ladrillo era mediante infrarrojos, a diferencia de su predecesora.

La segunda versión del ladrillo llegó al mercado en 2006: el Lego Mindsotorms NXT (Figura 1.1). Esta versión incluye mejoras en el microcontrolador, que le permiten cargar programas más complejos, además de poder procesar varios hilos simultáneamente; y en la comunicación con el ladrillo, que se puede realizar mediante USB o mediante Bluetooth, para comunicarse con otros bloques, además de con ordenadores o teléfonos. Los motores de la versión no son compatibles directamente pero se puede usar un adaptador.

Figura 1.1: Lego NXT [1].

En 2013 apareció el modelo Lego Mindstorms EV3, con un procesador nuevo, el ARM9 [10] que ejecuta Linux, el cual permite tener una mayor capacidad de procesamiento, así como poder manejar tarjetas SD externas de hasta 32GB o una antena Wifi, haciendo posible su conectividad mediante USB, Wifi(opcional mediante el puerto USB), Bluetooth o infrarrojos. Además incorpora los sensores anteriores mejorados e introduce el sensor de giro [11].

A partir de estos sets, en 2019 se lanzó el set Lego Education Spike Prime, con 3 motores y sensores para color, fuerza y distancia. En estos sets el ladrillo de las versiones anteriores, ha sido sustituido por un hub programable que cuenta con 6 entradas/salidas, el LCD se ha cambiado por una matriz de LEDs de 5×5, conectividad Bluetooth y giroscopio [12]. Utilizando el mismo hub que el set Spike, y esta vez sí en el ámbito de los Mindstorms, en 2020 salió el set Lego Mindstorms Robot Inventor.

3. Justificación del proyecto

Dadas las condiciones sanitarias del año pasado los alumnos de 3º de la carrera de electrónica industrial y automática, se vieron obligados en la asignatura de Control de Sistemas Mecatrónicos a sustituir el robot real, por el modelo LEGO EV3 en Coppelia-Sim del trabajo realizado por Alberto Martín Domínguez [13], el cual no coincidía con el robot NXT que se utiliza en la asignatura ni en el lenguaje (RobotC) usado para su programación. Por ello con este proyecto se busca proporcionar una herramienta de trabajo lo más parecida al robot real, con el que los alumnos puedan trabajar cuando no dispongan del modelo real, como si de éste se tratase.

4. Estructura

En esta sección se define la estructura que seguirá el proyecto. Concretamente se divide en 4 documentos: memoria, planos, pliego de condiciones y presupuesto.

4.1. Memoria

En primer lugar, este documento pretende introducir el mundo de la robótica y de LEGO, para ser capaces de entender mejor lo que se va a tratar en los diferentes capítulos.

En los siguientes capítulos de los que consta este documento, se van explicando tanto la elección de las herramientas con las que llevar a cabo la simulación, como los pasos para poder llevar a cabo una simulación como esta, así como una explicación del desarrollo de las funciones desarrolladas para el control del robot.

4.2. Planos

En este documento encontraremos varias imágenes, correspondientes a diversas vistas del robot, tomadas en el programa Studio 2.0. Así como las instrucciones para realizar el montaje del robot en la realidad.

4.3. Pliego de condiciones

En este documento se relatan las condiciones que se deben cumplir, tanto de software como de hardware, para que el usuario final pueda disfrutar de la simulación del robot.

4.4. Presupuesto

Este último documento del trabajo, tiene la intención de dar una idea aproximada del coste total que conllevaría la realización del proyecto.

En el presupuesto económico se recogen los costes que suponen realizar este proyecto desglosados en los diferentes elementos y recogidos por actividades. Al final de este encontramos una tabla que recoge el coste de las diferentes actividades a realizar, incluyendo los impuestos y el beneficio industrial.

Capítulo 2

Estudio de necesidades, factores a considerar: limitaciones y condicionantes.

En este capítulo se tratarán las necesidades, limitaciones y condicionantes impuestos por el cliente. Con lo que se concretará la forma final que debe tener el proyecto.

1. Especificaciones del diseño

El objetivo de este proyecto es proponer una alternativa al uso del robot físico del que solo se dispone en algunos laboratorios de la universidad.

Este sistema debe ser capaz de simular con gran precisión el robot real, de tal manera que permita poder ir implementando los cambios en el software simultáneamente en el robot simulado y en el robot real. Para ello se deberá elegir un lenguaje de programación compatible con el LEGO NXT. En este caso dado que se busca que se pueda usar con el mismo lenguaje de programación que el utilizado con el robot real, el lenguaje seleccionado es el RobotC.

La comunicación con el robot, que en el robot real ocurre con el puerto USB, en la simulación se deberá llevar a cabo usando la API del CoppeliaSim.

2. Limitaciones

Una de las principales limitaciones con la que se cuenta a la hora de realizar este proyecto es que hay que tener en cuenta que las piezas de las que dispone el set son limitadas, por ello se deberá ajustar a las piezas disponibles en el set LEGO Mindstorms Education NXT 9797, que está disponible en las prácticas de la asignatura.

Otra de las limitaciones es que se debe tener en cuenta que el simulador debe ser compatible con la mayoría de ordenadores, por lo que debe buscarse la mayor optimización posible, para una mayor fluidez.

Capítulo 3

Planteamiento de soluciones alternativas y justificación de la solución adoptada

A continuación, se va a explicar la selección de los programas con los que se trabajará durante el desarrollo del proyecto, tanto para el modelado como la simulación y el desarrollo de las funciones.

1. Herramientas de modelado 3D

Las herramientas utilizadas para la realización del modelo en 3D son Studio 2.0 [14] (Figura 3.1) y LeoCAD [15] (Figura 3.2). La elección de estas herramientas se basa en la experiencia previa en diseño de modelos Lego con Studio 2.0 y la necesidad de exportar en formato *.obj*. Por ello se ha utilizado LeoCAD, ya que son programas compatibles y se pueden exportar modelos entre ambos, aunque se podría haber realizado el modelo en LeoCAD y haberlo exportado directamente a *.obj*.

Studio 2.0 es una herramienta muy completa y fácil de utilizar con diversos puntos fuertes:

- El sistema de selección de piezas, podemos buscar piezas por su nombre o bien

buscar el número de serie de la pieza Lego en la web de [16], en la cual encontraremos el número que la identifica en el programa.

- Montar los modelos es muy sencillo, ya que se puede observar en tiempo real la posición final de la pieza, además de que calcula los ángulos y posibles conexiones que puede alcanzar cada pieza.
- Dispone de un motor de renderizado de imágenes con el que visualizar lo más realista posible el montaje final.
- Se puede realizar un libro de instrucciones según vamos añadiendo piezas, configurando los pasos a nuestro gusto.
- Comprobación de la estabilidad del modelo antes de llevarlo a la realidad, para asegurar que es viable su construcción.
- Dispone de una base de piezas interna muy extensa, con la cual además podemos realizar el pedido de las piezas necesarias para nuestro modelo.

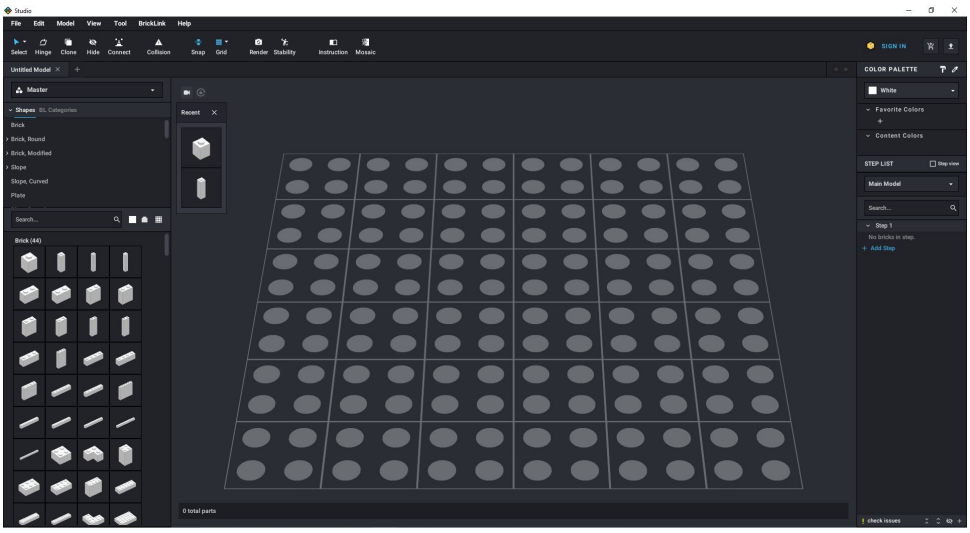


Figura 3.1: Interfaz de usuario de Studio 2.0.

LeoCAD es también una herramienta de diseño muy potente de la que destacan:

- Interfaz de usuario más sencilla.

- Simplicidad en el sistema de montaje, aunque no tiene los cálculos de agarre de Studio 2.0
- Una gran base de datos de piezas, con más de 4000 piezas disponibles.
- Gran compatibilidad con formatos externos, lo que facilita importar y exportar.
- Posibilidad de realizar un libro de instrucciones.
- Importar librerías de piezas diseñadas por terceros es una característica muy importante si se desea crear modelos únicos.

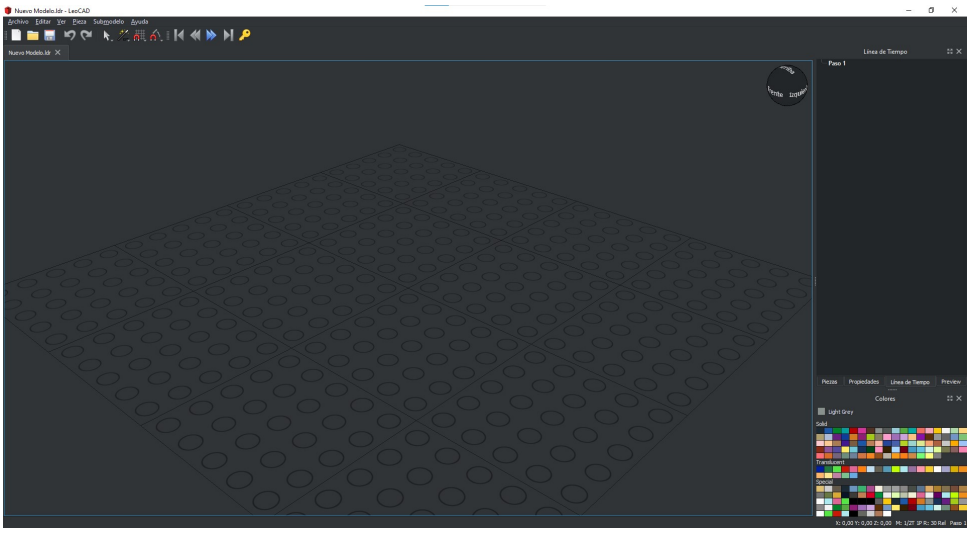


Figura 3.2: Interfaz de usuario de LeoCAD.

2. Herramientas para la simulación

Los dos programas que se barajaron para el desarrollo de la simulación fueron el CoppeliaSim y el Webots.

Webots [17] es un simulador de robots de código abierto, que además es multiplataforma. Entre sus características destaca:

- Licencia de educación.
- Una gran biblioteca con diferentes robots, sensores, actuadores...

- Posibilidad de importar modelos CAD.
- Programación con diversos lenguajes: Python , Java , MATLAB...
- Creación de escenas.

CoppeliaSim [18], es una potente herramienta de simulación en el campo de la robótica, con una gran cantidad de características como son:

- La compatibilidad entre plataformas y la portabilidad, con lo que podremos guardar el fichero con el que estemos trabajando en un equipo Windows y después abrirlo en uno Mac o Linux.
- Gran control de la simulación, gracias a que se puede personalizar desde diferentes ángulos:
    - Scripts embebidos, se pueden aplicar a escenas, simulaciones, robots, etc. Se programan en Lua.
    - Los add-ons son complementos que pueden actuar como funciones o trabajar en segundo plano, así como iniciarse con el inicio. Son parecidos a los scripts pero son más generales.
    - Los plugins solo se utilizan para proporcionar una simulación con comandos Lua personalizados, por lo que se utilizan junto con el primer método.
    - API remota, autoriza a una aplicación externa a conectarse mediante las funciones de Api remota [19].
    - Nodo ROS, permite que una aplicación externa se conecte al simulador a través del sistema operativo del robot.
    - Conexión por nodos TCP/IP entre otros, que permite que se conecte una aplicación externa.
- Una potente API con la que interactuar con diferentes aplicaciones.
- Compatibilidad con diferentes lenguajes de programación como son C, Lua, Java, Python, MATLAB ...

- Posibilidad de trabajar con cinemática inversa y directa.
- Detección de colisiones.

Cabe destacar que la versión que se usará para el desarrollo del trabajo es la versión EDU que es de libre uso para estudiantes. Esta versión no tiene limitaciones ni de uso de las funcionalidades de simulación ni de las de edición, aunque no puede ser usada con fines comerciales.

La elección final es el CoppeliaSim, ya que aunque ambos programas comparten muchas características en común, el CoppeliaSim es el programa que se ha estado utilizando en la asignatura de Robótica Móvil impartida en el grado de Electrónica, con lo que ya se tiene experiencia previa.

3. MATLAB

MATLAB [20] es uno de los programas más usados en el ámbito de la ingeniería, ya que está muy bien optimizado para la resolución de problemas, así como para el control, la robótica, el procesamiento de señales...

- Una de sus características principales es su lenguaje de programación llamado MATLAB, que está basado en matrices, y que es de muy fácil uso si se está acostumbrado a C.
- Otra característica a destacar es su compatibilidad con APIs externas, lo cual facilitará la integración de la API de CoppeliaSim para poder interactuar con el simulador. Para ello, si ya se dispone de una *toolbox* que incluya esta, será muy sencillo; por lo contrario, si se desea empezar el proyecto de cero, lo primero será extraer la API de las carpetas de CoppeliaSim, el fichero general se encuentra en *(Disco de instalación):\ProgramFiles\CoppeliaRobotics\CoppeliaSimEdu\programming\remoteApiBindings\lib\lib\(se selecciona la carpeta del S.O. que se use)*, pero los ficheros específicos para MATLAB están situados en*(Disco de instalación):/Program Files\CoppeliaRobotics\CoppeliaSimEdu\programming\remoteApiBindings\matlab*,

lo siguiente será copiar la carpeta *matlab* en el directorio en el que se trabaje, y ya se podrá utilizar la API de CoppeliaSim con MATLAB.

- Crear y compartir las *toolboxes* de MATLAB también es relativamente sencillo, ya que una vez que los ficheros *.m* que se deseen incluir estén terminados, se podrá utilizar la herramienta incluida en MATLAB para crearlos [21].

Otro de las lenguajes de programación con los que se podría haber llevado a cabo el proyecto es Python, [22] el cual es un lenguaje de programación de propósito general de alto nivel, que tiene como objetivo ayudar a los programadores a escribir código sin importar la magnitud del proyecto.

- Es un lenguaje bastante similar a Lua (el lenguaje que se usa en CoppeliaSim).
- Es compatible con CoppeliaSim.
- Lenguaje limpio, ordenado y legible.
- Hay una gran cantidad de compiladores *open-source* compatibles.

Aunque es un lenguaje relativamente sencillo, no es un lenguaje que esté tan extendido como MATLAB que se usa en una gran cantidad de asignaturas, por lo que con la finalidad de facilitar el uso del simulador se decidió implementar las funciones en MATLAB.

Capítulo 4

Descripción detallada de la solución adoptada

En este capítulo se tratará el diseño del robot en piezas Lego pasando por las herramientas de diseño 3D y la implementación del diseño en el simulador CoppeliaSim. A continuación se detallan la distintas tareas que se han realizado, para llevar a cabo el proyecto:

- Creación del modelo 3D.
- Importación a CoppeliaSim.
- Configuración de las propiedades del modelo.
- Programación de las funciones en CoppeliaSim y MATLAB.

1. Diseño del robot móvil

El diseño de este robot está orientado a que pueda ejercer dos funciones principales, que son seguir líneas y poder empujar objetos de un escenario. Por ello el montaje del robot incluye una pala en la parte delantera del mismo, que permitirá empujar los objetos. Además, en la parte posterior de la pala se situará un sensor de luz que se utilizará para detectar el color de la superficie sobre la que nos encontramos.

Se empezó a diseñar a partir del diseño del set 9797 [2] proporcionado por LEGO, en el cual se introducen diversos sensores a lo largo del montaje, como son el sensor de ultrasonidos, sensor de luz, sensor de sonido, sensor de tacto y 3 motores, 2 para controlar el movimiento del robot y otro para incorporarle un brazo con el que poder mover cosas. En el montaje del robot final no se implementaron ni el sensor de sonido ni el brazo, dado que no han sido considerados necesarios para la finalidad del robot.

