TRABAJO DE DIPLOMA

En opción al Título de Ingeniero en Automática

Título

Simulación de robótica móvil empleando CoppeliaSim

Autor

Raúl Samón Pérez

Tutor

MSc. Michel Sanz Pérez

Noviembre, 2023

UNIVERSIDAD DE ORIENTE
Facultad de Ingeniería Eléctrica
Departamento de Automática

TRABAJO DE DIPLOMA

Título

Simulación de robótica móvil empleando CoppeliaSim

Autor

Raúl Samón Pérez

Tutor

MSc. Michel Sanz Pérez

Noviembre, 2023

UNIVERSIDAD DE ORIENTE | **Facultad de Ingeniería Eléctrica**

Departamento de Automática

Hago constar que el presente Trabajo de Diploma fue realizado en la Universidad de Oriente como parte de la culminación de estudios de la especialidad de Ingeniería en Automática, autorizando a que el mismo sea utilizado por la Institución para los fines que estime convenientes, tanto de forma parcial como total, y que además no podrá ser presentado en eventos, ni publicados sin autorización de la Universidad.

_____

Nombre y firma del autor

Los abajo firmantes certificamos que el presente trabajo ha sido realizado según acuerdo de la dirección de nuestro centro y el mismo cumple con los requisitos que debe tener un trabajo de esta envergadura referido a la temática señalada.

_____

Nombre y firma del autor

_____	_____
Nombre y firma del Tutor	Fecha
_____	_____
Nombre y firma del Jefe de Carrera	Fecha
_____	_____
Nombre y firma del Jefe de Departamento	Fecha

Pensamiento

“Donde haya un error que enmendar, enmiéndalo tú. Donde haya un esfuerzo que todos esquivan, hazlo tú. Sé tú el que aparta la piedra del camino”

Gabriela Mistral

Dedicatoria

Este trabajo es dedicado a todo aquel que, en sus ansias de adquirir conocimientos sobre robótica, tiene la necesidad de trabajar con la plataforma de simulación CoppeliaSim. Espero encuentre en sus páginas la información necesaria y la motivación para continuar investigando.

Agradecimientos

A todos los profesores que han aportado a mi formación académica y han incentivado mi amor por su profesión, en especial a mi tutor Michel Sanz quien me ha guiado durante estos meses de trabajo.

A mis amigos y compañeros de aula que me han acompañado en este viaje y han convertido esta etapa de sacrificio en una experiencia inolvidable.

A mis viejos amigos, casi todos separados por la distancia o las complicaciones de la vida, siempre tendrán un lugar en mi corazón.

A todos mis familiares por creer en mi en todo momento, por su preocupación y por regalarme momentos especiales que me dan las fuerzas para seguir adelante.

A mi novia por su paciencia, su amor y su apoyo incondicional durante este período.

A mi hermana, cuya fe en mí, me ha impedido bajar la cabeza en los momentos de dificultad.

A mis padres, por darme todas las condiciones que permitieron hacer realidad este sueño; por enseñarme el significado del honor, respeto y disciplina; por estar en todo momento y ser capaces de sacarme una sonrisa hasta en los momentos más complejos. Su sacrificio y humildad marcan el camino que siguen mis pasos. Jamás ha existido un hijo con la dicha de tener los padres que he tenido yo. Todo lo que soy es gracias a ustedes. Esta también es su obra.

Resumen

La robótica ha revolucionado nuestra sociedad en múltiples aspectos desde el sector industrial hasta el asistencial, posicionándose como una de las áreas más prometedoras de la tecnología. Su desarrollo e investigación por parte del personal de la Facultad de Ingeniería Eléctrica de la Universidad de Oriente, hoy en día se ve obstaculizado por la escasez de medios para la experimentación práctica en robótica móvil, limitaciones que pueden ser solventadas mediante el uso de una herramienta de simulación.

El presente trabajo pretende mostrar la plataforma de simulación de robótica CoppeliaSim como vía para el estudio y experimentación en temáticas relacionadas a la robótica móvil, para lo cual se desarrollan varias escenas con el fin de simular algoritmos de seguimiento de trayectorias con posibles aplicaciones reales.

En un primer momento se crea el modelo 3D del robot educativo EduRoMAA y cuatro escenas de prueba para probar el comportamiento del robot frente a los algoritmos simples de seguimiento de línea y de pared. Posteriormente se trabaja con el modelo del dron *"Quadcopter",* desarrollado e implementado en el software por el propio fabricante, y se desarrollan dos escenas para simular algoritmos de seguimiento de trayectorias como son el "Guiado óptico" y el "Guiado por GPS" aplicado a la inspección de parques solares fotovoltaicos.

Para la programación de los algoritmos se emplean scripts en Lua, un lenguaje fácilmente interpretado por CoppeliaSim y que presenta una extensa gama de funciones que pueden ser consultadas en el manual de referencia de este software.

Palabras clave: CoppeliaSim, robótica móvil, seguimiento de trayectoria, simulación.

Abstract

Robotics has revolutionized our society in multiple aspects from the industrial to the healthcare sector, positioning itself as one of the most promising areas of technology. Its development and research by the staff of the Faculty of Electrical Engineering of the Universidad de Oriente, today is hindered by the scarcity of means for practical experimentation in mobile robotics, limitations that can be solved through the use of a tool simulation.

This work aims to show the CoppeliaSim robotics simulation platform as a way to study and experiment on topics related to mobile robotics, for which several scenes are developed in order to simulate trajectory tracking algorithms with possible real applications.

Initially, the 3D model of the EduRoMAA educational robot and four test scenes are created to test the behavior of the robot against simple line and wall following algorithms. Subsequently, we work with the “Quadcopter” drone model, developed and implemented in the software by the manufacturer itself, and two scenes are developed to simulate trajectory tracking algorithms such as “Optical Guidance” and “GPS Guidance” applied. to the inspection of photovoltaic solar parks.

To program the algorithms, scripts are used in Lua, a language that is easily interpreted by CoppeliaSim and that presents a wide range of functions that can be consulted in the reference manual of this software.

Keywords: CoppeliaSim, mobile robotics, trajectory tracking, simulation.

Listado de Imágenes

Índice

Introducción

La robótica es una rama de la ingeniería que se encarga del diseño, construcción, operación y uso de robots. Un robot es una máquina automática programable capaz de realizar determinadas operaciones de manera autónoma y sustituir a los seres humanos en tareas pesadas, peligrosas o repetitivas [1].

Dentro de los tipos de robots se encuentran los robots móviles, los cuales poseen la capacidad de moverse en su entorno y no se fijan a una ubicación física [2]. Esta virtud les permite extender las aplicaciones con respecto a los robots fijos incapaces de poder realizar navegación autónoma. Esta área de la robótica desarrolla robots capaces de obtener información sobre el medio que los rodeas y actuar durante un tiempo prolongado sin intervención humana. Esto implica que el robot posea sensores para percibir el entorno y comportarse de forma adecuada [3]. En la actualidad se aplican una gran variedad de métodos para la planeación y seguimiento de trayectorias que permitan a los robots cumplir con su objetivo [4].

Lejos de solo desarrollarse en el ámbito industrial, en los últimos años la robótica ha experimentado un aumento notable de su incidencia en nuestras vidas privadas. En la actualidad está siendo utilizada en todos los ámbitos que nos rodean sirviendo para un amplio abanico de propósitos desde salvar vidas en medicina hasta combatir el fuego en un incendio. Su crecimiento vertiginoso en el sector doméstico y su papel clave en el mundo de la salud, transporte y la exploración del espacio, evidencian que la robotización de la sociedad continuará aumentando, por lo que constituye una necesidad preparar lo mejor posible a los estudiantes para esta nueva realidad.

Desde el punto de vista educacional, la robótica se puede definir como un área multidisciplinar que comprende áreas de ciencia, tecnología, ingeniería y matemáticas, entre otras. Es este el motivo por el cual se puede afirmar que no existe un único camino para sumergirse en ella. Una persona que trabaje en este campo debe poseer los conocimientos básicos de otras materias tales como la electrónica, informática, biotecnología, ciencia cognitiva, etc [5].

Con el estudio de la robótica móvil y sus diferentes configuraciones es posible desarrollar aplicaciones en el campo del entretenimiento, la salud, táctica militar, entre otras. Con el desarrollo de nuevas aplicaciones, y nuevos frentes de acción, la investigación en la robótica móvil juega un papel importante en el área académica [4].

En muchos países esta disciplina se va abriendo camino en todos los niveles educativos, desde primaria hasta estudios de posgrado, por lo que ya forma parte del currículo educativo siendo a su vez un foco importante de investigación. Existen una gran cantidad de trabajos investigativos a nivel mundial relacionados con la robótica en general. De ahí el llamado del presidente de la República Miguel Díaz-Canel Bermúdez a emplear la robótica para hacer más eficientes procesos productivos y de servicios logrando así llegar más rápido a soluciones, propiciando un mayor rendimiento y productividad [6].

Los investigadores de la Facultad de Ingeniería Eléctrica (FIE) de la Universidad de Oriente, atendiendo el llamado de su presidente y en su afán de mantenerse a la vanguardia de la enseñanza, también se han adentrado en el mundo de la robótica. Esto se evidencia en la realización de trabajos investigativos como es el “Diseño e implementación de un Entorno Virtual de un manipulador ArmX de cuatro grados de libertad” [7], en el cual se diseña e implementa una interfaz gráfica que permite visualizar e interactuar con un entorno virtual del robot ArmX utilizando los softwares Matlab y SolidWorks y la plataforma Arduino. Aunque este trabajo presentó resultados positivos uno de los inconvenientes fue la conexión entre los softwares Matlab y SolidWorks, la cual no puede establecerse de manera directa, sino que requiere de otro programa intermedio. Por lo que el proceso en general resultó bastante tedioso surgiendo la necesidad de emplear un software que permite tanto el diseño como la programación del modelo.

Sin embargo, la parte práctica tradicionalmente ha sido un problema por la obsolescencia y/o ausencia de medios, los cuales resulta difícil mantener o actualizar, dadas las limitaciones económicas del país, como consecuencia del férreo y genocida bloqueo económico aplicado por el gobierno de los EEUU. Esto implica asignar a cada alumno un número limitado o nulo de turnos de acceso a estos medios, lo que hace que en general que la parte práctica haya sido bastante deficiente.

Por otro lado, debido al aumento de la capacidad de procesamiento de los ordenadores se ha popularizado internacionalmente el uso de plataformas de simulación. Estas cuentan con la capacidad para hacer un modelado 3D y la renderización de un robot y su entorno, y con un motor de física que genere un

movimiento del robot completamente realista. El uso de una plataforma de simulación se presenta como la solución a las carencias existentes en esta disciplina.

Es por ello que se establece como **problema de la investigación:** Limitaciones en el uso de entornos virtuales para la simulación de robótica.

Objeto: Simulación de robótica en entornos virtuales.

Campo: Simulación de algoritmos de control en robótica móvil en entornos virtuales.

Objetivo: Desarrollo de algoritmos de control para el robot EduRoMAA modelado en distintos escenarios virtuales dentro del entorno virtual CoppeliaSim.

De esta manera la **idea a defender** es: El desarrollo de algoritmos de control para el robot móvil EduRoMAA modelado en escenarios virtuales del CoopeliaSIM ofrece potencialidades para la simulación de robótica.

En la siguiente lista se describen las **tareas** concretas que se han debido realizar durante la totalidad del trabajo:

1. Listar los requisitos que debe poseer la herramienta para su uso por parte de los investigadores de acuerdo a sus necesidades y las condiciones de la facultad.
2. Analizar los simuladores recomendados para robótica móvil a nivel internacional para, de cara a los requisitos previamente listados, seleccionar el más adecuado.
3. Estudiar las herramientas y formas de trabajo existentes dentro del simulador escogido.
4. Diseñar el modelo del robot EduRoMAA y los diferentes escenarios virtuales.
5. Desarrollar e implementar algoritmos de control al modelo para evaluar su comportamiento dentro de los escenarios.
6. Desarrollar dos escenas sobre una posible aplicación real de la robótica móvil.

Métodos científicos utilizados:

1. Histórico-lógico en la revisión bibliográfica referente y la elaboración del marco teórico de la investigación.

2. Análisis-síntesis para evaluar las fuentes de información y criterios referentes al objeto de estudio.
3. Hipotético-deductivo para establecer algoritmos de control de robótica móvil.
4. Simulación para evaluar los métodos propuestos.
5. Experimental para el diseño del modelo y obtención de los gráficos correspondientes a la trayectoria del robot.
6. Estadísticos para analizar y evaluar los resultados experimentales obtenidos.

Estructura de la tesis:

Esta investigación se ha estructurado en Introducción, dos capítulos, conclusiones y recomendaciones. En el capítulo 1 se realiza una descripción detallada del simulador escogido a partir del estudio de la documentación de los referidos en trabajos investigativos de otras universidades del mundo. En el capítulo 2 se crea el modelo 3D de un robot móvil educativo y varios escenarios con el fin de probar distintos algoritmos de control simples de robótica móvil. Luego se desarrollan dos escenas basadas en una aplicación de robótica móvil como es la inspección de parques solares fotovoltaicos. Finalmente se exponen las conclusiones y recomendaciones derivadas del estudio, así como un conjunto de anexos y la bibliografía empleada que fundamentan la obtención de los resultados.

Este trabajo se enmarca dentro del cumplimiento de una de las tareas del proyecto “Aplicación de la Inteligencia Artificial en el mejoramiento del índice de disponibilidad de generadores fotovoltaicos (DEHOT^{PV})”, cuyo objetivo es el desarrollo y aplicación de métodos para incrementar el índice de rendimiento de los generadores fotovoltaicos conectados a red, a través del empleo de técnicas de aprendizaje automático. Dicho proyecto es financiado por la Universidad de Oriente y dirigido por DrC. Leonardo Peña Pupo.

Capítulo 1 Marco teórico referencial

En este capítulo se definen los requisitos que debe cumplir la herramienta de simulación en base a las necesidades de los investigadores de la facultad y luego se realiza la búsqueda de la que mejor se adapte a los mismos. Posteriormente se lleva a cabo un estudio detallado del funcionamiento de la alternativa escogida con el fin de desarrollar las habilidades necesarias para utilizar el software de manera correcta, conocer las funcionalidades que este ofrece a los usuarios y verificar que su selección es la adecuada.

1.1 Entornos virtuales para la simulación de robótica

Una plataforma de simulación robótica es una aplicación informática dotada de una serie de herramientas y capacidades que hacen posible que el usuario pueda simular o recrear tareas realizadas por robots, visualizándolas por pantalla y comprobando así su correcto desarrollo. Se puede entender como una herramienta de visualización donde el usuario, que tiene los modelos que representan a los robots y el código para controlarlos, puede ver si la tarea que pretende llevar a cabo se ejecuta según lo previsto. El entorno de simulación donde se realizan estas pruebas debe ser por tanto lo más fiel a la realidad que se pueda, imitando en lo posible las características del lugar concreto donde se realizaría la tarea [8].

Mientras que el mundo real es obviamente complejo y con múltiples variables aleatorias y difícilmente predecibles, un entorno de simulación ayuda a simplificar el problema que se quiere abordar, ya que proporciona un control absoluto sobre el entorno y los experimentos que se vayan a realizar. Además, la simulación ahorra gran cantidad de costes, tanto materiales como de tiempo en ensayos. Al mismo tiempo, permite reproducir los experimentos o prácticas que se realicen tantas veces como sea necesario, algo que resulta difícil con robots reales ya que las condiciones siempre cambian en un entorno real, como por ejemplo la iluminación, el estado de las baterías, etc. [9]

Para el ámbito de la robótica educativa, disponer de un modelo de simulación que represente al robot real de forma fidedigna ofrece grandes ventajas desde el punto de vista didáctico. Como es de imaginar, la primera ventaja que se puede destacar es el coste de adquisición [5], razón por la cual un modelo de simulación permitiría al centro proporcionar a los alumnos el desarrollo de las habilidades sin necesidad de que haya

un robot físico para cada uno de ellos. El uso de software de simulación también es muy útil en casos de docencia no presencial, como puede ser la situación vivida recientemente como consecuencia del virus Covid-19. Además, al familiarizarse de esta forma con el robot, su programación y control, cuando se emplee el robot real los riesgos de averías derivadas de un mal uso o al desconocimiento se minimizan.

1.2 Requisitos para la selección de la plataforma de simulación

Para realizar una correcta elección del simulador a utilizar se listaron los requisitos que el sistema debía cumplir para los cuales el simulador objetivo debía ofrecer soluciones. A continuación, se muestra la lista de los requisitos establecidos:

- Libre o gratuito.
- Ligero ya que los laboratorios de la facultad no disponen de computadoras potentes contando con un procesador Core i3, 4GB RAM y sin tarjeta gráfica dedicada.
- Proporcionar o permitir la creación de un robot diferencial (dos ruedas delanteras y un punto de apoyo trasero).
- Posibilidad de simular sensores de distancia y de visión para la simulación de estrategias de control.
- El simulador debe permitir crear paredes y otros obstáculos, y modificar el color de los objetos y asociarles texturas, con el fin de construir escenas afines a la realidad para así evaluar el comportamiento del modelo.
- Soportar varios lenguajes de programación.

1.3 Análisis de los simuladores considerados

No son pocos los simuladores disponibles, tanto totalmente libres como comerciales, que se encuentran referidos en internet, sin embargo, la lista de preferidos o recomendados en trabajos investigativos, ya sean técnicos o educativos, se enfoca en plataformas como: CoppeliaSim [10], Webots [11], AirSim [12] y Gazebo [13]. A través del estudio del resumen de la documentación de cada uno de ellos se hizo una selección de la mejor alternativa atendiendo a los requisitos previamente mencionados. A continuación, se describen brevemente sus características.

CoppeliaSim:

CoppeliaSim, previamente llamado V-Rep es actualmente desarrollado por la compañía suiza Coppelia Robotics AG. Este es un simulador que posee una versión educacional que es muy utilizada en otras universidades en asignaturas similares de robótica. Permite la programación en Lua, el lenguaje principal dentro de la aplicación, pero también permite otros lenguajes, tales como Python y C, entre otros. Incluye la simulación de varios sensores, tales como el sensor de distancia y el de visión; además, permite la construcción de diferentes escenarios y robots mediante la combinación de figuras geométricas básicas como cilindros, esferas y cubos. Finalmente, el simulador es bastante sencillo y ligero, por lo que no da problemas de rendimiento a los computadores actuales existentes en los laboratorios (en el manual ni siquiera se especifican los requisitos de hardware). En la figura se puede ver una imagen que muestra la apariencia del simulador que, a cambio de la ligereza, no es foto-realista [14].

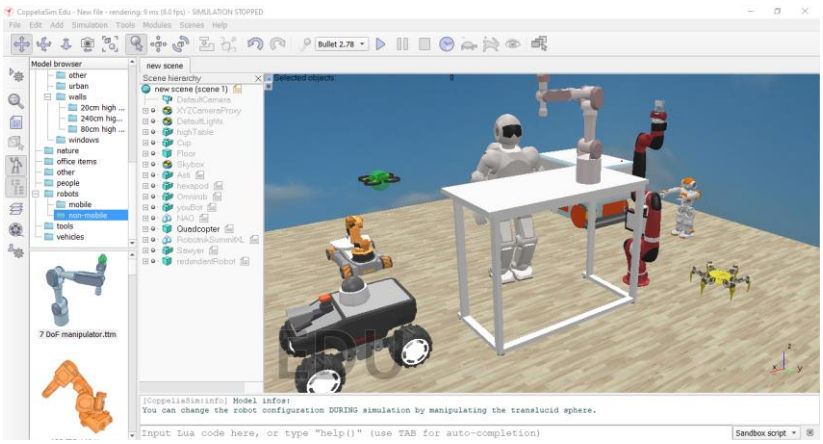


Figura 1 Apariencia de CoppeliSim

AirSim:

AirSim ha sido desarrollado por Microsoft utilizando el motor de videojuegos Unreal Engine, y publicado en 2017 bajo la licencia MIT. Es un simulador muy utilizado en el campo de la investigación en robótica móvil, para aplicaciones como conducción autónoma de vehículos o drones. Este simulador soporta una gran variedad de lenguajes, entre los que se encuentra Python; ofrece una gran variedad se sensores, entre los que se encuentran la cámara y el sensor de distancia, además de otros más complejos, como es el caso del GPS o la IMU3. Sin embargo, el simulador está especializado en vuelos y conducción autónoma, y por ello presenta una interfaz

fotorrealista, mucho más pesada computacionalmente de lo que se desea, teniendo unos requisitos recomendados de 4GB de memoria RAM en tarjeta gráfica dedicada. En la figura se puede ver una imagen de ejemplo de este simulador en la versión de complejidad mínima [14].