La configuración que sigue este montaje, es una configuración de tipo triciclo, ya que dispone de dos ruedas delanteras fijas y de una rueda trasera orientable descentrada. Gracias a estar invertido el triciclo, podremos controlar la posición del robot cuando avancemos hacia delante, mientras que al ir marcha atrás con la rueda caster, el movimiento será poco preciso.

Figura 4.1: Estructura básica NXT [2].

Con esta estructura, ya se puede añadir el resto de las piezas que completan el montaje, como serían el sensor de ultrasonidos en la parte superior, el sensor de luz, así como el sensor de tacto trasero.

Una vez que estaban situados los sensores, comenzó el diseño de la pala, la cual estaba limitada a la cantidad de material, ya que uno de los objetivos era realizar el montaje solo con los materiales que contiene un set oficial.

Figura 4.2: Montaje final.

Una vez que el diseño ya está confirmado, el siguiente paso era el traslado de la realidad al modelo 3D.

A continuación se van a describir los principales elementos que forman parte de la construcción del robot NXT.

1.1. Ladrillo NXT

El ladrillo NXT es la pieza central tanto del diseño como de la programación, ya que es la pieza que contiene el procesador. Las características de este ladrillo son:

- Microprocesador ARM7TDMI.
- Memoria RAM → 64KB. Memoria Flash → 256KB.
- Comunicación por Bluetooth 2.0.
- Comunicación USB 2.0.
- 4 Puertos de entrada, para la conexión con los sensores.
- 3 Puertos de salida, para controlar los actuadores.
- Altavoz.
- 4 Botones.

- Pantalla LCD.
- Alimentación mediante 6 pilas AA o una batería.

Figura 4.3: Ladrillo NXT [3].

1.2. Sensores

En este diseño se han seleccionado 3 sensores de los que incluye el set 9797 [2] y 2 motores de los 3 que incluye.

El sensor de ultrasonidos permite medir distancias, mediante un sistema de emisor/receptor. Esta técnica está basada en excitar una membrana magnetoestrictiva con impulsos eléctricos para generar una onda, que después de impactar rebotará y será recogida por el receptor, invirtiendo el proceso. Este sensor diseñado por LEGO tiene una capacidad para medir distancias desde los 0 a los 255 cm con una precisión de ± 3 cm [23]. Hay que tener en cuenta que la lectura del valor 255 cm puede indicar tanto que es la distancia máxima como que hay un error en la medida.

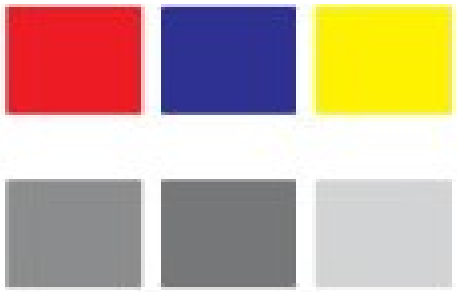
Figura 4.4: Ultrasonidos NXT [4].

El sensor de tacto, es un sensor que actúa como un botón, ya que puede presionarse o soltarse y el sensor detecta estas diferencias.

Figura 4.5: Sensor de tacto NXT [5].

El sensor de luz permite elegir entre luminosidad y oscuridad, para medir la luminosidad de una habitación y para determinar la luminosidad de los colores, lo cual permite calcular cuáles son, según los resultados que devuelva [24].

(a) Sensor de luz NXT [25].



(b) Comparación lectura colores [26].

Figura 4.6: Sensor de luz y detección.

1.3. Actuadores

El motor LEGO NXT contiene un sensor de rotación con una precisión de ±1°, este sensor permite una gran precisión en el movimiento del motor. Una rotación del motor equivale a 360° [24]. Sus características principales son 170 rpm máximo, un par de rotación de 27 N·cm y un par de 50 N·cm máximo en parado, que serán los valores ideales con los que se trabajará [27].

Figura 4.7: Servomotor NXT [6].

2. Modelado 3D

Para pasar el robot real a una versión CAD del mismo se empleó el programa Studio 2.0 (Sección 1). Se comenzó a seguir el mismo proceso de montaje que se había usado para el robot real. Se empezó a construir la estructura básica.

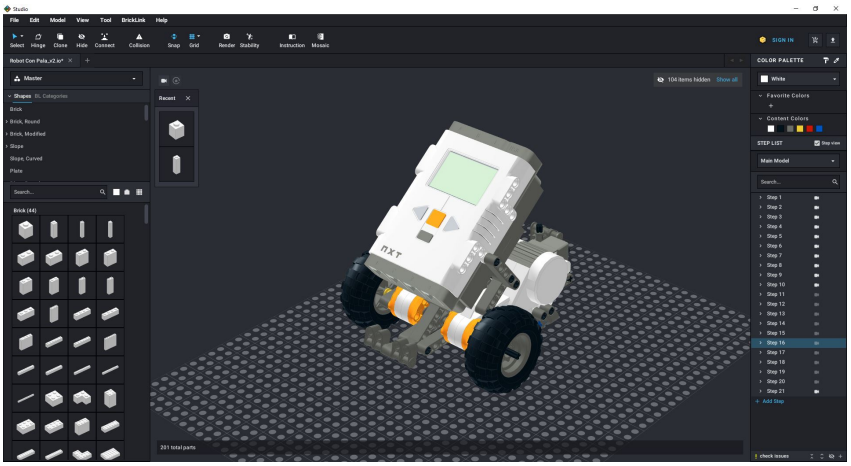


Figura 4.8: Estructura básica NXT modelado 3D.

Se prosiguió añadiendo los sensores al modelo.

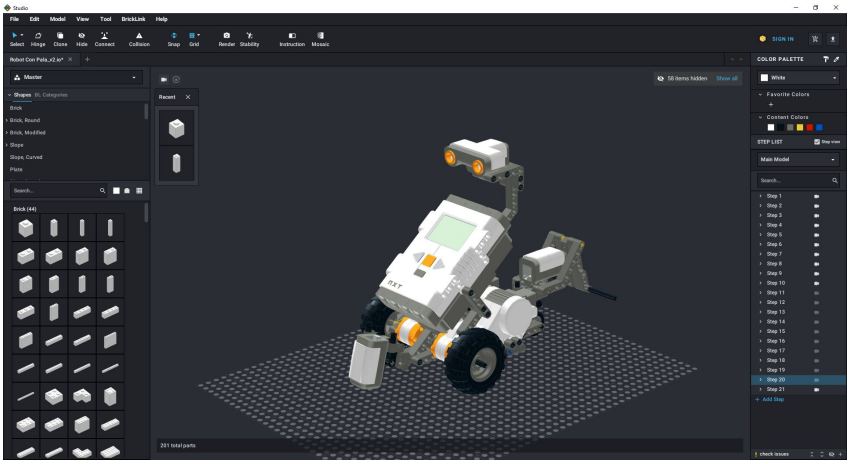


Figura 4.9: Modelado 3D sensorizado.

Para completar el modelo se añadió la pala, que fue la parte más complicada de modelar debido a los ángulos que formaban las piezas entre sí.

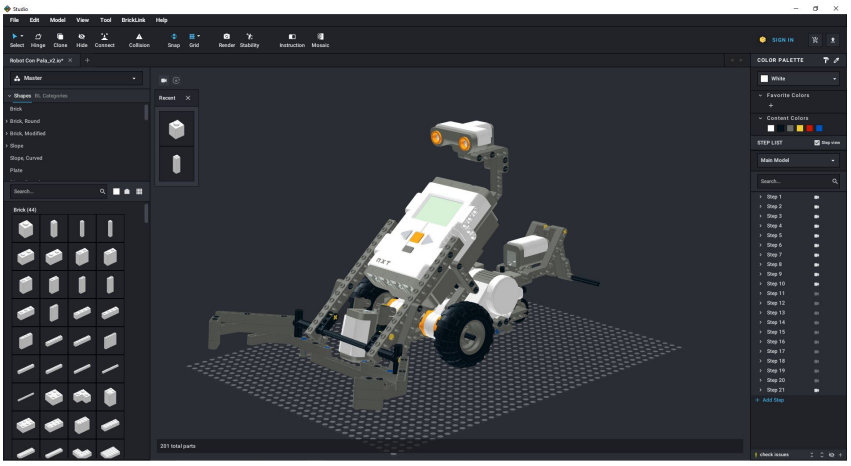


Figura 4.10: Modelado 3D final.

En la parte planos (Parte II), se encuentran más imágenes correspondientes a al alzado, perfil y planta del robot, así como las instrucciones de montaje del robot.

3. Modelado en CoppeliaSim

3.1. Importación

El modelado en CoppeliaSim comienza con la exportación del modelo 3D (Figura 3.2) diseñado con Studio 2.0 al formato LDraw *.ldr*. Trabajando con el diseño en un formato comprensible por LeoCAD, se puede hacer uso de la función exportar de LeoCAD, con la cual podremos a exportar al formato *.obj* accediendo a $Archivo \rightarrow Exportar \rightarrow Wavefront$.

Con el modelo 3D en formato *.obj*, ya se puede importar con CoppeliaSim, este formato permitirá importar el modelo con los colores originales, siempre que así lo deseemos, ya que también se pueden importar solo las formas.

Para importar el modelo se seleccionará $File \rightarrow Import \rightarrow Mesh...$. Tras esto aparecerá una pestaña de confirmación:

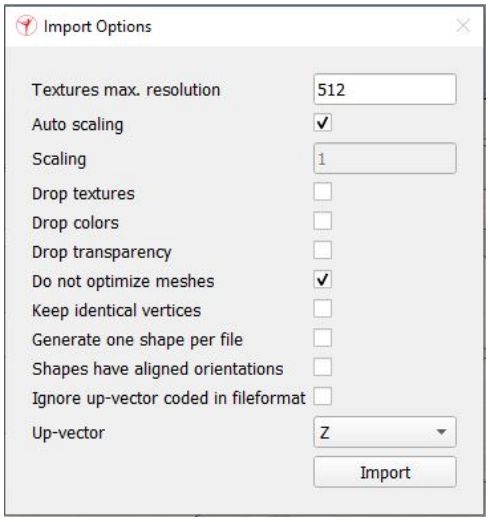


Figura 4.11: Opciones de importación.

En esta pestaña es importante clicar en *Do not optmize meshes*, con lo cual creará una nueva forma (*shape*) por cada pieza de Lego que hayamos insertado en el modelo, lo cual permitirá más tarde agruparlas para tener una pieza general y otras externas para hacer las animaciones de movimiento. Además se debe elegir el $Up - Vector \rightarrow Z$, para que encajen los vectores del modelo de Studio 2.0 con los del modelo importado.

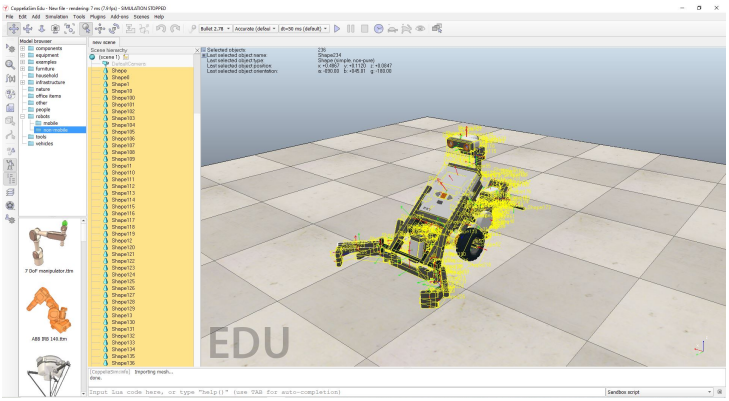
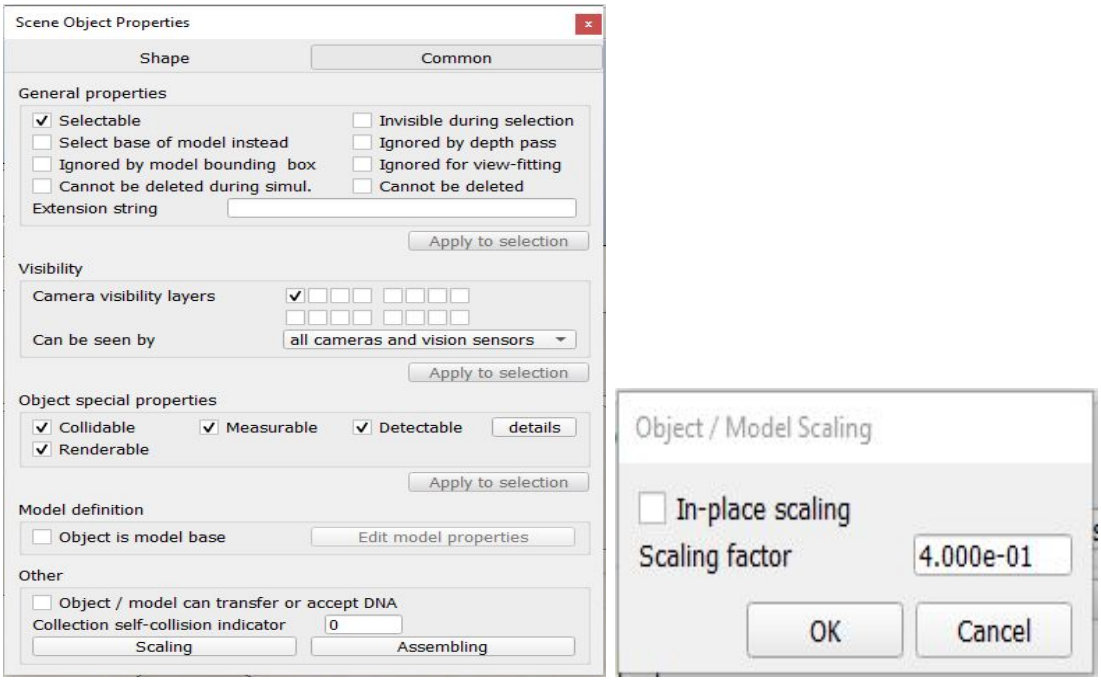


Figura 4.12: Formas importadas.

3.2. Escalado

Una vez que ya se han incorporado las formas a la escena de CoppeliaSim, se deberá proceder a su escalado, ya que aunque al importar se selecciona que la escala es 1mm, el formato de dibujo de LeoCAD al exportar impone una relación de 1 LDU (*LDraw Unit*),

que equivale a 0.4mm [28], por tanto este será el factor de forma que se deba aplicar al modelo. Para ello se entra en las propiedades de cada objeto y se clica en el botón *Scaling*, que lleva al *Model Scaling*.



(a) Propiedades del objeto. (b) Factor de escalado.

Figura 4.13: Escalado del modelo.

3.3. Agrupado

El siguiente paso del modelado es, como se ha mencionado en el apartado anterior (Subsección 3.1), agrupar todas las piezas que hemos incluido en CoppeliaSim, según si van a ser partes inmóviles o si van a estar asociadas con algún eje de movimiento.

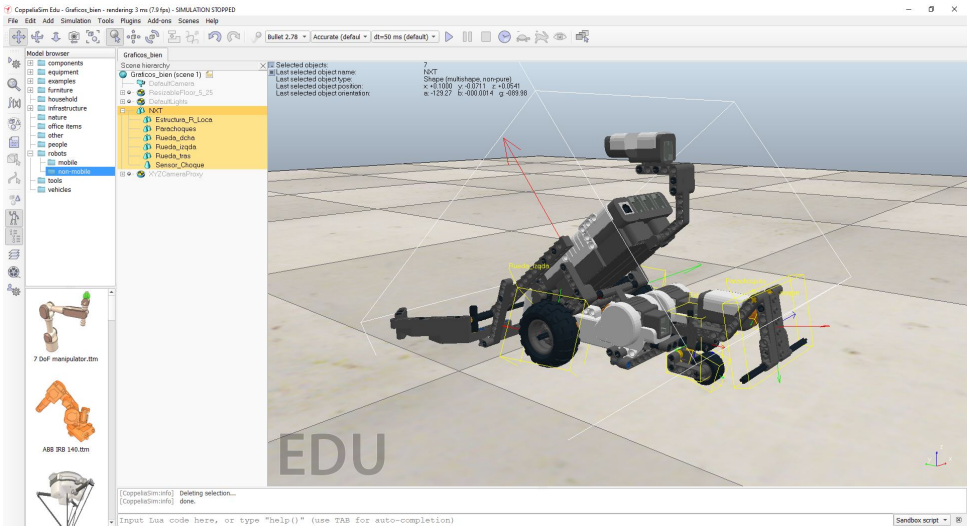


Figura 4.14: Formas agrupadas.

De esta manera se han distribuido en:

- El NXT siendo esta forma la que contiene la mayor parte de las piezas
- El sensor de choque es la pieza naranja del sensor de tacto.
- El parachoques es la parte móvil que hará contacto con el sensor de tacto.
- La estructura de la rueda loca, que se moverá girando de acuerdo al eje Z.
- Las 3 ruedas.