Figura 2 Apariencia de AirSim

Webots:

Webots es otro de los simuladores más famosos de código abierto, creado en 1996 en el Instituto Federal Suizo de Tecnología en Lausanne. Se puede programar con el lenguaje Python y soporta casi todos los sensores requeridos. Permite la creación de varios tipos de robots y escenarios con formas geométricas básicas de forma similar a Coppelia; por lo que también es bastante ligero, pidiendo únicamente unos requisitos mínimos de 2GB de memoria RAM y un procesador de cuatro núcleos de 2GHz o más, el uso de tarjeta gráfica dedicada únicamente es recomendado para un mayor rendimiento [14].

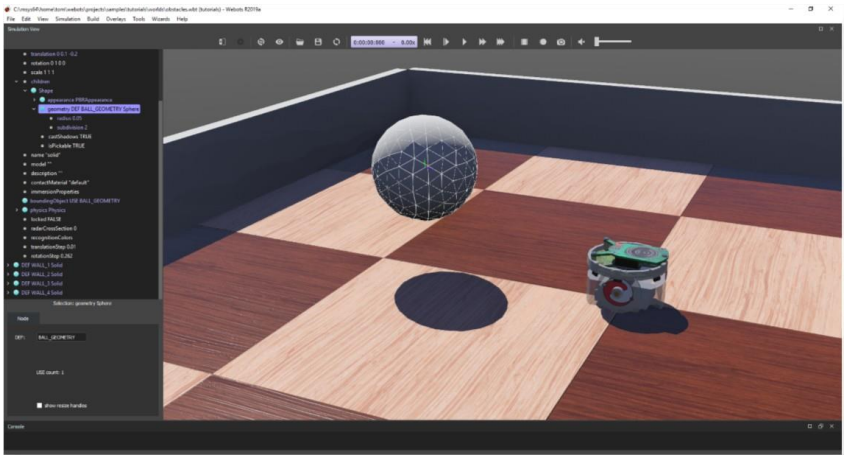


Figura 3 Apariencia de Webots

Gazebo:

Gazebo es un simulador perteneciente a la compañía estadounidense Willow Garage. Tiene una gran dependencia en ROS (conjunto de librerías que permite establecer una estructura de relaciones entre los distintos robots que estén en una determinada simulación, pudiendo llevar a cabo el control de los mismos). Permite trabajar fundamentalmente desde Ubuntu, Debian o Mac, sin embargo, es necesario usar Gazebo con una distribución de Linux, al estar ROS desarrollado para trabajar sólo en este sistema operativo [8]. Permite la creación de varios tipos de robots y escenarios con formas geométricas básicas de forma similar a Coppelia, aunque a diferencia de esta demanda los siguientes requisitos mínimos de hardware: 4GB de memoria RAM, procesador de cuatro núcleos Intel I5 o equivalente y tarjeta gráfica dedicada de 1GB.

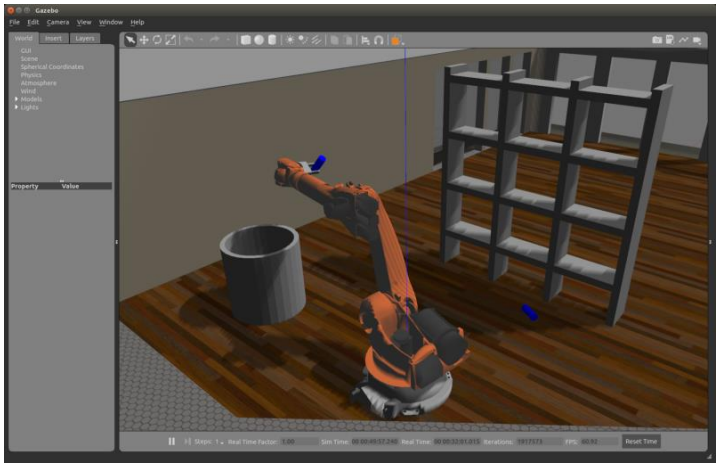


Figura 4. Apariencia de Gazebo

Otros:

Aparte de los simuladores descritos en las secciones anteriores, cabe destacar la existencia de otras alternativas que podrían haber sido consideradas en más detalle, como es el caso de ARGoS [15], pero en una primera fase de análisis se vieron muy similares a Webots, y se consideró mejor descartarlos.

1.3.1 Selección del simulador a emplear

La decisión de qué simulador utilizar de base es sin duda muy importante, ya que condiciona en gran medida todo el trabajo desarrollado. En la tabla siguiente se puede apreciar a modo resumen una comparativa entre los simuladores.

Tabla 1 Requisitos para la selección

Requisitos:	AirSim	Gazebo	Webots	Coppelia
Libre/Gratis	Si	Si	Si	Si
Ligero	No	No	A medias	Si
Simulación de sensores realista	Si	Si	Si	Si
Requisitos del robot	Si	Si	Si	Si
Requisitos de escenario	Si	Si	Si	Si
Soporte de varios lenguajes de programación	Si	Si	Si	Si

El simulador escogido fue CoppeliaSim, ya que además de cumplir con todos los requisitos listados, sobresale al resto de alternativas por su ligereza y simplicidad, características indispensables en una herramienta educativa.

El siguiente paso realizado en el proyecto fue un estudio más exhaustivo del funcionamiento de esta plataforma, cuyo logo se muestra a continuación. Para aprender a utilizar el software y verificar que la selección era adecuada.

Figura 5. Logo de CoppeliaSim

1.4 CoppeliaSim (V-REP). Descripción.

El simulador de robots CoppeliaSim es uno de los programas más utilizados para la creación y simulación de cualquier robot. Posee un entorno de desarrollo integrado, basado en un control de arquitectura en que cada objeto o modelo puede ser controlado individualmente a través de scripts. Los scripts son secuencias de comandos que no se compilan a código máquina, sino que los ejecuta e interpreta el propio programa [16].

En su versión Edu para estudiantes gratuita es posible construir elementos mediante la combinación de figuras geométricas básicas o hacer uso de robots y elementos prefabricados que están disponibles en la carpeta que se encuentra en el buscador

del modelo. También es posible agregar diferentes tipos de sensores para cuantificar o reaccionar a estímulos del entorno y propios. El entorno permite agrupar múltiples elementos bajo un objeto principal para asociar los elementos y de este modo escribir un script que indique al objeto como debe comportarse [17].

CoppeliaSim, 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, proximidad y fuerza.
- 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.
- Permite importar modelos creados en programas CAD.
- Otras características, como función de grabación de vídeo, simulación de pintura, etc [18].

1.4.1 Interfaz de CoppeliaSim

CoppeliaSim cuenta con una interfaz que está compuesta por diversos elementos necesarios para la simulación, los cuales componen las escenas. Cuando se abre un

ambiente en el simulador se accede a un sistema que por defecto incluye una superficie plana simple, como muestra la figura siguiente:

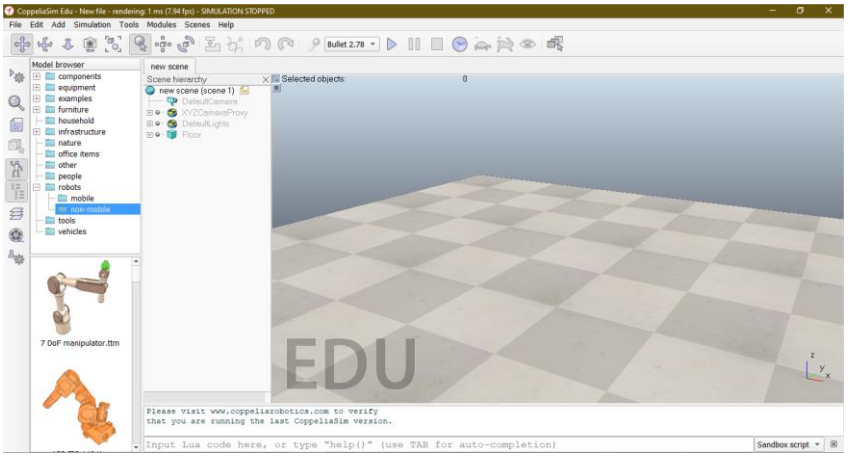


Figura 6. Escena por defecto de CoppeliaSim

Es posible modificar estos escenarios incluyendo paredes, ventanas y otros elementos de arquitectura como escaleras, baños o puertas. Si se desea cambiar las dimensiones o la posición del elemento se puede hacer seleccionando el elemento y el modificador de propiedades. Al desplegarse el menú de selección es posible modificar la geometría, las texturas u otras propiedades. Es posible también abrir múltiples escenarios independientes en una sola pestaña del mismo modo en el que trabaja un navegador. Se puede abrir un nuevo archivo o escenario o abrir un archivo previamente almacenado [17].

A continuación, se describen por separado los principales elementos y herramientas que contiene una escena:

- Escenario: Es donde tiene lugar la simulación. Dispone de un eje de coordenadas cartesianas para saber en todo momento la orientación y ubicación de cada uno de los elementos de la escena [16]. En su parte superior muestra información del último objeto seleccionado en la escena como por ejemplo su posición, orientación, nombre o tipo de objeto.

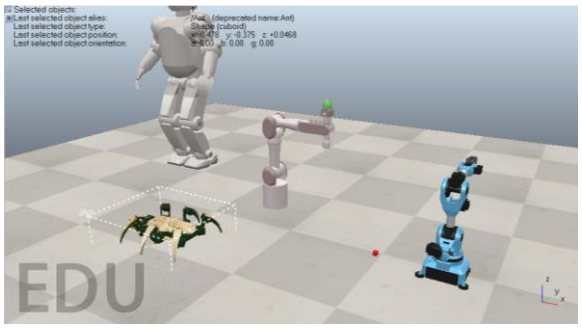


Figura 7. Escenario

- Barra menú: Esta barra contiene, como muestra la siguiente figura, todas las funcionalidades, información y herramientas de la simulación. Las principales funciones son para crear, guardar o cargar escenas, herramientas de la simulación, añadir/modificar objetos o formas y menú de ayuda.

Figura 8. Barra de Menú

- Barra de herramientas: Contiene variedad de herramientas para la gestión de la simulación, el movimiento de los objetos y la gestión de los motores físicos. Véase figura 9.

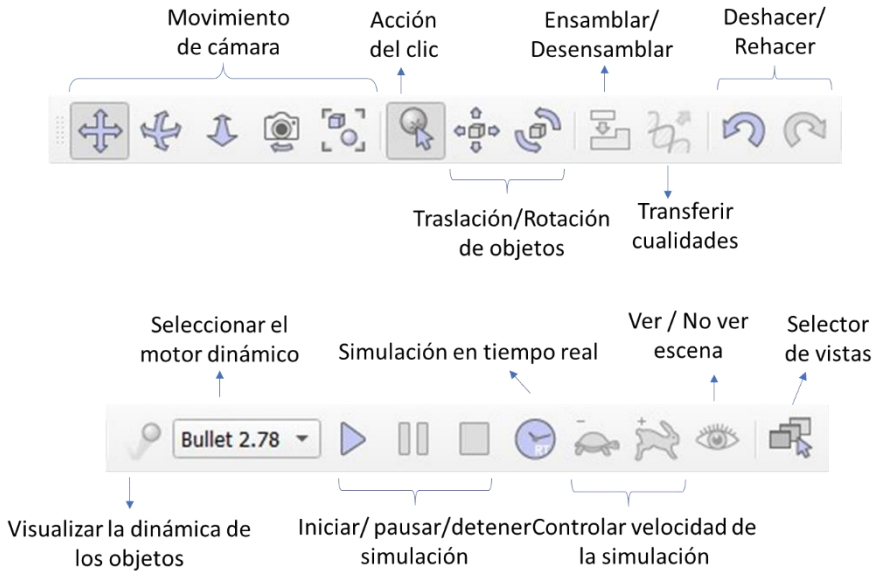


Figura 9. Barra de herramientas

- Barra de herramientas 2: Contiene herramientas para, como se muestra en la figura de izquierda a derecha: configurar la simulación; acceder a las

propiedades de un objeto; editar o añadir scripts; visualizar el buscador de modelos; visualizar la jerarquía de escena; visualizar las capas; grabar escena; y acceder a la configuración de usuario:

Figura 10. Barra de herramientas 2

- Buscador de modelos: Herramienta para buscar modelos ya preparados en CoppeliaSim. CoppeliaSim incluye una gran variedad de formas, objetos, sensores, robots y una amplia cantidad de componentes para realizar una simulación sin la necesidad de diseñar todos los objetos. Esta herramienta se muestra a continuación.

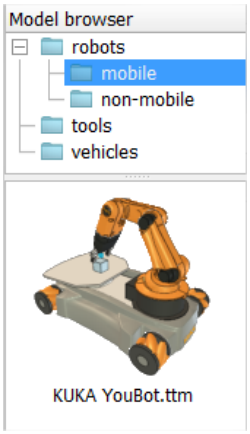


Figura 11. Buscador de modelos

- Jerarquía de escena: Esta ventana muestra el contenido de una escena, es decir, los objetos que la componen. Los mismos 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 [19]. Pueden estar ordenados jerárquicamente como objetos padre o hijos. En la figura se muestra un ejemplo de una jerarquía de escena donde se añadió solamente el robot Jaco desde el buscador de modelos.

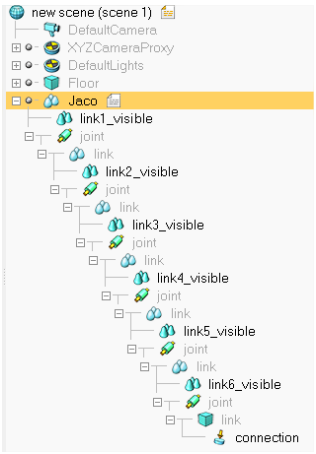


Figura 12. Ejemplo de jerarquía de una escena

- Barra de estado: Ventana que muestra todas las acciones que el usuario realiza en la escena además de los errores (en rojo) que puedan ocurrir en esta, como muestra la figura 13.

```
[sandboxScript:info] Simulation stopped.
[sandboxScript:info] Simulation started.
31
[/EDUROMAA@childScript:error]
45: in sim.readVisionSensor: one of the function's argument type is not correct.
stack traceback:
    [C]: in function 'simReadVisionSensor'
    [string "/EDUROMAA@childScript"]:45: in function 'sysCall_sensing'
[sandboxScript:info] Simulation suspended.
```

Figura 13. Barra de estado

- Menú desplegable: Se muestra cuando se hace clic con el botón derecho, dependiendo de la ventana en que se presione muestra una variedad de herramientas vinculadas la misma.

- Ventana propiedades de objeto: Ventana importante para configurar el comportamiento de cualquier objeto, se puede acceder de dos formas seleccionándola en la barra de menú o haciendo clic dos veces en la imagen del objeto de la jerarquía de escena. En la figura 14 se puede apreciar esta ventana [1].

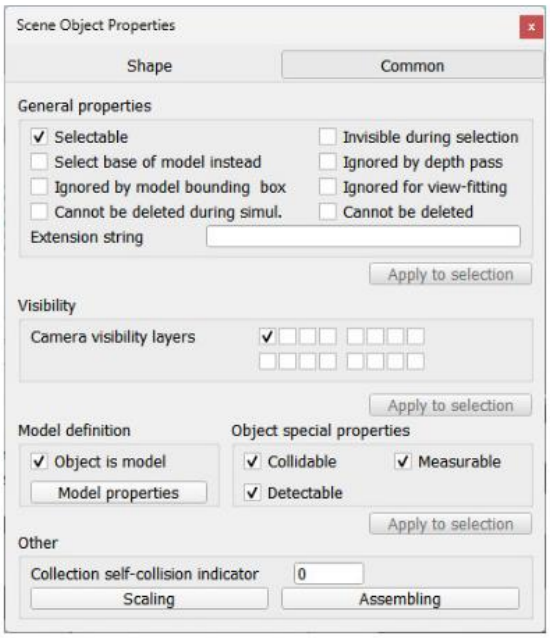


Figura 14. Ventana de propiedades del objeto

1.4.2 Objetos de escena

La creación de la simulación está basada principalmente en objetos de escena. Al generar uno de estos objetos, aparecen tanto en la página de escena como en la jerarquía. Los objetos de escena pueden ser de varios tipos, pero los usados principalmente son los de la figura siguiente:

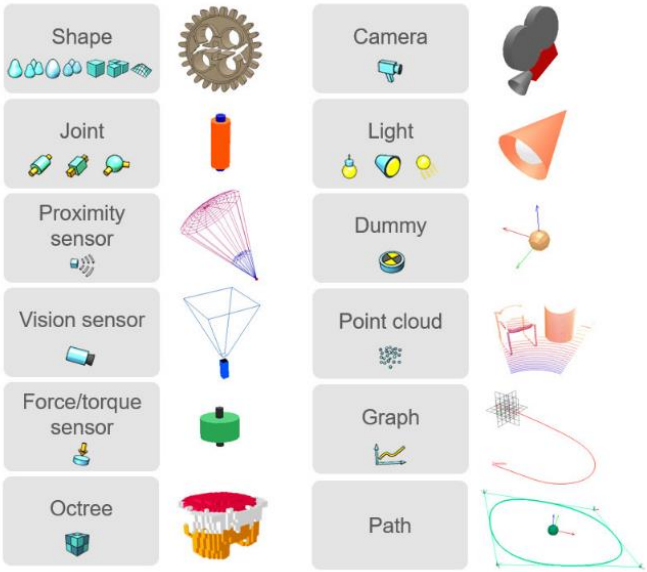


Figura 15. Objetos más comunes de CoppeliaSim

Los objetos empleados en la realización de la simulación de este proyecto se explicarán detalladamente a continuación.

Formas:

Las formas son un objeto solido compuesto por caras triangulares y conforman la estructura física de cualquier modelo. Se pueden importar, exportar, editar y ordenar jerárquicamente para hacer que unas dependan de otras y crear así objetos más complejos como robots [3].

Por defecto todas las formas importadas son simples, y estas se pueden agrupar o dividir. Se clasifican en puras, convexas, aleatorias y campos de altura (*heighfield)*.

Las formas puras representan formas primitivas (como un cubo, un cilindro o una esfera). Una forma pura es más adecuada que el resto de las formas para el cálculo dinámico de la respuesta ante colisión. Pueden ser simples o compuestas dependiendo de si el objeto está formado por una sola malla o por varias mallas agrupadas. El ícono de este tipo de formas se muestra en la figura 16.

Las formas aleatorias pueden representar cualquier malla, y son importadas desde un software de diseño 3D. No están optimizadas ni recomendadas para el cálculo dinámico de la respuesta ante colisión. También pueden ser simples o compuestas, y por tanto tener varios colores o atributos visuales. El ícono de este tipo de formas se muestra en la Figura 16.