3.4. Piezas dinámicas

A continuación, con la capa externa o gráfica ya importada en CoppeliaSim, se deberán crear las piezas dinámicas, esto es debido a que si se deseara realizar simulaciones con el modelo original, requeriría una gran cantidad de recursos del ordenador, produciendo una simulación muy ralentizada. El diseño de las piezas dinámicas va acorde a la posibilidad de que el robot impacte, así como la necesidad de que las piezas sean móviles, ya que es imprescindible modelarlas en forma dinámica para que giren.

Entre todos los tipos de formas que tiene CoppeliaSim se va a utilizar las formas puras simples, que son las que están mejor optimizadas para las simulaciones de colisiones, proporcionando así una experiencia más fluida.

A la hora de crear el modelo dinámico, deberemos primero seleccionar la forma que queremos editar, ya que ésta se cargará dividida en triángulos cuando se clica en el botón *Toggle shape edit mode*. Esta pestaña de edición tiene 3 modos:

- *Triangle edit mode* que es el que viene por defecto, y que divide la forma en pequeños triángulos en los que la separa, para crear la pieza.
- *Vertex edit mode* que carga la forma gráfica resaltando los vértices de la misma para poder seleccionarlos para crear la pieza.
- *Edge edit mode* que resalta los bordes de la pieza, con los que crear la pieza.

En este caso se ha hecho uso del *Triangle edit mode*. Una vez que las piezas están listas para seleccionarlas, se pueden seleccionar de diferentes maneras:

1. Pulsando *Shift* + arrastrando mientras se cogen los triángulos deseados.
2. Clicando uno por uno los triángulos en la barra de la izquierda, aunque este método es poco preciso ya que no se sabe que triángulo corresponde con que parte.
3. Seleccionando los triángulos de la zona desada con *Control* + *Clic*, en cada uno de los triángulos.

Usando estas herramientas, solo queda ya diseñar las piezas dinámicas que se consideren necesarias.

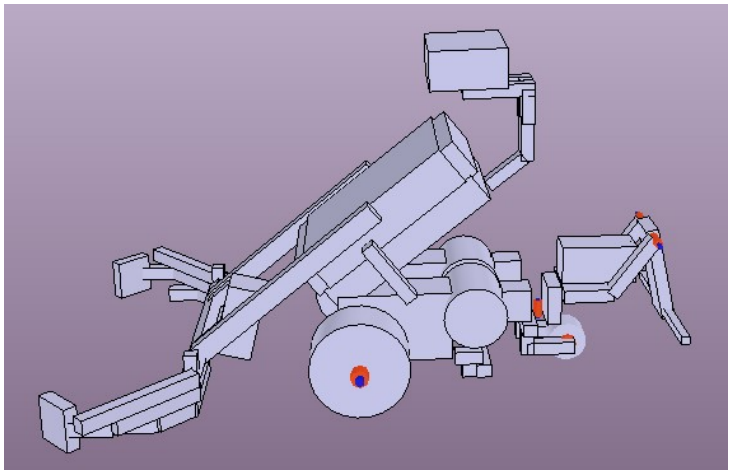


Figura 4.15: Formas dinámicas.

3.5. Propiedades dinámicas

Una vez que ya están las formas dinámicas definidas, el siguiente paso es agruparlas al igual que se hizo con las formas estéticas (Subsección 3.3), para después añadirles las propiedades, en este caso solamente se le han añadido a las ruedas, para que tengan coeficiente de rozamiento con el suelo.

Para ello lo primero es invertir las capas, para ver solamente las piezas dinámicas. Después se deberá clicar en la rueda que se desee, se abrirán sus propiedades de escena y se entrará en *Show dynamic properties dialog*, para ir a *Edit material* y en *Apply predefined settings* hay que seleccionar *highFrictionMaterial*. Con estos ajustes se evitará que las ruedas deslicen y permitirá controlar el robot con mayor precisión.

3.6. Modelado de actuadores

Para emular los motores (Subsección 1.3) que incluye el set NXT, en CoppeliaSim se pueden modelar tres tipos de articulaciones: las de revolución, prismáticas y esféricas. Para la emulación de los motores, la articulación de revolución es la más adecuada, ya que permite incorporarle las características propias de un motor.

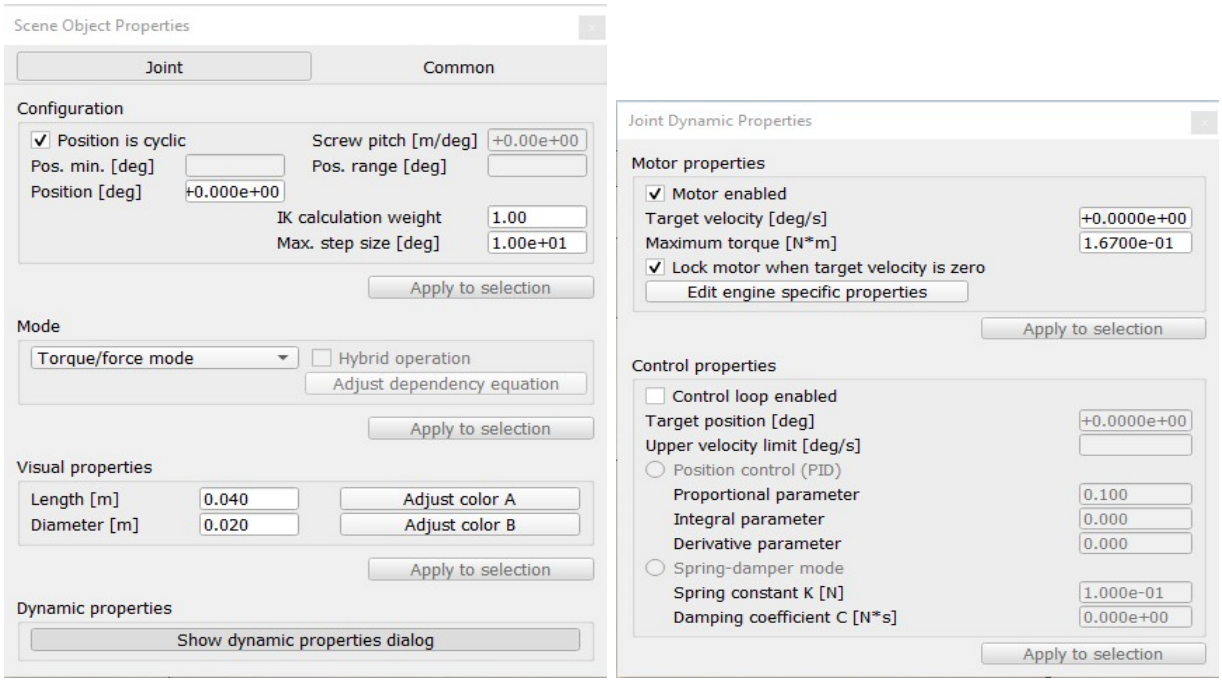
Una vez que se crea la articulación el primer paso es orientarla, ya que lo normal es que aparezca en el sentido del eje Z. Para ello usaremos la herramienta *Object/item rotate* y seleccionando la articulación y la giramos 90° en el eje Y, para orientarla en el sentido del eje de la rueda. Para posicionar la articulación en el eje de la rueda se usa la herramienta *Object/item shift*, seleccionando primero la articulación y luego la rueda, pulsaremos en *Position* →*Apply to position* con lo que la articulación estará situada en su posición final.

Ahora solo queda configurar las propiedades dinámicas de las ruedas.

El primer paso es entrar en las propiedades del objeto, donde se debe desactivar la casilla de *Position is cyclic*, para poder cambiar la *Pos. min. [deg]* a $-180°$ y el *Pos. range [deg]* a 360°, con lo que el programa dejará de gestionar el movimiento de ésta. Después hay que asegurarse de que el modo *Torque/force mode* está seleccionado. Por último quedará configurar las propiedades dinámicas de la articulación, donde se debe clicar en la opción *Motor enabled*, también hay que configurar el par máximo que es capaz

de otorgar.

Repetiremos el proceso para la otra rueda.



(a) Propiedades del motor.

(b) Propiedades dinámicas del motor.

Figura 4.16: Configuración del motor.

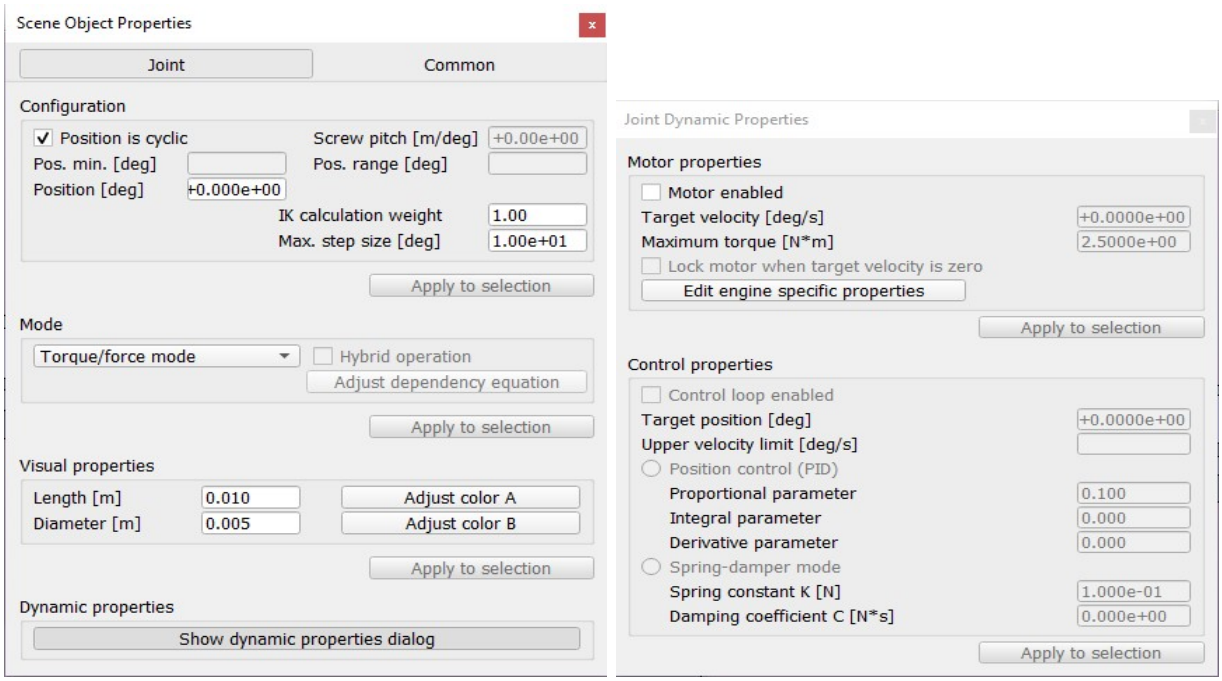
3.7. Modelado de ejes

En este apartado se seguirán los mismos pasos que para crear los actuadores, pero configurándolos de otra manera, para que se puedan mover con libertad.

Los ejes que tendremos que modelar de esta manera son: el que sujeta el parachoques, el eje que sujeta la rueda trasera y el eje de la rueda trasera, ya que en este caso la rueda trasera actuará como una rueda loca de tipo rotatoria.

En este caso no vamos a necesitar habilitar el motor, por tanto, dejaremos los ajustes por defecto de las propiedades del objeto, y se debe revisar que en las propiedades dinámicas de la articulación no esté activado el motor.

Repetiremos el proceso para el resto de ejes.



(a) Propiedades de la articulación. (b) Propiedades dinámicas de la articulación.

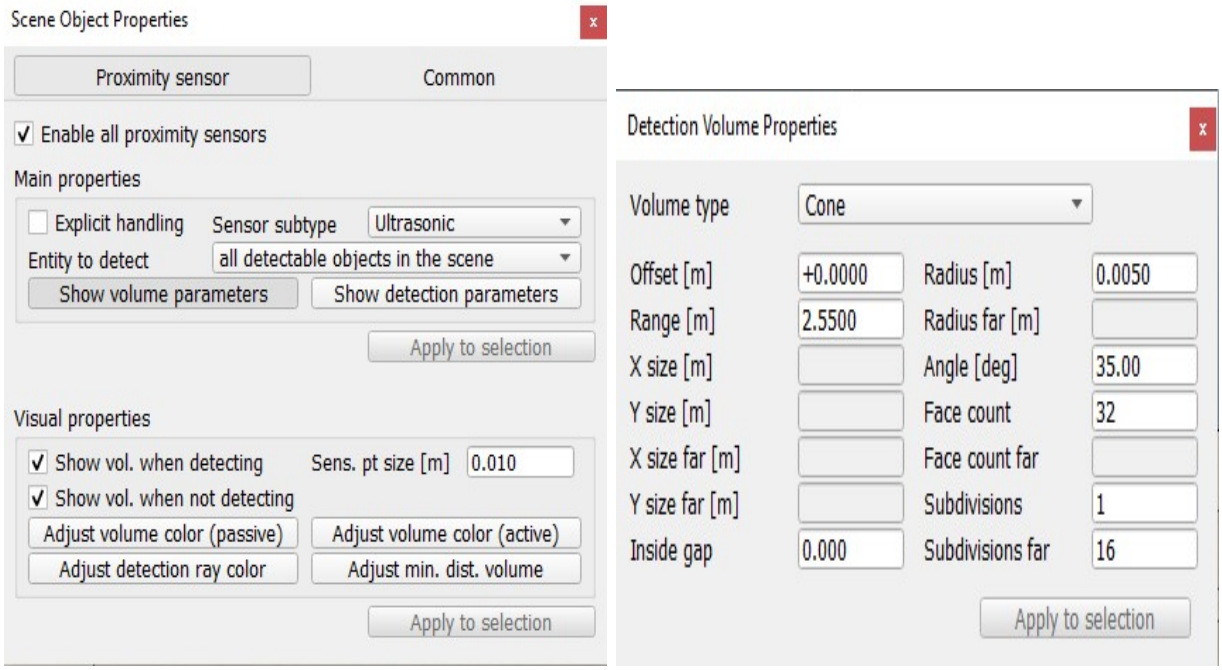
Figura 4.17: Configuración de los ejes.

3.8. Modelado de sensores

Para modelar los sensores, se puede usar la gran variedad de sensores que ya están incluidos en CoppeliaSim y configurarlos con las características de los sensores reales.

Para modelar el sensor de ultrasonidos, se tendrán en cuenta las características mencionadas previamente cuando se describió el sensor (Subsección 1.2). También se tendrá en cuenta una característica que es del último modelo del sensor, dado que no está disponible de los modelos anteriores, el cono de dispersión de la onda que genera el sensor es de $\pm 35°$ [29].

A continuación se tienen que introducir los datos en las *Detection Volume Properties*, para ello deberemos primero acceder a las propiedades del objeto y clicar en *Show volume parameters*.



(a) Propiedades del ultrasonidos. (b) Propiedades del volumen de detección.

Figura 4.18: Configuración del ultrasonidos.

Otro de los aspectos que permite CoppeliaSim modelar del sensor de ultrasonidos es, el ángulo de detección. Gracias a esta configuración se obtiene un modelado más realista, ya que al seleccionar esta opción se introduce el ángulo máximo entre el rayo de detección y el vector normal de un objeto [30]. En la programación de la función dedicada a la lectura de los datos de este sensor se ha incorporado un factor de aleatoriedad como en el robot real gracias al uso de la función *random*. Hay que mencionar también, que se ha inclinado levemente el sensor, para que tenga una mejor detección en distancias más cercanas al robot.

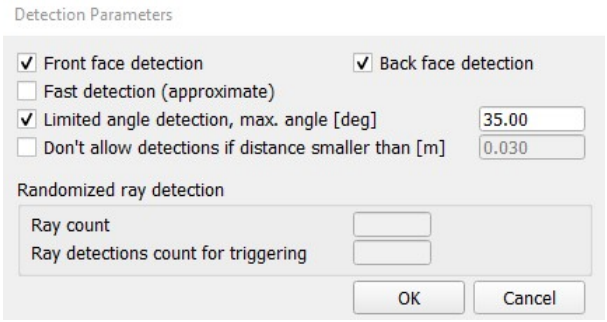


Figura 4.19: Configuración ángulo límite.

El sensor de luz es algo más complicado de configurar, pero con ayuda de la documentación oficial se aclara el proceso. Aunque el sensor dispone de dos modos de funcionamiento (Subsección 1.2), solo se va a implementar el de luz reflejada, ya que con ese es más que suficiente para las funciones que se le han planteado. Hay que destacar que para que el sensor haga una buena lectura de la línea, tiene que tener un grosor mínimo de 1 mm. A la hora de programar la función se ha usado el mismo método para incorporar aleatoriedad al robot, además de añadirle un offset, para que los resultados sean lo más parecidos a los reales.