El tercer tipo son las formas convexas, que son mallas importadas como las aleatorias pero optimizadas para el cálculo dinámico de la respuesta ante colisión (aunque se emplearán formas puras siempre que se pueda). También pueden ser simples o compuestas. El ícono de este tipo de formas se muestra en la Figura 16.

Tipos de Formas	Ícono
Puras	
Aleatorias	
Convexas	

Figura 16. Tipos de formas

Por último, se encuentran los campos de altura que pueden representar un terreno como una cuadrícula regular, donde solo cambian las alturas. Los campos de altura

también pueden considerarse formas primitivas y están optimizados para el cálculo dinámico de la respuesta a colisiones. Un ejemplo se muestra en la siguiente figura:

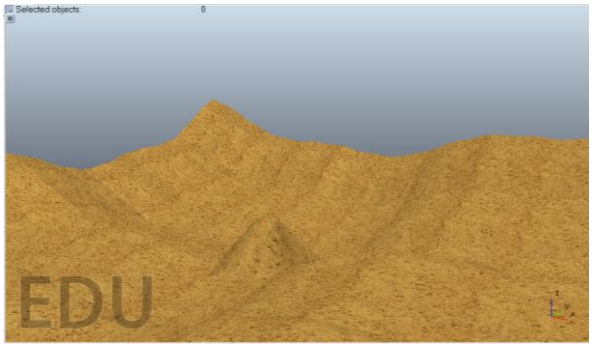


Figura 17. Objeto tipo campo de altura

Accediendo a la ventana de propiedades de estos objetos se le puede cambiar varios atributos, como cambios en su visualización, definir la capa a la que pertenece y atribuirles cualidades dinámicas. Esto último se logra marcando la casilla “cuerpo es dinámico”, lo cual posibilita cambiar en esta misma ventana cualidades como su peso o su momento de inercia. Para que a un objeto le afecte la gravedad tiene que ser dinámico, de lo contrario se estaría creando un objeto flotante. En la figura 18 se puede apreciar esta ventana.

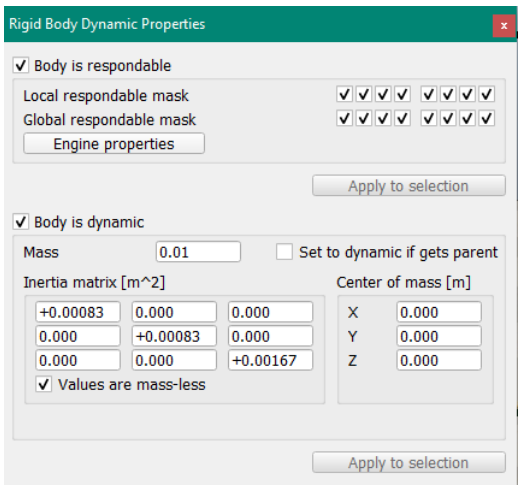


Figura 18. Ventana de propiedades dinámicas del objeto

Los objetos tipo Forma pueden ser capaces de colisionar (*Collidable*), renderizarse (*Renderable*), ser medibles (*Measurable*) y detectables (*Detectable*), propiedades que se establecen en el menú de propiedades del objeto. En dependencia de lo que se marque en la ventana, los objetos interactúan de forma diferente con el entorno que les rodea:

- Colisionables: Atribuye la capacidad de poder colisionar con otros objetos que tengan habilitado este mismo atributo.
- Medibles: Atribuye la capacidad de ser medible, por lo que permite que se calcule la distancia entre objetos que tengan habilitado este mismo atributo.
- Detectables: Atribuye la capacidad de ser detectado por un sensor de proximidad.
- Renderizables: Atribuye la capacidad de ser detectado por un sensor de visión.

Articulaciones

Las articulaciones permiten dotar a un objeto de movimiento al establecer con él una relación de jerarquía donde dicho objeto es hijo de la articulación. Pueden girar 360 grados, como es el caso de las ruedas, o poseer límites físicos como la posición en los brazos robóticos. El tamaño de la articulación dentro de la escena no es más que una cuestión estética, debido a que el movimiento del objeto está condicionado por la posición de la articulación y un correcto establecimiento de la relación de jerarquía. Como se puede apreciar en la siguiente figura, las articulaciones en CoppeliaSim pueden ser de tres tipos:

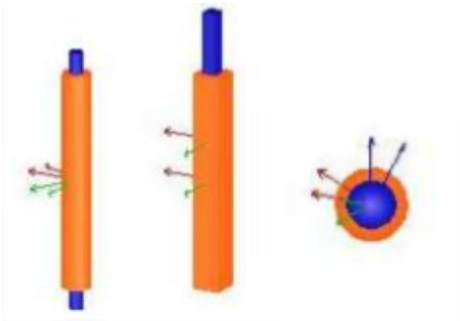


Figura 19. Tipos de articulaciones

- De revolución (*Revolute*): Las articulaciones de revolución son las que giran alrededor de un eje cuyo ángulo de inclinación es el parámetro principal a tener en cuenta durante su configuración. Poseen un grado de libertad.
- Prismáticas (*Prismatic*): Las articulaciones prismáticas son las que desplazan un eje permitiendo la traslación entre dos objetos. Poseen un grado de libertad.
- Esféricas (*Spheric*): Las articulaciones esféricas son las que cambian su orientación girando alrededor de un punto. Permite rotaciones entre dos objetos

y posee 3 grados de libertad debido a que equivalen a 3 articulaciones de revolución que interceptan en un mismo punto.

Las articulaciones pueden trabajar en varios modos:

- Pasivo: La articulación se mantendrá fija a no ser que su posición se modifique por programación.
- Dinámico: Funcionan a través del motor de físicas del propio simulador. Se puede establecer un control de parámetros como su velocidad, torque y posición. Sus valores de referencia se pueden establecer por programación.
- Cinemática inversa: Su posición de referencia es establecida por el módulo de cinemática inversa.
- Dependiente: Su posición depende de otra articulación [20].

Sensores

La plataforma cuenta con tres tipos de sensores cuyos parámetros son configurados por el usuario, quien debe establecer los valores de los sensores reales o los que desee según la escena que quiera desarrollar.

Sensor de proximidad:

Con este objeto se logra detectar otros objetos que tengan la característica de ser detectables (*detectable*). Para incluir un sensor a un objeto se le añade como un objeto hijo y se selecciona el tipo y rango de detección que tiene [10].

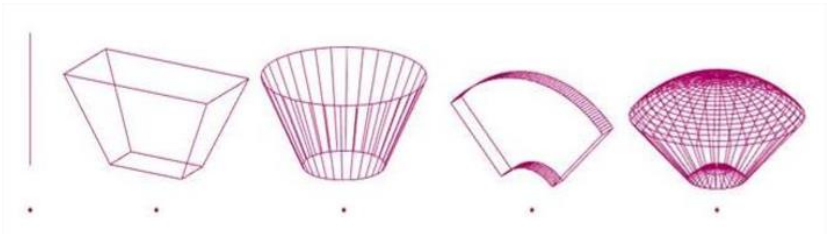


Figura 20. Tipos de sensores de proximidad

Sensores de visión:

Los sensores de visión permiten detectar objetos con la propiedad de ser *Respondables*, se puede seleccionar su resolución y acceder al contenido de la imagen mediante una API interna del propio simulador. Hay dos tipos de sensores de

visión: de proyección ortogonal y de perspectiva. El tipo de proyección en perspectiva es el adecuado para simular cámaras y será el que emplearemos [10].

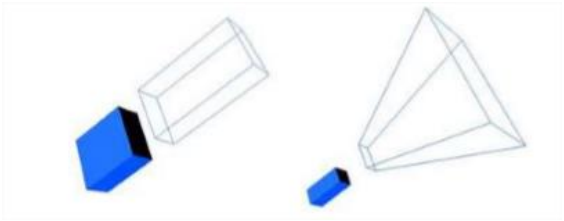


Figura 21. Tipos de sensores de visión

Sensores de fuerza:

Miden el par y la fuerza en sus tres ejes principales. Trabajan como una unión entre dos objetos que puede llegar a romperse si se sobrepasa determinada fuerza o par, pudiendo definir esa fuerza o par en el menú de propiedades del objeto [10].

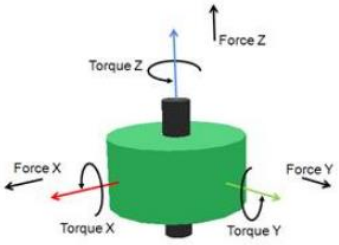


Figura 22. Sensor de fuerza

Otros tipos de objetos

• Cámaras: Una cámara es un objeto que permite ver la simulación desde diferentes puntos de vista o hacer que esta cámara siga a una determinada forma para poder ver su comportamiento en simulación sin necesidad de seguirlo manualmente.

• Luces: Las luces son objetos que permiten iluminar la escena.

• Ficticios (*dummy*): Un objeto ficticio que normalmente se utiliza para ser tomado como una referencia. Tienen la capacidad de colisionar, ser medibles y detectables, pero no visibles por sensores de visión.

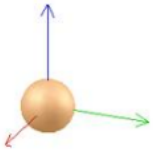


Figura 23. Objeto tipo ficticio

• Camino: Objeto que contiene varios puntos de control que facilitan la creación de caminos o trayectorias ya sean abiertas o cerradas [10].

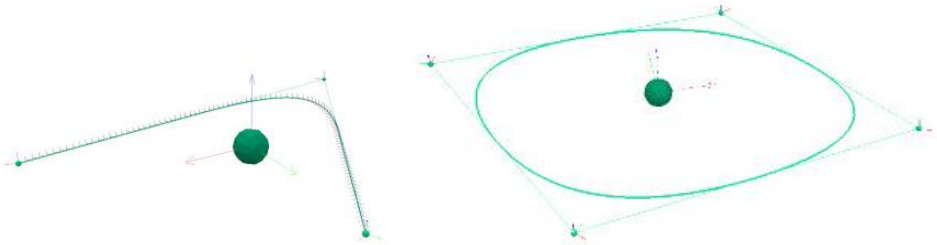


Figura 24. Objeto tipo camino

1.4.3 Módulos de cálculo

CoppeliaSim incluye cuatro módulos de cálculo: el de detección de colisión, cálculo de distancia mínima, dinámica y cinemática inversa. El módulo de detección de colisión calcula las colisiones que se pueden producir entre dos objetos, aunque no calcula la respuesta de los mismo ante la colisión. Para que los objetos puedan colisionar deben estar definidos como colisionables. La información obtenida de este módulo se puede extraer y representar en gráficos para su mejor visualización. La figura 25 muestra cómo el módulo de detección de colisiones no calcula las respuestas ante las mismas.

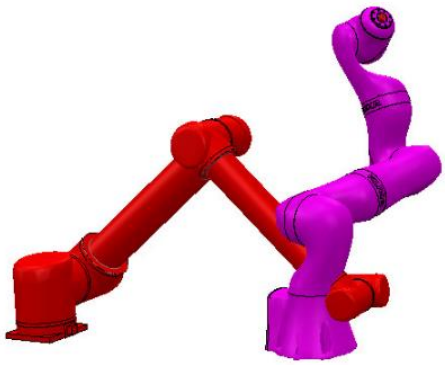


Figura 25. Ejemplo del módulo de detección de colisiones

El módulo de cálculo de distancia mínima calcula distancias mínimas entre dos objetos, definidos como medibles. Sin embargo, este módulo solo puede medir distancias, y no actuar en función de las medidas. En este caso, la información que aporta este módulo se puede guardar en gráficos. En la figura 26 se observa el funcionamiento del módulo de cálculo de mínima distancia.

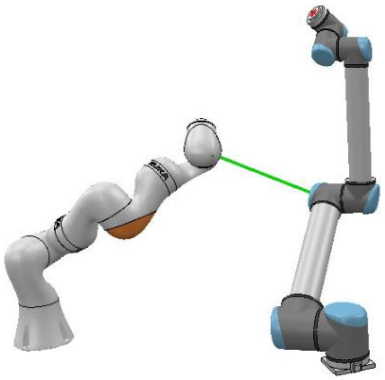


Figura 26. Ejemplo del módulo de cálculo de distancia mínima

El módulo de cinemática inversa permite resolver problemas cinemáticos de cualquier mecanismo, contando para esto con dos modos: el modo de cinemática inversa (IK *mode*) y el modo de cinemática directa (FK *mode*). El primero permite calcular los valores de posición y orientación de una articulación partiendo del movimiento final. Por otro lado, el modo de cinemática directa permitiría calcular el efecto final de un mecanismo partiendo de los valores de sus articulaciones. La figura 27 muestra un esquema del modo de cinemática directa, con el que se puede saber el movimiento final de un objeto a partir del valor de las articulaciones.

Figura 27. Esquema del modo de cinemática directa

Por último, el módulo de dinámica simula objetos dinámicos, y calcula las interacciones entre ellos como, por ejemplo, respuestas de colisión. Actualmente CoppeliaSim cuenta con cuatro motores distintos para desarrollar la simulación: el Bullet, cuyo logo se muestra en la figura 28; el ODE (Open Dynamics Engine), el Vortex y el Newton. En este trabajo en concreto todas las simulaciones se harán con el motor Bullet, un motor de física 3D, de software libre para el cálculo de colisiones, dinámicas de cuerpo rígido y blando, usado sobre todo en videojuegos [3].

Figura 28. Logo del motor de físicas Bullet

Un motor de física permite simular interacciones entre objetos que están cerca de las interacciones con objetos del mundo real. Permite que los objetos caigan, choquen y reboten, pero también permite que un manipulador agarre objetos, una cinta transportadora impulse piezas hacia adelante o un vehículo ruede de manera realista sobre terreno irregular. La siguiente figura ilustra dicha simulación [10]:

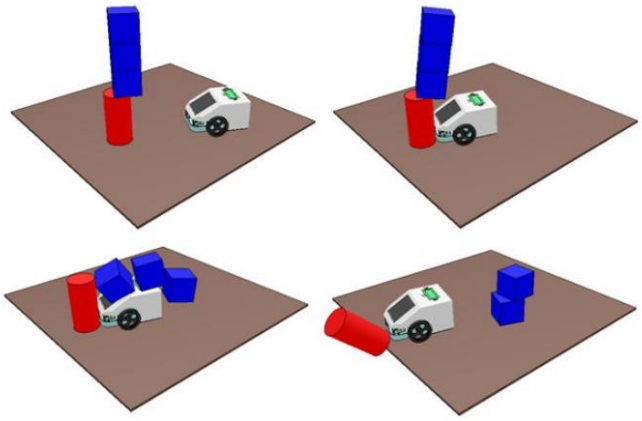


Figura 29. Escena para demostrar la interacción entre objetos

A diferencia de muchos otros paquetes de software de simulación, CoppeliaSim no es un simulador centrado puramente en un motor de física. Más bien puede verse como un simulador híbrido, que combina cinemática y dinámica para obtener el mejor rendimiento en diversos escenarios de simulación. Hoy en día, los motores de física todavía se basan en muchas aproximaciones y son relativamente imprecisos y lentos, por lo que siempre que sea posible se debe intentar utilizar la cinemática (por ejemplo, para manipuladores robóticos) y confiar únicamente en la dinámica cuando de otro modo no sea posible (por ejemplo, la pinza de un manipulador robótico, dispositivos móviles) [10].

1.4.4 Programación

Para programar el comportamiento de un modelo o una escena, CoppeliaSim ofrece varias alternativas: scripts, *add-on*, *plugins*, API (*Application Programming Interface*) externa, nodos de ROS y nodos BlueZero. Entre estos los scripts embebidos

(*embedded scripts*) programados en Lua constituyen el método más sencillo de personalizar la simulación.

LUA es el lenguaje por defecto que ofrece la versión educativa de CoppeliaSim. Este es un lenguaje de programación escrito en C, cuyo principal objetivo es mejorar el uso de estructuras dinámicas y generar una mejora en el testeo con respecto a lo que se hace en C. Es un lenguaje gratuito y de software libre, empleado en programación de procedimientos, programación orientada a objetos, programación funcional, programación basada en datos, etc. Su sintaxis y su forma de escribirse y ejecutarse lo hace ideal para creación de scripts. [21]

Figura 30. Logotipo de LUA

Dentro del simulador existen dos tipos de scripts: los de simulación, que se ejecutan durante la misma para personalizar a esta o al modelo; y los scripts de personalización, que editan la escena o el simulador mientras la simulación está parada.

En los scripts de simulación es importante diferenciar el script principal y los scripts hijos. El script principal es donde se escribe el código necesario para que la simulación se pueda ejecutar, es creado por defecto y se recomienda no modificar. Por otro lado, los scripts hijos están asociados al modelo o a componentes del mismo y controlan realmente el comportamiento del modelo en la simulación. Estos están divididos en 4 funciones principales:

- función sysCall_init(): Se ejecuta solo una vez al iniciar la simulación. Es donde se inicializan las variables y se obtienen los manejadores de los objetos. Cada sensor, objeto o articulación tiene su manejador, que tiene que ser llamado por unas funciones que obtienen el nombre de estos para poder usarlos en las funciones siguientes permitiendo su control. Sin su manejador un actuador no puede ser controlado. Es la única función del script que no es opcional [2].

- función sysCall_actuation(): En esta función se espera que se muevan los diferentes actuadores o articulaciones. Se mantiene ejecutándose continuamente durante la simulación.
- función sysCall_sensing(): Es donde se lee la información de los sensores. Se mantiene ejecutándose continuamente durante la simulación.
- función sysCall_cleanup(): Función de finalización utilizada para limpiar, no es muy habitual. Se ejecuta solo una vez al finalizar la simulación.

Desde los scripts se pueden llamar a las funciones de la API interna de CoppeliaSim, para controlar, actuar, personalizar y obtener datos de la simulación. Estos comandos comienzan por el prefijo “sim.” y se encuentran en la página oficial del propio simulador [10]. Existen funciones de todo tipo, como son las que permiten:

- Obtener el manejador de un objeto.
- Obtener la posición u orientación de un objeto.
- Modificar la posición u orientación de un objeto.
- Conocer el tiempo de simulación.
- Establecer la velocidad o posición de una articulación.
- Leer los valores de los sensores.

1.5 Conclusiones parciales

Se analizaron las herramientas de simulación más utilizadas en otras universidades a nivel mundial y se concluyó que la que mejor se adapta a los requisitos de los investigadores del departamento de nuestra facultad es CoppeliaSim.

Se desarrolló un estudio exhaustivo de la plataforma seleccionada que permitió adquirir los conocimientos necesarios para comenzar a trabajar en ella. Durante este proceso quedó demostrado que:

- CoppeliaSim es un software sencillo, que permite crear simulaciones con múltiples opciones de configuración y objetos.
- La herramienta dispone de una gran variedad de parámetros, utilizados para describir las propiedades físicas de los objetos, desde su apariencia hasta sus

movimientos permitiendo la creación de modelos y escenas muy similares a la realidad.

- Posee varios módulos de cálculos y motores físicos que posibilitan que la simulación tenga una gran semejanza con la realidad.
- Brinda la posibilidad de crear de manera sencilla caminos que pueden ser recorridos por los robots y graficar en tiempo real la trayectoria recorrida.

Capítulo 2 Desarrollo de algoritmos de control para el robot EduRoMAA dentro del entorno virtual CoppeliaSim

En este capítulo se realiza una descripción del robot educativo EduRoMAA perteneciente a los investigadores de la FIE. Posteriormente se diseña el modelo 3D del mismo dentro de la plataforma de simulación y se evalúa su comportamiento en distintos escenarios ante distintos algoritmos de control simples de robótica móvil. Finalmente, con vistas a una futura aplicación, como es la inspección de parques solares fotovoltaicos, se desarrollan dos escenas para simular algoritmos de seguimiento de trayectorias como son el “Guiado óptico” y el “Guiado por GPS”.