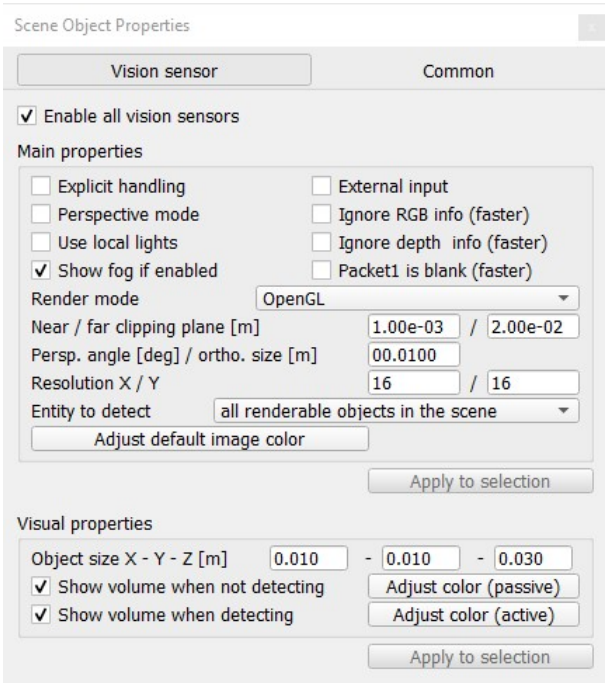


Figura 4.20: Configuración sensor de luz.

Por último solo queda por implementar el sensor de tacto (Subsección 1.2). Para configurar este sensor se ha decidido optar por la configuración de serie, en la cual el sensor devuelve el valor medio de una muestra de 5 medidas, ya que es la forma de tener una mejor optimización y asegurarse de que los valores que devuelven son verdaderos y no debidos a una mala interpretación por parte de CoppeliaSim.

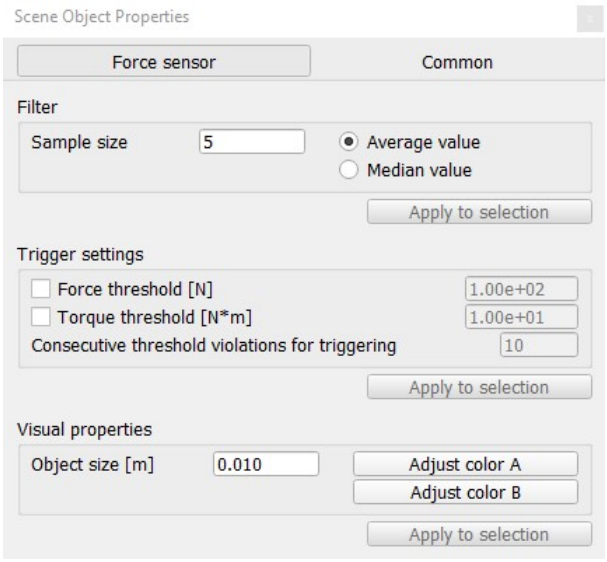


Figura 4.21: Configuración sensor de luz.

3.9. Diseño de la UI (Interfaz de usuario)

Para el diseño de la UI se ha empleado la versión V-Rep Pro Edu, ya que en la versión 4.1 de CoppeliaSim la edición de las interfaces está más limitada, y aprovechando la compatibilidad entre las versiones y que los elementos usados son comunes a las dos, no provocaría errores, así que se optó por esta solución.

Hay que mencionar que la resolución de la pantalla es superior a la real, con el fin de que el ladrillo de la UI tuviera un tamaño adecuado, ya que se tiene que seleccionar la misma resolución para todos los cuadrados de la UI.

La resolución que se ha seleccionado para cada cuadrado es de 14×14 píxeles, y la UI está formada por 18 filas y 11 columnas. A la hora de crear la UI se diferenció entre *button* y *label* para las casillas, siendo botones los propios botones del robot y los de la UI, mientras que el resto de elementos que la conforman son label. Además como se puede observar, los botones no tienen la misma forma ya que solo se pueden implementar botones cuadrados o rectangulares.

(a) Interfaz de usuario.

(b) Ladrillo real [31].

Figura 4.22: Comparación diseño de ladrillos.

4. Modelo final

Tras la importación de las formas estéticas, así como el modelado de las piezas dinámicas gracias a las formas estéticas; la implementación de los ejes de giro y de los motores, y de los sensores. Solo quedará establecer la relación jerárquica de los componentes.

Para ello, la parte principal del robot debe ser el esqueleto, es decir, el conjunto de piezas dinámicas más grande, al que se le añadirá el resto de piezas que conforman el robot. Debajo del esqueleto, se posicionarán todos los sensores y los ejes y motores. Como los ejes y los motores son piezas que van a tener animaciones asociadas, dentro de cada uno de ellos se deberá introducir la pieza dinámica del elemento que corresponda, y a su vez se introducirá la forma estética en el nivel inferior de la pieza dinámica.

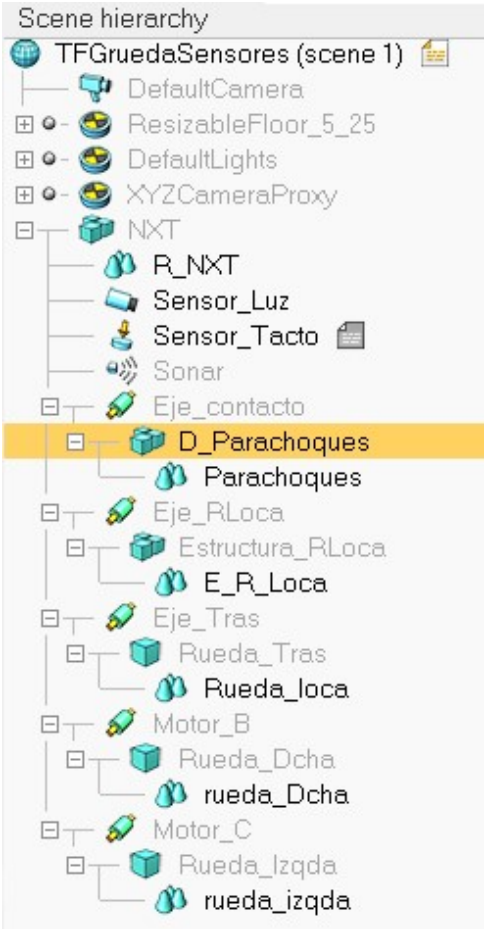


Figura 4.23: Estructura jerárquica.

Una vez que ya está ensamblado todo, tendremos el modelo listo para las simulaciones. En la siguiente imagen se puede observar el ensamblaje dinámico y de los sensores y actuadores.

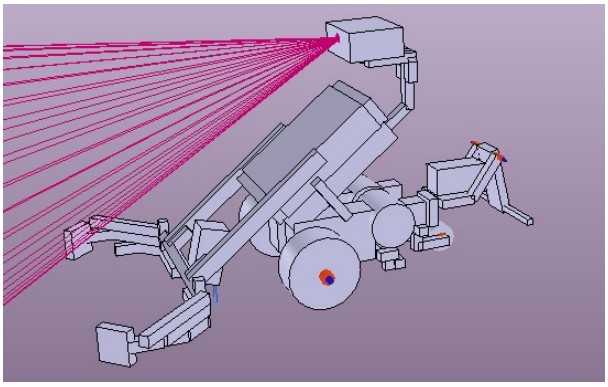


Figura 4.24: Dinámicas con sensores.

A continuación se puede observar el resultado final desde la capa estética.

Figura 4.25: Resultado final estético.

5. Creación de las funciones de manejo del robot

El primer paso para el desarrollo de las funciones es investigar las funciones disponibles en el lenguaje que se quiere emular, para ello es necesario acceder a los recursos oficiales de RobotC [32].

Una vez que se seleccionan las funciones, el siguiente paso es la comunicación con la API. Es precisa esta comunicación para que se puedan ejecutar las funciones desde MATLAB, ya que hay que implementarlas en CoppeliaSim, pero para poder desarrollar los futuros códigos que controlarán el robot, hay que crear los manejadores de las funciones en MATLAB.

Para este caso, se han realizado pruebas de conexión a la API con los diferentes puertos (*pruebaconexión.m*) y se estuvo investigando como trabajar con la API, pero finalmente se ha decidido reutilizar las funciones de conexión con la API creadas por Alberto Martín [13] y modificarlas para las circunstancias que se daban en nuestra versión de CoppeliaSim, como son las diferentes funciones y los puertos con los que se había decidido trabajar.

5.1. Funciones en CoppeliaSim

Una vez que la conexión a la API está garantizada, el siguiente paso es empezar a escribir el código en un *non-threaded child script*, dentro de un objeto de tipo *dummy*, el

cual contiene todas las funciones a las que se puede acceder desde MATLAB, así como otras funciones de control de la API.

Hay que destacar que la documentación oficial y el foro de CoppeliaSim han sido de gran utilidad a la hora de resolver problemas.

Para el desarrollo de este *script*, el primer paso es nombrar a todo correctamente en la jerarquía de la escena, para que las relaciones funcionasen correctamente. Una vez los nombres están claros, el siguiente paso es incorporar todos los datos mencionados anteriormente (Subsección 1.2) sobre los sensores y actuadores en variables globales, así como añadir otras relacionadas con el uso de la UI, para poder hacer uso de los botones. Además de añadir las variables para el uso de los sensores, también hay que llamar al manejador de los sensores y actuadores.

Las funciones que se encuentran aquí son:

- setMotor
- stopMotor
- getMotorEncoderC
- getMotorEncoderB
- resetMotorEncoder
- getTouchValue
- getUSDistance
- getLightValue
- displayStringAt
- eraseDisplay
- getButtonPress
- getTimer
- Así como otras funciones que se precisan para el uso de la API desarrollada por Alberto Martín [13].

Una vez ya están todas las funciones creadas, solo falta que se añadan en el fichero *nxtRemoteApi.m*, todas las funciones que devuelvan datos desde CoppeliaSim, para que se pueda trabajar con esos datos en MATLAB.

5.2. Funciones en MATLAB

A la hora de realizar la programación en MATLAB se han creado constantes con las que emular mejor el nombre de las variables que se usan en RobotC, las cuales se encuentran en el archivo *constantes.m*. Estas constantes se dividen en constantes de motores (motorB y motorC), de puertos (S1, S2, S3 y S4) y los botones (buttonLeft, buttonRight, buttonEnter, buttonBack). Con las constantes declaradas, ya se puede empezar a crear las funciones que serán las encargadas de utilizar la API, para hacer las llamadas a las funciones que se encuentran en el *script* de CoppeliaSim.

Una vez que están claras todas las funciones que se van a desarrollar, teniendo en cuenta de que a las funciones que se han desarrollado en CoppeliaSim, hay que sumarles las funciones que no existen en RobotC, dado a que en el robot original, los sensores se conectan a los diferentes puertos, y en este caso requiere del uso de funciones. Por tanto, habrá que crear unos comandos para controlar estas conexiones y las funciones espejo a las de CoppeliaSim, que serán los manejadores de estas, que las llamarán a su uso.

Dentro de los comandos espejo respecto a CoppeliaSim, habrá que declarar dos tipos diferentes, las que reciben datos cuando son llamadas y las que solo envían la señal, ya que las que devuelven datos, deberán ser dadas de alta en el archivo *nxtRemoteApi.m*

Todas las funciones disponibles en el simulador se encuentran detalladas en el Manual del usuario (Capítulo 10), en el cual se relatan los parámetros que admiten.

5.3. Creación de la toolbox

Una vez que ya se tienen las funciones desarrolladas, y ya se ha comprobado su funcionamiento, solo queda crear la *toolbox* para que se pueda distribuir fácilmente.

Para ello el primer paso es ir a MATLAB y con abrir la opción *Package toolbox*, en la cual se seleccionará la carpeta en la que se encuentran los ficheros de la API y los archivos *.m*, una vez se añada un nombre a la *toolbox* y una versión de desarrollo, al cliclar en

Package estará disponible la toolbox para su distribución.

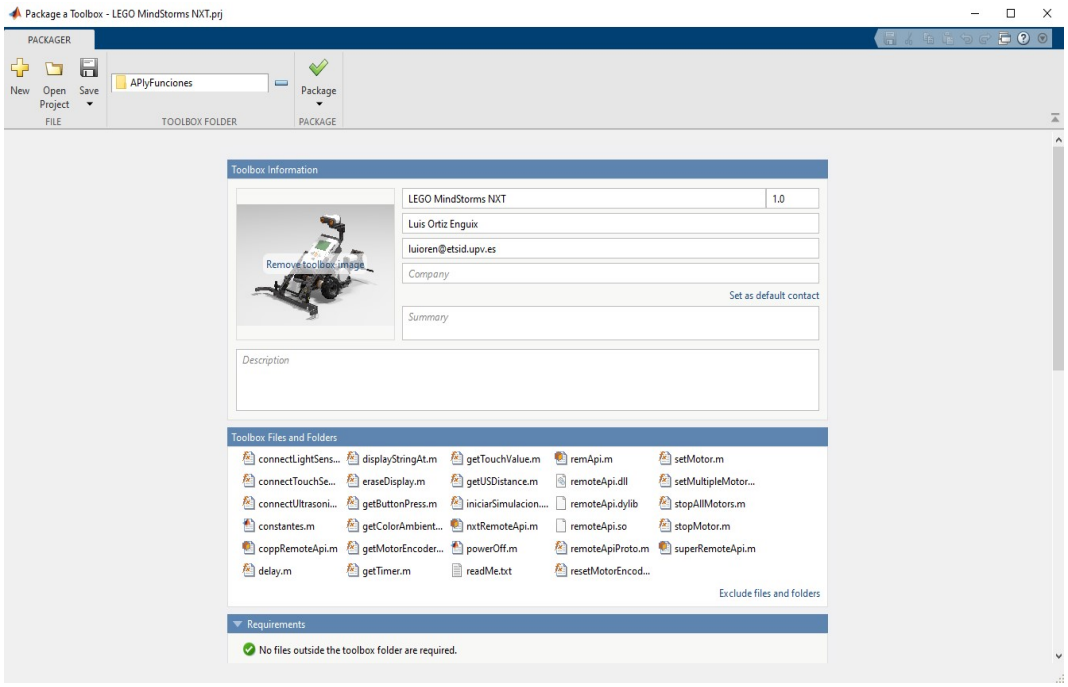


Figura 4.26: Creación de la toolbox.

Capítulo 5

Resultados

En este capítulo se evaluará el resultado de la simulación mediante la implementación de 2 escenas, una en la que el robot deberá seguir una línea y otra en la cual el robot deberá limpiar el escenario en el menor tiempo posible. Los códigos de estas pruebas se encuentran en los anejos.

1. Siguelíneas

Esta primera tarea consiste en conseguir que el robot siga un camino pintado mediante líneas negras de un grosor. Para ello se utilizará el sensor de luz del robot. Se valorará tanto el seguimiento como la rapidez en conseguir llegar a la meta.

Para resolver esta tarea, tras la realización de diversas pruebas, se optó por la utilización de un regulador PID, en el cual se tenían en cuenta los errores integral, derivativo y el error anterior, con el fin de calcular el error que nos separaba de la línea negra. Tras obtener el error, se calculaba la velocidad con la que se debía de actuar en cada rueda. Como medida de prevención se decidió implementar el uso de bucles de control, para que no se acumulara demasiado error integral, lo que se convertiría en un inconveniente. Además de este sistema, también se implementó otro sistema de prevención, en las velocidades de las ruedas, dado que no pueden trabajar por encima del 100 %, y en caso de que se superase este se pondría automáticamente en el 100 % antes de actuar.

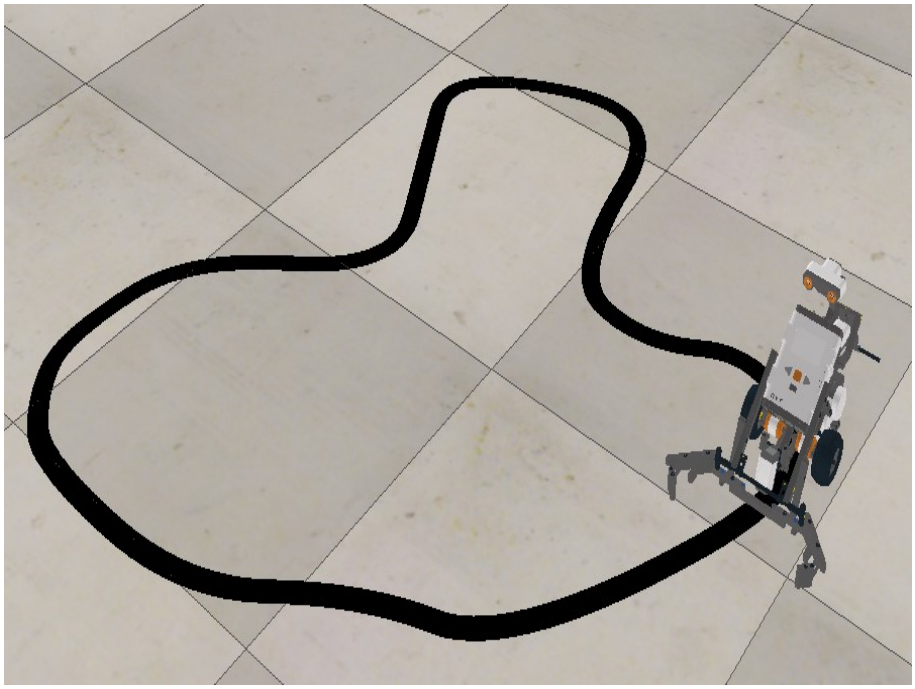


Figura 5.1: Escena del robot siguelíneas.

2. Robot Limpiador

La segunda tarea a realizar consiste en la programación de un robot que sea capaz de limpiar los objetos ubicados en el interior en un entorno circular de 1500 mm de radio. El robot deberá limpiar todos los objetos lo antes posible pero sin salir del área de limpieza en ningún momento. Para ello se pueden utilizarlos sensores de luz, de distancia y el de contacto.

A la hora de desarrollar este ejercicio, había que conseguir que no se saliera el robot del espacio mientras sacaba los cilindros.

Con este objetivo se desarrolló un programa en el cual se fijaba un cilindro como objetivo y el robot no paraba hasta sacarlo de la zona delimitada. Sin embargo, de este modo, cuando el robot sacaba el cilindro, también se salía de la zona y debía reincorporarse marcha atrás, por lo que no era un método de limpieza muy eficiente. La estrategia se replanteó para que el robot girase ligeramente al ir marcha atrás después de salirse. Así, al volver al círculo, también puede utilizar el sensor de ultrasonidos para localizar el siguiente objetivo. También es importante mencionar que si se deja el robot sin ninguna temporización en el caso de que no encuentra el objetivo, se atasca y tarda mucho en

encontrar el último cilindro. Por ello se implementó un contador de vueltas, con el que el robot, una vez llega a un número máximo de vueltas, cambia de dirección para ampliar el rango de búsqueda.

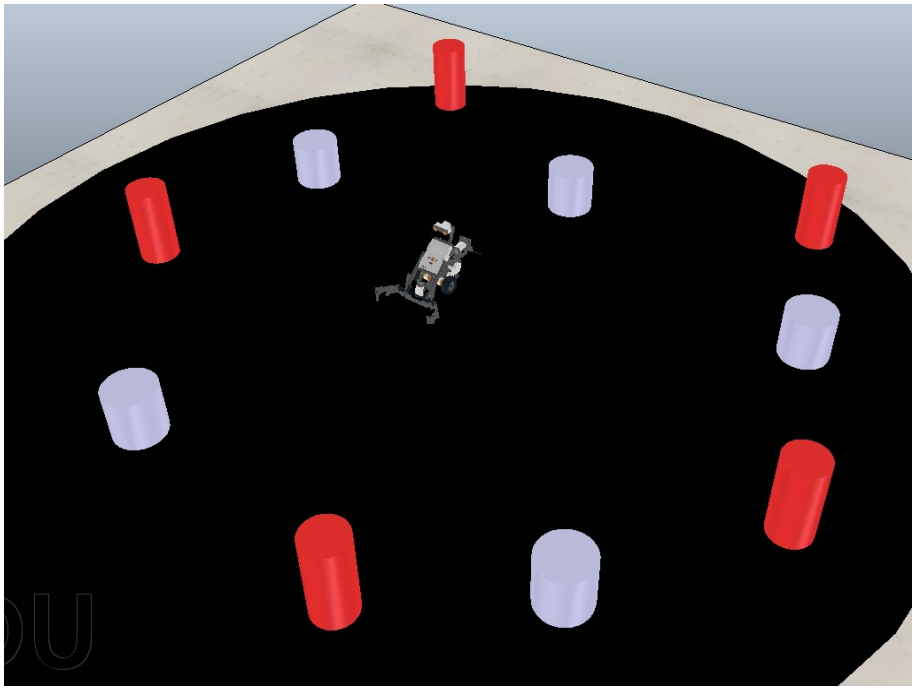


Figura 5.2: Escena del robot limpia obstáculos.

3. Compatibilidad del simulador

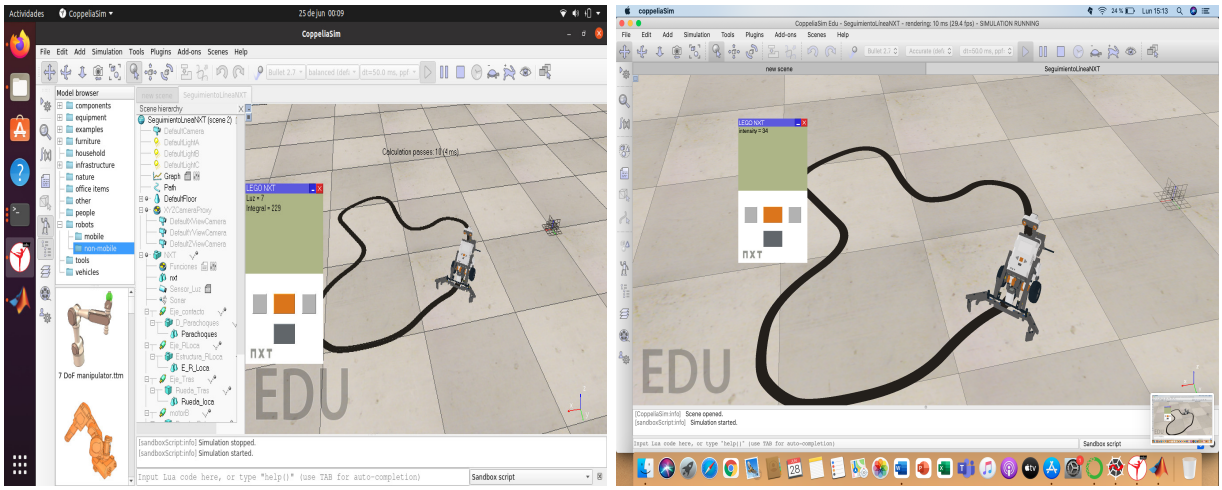
Uno de los objetivos era poder hacer funcionar el simulador en un gran abanico de equipos, por ello se han llevado a cabo las mismas pruebas que se han realizado en Windows, en los dos otros sistemas operativos compatibles con CoppeliaSim. La prueba que se ha llevado a cabo ha sido para ambos sistemas operativos el seguimiento de línea.

Por disponibilidad de software, se han realizado las pruebas en versiones anteriores de MATLAB, como son la R2019b y la R2018b, lo que nos confirma que aunque es preferente trabajar con las versiones en las que se ha desarrollado, también es compatible con otras.

Para el caso de MacOS, gracias a la ayuda de mi tutor, he podido comprobar que tanto la *toolbox* como el modelo en CoppeliaSim son compatibles con este sistema operativo.

Para poder comprobar la compatibilidad del proyecto utilicé otro ordenador, con una distribución de Ubuntu, en la que instalé los dos programas utilizados MATLAB y

CoppeliaSim. A la hora de instalar la *toolbox*, la instalé por comandos, pero el resultado finalmente fue satisfactorio.



(a) MacOS. (b) Ubuntu.

Figura 5.3: Compatibilidad SO's.

Capítulo 6

Conclusión

Para el desarrollo de este proyecto se había propuesto una serie de objetivos, con el fin de tener una buena experiencia de simulación; entre los objetivos principales encontramos el modelado 3D del robot o el uso de MATLAB para manejo del robot simulado y de la comunicación, entre otros. Una vez cumplidos los objetivos propuestos, se pueden extraer las conclusiones del proyecto.

Primero, el modelado 3D del robot ha sido relativamente sencillo, gracias a la herramienta seleccionada, ya que prácticamente era como montarlo en la vida real y tras montarlo en el laboratorio, se tenía clara ya la orientación de la mayoría de las piezas.

Segundo, se puede afirmar que la parte con la que se trabajó en el simulador fue de gran utilidad haber estado matriculado en la asignatura de Robótica Móvil, ya que se partía de una gran base en el simulador a falta de la comunicación API.

Tercero, el desarrollo de la comunicación entre MATLAB y CoppeliaSim no habría sido sido posible sin el trabajo previo realizado por Alberto Martín [13].

Cuarto, el uso de MATLAB como herramienta para el desarrollo de las funciones para el manejo del robot, ha sido muy acertada, ya que dispone de una gran compatibilidad con CoppeliaSim y su uso está muy generalizado en el mundo de la ingeniería, lo que facilitará la adaptación del usuario final al simulador.

Quinto, en las pruebas realizadas al simulador, no se han encontrado fallos, además los elementos de aleatoriedad incorporados a los sensores, le proporciona un acercamiento al modelo real, haciéndolo ideal para trabajar en ausencia de éste.

Sexto, con el fin de comprobar que el simulador puede utilizarse en la mayor cantidad

de equipos posible, se han realizado pruebas satisfactorias de su uso, tanto en Windows como Ubuntu y MacOS.

En conclusión, se puede decir que este simulador cumple con todos los objetivos propuestos para este proyecto, resultando en un simulador muy completo.

1. Mejoras

A continuación se hablará sobre las posibles mejoras que se pueden incorporar con el fin de que el usuario tenga la mejor experiencia posible, si se dispusiera de una mayor cantidad de tiempo.

- Mejorar la animación de las ruedas, probando a modelar el robot con otro programa de diseño 3D.
- Añadir más funciones de las que se han considerado necesarias para el fin con el que ha sido enfocado este robot, como pueden ser añadir más funciones dentro del grupo Display o algunas funciones más específicas del tipo Motor Commands.
- Incorporar otra forma de introducir la aleatoriedad al modelo, ya que con el código actual solo suma y sería interesante que también restara para que resulte en una dispersión de los datos más real.

Parte II

Planos

Capítulo 7

Planos

En este apartado vamos a ver el modelo 3D realizado con la herramienta de diseño Studio 2.0 y exportado gracias a la ayuda de otra herramienta para diseño de modelos LEGO, LeoCAD, con la cual se exportó el modelo a un formato compatible con CoppeliaSim. A la hora del modelado, en este modelo como se ha mencionado anteriormente, parte de las piezas originales que contiene el kit 9797 de LEGO. En las imágenes que acompañan al modelo, las unidades de diseño en los programas son el LDU, 1 LDU (*LDraw Unit*), equivale a 0,4 mm [28], por lo que el modelo en las imágenes es más pequeño que el original, pero después se convirtió a escala 1/1 al importarlo en CoppeliaSim.

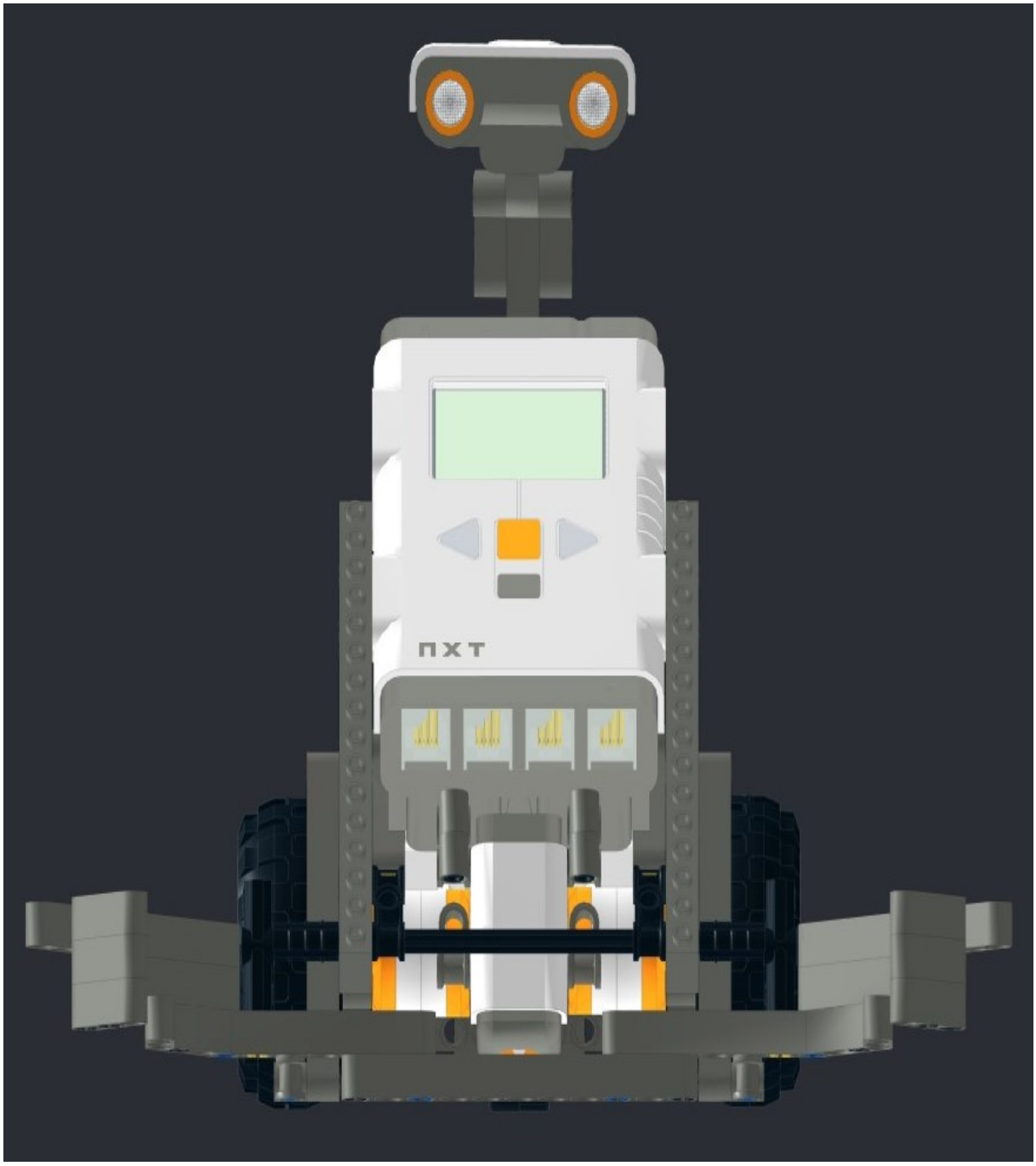


Figura 7.1: Alzado.

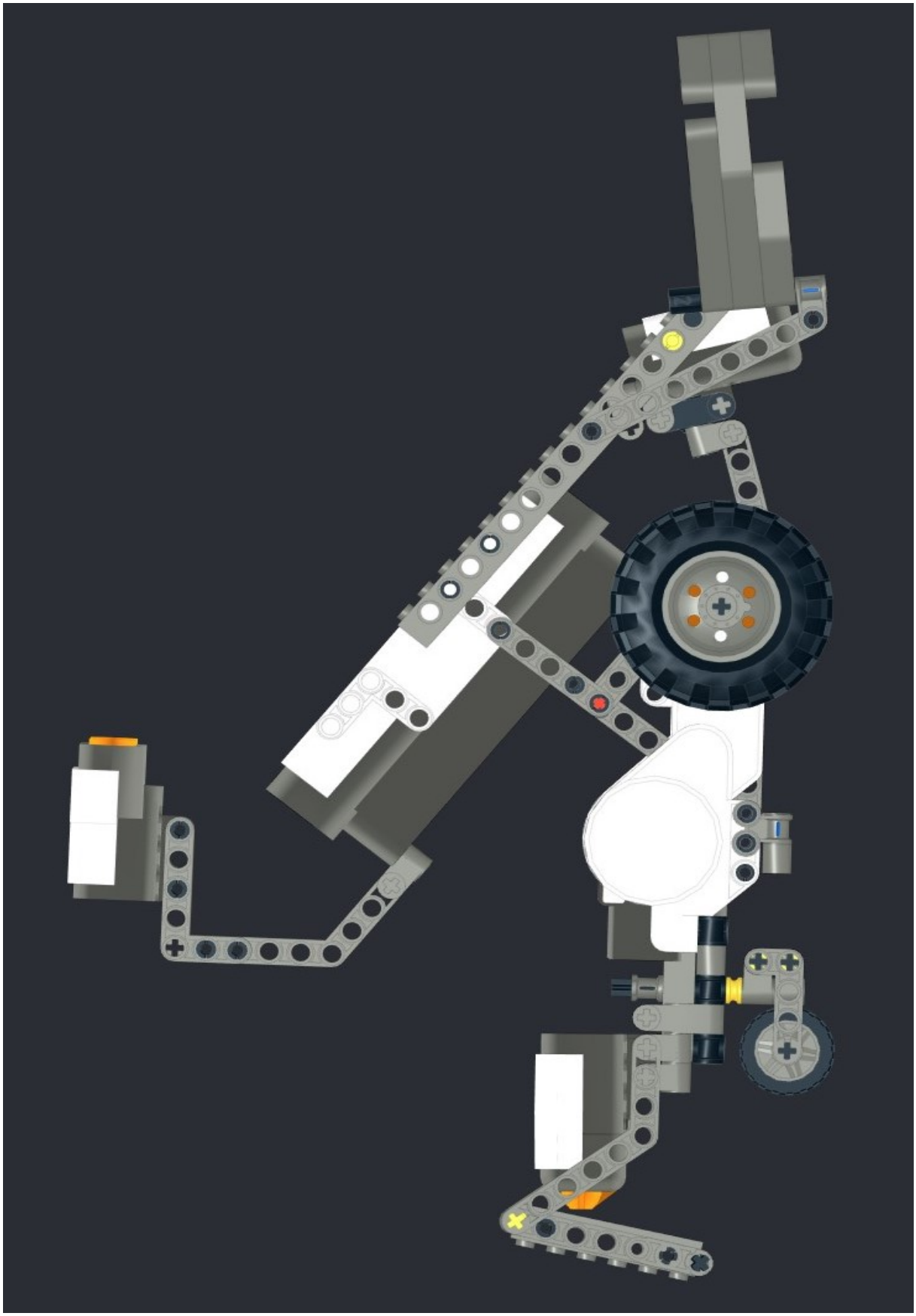


Figura 7.2: Perfil izquierdo.