2.1 Robot EduRoMAA

El EduRoMAA es un robot móvil triciclo fabricado mediante corte láser e impresión 3D que posee las siguientes características [22]:

- El robot es de tracción diferencial, es decir, realiza el cambio de dirección modificando la velocidad relativa de sus ruedas izquierda y derecha.
- Dispone de dos ruedas de tracción controladas de forma independiente a fin de generar la trayectoria deseada.
- Posee dos motores reductores 48:1 de corriente directa con un amplio voltaje de funcionamiento que permite regular su velocidad.
- Cuenta con sensores de bajo costo comunes en robótica, tales como: sensores de distancia por ultrasonidos y sensores infrarrojos para la detección de línea.
- El comportamiento del robot es controlado por un ARDUINO MEGA [23].

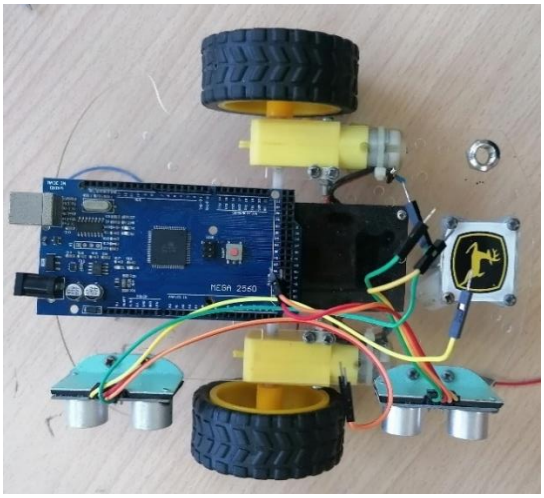


Figura 31. Robot EduRoMAA

La singular forma del robot EduRoMAA y sus características lo convierten en una gran herramienta para la aplicación en el área de la robótica móvil. De este modo, es posible profundizar en un campo que actualmente está adquiriendo gran relevancia.

2.2 Creación del modelo 3D en CoppeliaSim

El modelo del EduRoMAA en CoppeliaSim está formado por 3 componentes básicos. Primeramente, los distintos tipos de formas que se emplean para representar todas las piezas del robot. Por otro lado, las articulaciones conectan los distintos componentes del modelo y se emplean para simular los ejes de los motores y la rueda loca, y finalmente los sensores de proximidad y visión, que simulan los sensores ultrasónicos e infrarrojos que presenta el robot.

2.2.1. Componentes del modelo 3D: Formas

Estos elementos poseen dos componentes: una visual, y otra dinámica empleada para realizar los cálculos de la simulación. Esta última es del tipo forma pura, mientras que la parte visual es una forma aleatoria.

El diseño de la parte visual, para lograr una buena semejanza con la realidad, fue extraída de las fuentes [22], [20], [24], quienes realizaron el modelado de las piezas del robot en programas como FreeCad, KidCad y SolidWorks, y posteriormente fueron exportados a CoppeliaSim. Tanto la parte dinámica como la parte visual están diseñadas a escala real, respetando las medidas de las piezas reales.

Para importar estos archivos se accedió a Archivo / Importar / Malla. Luego se seleccionaron las piezas y en el menú de propiedades del objeto se ajustaron sus dimensiones. Posteriormente, para la creación de la parte dinámica a partir de la parte visual, se extrajo la forma pura en la opción: Alternar modo de edición de forma / Invertir Selección / Extraer forma*;* y se seleccionó el tipo de forma pura en dependencia de la forma de la pieza importada*.*

Tomando por referencia las ruedas, su componente dinámico consiste en una forma cilíndrica. Tanto la parte visual (izquierda) como dinámica (derecha) de las ruedas se muestran en la figura:

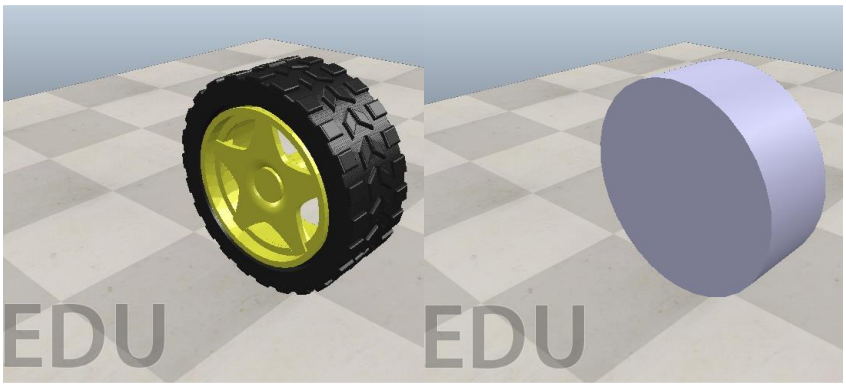


Figura 32. Parte visual y dinámica de una rueda del EduRoMAA

Todos los componentes modelados para el cálculo dinámico tienen marcada, en el diálogo de propiedades dinámicas, las opciones “Cuerpo responde” y “Cuerpo es dinámico”, para que se produzcan reacciones de colisión en caso de que dos objetos impacten, y para que varíe su posición y orientación durante la simulación. Aunque no es parte del modelo, el suelo no debe cumplir la condición de dinámico, aunque si debe cumplir la primera condición. En ese mismo diálogo se procede a personalizar la masa y los momentos de inercia de las distintas componentes, así como su material. Por defecto, todas las componentes del modelo tendrán el mismo material, excepto las ruedas, a las que se les variará su valor de fricción para mejorar su comportamiento.

2.2.2 Componentes del modelo 3D: Articulaciones

De los tres tipos de articulaciones que ofrece el simulador, sólo se empleará un tipo en el modelo en cuestión, la articulación de revolución, ya que todos los motores generan un movimiento de giro alrededor de un eje. El modelo cuenta con cuatro articulaciones de revolución, dos para simular los motores de las ruedas y otros dos más para propiciar el movimiento libre de la base y la rueda loca respectivamente. La distribución de las cuatro articulaciones coincide con el eje de cada elemento al que pertenecen. Aunque en la siguiente figura se puede observar estos componentes en la parte visual del modelo, vale recordar que los mismos pertenecen a la parte dinámica.

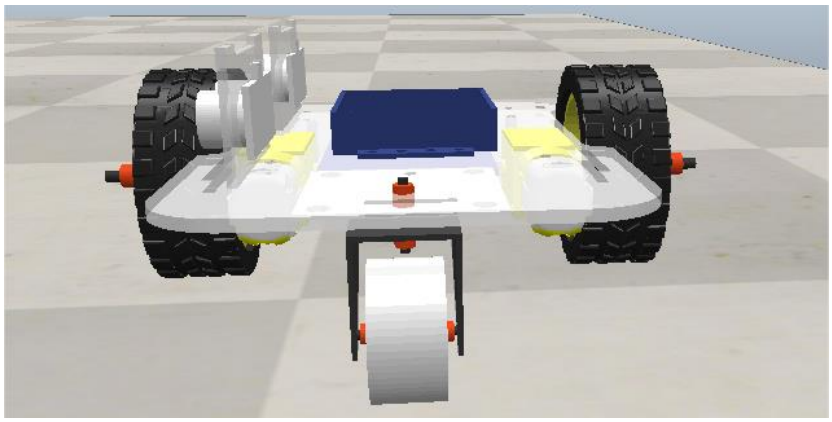


Figura 33. Ubicación de las articulaciones

Las articulaciones de las dos ruedas laterales serán controladas por velocidad, característica que se puede personalizar en el diálogo de propiedades dinámicas de la articulación y que permite asignar una velocidad objetivo (*target velocity*) en grados por segundo. Además, cada articulación tiene asignado un par máximo por defecto que le permita alcanzar la velocidad objetivo de forma casi instantánea para no tener en cuenta la aceleración del motor y simplificar el cálculo (2.5 N.m).

2.2.3 Componentes del modelo 3D: Sensores

El modelo consta de 6 sensores, de los cuales 4 son de proximidad y 2 de visión. Los de proximidad son de tipo cilíndrico y se emplean para simular los ultrasónicos. Estos, como ya se ha explicado detectan un objeto (siempre que esté configurado como detectable) y devuelven la distancia a la que se encuentra el mismo. Por otro lado, los dos sensores de visión permanecen instalados bajo la parte frontal con el objetivo de leer la intensidad del color del suelo y devolver dicho valor. Al implementar un sensor de visión en el modelo se genera una ventana auxiliar en la que se muestran todos los elementos detectados por la cámara. En la figura 43 se muestra esta ventana, en la que aparece la diferencia de color en el suelo detectado por ambos sensores:

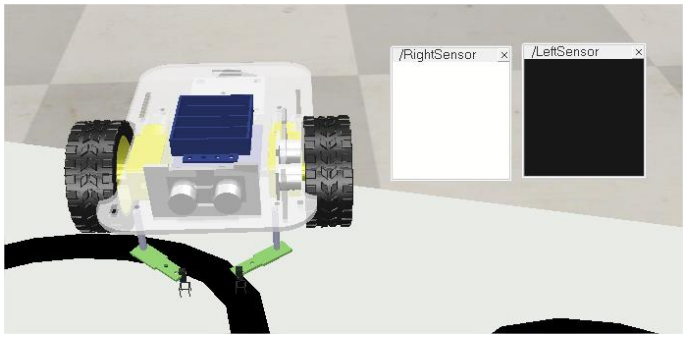


Figura 34. Ejemplo sensor de visión

2.2.4 Relación entre los componentes del modelo 3D del robot

En este apartado se explicará cómo están conectados entre sí los distintos elementos que componen el modelo del EduRoMAA para que se comporten de forma coordinada, cumpliendo las restricciones del modelo real. Los objetos que forman el modelo conforman un árbol de jerarquía en donde todos dependen de uno principal que es considerado la base del modelo. En este caso la base es la malla correspondiente a la parte dinámica de la base del robot, llamada “EduRoMAA”.

Dentro del árbol de jerarquía, se configuran dos tipos de relación de dependencia que definen los movimientos del modelo. El primero es de dependencia directa, es decir, un objeto depende directamente de otro jerárquicamente superior, de forma que no habrá movimiento relativo entre ambos. En este caso, si el objeto superior se moviese, los que dependen de él se moverían con él. Esta relación se emplea, por ejemplo, entre los sensores y la base del modelo. La segunda relación es de dependencia indirecta, a través de una articulación, la cual permite que el objeto inferior jerárquicamente tenga un movimiento relativo con respecto al superior y es la usada entre las articulaciones de las ruedas y la base del robot. En general, la estructura sería: objeto superior → articulación → objeto inferior [3].

Las formas correspondientes a la parte dinámica del modelo están un nivel jerárquico por encima de las correspondientes a la parte visual. De este modo, esta última es simplemente un objeto estático y estético cuya posición depende de la parte dinámica, la cual sí posee las propiedades dinámicas y reactivas.

Las articulaciones, a su vez, deben ser superiores jerárquicamente a las formas que se quiere que dependan de ellas e hijas o pertenecientes al modelo general del robot.

Dicho esto, la relación de jerarquía de este robot móvil es la siguiente:

1. Toda forma pura que depende de una articulación se agrupa seleccionando, haciendo click derecho > Editar > Agrupar / Fusionar > Agrupar las formas seleccionadas, y, posteriormente, se hace hija de la articulación de la que depende.

2. Las formas correspondientes a la parte visual y dependientes de articulaciones se agrupan y se hacen hijas de estas.

3. Las formas correspondientes a la parte dinámica que no dependen de ninguna articulación se agrupan.

4. El resto de las formas de la parte dinámica que no dependen de ninguna articulación también se agrupan creando el objeto base.

5. Los sensores, articulaciones y la agrupación de las formas de la parte visual se convierten en hijos de la base del robot.

En la figura, se muestra finalmente el árbol de jerarquía del modelo del EduRoMAA.

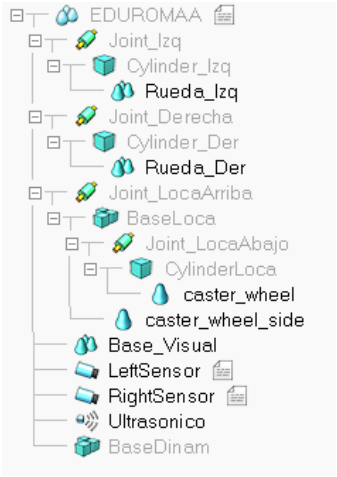


Figura 35. Árbol de jerarquía del EduRoMAA

Una vez realizado este procedimiento a todas las piezas del robot, se obtuvo la parte visual y la dinámica como se muestra a continuación:

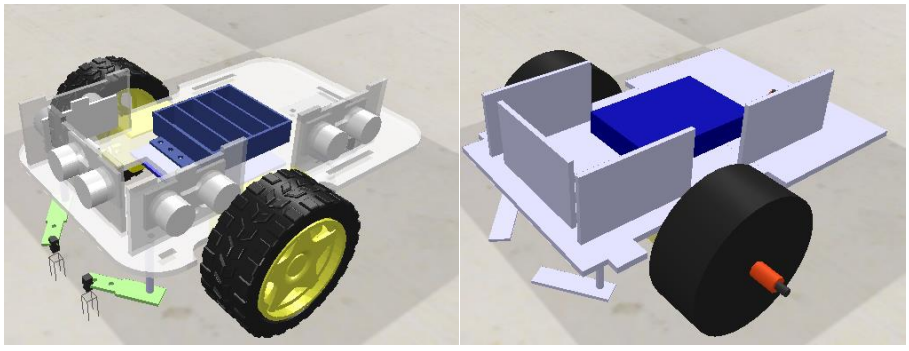


Figura 36. Parte visual y dinámica del robot

2.3 Experimentos

Una vez creado el modelo 3D del robot, se procede a verificar cuán efectivo es su comportamiento ante determinados algoritmos simples de robótica móvil educativa. Con este objetivo se crearon las cuatro escenas de prueba que se muestran a continuación.

2.3.1 Creación de escenas

La primera consiste en generar una línea negra impresa en el suelo creando una trayectoria cerrada para implementar el algoritmo del seguimiento de línea. Para ello se añade desde la barra de menú un objeto tipo camino, en este caso, se selecciona la opción cerrado, ya que se desea establecer una línea cerrada. Posteriormente se accede al menú principal del objeto y se habilita la casilla “Generar forma extruida”. Por último, se accede a los puntos de control (hijos del objeto camino) y se cambian de posición hasta lograr la trayectoria deseada. Se modifica el color y la altura en el menú de propiedades del objeto y se añade una forma plana de color blanco que constituye la pista sobre donde se encuentra la línea como muestra la figura.

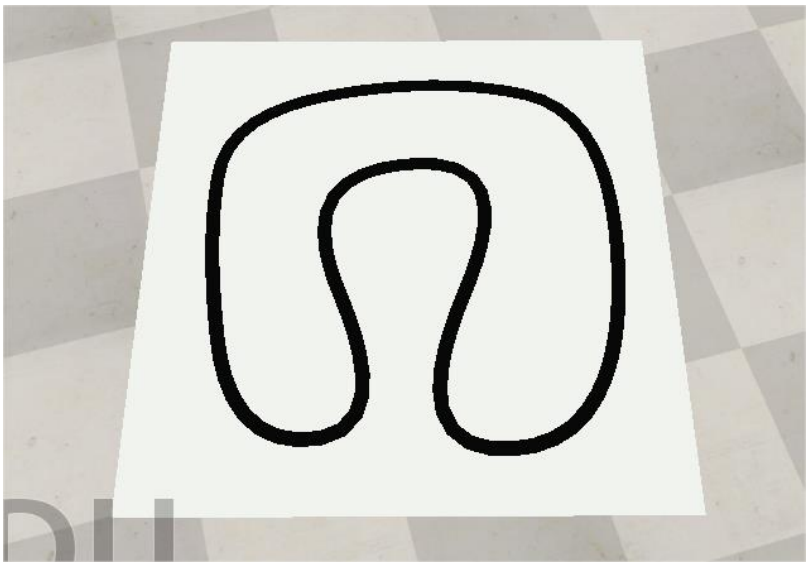


Figura 37. Escena 1

Para la segunda escena se modifica la forma de la trayectoria creada en el primer escenario a través de modificaciones en la posición de sus puntos de control. Luego dentro del menú de propiedades del objeto se aumenta su altura generando las paredes de este nuevo escenario, como se muestra en la figura siguiente:

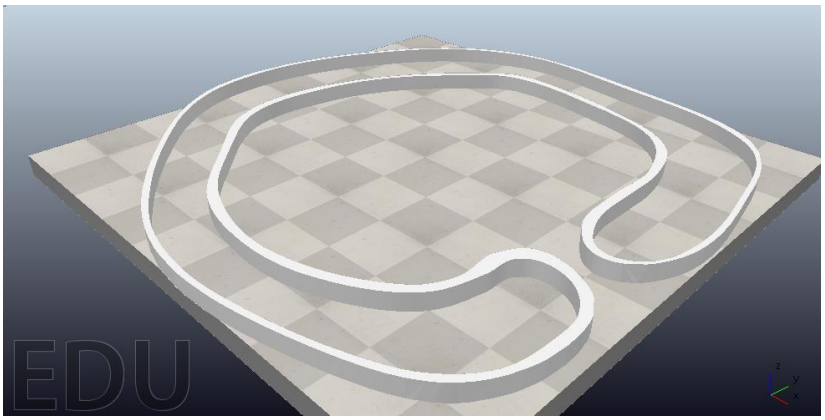


Figura 38.Escena 2

Para la tercera escena se crea un laberinto que permita la implementación del algoritmo de seguimiento de una pared. Para crear esta escena se insertan varias formas tipo cubos, a las cuales se le editan sus dimensiones y su posición con el fin de crear el laberinto. Al final del laberinto se coloca una línea negra que indica la meta. En la siguiente figura se puede apreciar el laberinto resultado de la aplicación de este procedimiento.

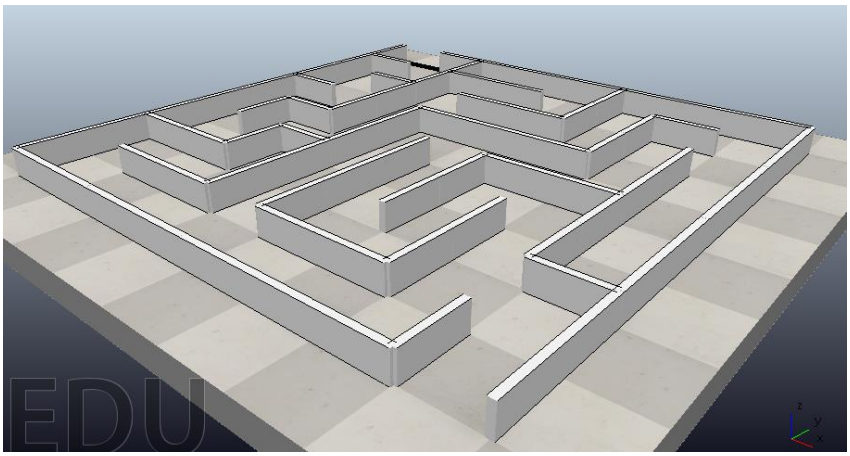


Figura 39.Escena 3

Para la última escena se crea una trayectoria con algunos objetos en medio para la implementación del algoritmo de seguimiento de línea con evasión de obstáculos. La creación de la línea se realiza mediante el mismo procedimiento de la escena 1, a diferencia de que esta vez se selecciona la trayectoria de tipo abierta y se añaden desde el buscador de modelos algunos objetos predefinidos, los cuáles se colocan encima de la línea, como muestra la figura siguiente.