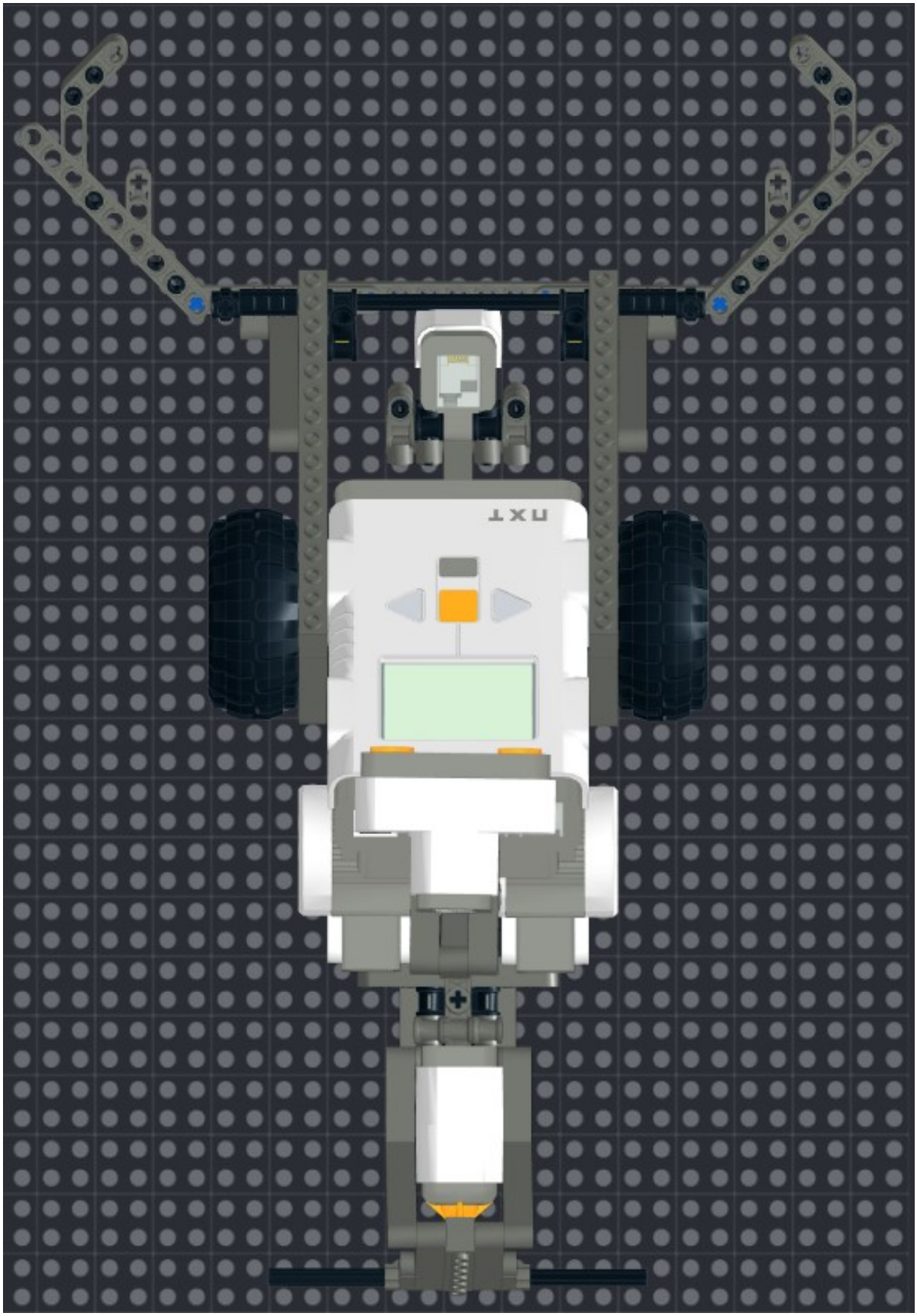
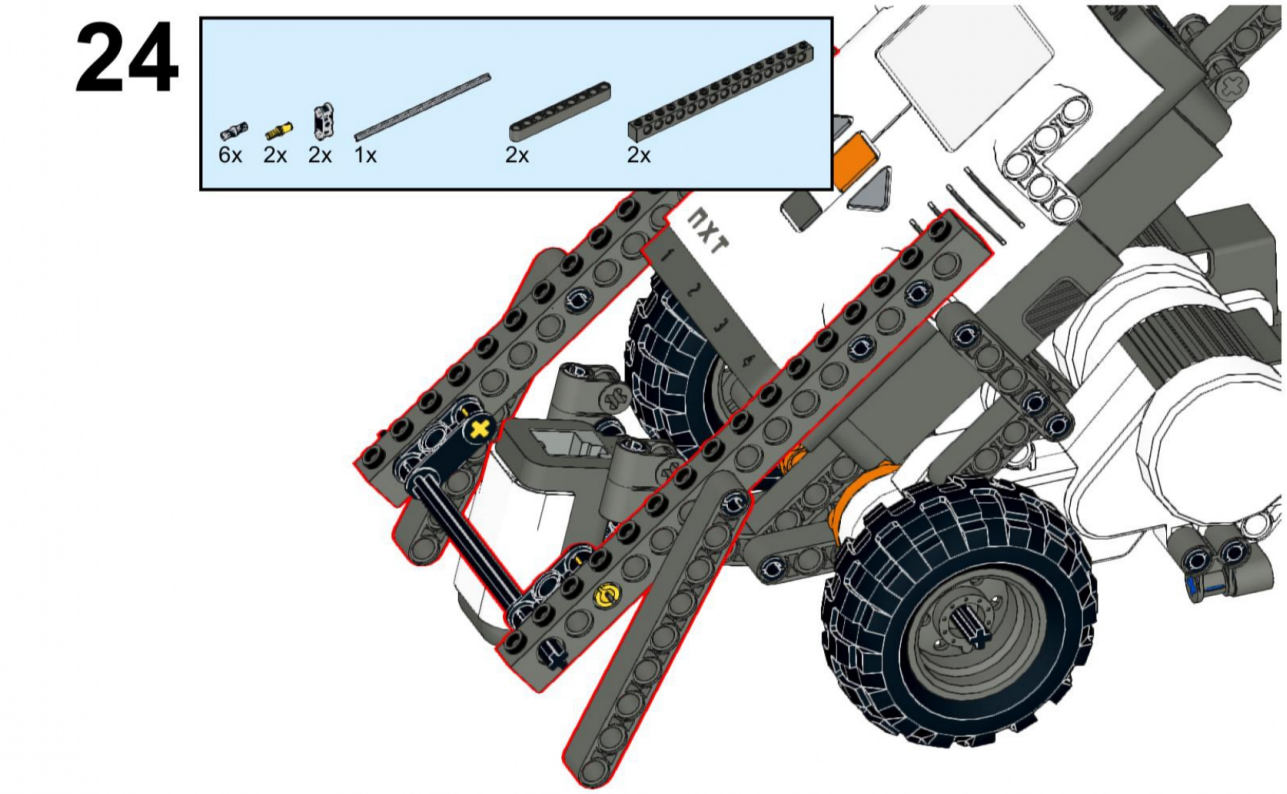


Figura 7.3: Planta.

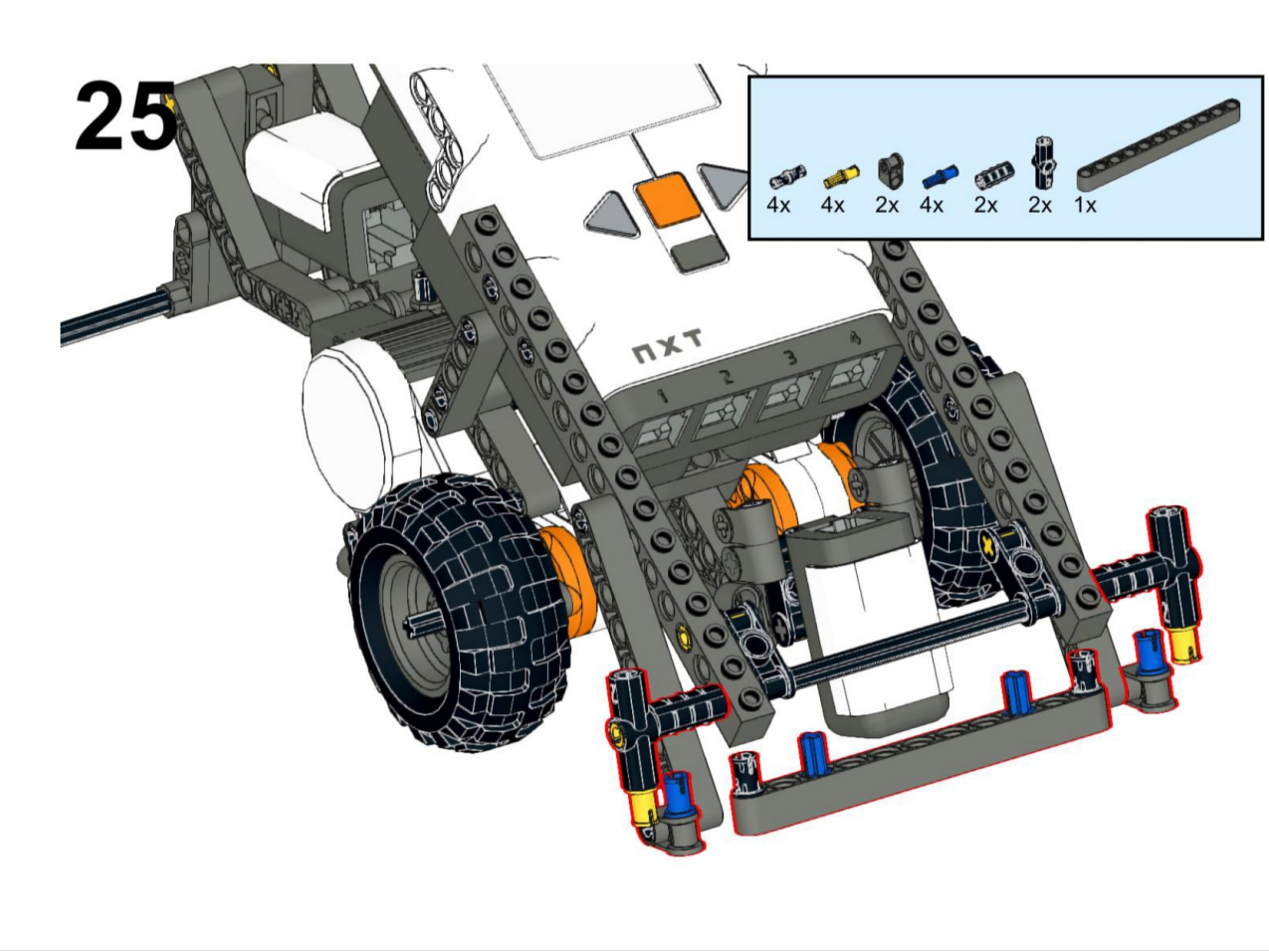
1. Montaje

Para realizar el montaje, se parte de las instrucciones que acompañan al set 9797 [2] de LEGO, hasta el apartado 23, después se realizan los pasos que se detallan en las instrucciones que vienen a continuación, y por último se seguirá con los pasos 26, 27 y 28 del manual de instrucciones.

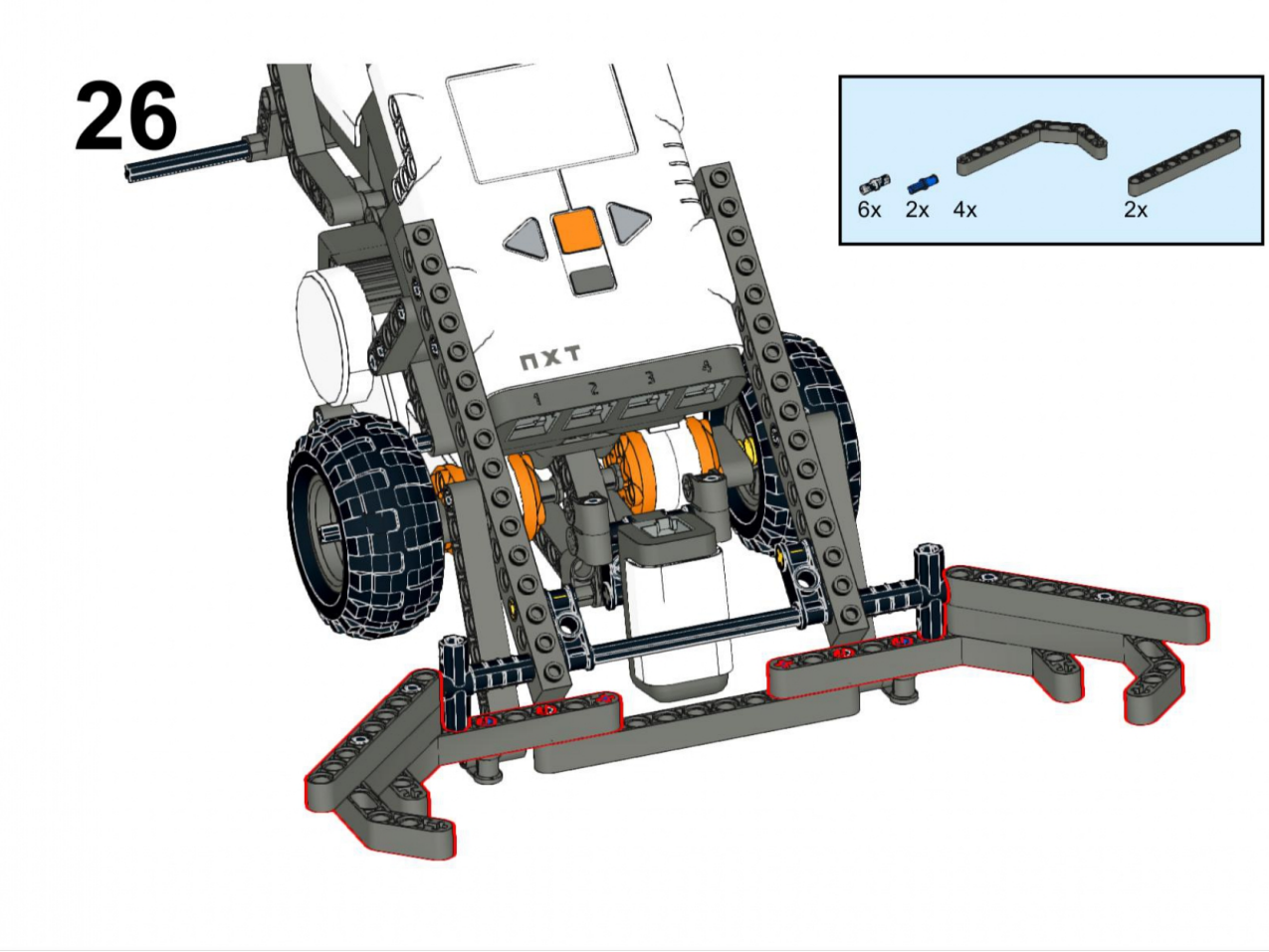
1.1. Montaje de la pala paso 1: estructura exterior