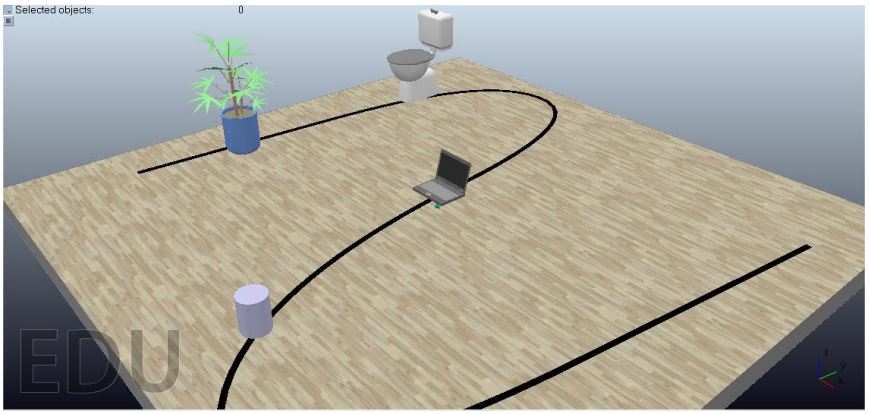


Figura 40.Escena 4

2.3.2 Creación de los algoritmos de control:

Los algoritmos de control son fundamentales en la robótica móvil, ya que permiten que el robot realice movimientos precisos y eficientes para garantizar el cumplimiento de las tareas asignadas. Estos permiten además la adaptabilidad del robot a cambios en el entorno lo cual es crucial para su utilidad práctica en aplicaciones como logística, la exploración y la vigilancia. A continuación, se describen algunos de los algoritmos de control más comunes de robótica.

2.3.2.1 Seguimiento de Línea

Una solución para lograr que los robots sigan una ruta determinada, son los seguidores de línea. Estos son robots móviles equipados de sensores ópticos que les permiten seguir una línea que se encuentra impresa en el piso y la cual guía el camino del robot. Aunque esta solución es sencilla, limita el robot a una única trayectoria [4]. El algoritmo consiste en establecer una velocidad de avance lineal mientras los sensores ópticos del robot no detecten el color negro en el piso. Al detectar dicho color se modifica la velocidad de los motores para que el robot gire hacia la dirección en la que se está detectando el color negro. En caso de que el color sea detectado por ambos sensores se establece una velocidad de avance lineal. A continuación, se muestra un flujograma de este algoritmo.

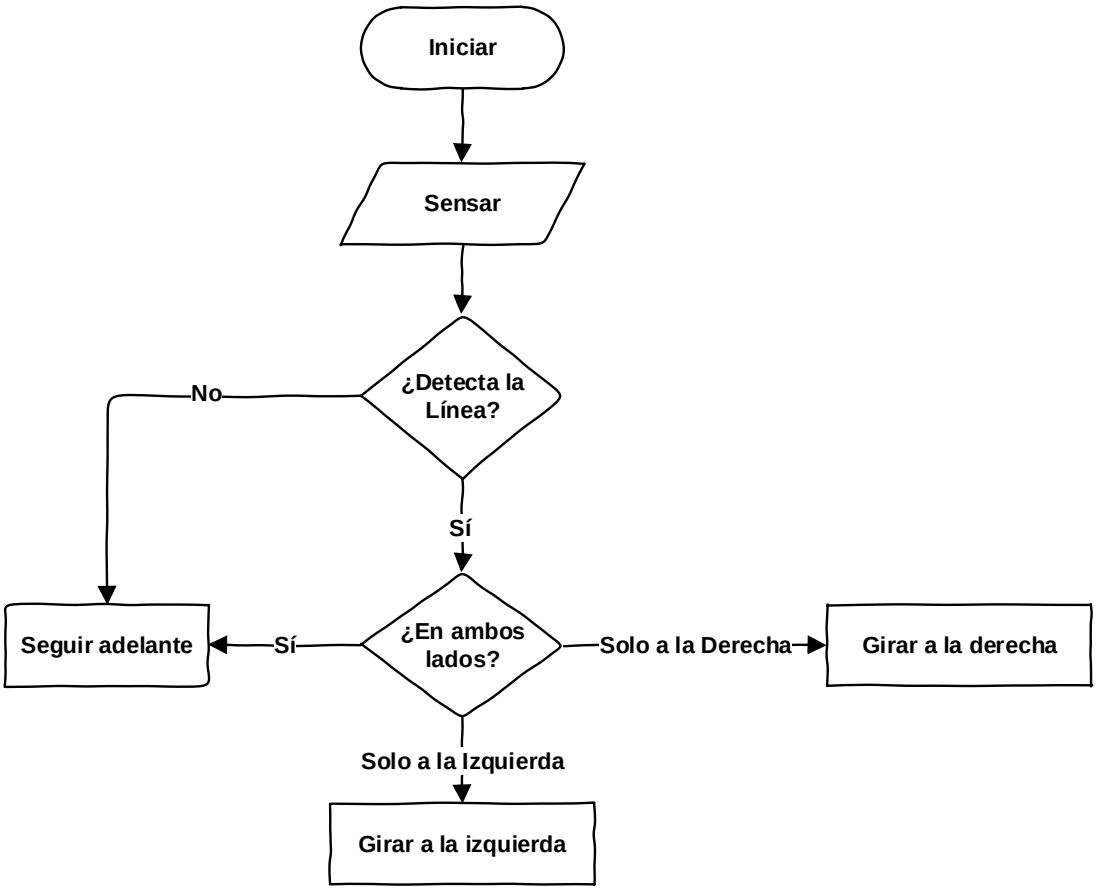


Figura 41. Flujograma del algoritmo del seguidor de línea

2.3.2.2 Seguimiento entre paredes

El 2do algoritmo consiste en colocar entre dos paredes al robot equipado con sensores de proximidad (en este caso ultrasónicos) para lograr que este se desplace equidistante a las mismas. Para este algoritmo se establecen las velocidades de las ruedas del robot en función de la distancia de separación de las paredes entregada por los sensores ultrasónicos. En todo momento el robot se desplazará hacia el lado cuyo sensor de proximidad detecta una distancia mayor, buscando que la misma se reduzca hasta alcanzar el valor del lado opuesto, momento en el cual se establece una velocidad de avance lineal. En caso de que el sensor ultrasónico instalado en la parte frontal del robot detecte una pared enfrente se detendrá. A continuación, se muestra un flujograma de este algoritmo.

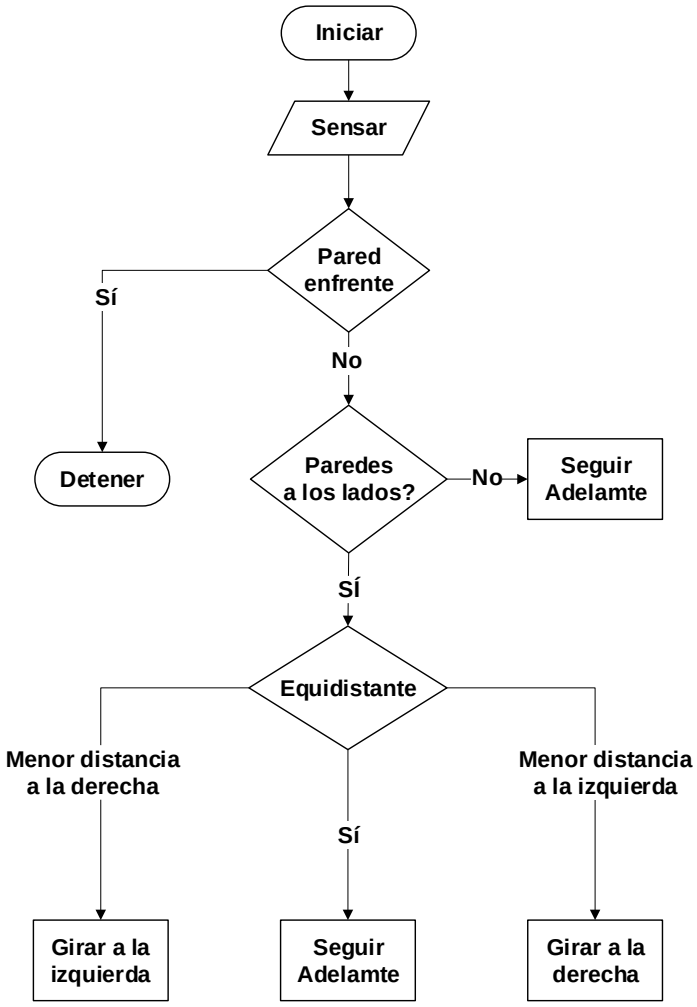


Figura 42. Flujograma del algoritmo de seguimiento entre paredes

2.3.2.3 Seguimiento de una pared

El 3er algoritmo es uno de los métodos más utilizados en navegación para que un robot pueda moverse de manera autónoma apoyándose en el uso de los sensores ultrasónicos instalados en su parte lateral izquierda. Una vez el robot detecta la pared debe desplazarse en paralelo a ella. Si se aleja demasiado gira a la izquierda para acercarse a ella y viceversa, manteniéndose así a una distancia segura. En caso de que el sensor instalado en la parte frontal del robot detecte una pared enfrente, el robot girará a la derecha si se detecta pared a su izquierda, o girará a la izquierda en caso contrario. A continuación, se muestra un flujograma de este algoritmo.

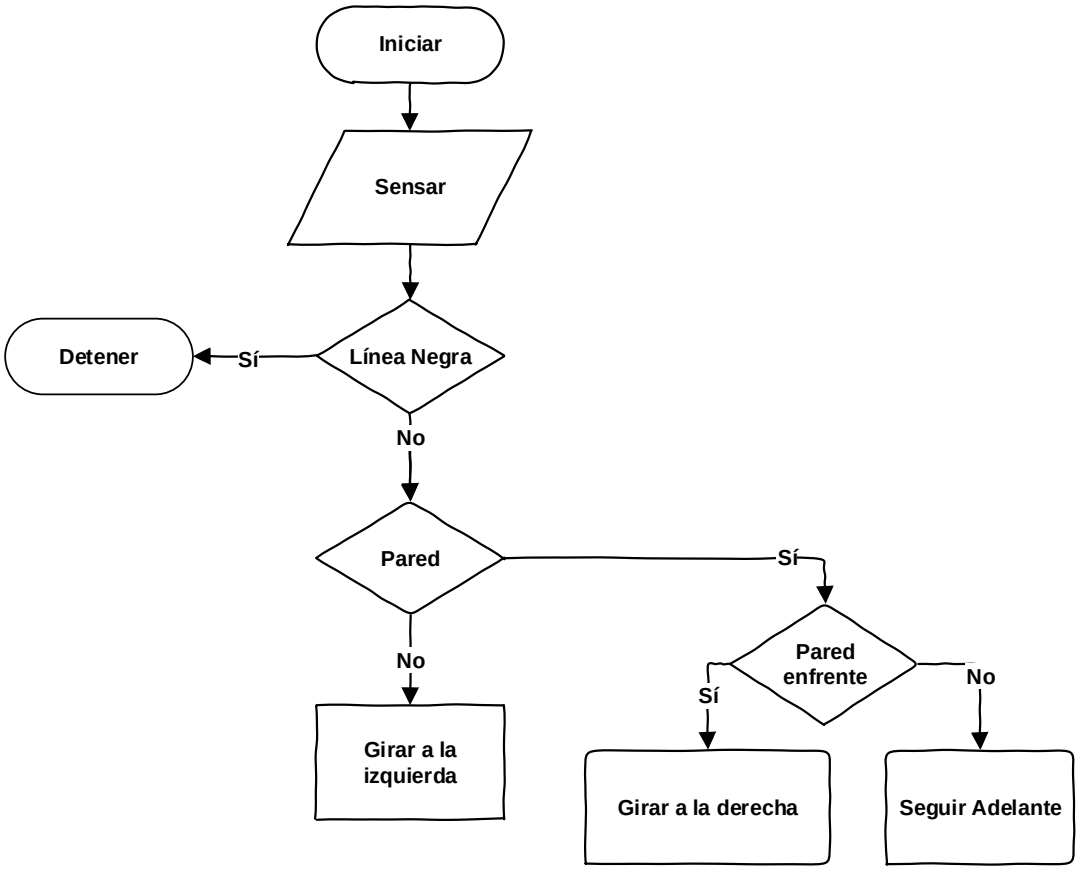


Figura 43. Flujograma del algoritmo del seguidor de pared

Este algoritmo puede ser implementado de diferentes maneras utilizando diferentes tipos de sensores. En este caso se utilizan los sensores ultrasónicos del EduRoMAA para navegar en un laberinto.

2.3.2.4 Seguimiento de línea con evasión de obstáculos

Este algoritmo de control permite que el robot evite obstáculos de manera autónoma, algo esencial para garantizar su seguridad y la de su entorno. Se basa en la detección de objetos a través de sus sensores de proximidad y el uso de esa información para evadirlos y mantener al robot cerca de la línea que guía su camino. Este es el motivo por el cual se puede afirmar que este algoritmo es integrador de los anteriores.

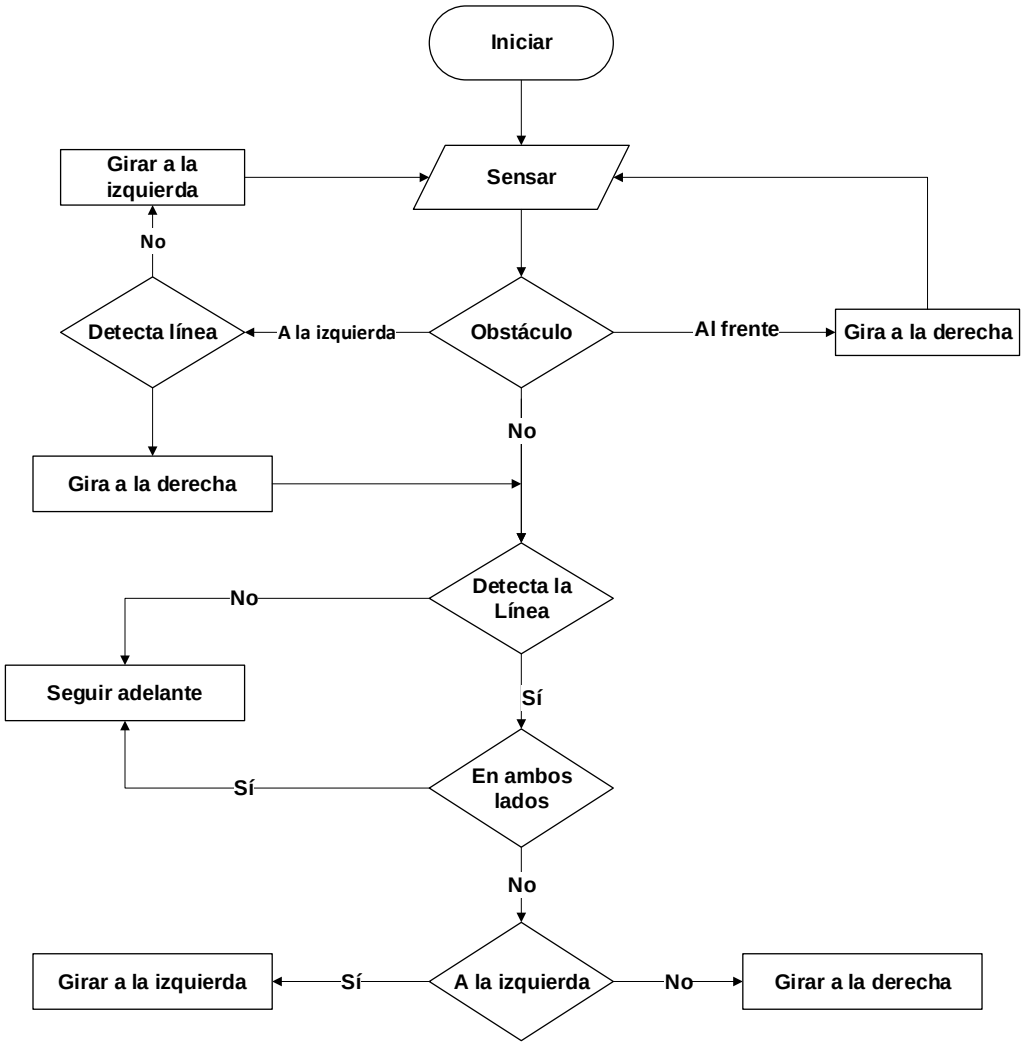


Figura 44. Flujograma del algoritmo del seguidor de línea con evasión de obstáculos

2.3.3 Resultados de las simulaciones

Escena1

Durante la implementación del primer algoritmo: seguimiento de una línea, el robot presentó un comportamiento adecuado apoyándose en los valores entregados por los sensores de visión. Para este algoritmo se accede a la información de los sensores de visión a través de la API interna de CoppeliaSim *sim.ReadVisionSensor* y en consecuencia, para seguir la trayectoria, se le asignan velocidades a las articulaciones por medio de la API *sim.SetTargetVelocity* (Ver Anexo1). La siguiente figura muestra al modelo durante el recorrido.

Figura 45. Simulación de la escena 1

Inicialmente se colocó el robot fuera de la pista y se desplazó en línea recta hasta que sus sensores detectaron el color negro, momento a partir del cual comenzó a seguir la trayectoria deseada sin dificultades. El gráfico siguiente muestra en azul y naranja la trayectoria del robot antes y después de detectar la línea respectivamente.

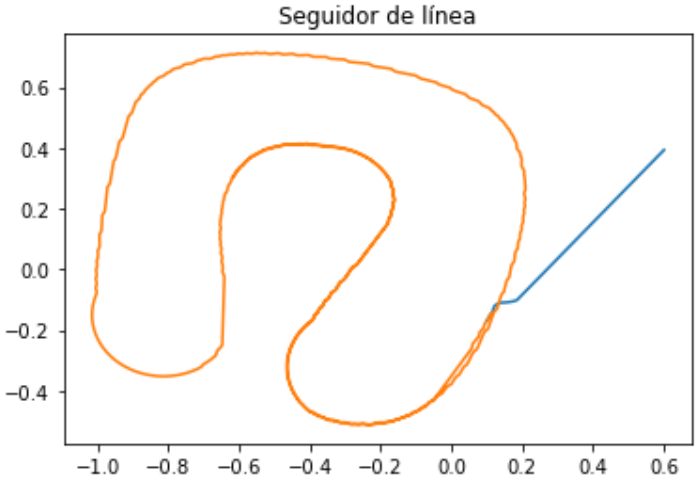


Figura 46. Trayectoria recorrida por el modelo durante la escena 1

Escena 2

La segunda escena se desarrolló de manera exitosa. En la misma el robot se mantuvo desplazándose por el centro del camino equidistante a las paredes utilizando la información entregada por los sensores de proximidad instalados en sus laterales. El mismo se detuvo cuando el sensor frontal detectó una pared enfrente. Para conseguir dicho objetivo, se accedió a los valores de distancia entregados por los sensores ultrasónicos a través de la API *sim.ReadProximitySensor*. Con estos datos se calculó una distancia promedio que será la referencia y de variar la misma, el robot variará la velocidad de sus motores a través de la API *sim.SetJointTargetVelocity*.

La figura siguiente muestra el robot durante su desplazamiento por el centro del camino entre las paredes.

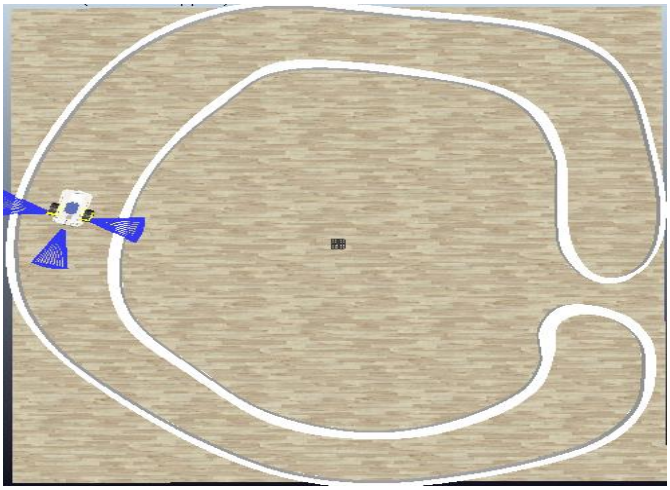


Figura 47. Simulación de la Escena 2

El gráfico siguiente muestra la trayectoria recorrida por el modelo.