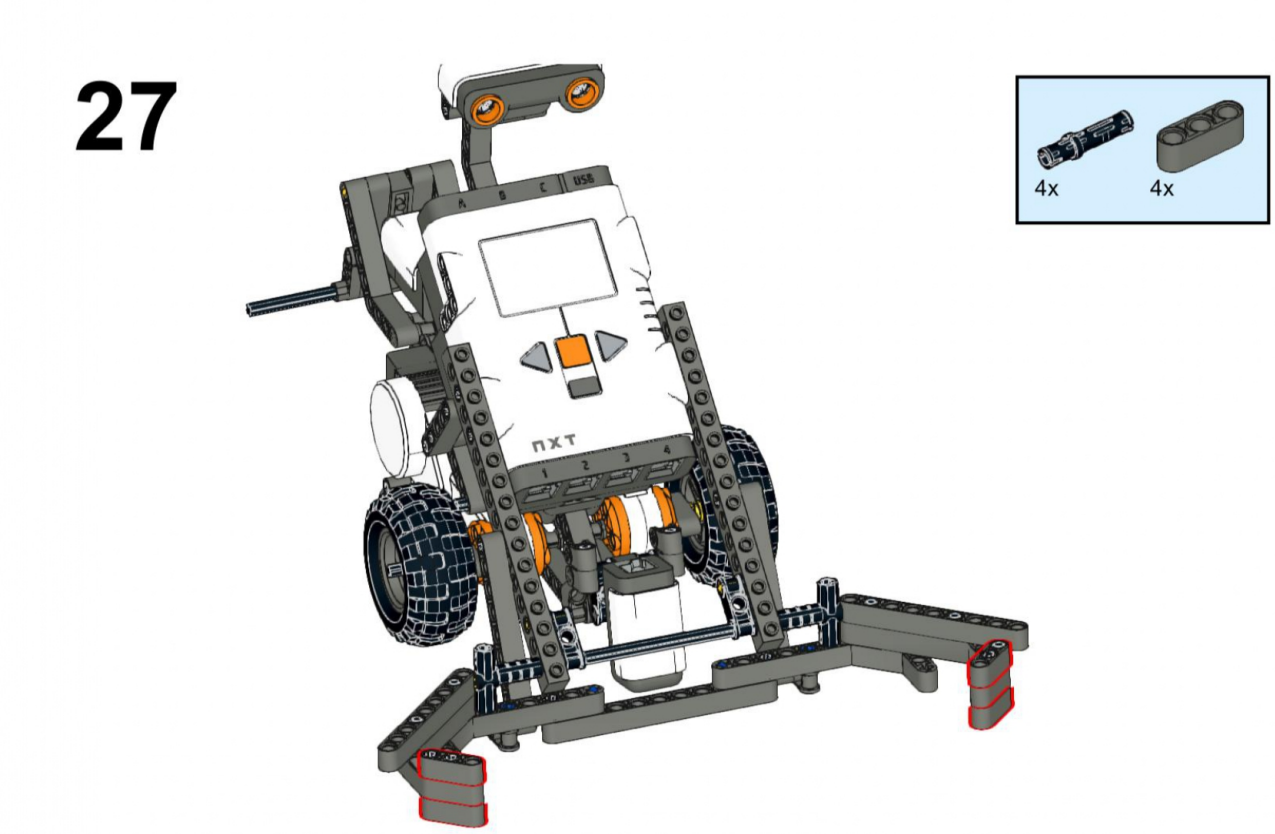
1.2. Montaje de la pala paso 2: estructura frontal



1.3. Montaje de la pala paso 3: abrazaderas



1.4. Montaje de la pala paso 4: borde abrazaderas



Parte III

Pliego de condiciones

Capítulo 8

Pliego de condiciones

1. Objeto

La presente especificación técnica se refiere al proyecto de simulación del robot NXT mediante la herramienta de simulación CoppeliaSim y controlado por medio de MATLAB. Se incluirá por tanto el desarrollo del modelo 3D, su implementación en el simulador, la configuración del modelo en el simulador, el desarrollo de las funciones para controlarlo mediante MATLAB.

2. Características

2.1. Objetivos

El simulador deberá cubrir los siguientes objetivos:

- Contar con un lenguaje parecido a RobotC.
- Aprovechar la API de la que dispone CoppeliaSim.
- Poder exportar los datos a ficheros externos.

2.2. Alcance

El alcance del presente pliego, es el de adquirir un sistema de simulación de los robots LEGO de los que dispone la ETSID, debe satisfacer las siguientes necesidades:

- Ofrecer una experiencia de uso parecida al modelo real.
- Compatibilidad en el lenguaje de programación con el modelo real.
- Poder ejecutar el simulador en cualquier ordenador que cumpla con los requisitos de CoppeliaSim.

3. Condiciones de uso, mantenimiento y ejecución

El ingeniero solo es responsable del desarrollo del modelo en el simulador y de las funciones para poder trabajar con él. Dada la temporalidad en la que se realiza este proyecto las versiones (Sección 2) de software que se están utilizando son las que el usuario final deberá instalar en su equipo con el fin de poder simular el robot. Aunque dada la gran compatibilidad de CoppeliaSim se puede valorar el uso de las versiones futuras por parte del usuario final, pero sin garantizar un funcionamiento acorde a lo descrito en este documento. Tampoco se deben cambiar las funciones de MATLAB para evitar errores.

3.1. Software

Las versiones que se deben instalar de software, son las siguientes:

- MATLAB R2020a, ya que es la versión con la que se han desarrollado las funciones de control, y la cual es la versión con la que se ha creado la toolbox que contiene las funciones y las librerías de la API.
- CoppeliaSim v4.1, ya que a pesar de la compatibilidad entre las versiones, se mejor si se usa la versión para la que fue desarrollado, sobre todo por cambios en las funciones de la API.

3.2. Hardware

Las recomendaciones de hardware son un poco globales. El ordenador con el que trabaje debe tener mínimo un procesador i5 de 7^{a} generación en adelante, y espacio en el disco duro suficiente, al menos 15 GB disponibles, para poder instalar los dos programas

y un directorio raíz en el que trabajar. Además, es recomendable que el equipo monte una tarjeta gráfica con un mínimo de 2GB de VRAM, la cual nos aportará una mayor cantidad de fotogramas, obteniendo una mayor fluidez en las simulaciones. En otros equipos se puede mover el simulador, pero la fluidez no está garantizada.

4. Pruebas de servicio

El simulador será entregado tras haber pasado satisfactoriamente las pruebas mencionadas en la memoria (Sección 1), y tras validar las pruebas realizadas con el modelo real. También se comprobará que la temporalidad del simulador se corresponda con la realidad.

Parte IV

Presupuesto

Capítulo 9

Presupuesto

El presupuesto se muestra dividido en los materiales, la mano de obra así como los precios unitarios y los precios unitarios desglosados.

1. Amortización

En esta sección se calcula el coste de amortización del ordenador utilizado así como de la licencia utilizada y los materiales fungibles.

Las días de trabajo, utilizados en los cálculos que están a continuación equivalen a una jornada laboral de 8 horas.

El precio de adquisición del ordenador fueron 550 €. Si se tiene en cuenta que se amortizan en un plazo de 4 años, el precio por año es de 137,5 € por año, y el precio por día es de 0,37€.

$$Coste_{ordenador} = \frac{550}{365 \cdot 4} = 0{,}37 €/día \tag{9.1}$$

El precio de la licencia anual de MATLAB asciende a 800€ por año, por lo que el precio por día es de 2,19 €.

$$Coste_{MATLAB} = \frac{800}{365} = 2{,}19 €/día \tag{9.2}$$

El precio del material fungible es de 4,8 € repartidos entre folios, bolígrafos etc. Teniendo en cuenta que el proyecto ha durado 48 días, el coste por día del material

fungible es de 0,1 €.

$$Coste_{Material fungible} = \frac{4{,}8}{48} = 0,1 €/h \tag{9.3}$$

2. Materiales

En este apartado encontraremos tanto los materiales físicos como el ordenador y materiales fungibles, como las licencias utilizadas para el desarrollo del proyecto.

Código	Material	Precio (€/día)	Cantidad (días)	Total
EDU	CoppeliaSim 4.2	0€	8,75	0€
FNG	Materiales fungibles	0,1€	48	4,8€
MTB	MATLAB 2020	2,19€	31,25	68,44 €
LEO	LeoCAD	0€	0,125	0€
LTX	LaTeX	0	7,5	0€
PC	Ordenador	0,37	48,25	17,85€
STD	Studio 2.0	0€	0,625	0€
			Total materiales	91.09€

Tabla 9.1: Desglose materiales.

3. Mano de obra

Código	Mano de obra	Precio(€/día)	Días	Total
MO	Graduado en Ingeniería Electrónica Industrial y Automática	105,2€	48,25	5075,9€
			Total Mano de Obra	634,49€

Tabla 9.2: Mano de Obra.

4. Precios Unitarios

Código	Descripción	Precio
MD	Modelado 3D	80,85€
IM	Integración del modelo 3D	837,76€
MF	Diseño de las funciones en MATLAB	3448,12€
PF	Pruebas de funcionamiento	110,02€

Tabla 9.3: Precios unitarios.

5. Descomposición precios unitarios

MD	**Modelado 3D**	**Precio(€/día)**	**Días**	**Total**
MO	Graduado en Ingeniería Electrónica Industrial y Automática	105,2€	0,75	78,9€
MF	Materiales fungibles	0,1	0,75	0,075€
PC	Ordenador	0,37€	0,75	0,28€
%	Costes Directos Complementarios	2 %	79,26	1,59€
		Precio total por u.		80,85€
IM	**Integración del modelo 3D**	**Precio(€/día)**	**Días**	**Total**
MO	Graduado en Ingeniería Electrónica Industrial y Automática	105,2€	7,75	815,3€
MF	Materiales fungibles	0,1	7,75	0,775€
PC	Ordenador	0,37€	7,75	2,87€
%	Costes Directos Complementarios	2 %	821,33	16,43€
		Precio total por u.		837,76€
MF	**Diseño de las funciones en MATLAB**	**Precio(€/día)**	**Días**	**Total**
MO	Graduado en Ingeniería Electrónica Industrial y Automática	105,2€	31,25	3287,5€
MF	Materiales fungibles	0,1	31,25	3,13€
MTB	MATLAB	2,19	31,25	68,44€
PC	Ordenador	0,37€	31,25	11,56€
%	Costes Directos Complementarios	2 %	3380,51	67,61€
		Precio total por u.		3448,12€
PF	**Pruebas funcionales**	**Precio(€/día)**	**Días**	**Total**
MO	Graduado en Ingeniería Electrónica Industrial y Automática	105,2€	1	105,2€
MF	Materiales fungibles	0,1	1	0,1€
MTB	MATLAB	2,19	1	2,19€
PC	Ordenador	0,37€	1	0,37€
%	Costes Directos Complementarios	2 %	107,86	2,16€
		Precio total por u.		110,02€

Tabla 9.4: Descomposición Precios Unitarios.

6. Medición y Presupuesto

Código	Descripción	Medición	Precio Unitario	Importe
MD	Modelado 3D	1 u	80,85€	80,85€
IM	Integración del modelo 3D	1 u	837,76€	837,76€
MF	Diseño de las funciones en MATLAB	1 u	3448,12€	3448,12€
PF	Pruebas funcionales	1 u	110,02€	110,02€
			Presupuesto de Ejecución Material (PEM)	4476,75€
			13 % de Gastos Generales	581,98
			6 % de Beneficio Industrial	268,61
			Suma	5327,34€
			21 % de IVA	1118,74€
			Presupuesto de Ejecución por Contrata (PEC)	6446,08€

Tabla 9.5: Presupuesto.

El presupuesto de ejecución por contrata asciende a la cantidad de SEIS MIL CUATROCIENTOS CUARENTA Y SEIS EUROS CON OCHO CÉNTIMOS.

Valencia, Junio de 2021

Graduado en Ingeniería en Electrónica Industrial y Automática

Luis Ortiz Enguix

Parte V

Anejos

Capítulo 10

Manual de usuario

A continuación se va a detallar el proceso para la instalación la instalación del modelo y de la *toolbox* que lo controla, así como su utilización para desarrollos futuros por parte del usuario. Todos los archivos que se mencionan, así como las escenas y los códigos de los ejemplos del capítulo resultados (Capítulo 5), se pueden encontrar en: https://drive.google.com/drive/folders/1uDPy5y9a0GC6I7apFwaA2JgAQvYE51o8?usp=sharing

1. Instalación del software

En esta sección se hablará de dónde conseguir el software necesario y cuáles son las versiones que se deben instalar, tanto de MATLAB como del CoppeliaSim. En los archivos adjuntos al TFG se encuentran todos los ficheros que se han desarrollado y que se usarán a continuación.

1.1. MATLAB

Para poder instalar MATLAB, el primer paso es acceder a la web de software de la UPV, https://software.upv.es/index.php, dentro de la web encontraremos las instrucciones de como registrarnos en la web de mathworks, donde podremos encontrar las diferentes versiones de MATLAB. Dentro de las versiones, se deberá descargar la versión de MATLAB R2020a, que es con la que se ha desarrollado este trabajo.

1.2. CoppeliaSim

Para la instalación de CoppeliaSim, los pasos son parecidos, pero sin tener que entrar a la web de software upv, ya que tiene una versión de acceso libre para estudiantes y docentes, con todas las características, siempre y cuando no vaya a ser usado con fines comerciales. Para encontrar la versión con la que se ha desarrollado este proyecto, la v 4.1, se deberá acceder a https://coppeliarobotics.com/previousVersions, donde se encuentran las versiones anteriores de CoppeliaSim.

2. Modelo CoppeliaSim

Cuando ya se tiene todo el software necesario, el siguiente paso será importar el modelo a CoppeliaSim, donde ya podrá ser usado en escenas del programa.

Para importar el modelo, se deberá acceder a *File* → *Load model*, una vez ahí se abrirá el explorador de archivos de Windows, donde buscaremos nuestro modelo, el cual viene contenido en una escena, por ello se puede o modificar la escena en la que viene el modelo o extraer el modelo para añadirlo a otra escena que ya se tenga prediseñada. Para guardar el modelo, se deberá seleccionar el modelo en la *Scene hiearchy*, una vez está seleccionado, se volverá a acceder a *File*, pero esta vez se seleccionará *Save model as* → *CoppeliaSim model ....*

Ahora cuando se desee utilizar el modelo, se deberá clicar en *File* → *Load model ...* y ya podremos disfrutar del modelo del NXT en las escenas que se creen.

3. Instalación de la toolbox

En este paso se va a explicar cómo instalar la toolbox con la que se realizará el control del modelo una vez que se inicie la simulación.

Para instalar la toolbox, será tan fácil como acceder a la carpeta donde se encuentre el archivo (extensión *.mtlbx*) y clicar en él. Después de esto se abrirá MATLAB y después se abrirá el *Add-On Manager*. En éste aparecerá un aviso un aviso que indica que se ha realizado la instalación correctamente.

En caso de que no se consiguiera instalar la *toolbox* automáticamente, se puede abrir la carpeta en la que se encuentre el archivo con MATLAB e instalarla con los siguientes comandos en el *Command Window*.

```
toolboxFile = 'LEGO MindStorms NXT.mltbx';
installedToolbox = matlab.addons.toolbox.installToolbox(toolboxFile)
```

Por último otra opción es convertirla en un archivo *.zip* y extraer los ficheros. Con los ficheros extraídos en una carpeta, se puede trabajar en MATLAB con esa carpeta como raíz del programa, con lo que se tiene acceso a todas las funciones desarrolladas para el NXT.

4. Escenas de prueba

En los archivos adjuntos también encontraremos las escenas que se han utilizado para comprobar el funcionamiento del robot. Estas escenas se pueden usar para desarrollar el software propio con el que hacer pruebas antes de trabajar con el robot real.

5. Funciones y ficheros

Las diferentes funciones con las que se puede trabajar en MATLAB, para ejecutar la simulación, están inspiradas en las funciones disponibles en RobotC y son las siguientes:

API

Las funciones de conexión con la API se dividen en dos, las que proporciona el propio CoppeliaSim y las desarrolladas por Alberto Martín Domínguez en su TFG [13].

- **remApi.m, remoteApi.dll, remoteApi.so, remoteApi.dylib y remoteApiProto.m**: son los archivos proporcionados por CoppeliaSim para la conexión con el servidor mediante la API.
- **superRemoteApi**: Esta es la clase que maneja las comunicaciones con CoppeliaSim, y la cual será la clase padre de las siguientes clases.

- **coppRemoteApi**: inicia el cliente de API remota para el control de la simulación, es una clase hija de superRemoteApi. El puerto utilizado para la simulación es el 19997 que es el que viene por defecto. Si se diera algún fallo, se deberá comprobar en los archivos de programa que sea el mismo.

- **nxtRemoteApi**: inicia el cliente de API remota para el control del robot NXT. El puerto usado en este caso es el 19999, pero hay un amplio rango que se puede usar. Este es posible cambiarlo en el código embebido en el robot. Función modificada para hacerla compatible con las nuevas funciones.

Battery and Power Control

- **powerOff()**: Termina la simulación como si se apagase el robot real. Elimina las constantes y limpia el espacio de trabajo.