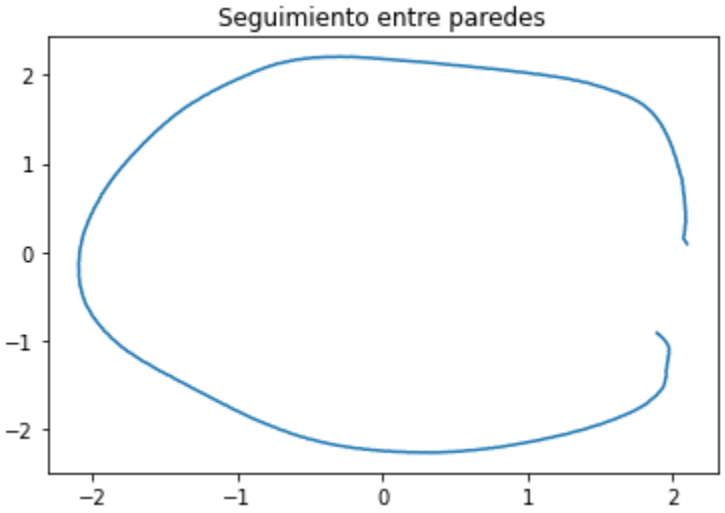


Figura 48. Trayectoria recorrida por el modelo durante la escena 2

Escena 3

En la tercera escena el robot logró resolver el laberinto siguiendo de manera efectiva la pared que detectaban sus sensores ultrasónicos a la izquierda y en frente, lo que permitió al robot recorrer todo el laberinto mediante el seguimiento de una de sus paredes.

Inicialmente se define la distancia de separación del robot y la pared durante su recorrido. De manera muy similar al algoritmo anterior, se accede a los valores de distancia entregados por los sensores de proximidad instalados en la parte izquierda del robot y se calcula la distancia promedio. Esta distancia es comparada con la referencia y en caso que sea mayor o menor se establecen las velocidades de las articulaciones para lograr que el robot se acerque o se aleje respectivamente (Ver Anexo3).

La imagen siguiente muestra al robot en al inicio del laberinto:

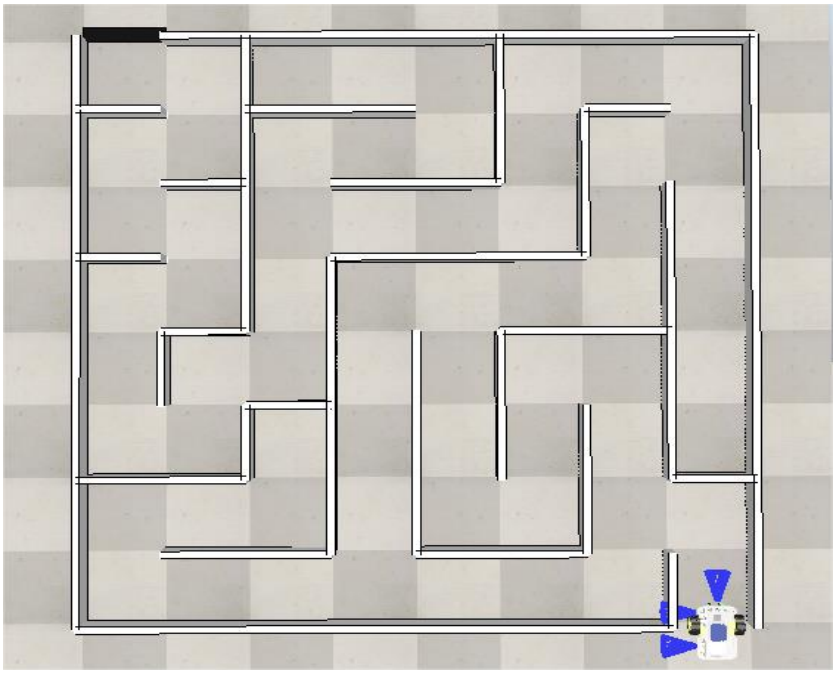


Figura 49. Simulación de la escena 3

El recorrido realizado por el robot desde su posición de inicio hasta la meta se muestra en el siguiente gráfico.

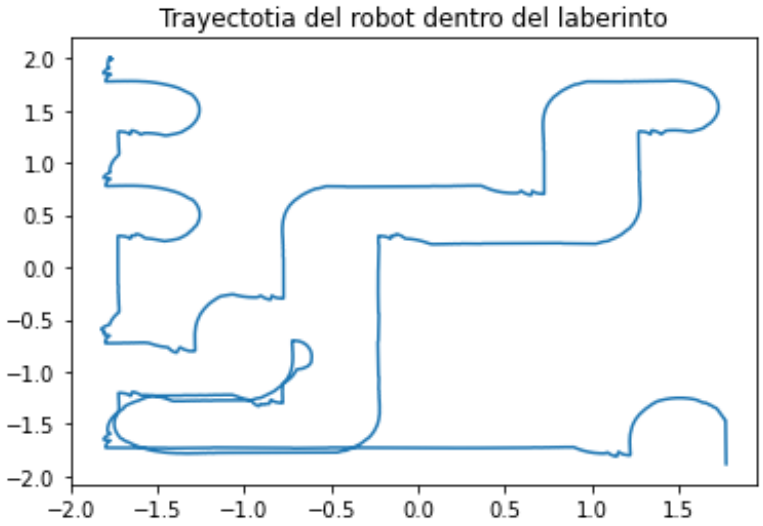


Figura 50. Trayectoria desarrollada por el robot en la escena 3

Escena 4

Finalmente, en la última escena se consiguió un desplazamiento adecuado de principio a fin utilizando los sensores de visión del robot para seguir la línea impresa en el piso. El robot utiliza el algoritmo del seguidor de línea para seguir la trayectoria y el del seguimiento de pared para que al detectar un obstáculo sea capaz de evitarlo sufriendo una pequeña desviación de la trayectoria.

Con el sensor de proximidad instalado en su parte frontal consiguió detectar los objetos en su camino. En dichos casos logró desviarse de manera efectiva y bordear dichos objetos, empleando los sensores de proximidad instalados a su izquierda, hasta reencontrarse con la línea. Sin dificultades el robot logró evadir los obstáculos encontrados en su camino para finalmente recorrer la trayectoria deseada que se muestra a continuación:

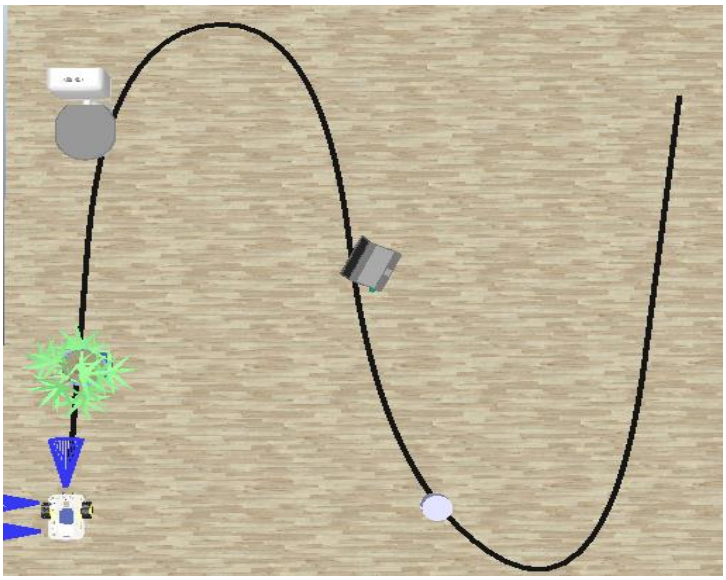


Figura 51. Simulación de la escena 4

La siguiente imagen muestra al robot durante la evasión del primer obstáculo:

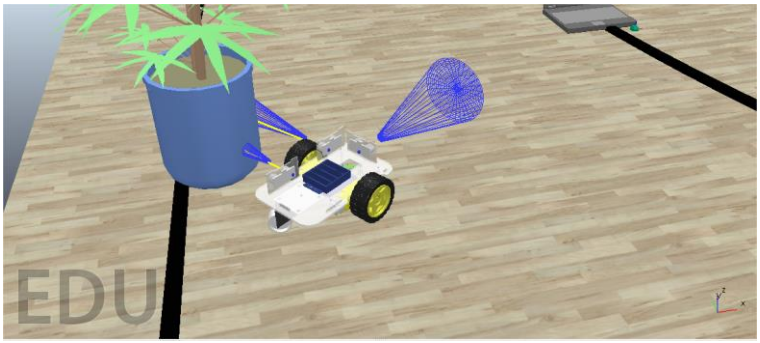


Figura 52. Robot durante la evasión del primer obstáculo de la escena 4

El siguiente gráfico muestra el camino recorrido por el robot durante la simulación de la escena.

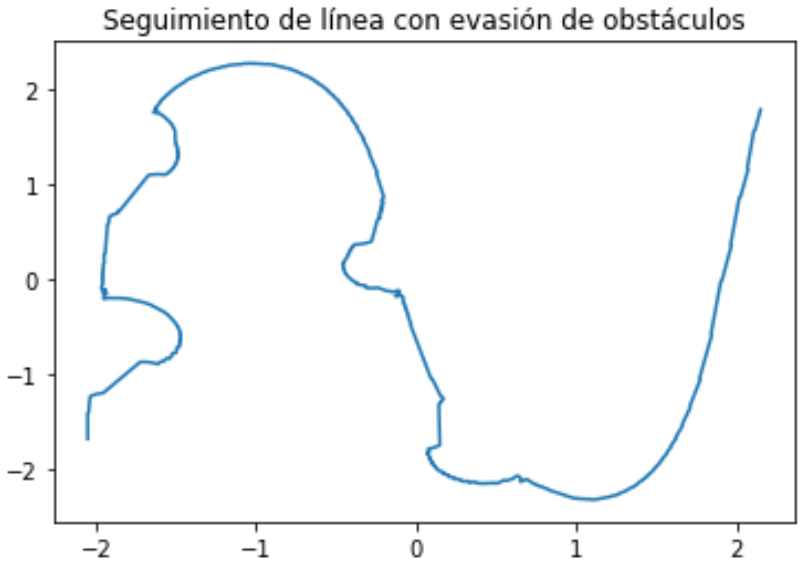


Figura 53. Camino recorrido por el robot en la escena 4

2.4 Programación de algoritmos en LUA y en C (Arduino)

Los algoritmos de control anteriormente mostrados fueron programados en Lua, sin embargo, para la implementación de los mismos en el robot real sería necesaria la programación de su controlador Arduino Mega. Es por ello que en la siguiente tabla se muestran las similitudes entre la programación en LUA y en C (Arduino).

Tabla 2. Similitudes en la programación de LUA y C (Arduino)

Etapa	LUA	ARDUINO
Inicialización	*function sysCall_init*() Obtener manejadores de los objetos a través de la API: *sim.getObject*	*void setup* () Declarar los pines a través de "*pinMode*"
Leer sensores	Dentro de: "*function sysCall_sensing()*" a través de la API: "*sim.read____Sensor*"	Dentro de: "*void loop* ()" a través de librerías específicas dependiendo del tipo de sensor a utilizar
Comparar	if	if
Establecer velocidad	Dentro de: *"function sysCall_actuation()"* a	Dentro de: "*void loop* ()" a través de: "*digitalWrite*"

	través de la API *sim.setJointTargetVelocity*	

2.5 Escenas basadas en la inspección de granjas solares fotovoltaicas

Las escenas desarrolladas se basan en una de las aplicaciones típicas de la robótica móvil como son los vehículos de guiado automático (AGV), empleados en la inspección de granjas o parques solares. Este apartado se enmarca dentro de una de las tareas del proyecto Ciencia y Conciencia financiado por la Universidad de Oriente: DEHOT^{pv}.

La inspección de parques solares constituye un método sencillo para conocer el estado de los paneles y consiste en la detección de puntos calientes mediante una cámara termográfica incorporada a un AGV que realiza el seguimiento de una trayectoria. Este es un servicio basado en un monitoreo ágil y eficaz que disminuye los problemas que afectan la producción de energía, supone una reducción en los costos de reparaciones tardías y disminuye la probabilidad de daños irreversibles contribuyendo a la reducción de los altos costos de producción que dichas afectaciones representan en el balance económico de la planta generadora [25]. Los vehículos encargados de realizar la inspección pueden ser terrestres o aéreos, y emplean, con el fin de recorrer la trayectoria deseada, distintos tipos de tecnología diferentes entre las que se encuentran el guiado óptico y el guiado por GPS.

El flujograma siguiente muestra una síntesis del algoritmo para realizar la inspección:

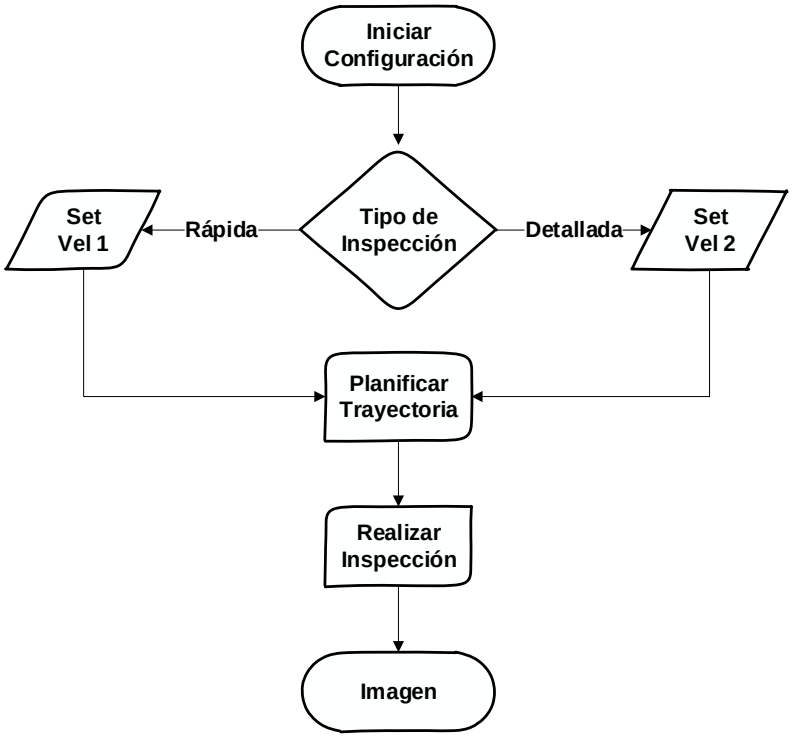


Figura 54 Flujograma de inspección de granja solar

Las escenas que se simulan a continuación están enfocadas en el seguimiento de la trayectoria del vehículo que propicia que la cámara realice una correcta toma de imágenes.

2.5.1 Inspección de una granja solar aplicando el guiado óptico en CoppeliaSim

La realización de la inspección de granja solar por un vehículo terrestre aplicando el guiado óptico consiste en seguir una línea de un color determinado por medio de los sensores ópticos que posee el robot instalados en su parte frontal inferior. Dicha línea se encuentra impresa en el piso y guía el camino del robot. Una variante de este método es el Guiado magnético, para el cual se sustituyen los sensores ópticos por magnéticos y se construye la línea de un material metálico.

Figura 55. Vehículo terrestre que realiza la inspección

2.5.1.1 Simulación de la inspección

La escena del parque solar se construyó por medio del modelo de un módulo solar diseñado en Solidworks [26] y exportado al simulador. Para el terreno se emplea el objeto que ofrece el programa para la simulación de un terreno de concreto. Para la creación de la pista se aplica el procedimiento explicado anteriormente en el epígrafe 2.3.1.

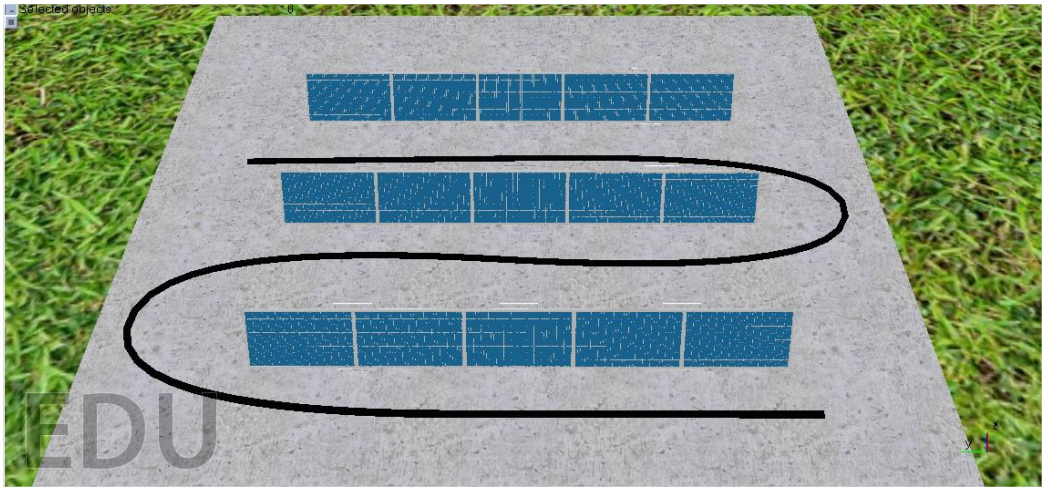


Figura 56. Escena de parque solar para la inspección con guiado óptico

Para esta escena se le incorporó al modelo 3D del EduRoMAA la cámara encargada de tomar las imágenes de los paneles y su base. Se cambió su color y se añadió, además, una tapa superior y cuatro laterales.

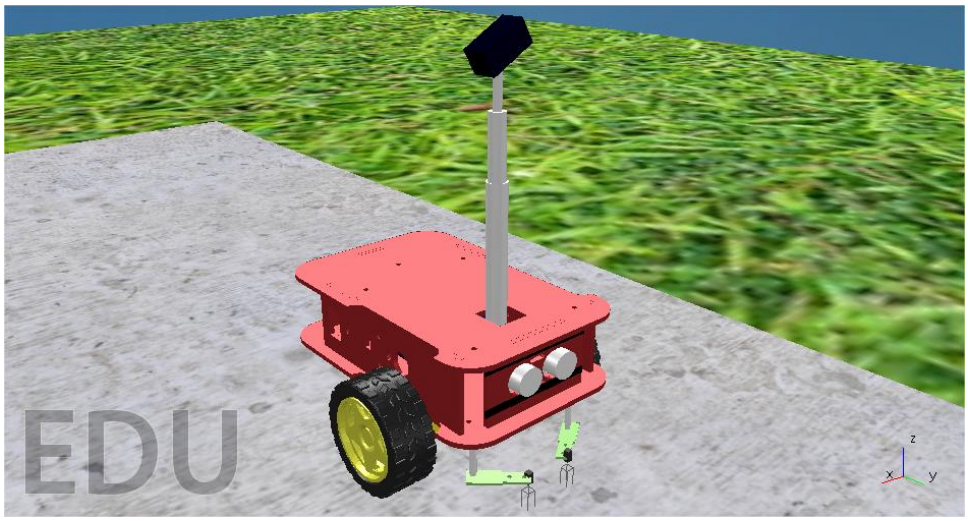


Figura 57. Modelo 3D del EduRoMAA con modificaciones

El algoritmo que se implementa es exactamente el del seguidor de línea explicado en el epígrafe 2.3.2.

2.5.1.2 Resultados de la simulación de la inspección aplicando guiado óptico

La escena se desarrolló de manera exitosa como se muestra en la imagen siguiente.

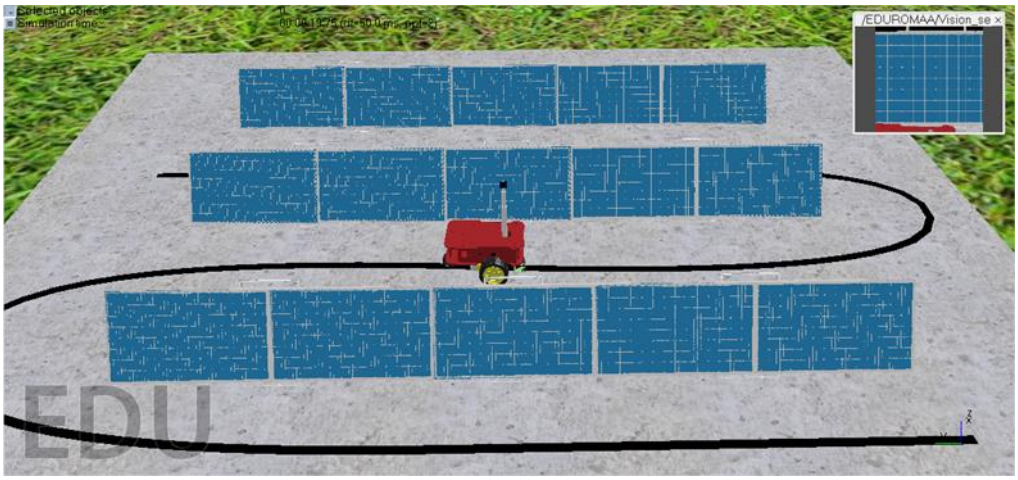


Figura 58. Escena de inspección de granja solar aplicando el guiado óptico

Para este tipo de guiado es necesario mantener constante la orientación de la cámara para realizar una correcta toma de imágenes. Para ello se emplean las API *sim.GetObjectOrientation*, que permite leer la orientación inicial de un objeto, y *sim.SetObjectOrientetion* para modificarla, algo que, en el mundo real resulta mucho más complicado e implica la utilización de una unidad de medición inercial.

Este tipo de guiado presenta inconvenientes como son las irregularidades del terreno q causan el movimiento de la cámara provocando que la calidad de la imagen no sea

optima, por lo que debe realizarse en un terreno llano y sin obstáculos que permita además la impresión de una línea. Además, se limita a los parques ubicados en la superficie terrestre, ya que resulta muy difícil el desplazamiento de este tipo de AGV en los parques instalados sobre techos y cubiertas e imposible en los parques solares flotantes.

2.5.2 Inspección de una granja solar aplicando el guiado GPS en CoppeliaSim

Esta escena se basa en el guiado GPS que permite seguir una trayectoria formada por un conjunto de coordenadas (*waypoints*). La escena consiste en la simulación del algoritmo de "Persecución pura" aplicado a un dron para el seguimiento de la trayectoria necesaria en la inspección de una granja solar, como muestra la siguiente figura.

Figura 59. Inspección de granja solar aplicando guiado GPS

2.5.2.1 Algoritmo de persecución pura

Es un algoritmo de rastreo que funciona calculando la curvatura que deberá tomar un vehículo desde su posición actual hasta su meta. El objetivo del algoritmo es elegir una referencia que esté a cierta distancia por delante del vehículo y dentro del camino que este debe seguir, como se muestra en la figura 48. Esta distancia comúnmente es nombrada *Look Ahead* y mientras más pequeño sea su valor, será menor el error de navegación del vehículo [27].

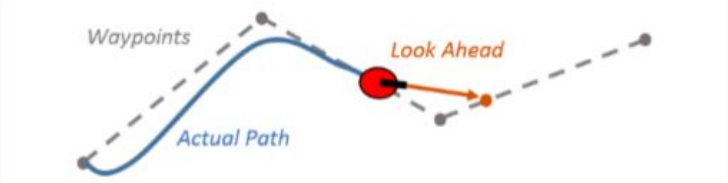


Figura 60. Algoritmo de Persecución Pura

Dicha distancia es la principal variable a tener en cuenta en este método. Normalmente, su valor es fijo a lo largo de todo el camino y se decide su magnitud antes de comenzar [27]. El algoritmo se expresa como se muestra en la figura siguiente.

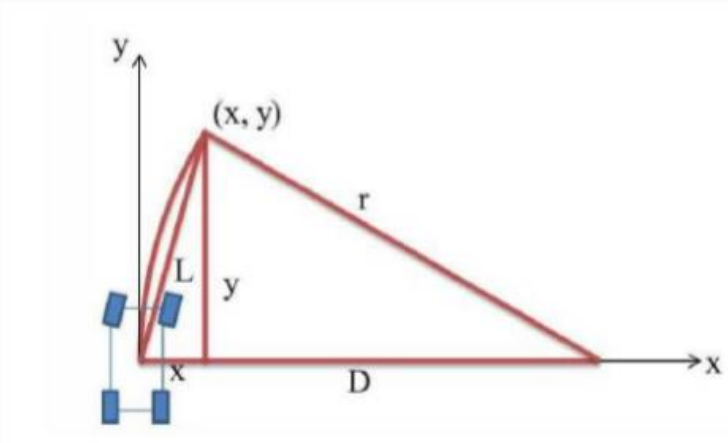


Figura 61. Parámetros del algoritmo de persecución pura

Los ejes x e y son el sistema de coordenadas del vehículo. El punto (x,y) está situado a cierta distancia L del vehículo y r es el radio de curvatura del arco . La relación entre las variables x, L y r es la siguiente:

$$D + X = r \tag{1}$$

$$D^2 + y^2 = r^2 \tag{2}$$

$$x^2 + y^2 = L^2 \tag{3}$$

De las ecuaciones (1), (2) y (3) se concluye que:

$$r^2 - 2rx + x^2 + y^2 = r^2$$

$$r = \frac{L^2}{2x}$$

Finalmente, se puede calcular el radio de curvatura requerido para que el vehículo vaya por el camino deseado, eligiendo correctamente una distancia *Look Ahead* y calculando el error del camino X [28].

2.5.2.2 Creación de la escena en CoppeliaSim

Para la creación de la escena se emplea el objeto tipo campo alto para la simulación de un terreno, el mismo modelo de los paneles de la escena anterior y el modelo genérico de un dron de cuatro hélices: *"Quadcopter"* ofrecido por el propio software y que incluye una cámara dirigida hacia el suelo. Este modelo posee dentro de su script principal las ecuaciones que permiten calcular durante la simulación la distancia *Look Ahead* y en correspondencia establecer las velocidades de los motores de sus hélices.

Para generar la ruta de vuelo se emplea un objeto tipo *Path*, que permite generar una trayectoria por medio de la unión de un conjunto de "puntos de control" equivalentes a las coordenadas GPS de la vida real, como se muestra en la siguiente figura.

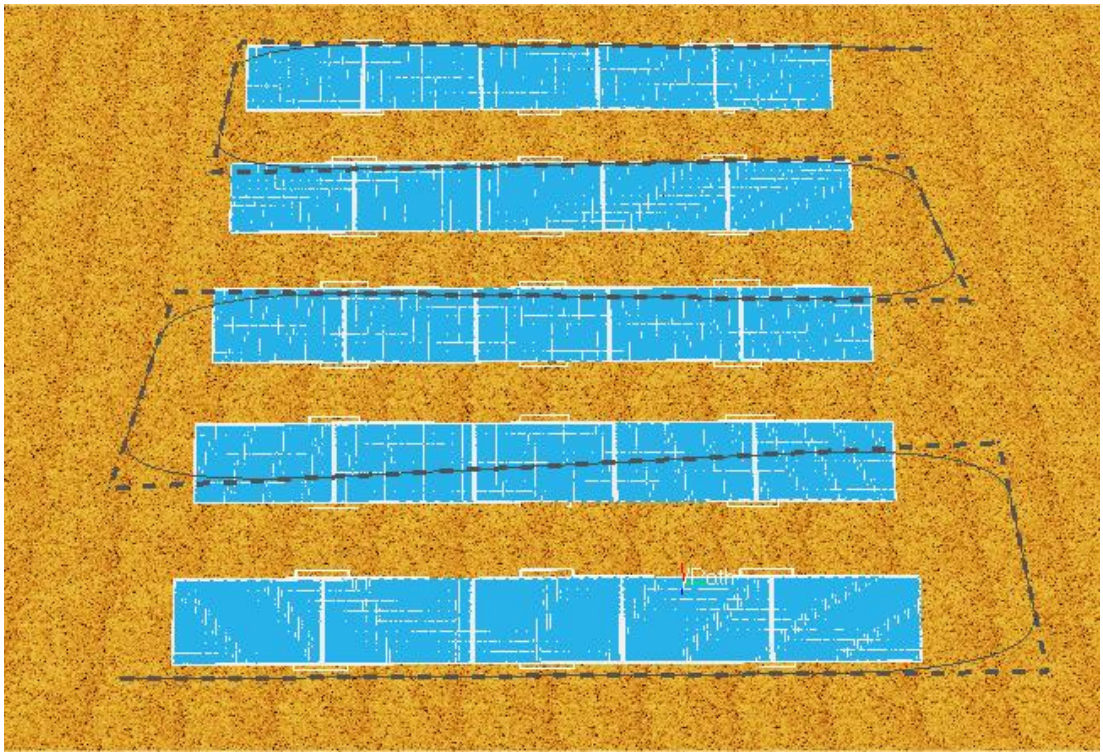


Figura 62.Escena de inspección de granja solar aplicando guiado GPS

2.5.2.3 Implementación del algoritmo de persecución pura en CoppeliaSim

Para la simulación del algoritmo en CoppeliaSim, el modelo del dron ofrece por defecto una referencia asignada en su script por medio de la API interna *sim.setObjectParent*

la cual no es más que un objeto ficticio al que se le modifica su script (ver Anexo5) y por medio de la API *sim.followPath*, se le ordena desplazarse por los puntos de control de la trayectoria generada anteriormente y a qué velocidad hacerlo.

El proceso para lograr que el dron se desplace a una distancia *Look Ahead* de la referencia tiene lugar dentro del script del *Quadcopter*. Mediante la API *sim.getObjectPosition* se obtiene la posición del modelo y de la referencia para realizar los cálculos de los valores de velocidad que deben tener las hélices para lograr el desplazamiento deseado a través de la API *sim.setScriptSimulationParameter*.

De esta manera el dron sigue la referencia desplazándose a su vez por la trayectoria anteriormente establecida. Como resultado de este algoritmo se obtiene el efecto que pretende conseguir el método del algoritmo de persecución pura.

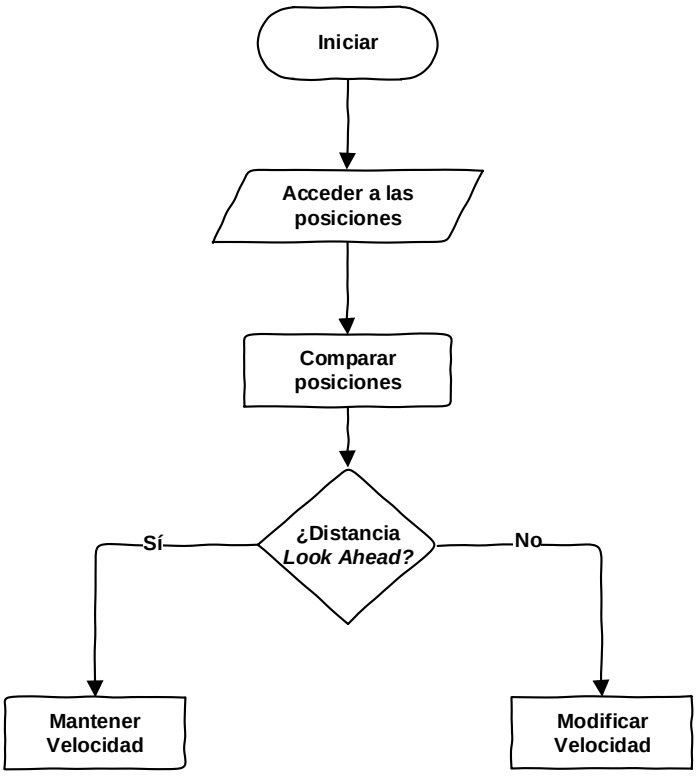


Fig. 1 Algoritmo de persecución pura en CoppeliaSim

Si bien el control dinámico del vuelo de un dron resulta complicado, esta situación en la actualidad está resuelta por la mayoría de los fabricantes, haciendo que los usuarios se enfoquen principalmente en la aplicabilidad de los mismos en diferentes entornos.

2.5.2.4 Resultados de la simulación de la inspección aplicando guiado GPS

La escena se desarrolló de manera exitosa como se muestra en la imagen siguiente:

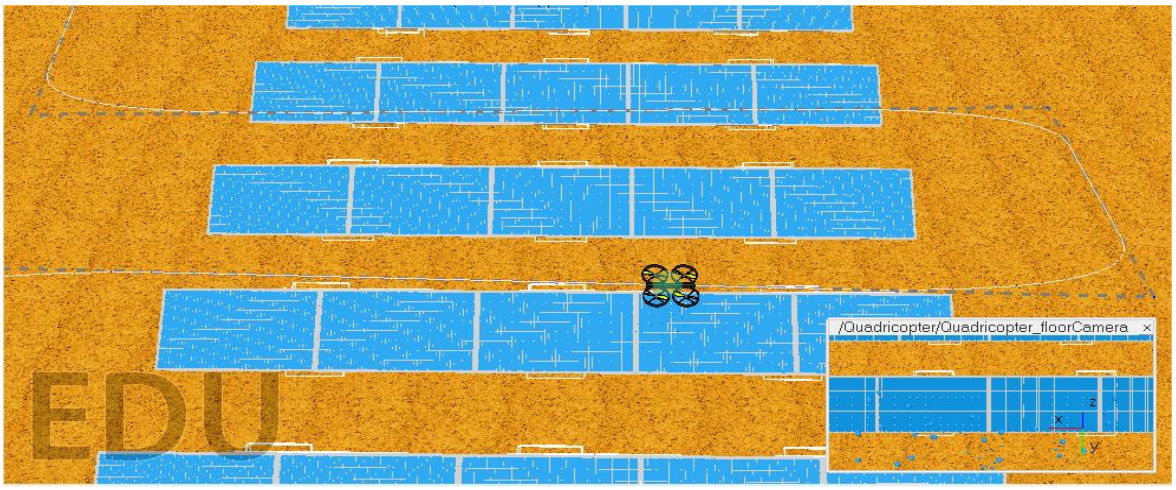


Figura 63. Escena de inspección de granja solar aplicando el guiado GPS

Para este tipo de guiado no es necesario cambiar la orientación de la cámara ya que el dron mantiene constante la suya.

La figura siguiente muestra el máximo sobrepaso, las oscilaciones y el error de posición durante los primeros segundos de la inspección.

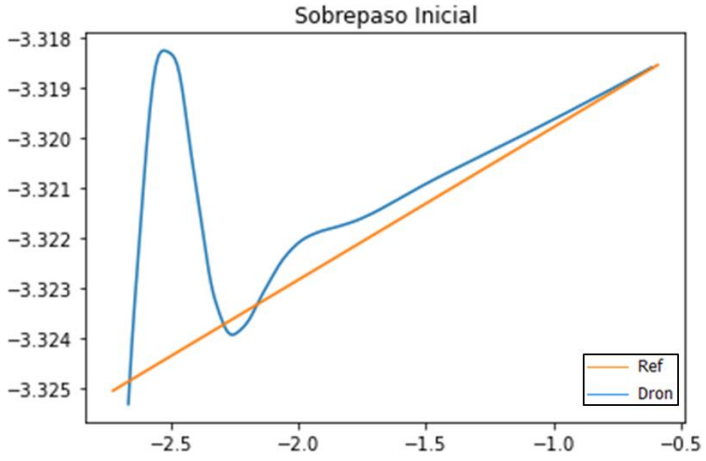


Figura 64. Comparación referencia-trayectoria en los primeros segundos

La figura muestra el recorrido del dron y de la referencia una vez completada la trayectoria correspondiente. Se puede apreciar un error de desviación prácticamente nulo debido en parte a que la distancia *Look Ahead* escogida fue prácticamente cero.

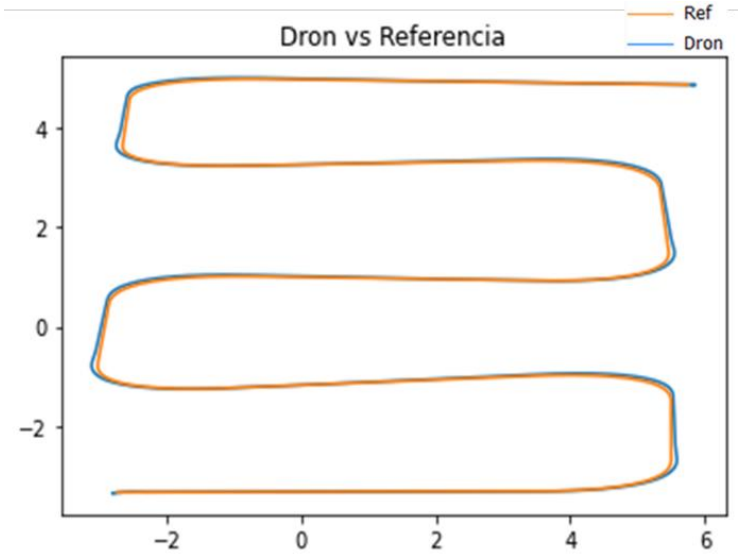


Figura 65. Trayectoria Dron-Referencia al finalizar la inspección

Tras la aplicación de cualquiera de los dos métodos utilizados para realizar la inspección, posterior al procesado de las imágenes por medio de una inteligencia artificial, el resultado será un gráfico que muestre la localización exacta de los puntos calientes y su temperatura.

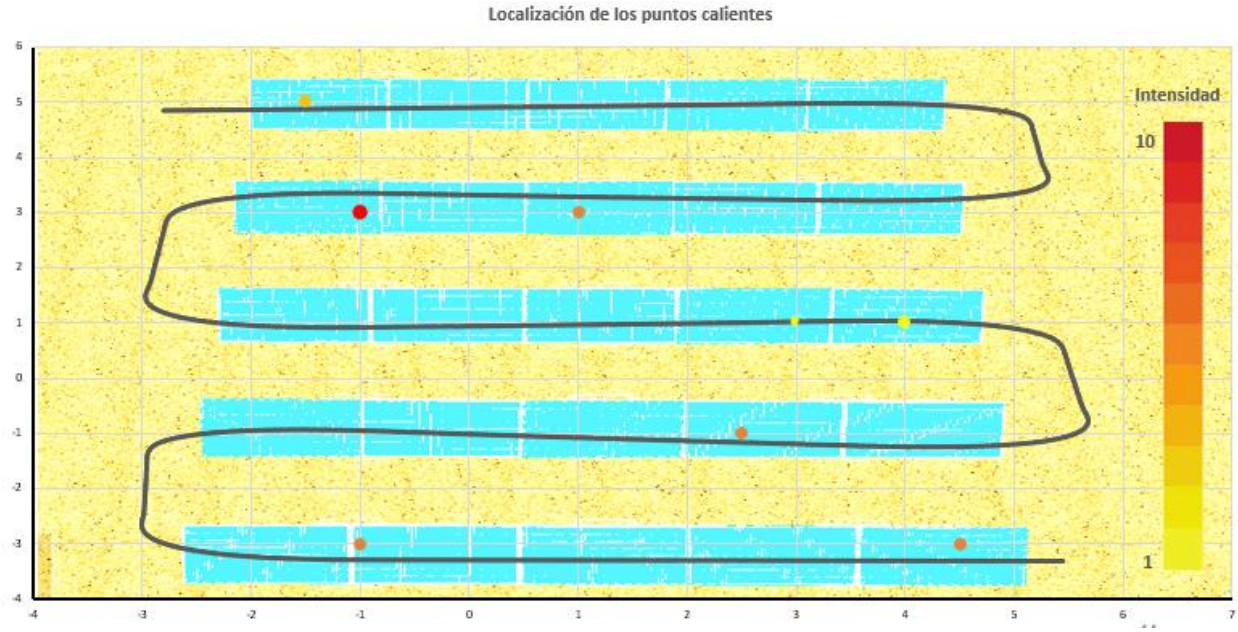


Figura 66. Resultado de la inspección

2.6 Conclusiones parciales

Durante este capítulo se explotó las potencialidades de la herramienta a partir del trabajo realizado que permitió el cumplimiento de las siguientes tareas:

- Se diseñó el modelo 3D del robot EduRoMAA auxiliándose en las bondades del software, que permite importar objetos desde programas CAD posibilitando un alto nivel de semejanza entre el modelo y el robot real.
- Se crearon diferentes escenarios virtuales mediante la inserción de objetos ofrecidos por el simulador que facilitaron en gran medida la realización de esta tarea.
- Se desarrollaron algoritmos de control que fueron implementados al modelo y se evaluó su comportamiento dentro de los escenarios obteniendo resultados satisfactorios de acuerdo a lo esperado.
- Se desarrollaron dos escenas basadas en la inspección de granjas solares fotovoltaicas demostrando la potencialidad de la herramienta para la simulación de escenas reales.
- La inspección de parques solares por guiado GPS aplicado a un dron se puede realizar a cualquier tipo de parque solar sin importar su ubicación, lo que constituye una ventaja importante respecto al empleo del otro guiado.