Buttons

- **getButtonPress(button)**: Devuelve si el botón especificado está pulsado o no. Los botones pulsados devolverán un valor de 1 y los botones sin presionar devolverán un valor de 0. Los argumentos de entrada que acepta son: *buttonRight, buttonEnter, buttonLeft y buttonBack.*

Connections

- **ConnectLightSensor(Puerto)**: Permite realizar la conexión virtual del sensor de luz a un puerto del robot. Los argumentos de entrada que acepta son: *S1, S2, S3 y S4.*

- **ConnectTouchSensor(Puerto)**: Permite realizar la conexión virtual del sensor de tacto a un puerto del robot. Los argumentos de entrada que acepta son: *S1, S2, S3 y S4.*

- **ConnectUltrasonicSensor(Puerto)**: Permite realizar la conexión virtual del sensor de ultrasonidos a un puerto del robot. Los argumentos de entrada que acepta son: *S1, S2, S3 y S4.*

Constants

- **constantes.m**: En este fichero encontraremos las constantes que se usarán en las diferentes funciones, así como las llamadas a iniciar la simulación.

Inicio

- **iniciarSimulacion(script)**: Esta función es la encargada de iniciar la simulación.

LCD Display

- **eraseDisplay()**: Limpia la pantalla del ladrillo del NXT.
- **displayStringAt(x, y, string)**: Le da formato a una cadena de texto y la muestra en cualquier coordenada (X,Y) de la pantalla LCD. La coordenada y va de 1 a 8.

Motor Commands

- **getMotorEncoder(motor)**: Devuelve el valor del codificador del motor seleccionado. Los argumentos de entrada que acepta son: *motorC y motorB.*
- **resetMotorEnconder(motor)**: Restablece el valor del codificador del motor que se indique. Los argumentos de entrada que acepta son: *motorC y motorB.*
- **setMotor(motor, velocidad)**: Activa el motor que se seleccione a una velocidad indicada entre -100 y 100. Los argumentos de entrada que acepta son: *motorC y motorB.*
- **stopMotor(motor)**: Detiene el motor indicado. Las argumentos de entrada que acepta son: *motorC, motorB.*
- **setMultipleMotor**: Esta función permite seleccionar dos motores y ponerlos a trabajar a la misma velocidad. Los argumentos de entrada que acepta son: *motorC, motorB.*
- **stopAllMotors()**: Detiene todos los motores. No tiene argumentos de entrada.

Sensors

- **getLightValue(Puerto)**: Los valores devueltos por el sensor de color oscilan entre 0 y 100. Los objetos oscuros devuelven valores más bajos mientras que los objetos claros devuelven valores más altos. Los argumentos de entrada que acepta son: *S1, S2, S3 y S4* y tendrán que coincidir con el nombre usado en las funciones de conexión.

- **getTouchValue(Puerto)**: Devuelve el valor del sensor táctil conectado al ladrillo en el puerto seleccionado en su conexión. Esta función devuelve un 1 lógico si el sensor táctil está pulsado mientras que devuelve un 0 lógico si el sensor táctil no está pulsado. Los argumentos de entrada que acepta son: *S1, S2, S3 y S4* y tendrán que coincidir con el nombre usado en las funciones de conexión.

- **getUSDistance(Puerto)**: Devuelve el valor del sensor táctil conectado al ladrillo. La distancia es devuelta en cm. El valor máximo que devuelve es 255 que puede ser o la distancia máxima o un error. Los argumentos de entrada que acepta son: *S1, S2, S3 y S4* y tendrán que coincidir con el nombre usado en las funciones de conexión.

Timing

- **delay(tiempo)**: La ejecución del programa esperará el número especificado de ms.

- **getTimer()**: Devuelve el tiempo de ejecución en ms, el cual solo se corresponderá con la realidad si está seleccionado el modo *Real Time* en el simulador.

Capítulo 11

Pruebas

1. Código siguelíneas

```
1  %% Esperar al bot n central de robot para empezar
2  displayStringAt(0, 1, 'Programa siguelineas');
3  displayStringAt(0, 2, 'Presione el boton central');
4  displayStringAt(0, 3, 'para comenzar');
5  while(¬getButtonPress(buttonEnter))
6  end % Espera a pulsar el boton central
7  eraseDisplay();   % Limpia la pantalla
8
9  pause(1);
10
11 resetMotorEncoder(motorB); % Resetea el encoder rueda derecha
12 resetMotorEncoder(motorC); % Resetea el encoder rueda izquierda
13
14 %% Inicializaci n de los sensores
15
16 connectLightSensor(S1); %Sensor de luz en el puerto 1
17
18 %% Bucle de control
19
20 t1 = double(getTimer())   % Guarda el primer tiempo para poder ...
       calcular el tiempo de ejecucion
```

```
21 luz_linea = 6;   % Declara la variable luz_linea y la inicializa a 6
22 luz_suelo = 45;  % Declara la variable luz_suelo y la inicializa a 54
23 gris = (luz_suelo+luz_linea)/2;   % Calcula gris
24 Kp = 0.99;   % Declara el valor de Kp 0.85
25 Ki = 0.0000007;  % Declara el valor de Ki 0.0000012
26 Kd = -0.25;   % Declara el valor de Kd -0.142
27 error_anterior = 0;  % Declara error_anterior y la inicializa a 0
28 integral = 0;  % Declara integral y la inicializa a 0
29 derivada = 0;  % Declara derivada y la inicializa a 0
30 error = 0;     % Declara error y la inicializa a 0
31
32 while(¬getButtonPress(buttonEnter))   % Mientras no se pulse el boton ...
       central
33
34     luz = getLightValue(S1);   % Declara e inicializa la variable luz
35     error = gris - luz;   % Calcula el valor del error
36     integral = error + integral;  % Calcula el error integral
37     derivada = error - error_anterior;  % Calcula el error derivativo
38
39     limite_integral = 2000;   % Fija un l mite al error integral
40
41     if(integral > limite_integral)   % Si el error integral es mayor ...
           al limite
42         integral = limite_integral;
43     else if (integral < -limite_integral)   % Sino si el error ...
           integral es menor al limite negativo
44         integral = -limite_integral;
45         end
46     end
47
48     error = Kp*error + Ki*integral + Kd*derivada;   % Calcula el ...
           error total
49
50     limite_velocidad = 100;   % Declara el limite de velocidad del LEGO
51
52     velocidad_izq = 45 - error;   % Calcula la velocidad del motor ...
           izquierdo
```

```
53      velocidad_der = 45 + error;   % Calcula la velocidad del motor ...
            derecho
54
55      if(velocidad_izq > limite_velocidad)   % Si la velocidad del ...
            motor izquierdo es mayor al limite
56          velocidad_izq = limite_velocidad;
57      else if(velocidad_izq < -limite_velocidad)  % Si la velocidad del ...
            motor izquierdo es menor al limite negativo
58          velocidad_izq = -limite_velocidad;
59          end
60      end
61
62      if(velocidad_der > limite_velocidad)   % Si la velocidad del ...
            motor derecho es mayor al l mite
63          velocidad_der = limite_velocidad;
64      else if(velocidad_der < -limite_velocidad)   % Si la velocidad ...
            del motor derecho es menor al l mite negativo
65          velocidad_der = -limite_velocidad;
66          end
67      end
68
69      if( luz < 9)   % Si la luz es menor a 9
70          setMotor(motorB,10);  % Motor derecho
71          setMotor(motorC,-10);  % Motor izquierdo
72      else
73          setMotor(motorB,velocidad_der);  % Motor derecho
74          setMotor(motorC,velocidad_izq);  % Motor izquierdo
75      end
76
77      error_anterior = error;   % Guarda el error anterior, para ...
            calcular el error total
78
79      mensaje = ['Luz = ' num2str(double(getLightValue(S1)))];   % ...
            Muestra en la linea 1 el valor de la luz
80      displayStringAt(0, 1, mensaje);
81      mensaje = ['Integral = ' num2str(double(integral))]; % Muestra ...
            en la linea 2 el valor del error integral
```

```
82      displayStringAt(0, 2, mensaje);
83
84  end
85
86  stopAllMotors();
87  t2 = double(getTimer());    % Guarda el ultimo tiempo para poder ...
        calcular el tiempo de ejecuci n
88  tiempo = ['Tiempo = ' num2str((t2-t1)/1000)];   %Tiempo en segundos
89  display(tiempo);   % Muestra el tiempo en la consola de MATLAB
90
91  pause(1);      % Hace una pausa de 1 ms
```

2. Código limpia obstáculos

```
1   %% Esperar al bot n central de robot para empezar
2   displayStringAt(0, 1, 'Programa limpiador');     % Escribe texto en ...
        la l nea 1
3   displayStringAt(0, 2, 'Presione el boton central');      % Escribe ...
        texto en la l nea 2
4   displayStringAt(0, 3, 'para comenzar');      % Escribe texto en la ...
        l nea 3
5
6   %% Esperar al boton central de robot para empezar
7   while(¬getButtonPress(buttonEnter))
8   end  % Espera a pulsar el boton central
9   eraseDisplay();  % Limpia la pantalla
10
11  displayStringAt(0, 1,'PROGRAMAS LIMPIADOR');
12  displayStringAt(0, 3,'Elija el programa que desee');
13  displayStringAt(0, 5,'Boton izquierdo = Bordes');
14  displayStringAt(0, 7,'Boton derecho = Centrados');
15
16  t1 = double(getTimer())   % Guarda el primer tiempo para poder ...
        calcular el tiempo de ejecuci n
```

```
17 eraseDisplay();   % Limpia la pantalla
18
19 pause(1); % Hace una pausa de un 1 ms
20
21 resetMotorEncoder(motorC); % Resetea el encoder rueda izquierda
22 resetMotorEncoder(motorB); % Resetea el encoder rueda derecha
23
24 %% Inicializaci n de los sensores
25
26 connectLightSensor(S1); %Sensor de luz en el puerto 1
27 connectUltrasonicSensor(S2);%Sensor ultrasonidos en el puerto 2
28
29 %% Bucle de control
30
31 objeto = 0;   % Inicializa la variable objeto a 0
32 vuelta = 0;   % Inicializa la variable vuelta a 0
33
34 while(¬getButtonPress(buttonBack)) %Mientras no se pulse el Bot n ...
       Exit se ejecutar lo siguiente
35
36     luz = getLightValue(S1); % Lee y almacena el valor de la ...
           luminosidad del suelo
37     distancia = getUSDistance(S2); % Lee y almacena el valor de la ...
           distancia
38
39     if (luz ≥ 30)
40         setMotor(motorB, -60); % motor derecho
41         setMotor(motorC, -30); % motor izquierdo
42         delay(700); % pausa de 0.75s
43         objeto = 0;      % Objeto = 0 por lo que no ha detectado ...
               ning n objeto
44         vuelta = 0;      % Reinicia la variable vuelta
45     elseif (distancia < 255) || (objeto == 1)
46         setMotor(motorB, 100);    % motor derecho
47         setMotor(motorC, 100);     % motor izquierdo
48         objeto = 1;          %Ha detectado un objeto
49         vuelta = 0;          %Reinicia la variable vuelta
```

```
50          if (distancia > 20 && distancia < 255) % Si la distancia es ...
                menor a 255 y mayor a 20 entra en el bucle
51              objeto = 0;      % Objeto = 0 por lo que no ha detectado ...
                    ning n objeto
52          end
53      else
54          setMotor(motorB, 40);   % motor derecho
55          setMotor(motorC, 100); % motor izquierdo
56          vuelta = vuelta + 1; % Suma uno a la variable vuelta
57      end
58
59      if (vuelta ≥ 350) % Si acumula 350 en vuelta entramos en el bucle
60          setMotor(motorB, 100); % motor derecho
61          setMotor(motorC, 80);   % motor izquierdo
62      end
63
64      displayStringAt(0, 1, 'Programa bordes'); % Escribimos texto en ...
            la l nea 1
65      displayStringAt(0, 3, ['Luz = ' num2str(double(luz))]);    % ...
            Escribimos texto en la l nea 3
66      displayStringAt(0, 4, ['Distancia = ' ...
            num2str(double(distancia))]); % Escribimos texto en la l nea 4
67      displayStringAt(0, 6, ['Objeto = ' num2str(double(objeto))]);   ...
            % Escribimos texto en la l nea 6
68      displayStringAt(0, 7, ['Vuelta = ' num2str(double(vuelta))]);   ...
            % Escribimos texto en la l nea 7
69
70  end
71
72  stopAllMotors();
73  t2 = double(getTimer()); %Guarda el segundo tiempo para poder ...
        calcular el tiempo de ejecuci n
74  tiempo = ['Tiempo = ' num2str((t2-t1)/1000)];  %Tiempo en segundos
75  display(tiempo); %Muestra en la consola de comandos de MATLAB el ...
        tiempo de ejecuci n
76
77  pause(1)    % Para tras un delay de 1 ms
```

Capítulo 12

Bibliografía

[1] Lego mindstorms 8547 - nxt 2.0 v24. URL: https://www.amazon.es/LEGO-Mindstorms-8547-NXT-2-0/dp/B001V7RF9U.

[2] LEGO. Instrucciones set 9797. URL: https://www.lego.com/es-es/service/buildinginstructions/9797.

[3] Xander. Comparing the nxt and ev3 bricks. URL: http://botbench.com/blog/2013/01/08/comparing-the-nxt-and-ev3-bricks/.

[4] Sensor ultrasónico nxt. URL: https://www.pinterest.com.mx/pin/588142032563757133/.

[5] LEGO. Sensor de tacto. URL: https://www.lego.com/es-es/product/touch-sensor-9843.

[6] Los motores del lego mindstorms nxt. URL: http://rbtnxt.blogspot.com/2009/06/los-motores-del-lego-mindstorms-nxt.html.

[7] Desarrollo histórico y evolución de la robótica. URL: http://www.udesantiagovirtual.cl/moodle2/mod/book/tool/print/index.php?id=24899.

[8] Nuria Martinez Medina. Joseph marie jacquard. URL: https://www.rtve.es/noticias/20110923/jacquard-tejedor-informatico/463573.shtml.

[9] Robots. URL: https://en.wikipedia.org/wiki/Robot#Origin_of_the_term_'robot'.

[10] Lego mindstorms. URL: https://juegosrobotica.es/lego-mindstorm/#.

[11] Mindstorms. URL: https://en.wikipedia.org/wiki/Lego_Mindstorms.

[12] LEGO. Set spike™ prime de lego® education. URL: https://www.lego.com/es-es/product/lego-education-spike-prime-set-45678.

[13] Alberto Martín Domínguez. Modelado y simulación de un robot lego mindstorms ev3 mediante v-rep y matlab. URL: https://riuma.uma.es/xmlui/bitstream/handle/10630/12980/AlbertoMartinDominguezMemoria.pdf?sequence=1.

[14] BrickLink. Studio 2.0. URL: https://www.bricklink.com/v2/build/studio.page.

[15] Leocad. URL: https://www.leocad.org/.

[16] Bricklink. URL: www.bricklink.com.

[17] Cyberbotics. URL: https://cyberbotics.com/.

[18] Coppellia. URL: https://www.coppeliarobotics.com/.

[19] CoppeliaRobotics. Regular api reference. URL: https://coppeliarobotics.com/helpFiles/en/apiFunctions.htm.

[20] Matlab. URL: https://es.mathworks.com/products/matlab.html.

[21] MATLAB. Create and share toolboxes. URL: https://es.mathworks.com/help/matlab/matlab_prog/create-and-share-custom-matlab-toolboxes.html.

[22] Python. URL: https://www.python.org/.

[23] Luis España. Sensor ultrasónico nxt. URL: https://www.esmindstorms.com/sensor-ultrasonico-nxt/.

[24] CEEO. Nxt sensors. URL: http://www.legoengineering.com/nxt-sensors/.

[25] El sensor de luz. URL: http://rbtnxt.blogspot.com/2009/06/el-sensor-de-luz.html.

[26] Luis España. Sensor de luz nxt. URL: https://www.esmindstorms.com/sensor-de-luz-nxt/.

[27] Thomas Avery. Lego® 9v technic motors compared characteristics. URL: https://www.philohome.com/motors/motorcomp.htm.

[28] What are ldus and how do they relate to lego bricks? URL: https://bricks.stackexchange.com/questions/691/what-are-ldus-and-how-do-they-relate-to-lego-bricks.

[29] LEGO. Distance sensor lego specifications. URL: https://education.lego.com/v3/assets/blt293eea581807678a/blt64c2b9534cf10f68/5f8801b8bc43790f5c4389ea/techspecs_technicdistancesensor.pdf.

[30] CoppeliaRobotics. Proximity sensor detection parameter dialog. URL: https://www.coppeliarobotics.com/helpFiles/en/proximitySensorDetectionParameterDialog.htm.

[31] Descripción del material de robótica disponible para préstamos. URL: https://parapnte.educacion.navarra.es/prestamo-de-material-de-robotica/descripcion-del-material-de-robotica-disponible-para-prestamos/.

[32] Robotc a c programming language for robotics. URL: https://www.robotc.net/WebHelpMindstorms/index.htm.