Conclusiones Generales

El desarrollo de la robótica constituye un tema de investigación relevante a nivel mundial. El diseño de algoritmos de control cada vez más precisos y confiables sigue siendo un desafío con vistas a elevar el desempeño y aplicabilidad de estas máquinas. En esta investigación, se propuso el empleo del simulador de robótica CoppeliaSim como herramienta para la experimentación en el desarrollo de algoritmos de control de robótica móvil. Como resultado de la investigación, es posible afirmar que los objetivos trazados fueron logrados, arribando a las siguientes conclusiones a partir del análisis de los resultados alcanzados:

- CoppeliaSim permite crear escenas y modelos para comprobar algoritmos de manera muy similar a la realidad.
- El uso del simulador CoppeliaSim permite probar y depurar algoritmos sin el riesgo de dañar el robot o su entorno, lo que resulta especialmente importante en aplicaciones críticas o peligrosas, ahorrando recursos.
- Al no depender de la disponibilidad física de robots este simulador permite acelerar el proceso de desarrollo y prueba de algoritmos, economizando grandes cantidades de tiempo.
- La plataforma CoppeliaSim se muestra como una herramienta efectiva y útil para la formación y educación en robótica móvil, permitiendo practicar y experimentar sin restricciones de tiempo y espacio en escenarios pre elaborados donde es posible comprobar el comportamiento de distintos robots ante determinados algoritmos.
- Producto de los resultados de la simulación de las escenas basadas en una aplicación real se pudo afirmar que el guiado GPS permite la inspección de todo tipo de parques solar independientemente del tipo de terreno o su localización, siendo más efectivo que el guiado óptico.

Recomendaciones

- Continuar utilizando la plataforma en el desarrollo de simulaciones que requieran algoritmos más complejos.
- Explotar la flexibilidad brindada por el software al permitir la interacción mediante APIs externas con compiladores de otros lenguajes como Python, C/C++, y con el versátil software de programación y cálculo numérico Matlab.
- Incorporar el simulador a la docencia para desarrollar en los estudiantes habilidades de programación, creación e implementación de algoritmos de control de robótica móvil.
- Evaluar los algoritmos desarrollados en el entorno con la plataforma Arduino para analizar las respuestas del robot real.
- Aplicar el algoritmo de persecución pura a un dron real para poder integrar este trabajo con el de detección de puntos calientes.

Referencias

[1] J. Aporta Costela, «Control de flotas de robots utilizando el software CoppeliaSim en Python,» Almeria, 2021.

[2] Á. Vicente Cánovas, «Estudio de entornos virtuales para robots móviles. Desarrollo de aplicaciones de control de robots móviles con configuración diferencial.,» Valencia, 2020.

[3] R. Boa de la Fuente, «Desarrollo de un modelo en simulación en v-rep de un robot móvil basado en smartphone y soporte al lenguaje python,» La Coruña, 2019.

[4] L. E. Solaque Guzmán, M. A. Molina Villa y E. L. Rodríguez Vásquez , «Seguimiento de trayectorias con un robot móvil de configuración diferencial,» pp. 26 - 34, 2014.

[5] D. A. J. Hermo, «Creación de un modelo de simulación de robot móvil para conducción autónoma,» La Coruña, 2020.

[6] A. Perera Robbio, «Presidencia de Cuba,» 13 febrero 2023. [En línea]. Dispnonible: https://www.presidencia.gob.cu/es/noticias/inteligencia-artificial-el-bienestar-posible/. [Último acceso: 2 agosto 2023].

[7] D. Labrada, «Diseño e implementación de un Entorno Virtual de un manipulador ArmX de cuatro grados de libertad,»Santiago de Cuba, 2018.

[8] J. Valazco, «Análisis y comparación de las principales plataformas de simulación robótica y su integración con ros,» Madrid, 2019.

[9] J. M. Cañas Plaza, . M. A. Cazorla Quevedo y V. Matellan, «Uso de simuladores en docencia de robotica movil,» pp. 268-277, 2009.

[10] «Coppelia Robotics,» [En línea]. Dispnonible: https://www.coppeliarobotics.com/helpFiles/ index.html. [Último acceso: 26 Julio 2023].

[11] «Webots,» [En línea]. Dispnonible: https://cyberbotics.com/doc/guide/index.

[12] «AirSim,» [En línea]. Dispnonible: https://microsoft.github.io/AirSim/.

[13] «Gazebo,» [En línea]. Dispnonible: http://classic.gazebosim.org/. [Último acceso: 21 Julio 2023].

[14] A. Martınez Rodrıguez, «Desarrollo de una herramienta educativa de simulación para robótica móvil,» Málaga, 2016.

[15] «ARGoS,» [En línea]. Dispnonible: https://www.argos-sim.info/user_manual.php. [Último acceso: 27 mayo 2023].

[16] G. López Valero, «Diseño, estudio y simulación de un robot bípedo en CoppeliaSim (V-Rep) y construcción de un prototipo en impresión 3D,» Valencia, 2020.

[17] I. C. Millán, «Programación de robot móvil con prevención de colisiones CoppeliaSim,» Bogotá, 2021.

[18] A. M. Dominguez, «Modelado y Simulación de un Robot LEGO Mindstorms EV3 mediante V-REP y Matlab,» Málaga, 2016.

[19] B. M. Damián García, E. García Peralta, W. Campos y L. Cortez, «Modelado y simulación de un robot Autobalanceado mediante Coppelia Sim,» Chilpancingo, 2021.

[20] L. Armesto, «youtube,» [En línea]. Dispnonible: https://m.youtube.com/playlist?list=PLjzuoBhdtaXOYfcZOPS98uDTf4aAoDSRR. [Último acceso: 17 noviembre 2023].

[21] «LUA the programming language,» 14 Mayo 2023. [En línea]. Dispnonible: https://www.lua.org/about.html. [Último acceso: 1 Noviembre 2023].

[22] D. Gonzalez, «GitHub,» 3 Mayo 2023. [En línea]. Dispnonible: https://github.com/ciiiutnfrc/eduromaa. [Último acceso: 1 Noviembre 2023].

[23] «Arduino,» [En línea]. Dispnonible: https://arduino.cl/producto/arduino-mega-2560/. [Último acceso: 17 noviembre 2023].

[24] A. López, «Grabcad Comunity,» 4 Julio 2019. [En línea]. Dispnonible: https://grabcad.com/library/llanta-de-motor-reductor. [Último acceso: 1 Noviembre 2023].

[25] J. Ulloa Ayala, J. Castellanos Benavides y M. Montaño Gómez, «Implementación de la termografía con drones en la inspección de granjas solares.,» Bogotá, 2021.

[26] «Cults.,» [En línea]. Dispnonible: http://cults3d.com/es/etiquetas/panel%20solar. [Último acceso: 10 octubre 2023].

[27] H. Pérez León, «Generación y seguimiento de trayectorias para un vehículo aéreo multi-rotor,» Sevilla, 2018.

[28] R. Craig Coulter, «Implementation of the Pure Persuit Path Tracking Algorithm,» Pittsburgh, 1992.

Anexos

Anexo1. Código Seguidor de Línea

```
function sysCall_init()
    -- do some initialization here
    leftJoint = sim.getObject ('/Joint_Izq')
    rightJoint = sim.getObject ('/Joint_Derecha')
    leftSensor = sim.getObject ('/LeftSensor')
    rightSensor = sim.getObject ('/RightSensor')
    print (leftSensor)
end
function sysCall_actuation()
    v = 1
    dv = 5
    sim.setJointTargetVelocity (leftJoint, v)
    sim.setJointTargetVelocity (rightJoint, v)

    if (data_left ~= nil) then
    intensity_left = data_left [11]
    intensity_right = data_right [11]
    --para que gire siguiendo la linea a la izquierda
        if (intensity_right > 0.5 and intensity_left < 0.5) then
        sim.setJointTargetVelocity (rightJoint, v+dv)
        end
    -- para que gire siguiendo la linea a la derecha
        if (intensity_left > 0.5 and intensity_right < 0.5) then
        sim.setJointTargetVelocity (leftJoint, v+dv)
        end
    end
end
function sysCall_sensing()
    --Leer datos de los sensores de visión
    result,data_left = sim.readVisionSensor (leftSensor)
    result, data_right = sim.readVisionSensor (rightSensor)
end
```

Anexo2. Código de seguimiento entre paredes

```
function sysCall_init()
    lmotor=sim.getObjectHandle("./Joint_Izq")
    rmotor=sim.getObjectHandle("./Joint_Derecha")
    sensorF=sim.getObjectHandle('./SF')
    sensorL=sim.getObjectHandle('./SI')
    sensorR=sim.getObjectHandle('./SD')
    avg_default= 0.15
    avg_default1= 0.15
end
function sysCall_actuation()

    sim.setJointTargetVelocity(lmotor,2)
    sim.setJointTargetVelocity(rmotor,2)
    if (FWD==1) then
    sim.setJointTargetVelocity(lmotor,0)
    sim.setJointTargetVelocity(rmotor,0)
    end
    if (FWD==2) then sim.setJointTargetVelocity(lmotor,4) end
    if (FWD==3) then sim.setJointTargetVelocity(rmotor,4) end

end

function sysCall_sensing()
    --
    flag1,I= sim.readProximitySensor(sensorL)
    flag2,D= sim.readProximitySensor(sensorR)
    flag3,F= sim.readProximitySensor(sensorF)

    if ( flag3==1 and F<0.2) then FWD = 1 --Obstaculo en frente
    elseif (I < D) then FWD = 2
    elseif (I > D) then FWD = 3
    else FWD=0  end
    end
```

Anexo3. Código seguidor de pared para resolver laberinto

```
function sysCall_init()

    lmotor=sim.getObjectHandle("./Joint_Izq")
    rmotor=sim.getObjectHandle("./Joint_Derecha")
    sensorLF=sim.getObjectHandle('./I1')
    sensorLR=sim.getObjectHandle('./I2')
    sensorF=sim.getObjectHandle('./Sonar')
    leftSensor = sim.getObject ('/LeftSensor')
    avg_default= 0.2

end

function sysCall_actuation()

    sim.setJointTargetVelocity(lmotor,1)
    sim.setJointTargetVelocity(rmotor,1)

    if (avg == LR)then
        sim.setJointTargetVelocity(rmotor,5)
        sim.setJointTargetVelocity(lmotor,0)
    end
    if ( FWD==1) then
        sim.setJointTargetVelocity(lmotor,15)
        sim.setJointTargetVelocity(rmotor,-15)
    end
    if (FWD==2) then sim.setJointTargetVelocity(lmotor,3) end
    if (FWD==3) then sim.setJointTargetVelocity(rmotor,3) end
    if ( FWD==4) then
    sim.setJointTargetVelocity(lmotor,0)
    sim.setJointTargetVelocity(rmotor,0)
    end
    ------------------------------------------
```

```
     if (data_left ~= nil) then
        intensity_left = data_left [11]
        if (intensity_left < 0.5) then
        sim.setJointTargetVelocity (rmotor, 0)
        sim.setJointTargetVelocity (lmotor, 0)
        end
    end
end

function sysCall_sensing()

    flag1,LF=sim.readProximitySensor(sensorLF)
    flag2,LR = sim.readProximitySensor(sensorLR)
    flag3,F= sim.readProximitySensor(sensorF)

    if (flag1==0 and flag2 == 1) then
    avg = LR
    elseif (flag1==1 and flag2 == 0) then
    avg = LF
    elseif (flag1 ==1 and flag2== 1) then
    avg = 0.5*(LF+LR)
    else
    avg = avg_default
    end

    if ( flag3==1 and F<0.18) then FWD = 1
    elseif (avg < 0.17) then FWD = 2
    elseif (avg > 0.175) then FWD = 3
    else FWD=0  end
    ------------------------------------------------------
    result,data_left = sim.readVisionSensor (leftSensor)

    end
```

Anexo4. Código seguidor de línea con evasión de obstáculos

```
function sysCall_init()

    lmotor = sim.getObject ('/Joint_Izq')
    rmotor = sim.getObject ('/Joint_Derecha')
    leftSensor = sim.getObject ('/LeftSensor')
    rightSensor = sim.getObject ('/RightSensor')
    sensorLF=sim.getObjectHandle('./SI')
    sensorLR=sim.getObjectHandle('./SD')
    sensorF=sim.getObjectHandle('./SF')
    avg_default = 0.203
end

function sysCall_actuation()
    v = 1
    dv = 4
    sim.setJointTargetVelocity (lmotor, v)
    sim.setJointTargetVelocity (rmotor, v)

    if (FWD==1)then
        sim.setJointTargetVelocity (lmotor, v+dv)
        sim.setJointTargetVelocity (rmotor, 0)
    else
    if (data_left ~= nil) then
        intensity_left = data_left [11]
        intensity_right = data_right [11]
        if (intensity_left > 0.5 and FWD ==2 ) then
            sim.setJointTargetVelocity (lmotor, 2)
            sim.setJointTargetVelocity (rmotor,0)
        end
        if (intensity_left > 0.5 and FWD==3) then
            sim.setJointTargetVelocity (rmotor,2)
            sim.setJointTargetVelocity (lmotor,0)
        end
```

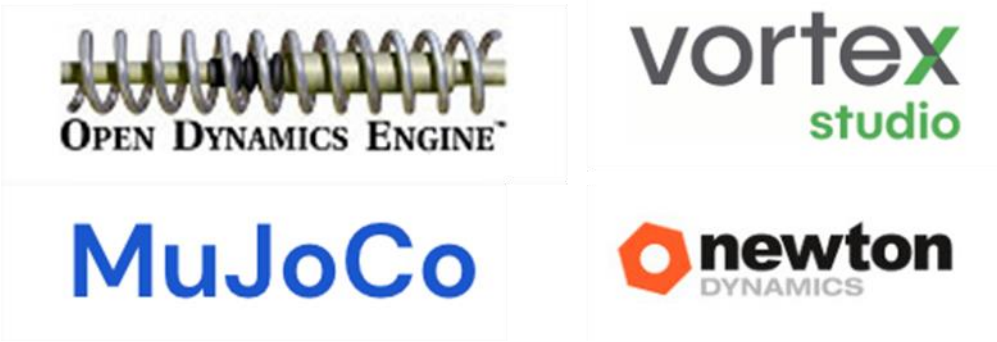
```
        if ( avg == LR)then
            sim.setJointTargetVelocity(rmotor,10)
            sim.setJointTargetVelocity(lmotor,0)
        end
        --Seguir línea girar a la izq
        if (intensity_right > 0.5 and intensity_left < 0.5) then
        sim.setJointTargetVelocity (rmotor, v+dv)
        sim.setJointTargetVelocity (lmotor, 0)
        end
        --Seguir línea girar a la der
        if (intensity_left > 0.5 and intensity_right < 0.5) then
        sim.setJointTargetVelocity (lmotor, v+dv)
        sim.setJointTargetVelocity (rmotor, 0)
        end
        if (intensity_left < 0.5 and intensity_right < 0.5) then
        sim.setJointTargetVelocity (lmotor, 10)
        sim.setJointTargetVelocity (rmotor, -10)
        end
    end
    end
end
```

```
function sysCall_sensing()
    --Leer valores de los sensores de visión
    result,data_left = sim.readVisionSensor (leftSensor)
    result, data_right = sim.readVisionSensor (rightSensor)
    --Leer valores de los sensores de proximidad
    flag1,LF = sim.readProximitySensor(sensorLF)
    flag2,LR = sim.readProximitySensor(sensorLR)
    flag3,F  = sim.readProximitySensor(sensorF)
    --Calcular distancia promedio detectada por los sensores laterales
    if (flag1==0 and flag2 == 1) then
    avg = LR
    diff = 0
    elseif (flag1==1 and flag2 == 0) then
    avg = LF
    diff = 0
    elseif (flag1 ==1 and flag2== 1) then
    avg = 0.5*(LF+LR)
    diff = LF - LR
    else
    avg = avg_default
    diff = 0
    end

    if ( flag3==1 and F < 0.5 ) then    FWD = 1 --detecta obstaculo
    elseif (avg < 0.2)          then    FWD = 2 --acercandose al objeto
    elseif (avg > 0.205)        then    FWD = 3 --alejandose del objeto
    else FWD = 0  end

end
```

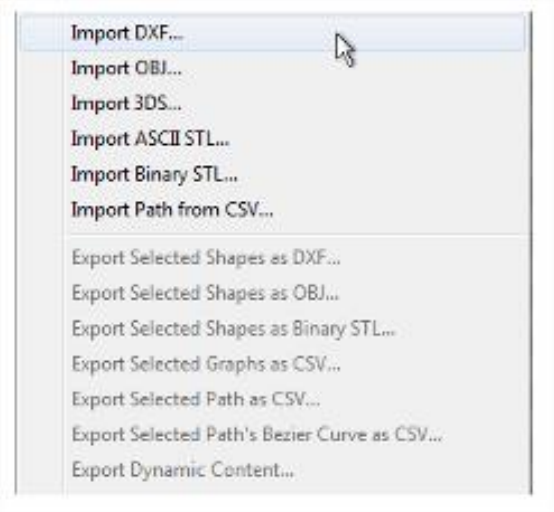
Anexo5. Código del Script de la referencia del dron

```lua
function sysCall_init()
    corout=coroutine.create(coroutineMain)
end

function sysCall_actuation()
    if coroutine.status(corout)~='dead' then
        local ok,errorMsg=coroutine.resume(corout)
        if errorMsg then
            error(debug.traceback(corout,errorMsg),2)
        end
    end
end

function coroutineMain()
    local thisObjectHandle = sim.getObjectHandle(sim.handle_self)
    local pathHandle = sim.getObjectHandle('Path')
    local changePositionOnly = 1
    sim.followPath(thisObjectHandle, pathHandle, changePositionOnly, 0, 0.17,15)
end
```

Anexo6. Motores de física de CoppeliaSim



Anexo7. Formatos en los que permite importar y exportar CoppeliaSim



Anexo8. Algunos modelos de robots móviles ofrecidos por el software

Anexo8. Algunos modelos de robots fijos ofrecidos por el software

Anexo9. Investigadores de la FIE experimentando con el robot real.