Escuela Superior de
Ingeniería y Tecnología
Sección de Ingeniería Industrial

Grado en Ingeniería Electrónica Industrial y Automática
TRABAJO FIN DE GRADO

Construcción de plataforma software para el diseño y la simulación de robots móviles

Carlos Sosa Marrero

Tutor: Santiago Torres Álvarez
Cotutor: Antonio L. Morell González

Julio de 2016

A Elena, Miguel y Alejandro,
porque su cariño será siempre
lo mejor de estos cuatro años.

Índice

Índice de figuras

Índice de cuadros

Resumen

Esta memoria describe las caracterícticas y posibilidades de Koala, una plataforma software para el diseño y testeo de robots móviles. Se detalla, asimismo, el trabajo realizado para su creación.

Koala está compuesta por dos módulos. El primero consiste en una aplicación desarrollada en el entorno de interfaces de usuario GUIDE de Matlab. Permite al usuario realizar de forma sencilla las labores de diseño de robots tipo diferencial, triciclo y cuatriciclo con huellas circulares o cuadrangulares. Este módulo se encarga, además, del control del robot en un determinado escenario.

El segundo módulo hace uso de la plataforma para la experimentación con robots V-REP. Es responsable de crear el robot a partir de la información recabada por el primer módulo y mostrar en pantalla su comportamiento en una serie de escenarios.

Abstract

This report describes the features and possibilities of Koala, a software platform for the design and testing of mobile robots. It also explains the work carried out for its creation.

Koala is composed of two modules. The first one consists in an application developed in the Matlab user inferfaces environment GUIDE. It allows the user to perform in a simple way the design tasks of differential, tricycle and quadricycle robots with circular and quadrangular footprints. This module is also in charge of controlling the robot on a particular scene.

The second module uses the robots experimentation platform V-REP. It is responsible for creating the robot based on the information gathered by the first module and dispaying its behaviour in a series of scenes.

1. Introducción

1.1. Objetivos

El objetivo del presente Trabajo de Fin de Grado es la construcción de Koala, una plataforma software para el diseño y la simulación de robots móviles basada en el entorno de desarrollo de interfaces de usuario GUIDE de Matlab que facilite el diseño de un robot móvil a partir de parámetros introducidos por el usuario (configuración de las ruedas, forma y dimensiones).

Asimismo, se persigue ofrecer al usuario una herramienta para el testeo y visualización en pantalla del comportamiento del robot en determinados entornos predefinidos o creados previamente por el usuario con el software de simulación V-REP.

De esta forma, se pretende conseguir un producto con un gran rango de aplicación en docencia e investigación que simplifique y agilice las labores de diseño y testeo de robots móviles y la programación necesaria para el control integral del mismo.

1.2. Antecedentes

Koala surge como evolución de una aplicación desarrollada en el marco de la asignatura Sistemas Robotizados para la simulación del comportamiento de un robot diferencial *Roomba* en Matlab. Esta aplicación solo permitía al usuario escoger el radio del cuerpo del robot.

Además, como se muestra en la figura 1, la visualización del comportamiento del robot se realizaba a través de un gráfico en dos dimensones, una opción limitada y que, además, hacía necesaria la existencia de funciones destinadas a graficar el robot en cada instante de tiempo y que fuera el propio Matlab el encargado de obtener la posición y orientación del robot en todo momento mediante una aproximación discreta.

Asimismo, la aplicación carecía de una interfaz gráfica de usuario, por lo que el radio del robot y el comportamiento deseado del mismo eran introducidos por línea de comandos.

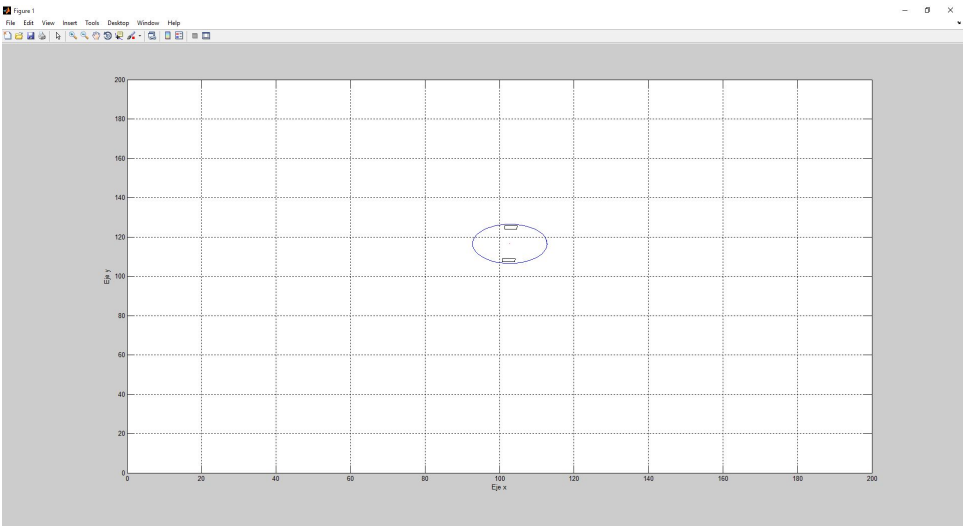

Figura 1: Aplicación para la simulación de un robot *Roomba*

1.3. Herramientas software utilizadas

Matlab en su versión R2010a. Se trata de un software matemático con múltiples aplicaciones en los campos del control, la robótica, la monitorización de la salud, el procesamiento de imágenes o las comunicaciones. Entre sus múltiples prestaciones destacan la manipulación y el tratamiento de datos y funciones y la implementación de algoritmos. Asimismo, incluye un editor de interfaces de usuario GUIDE que se utilizará para la creación de la interfaz de la plataforma desarrollada en el presente Trabajo.

El código desarrollado en Matlab es integrable con otros lenguajes. Esto facilitaría la exportación de la solución de control conseguida hacia algún tipo de sistema embebido (Arduino, Raspberry PI, etc.) si se optase por ello en una futura línea de trabajo.

V-REP (*Virtual Robot Experimental Platform*) en su versión educativa 3.3.1, la útlima lanzada hasta la fecha. Se trata de una plataforma software desarrollada por *Coppelia Robotics* para la experimentación con robots. Permite crear, editar, simular y evaluar cualquier sistema robótico creado por el usuario o escogido de su librería de modelos. Está basado en una arquitectura de control distribuido, lo que implica que los programas de control son enlazados directamente a los objetos de la escena. Asimismo, puede usar-

se como un programa independiente o integrarse de forma sencilla con otra aplicación, como es el caso de la plataforma objeto de este Trabajo.

2. Descripción general de la plataforma

2.1. Arquitectura de la plataforma

La plataforma software desarrollada se compone de dos grandes módulos que se detallan a continuación.

Módulo de la aplicación de Matlab La aplicación desarrollada en Matlab se encarga, a través de la GUI, de recoger las elecciones por parte del usuario en lo que respecta al escenario en el que ha de tener lugar la simulación, así como en lo referente a las características del robot cuyo comportamiento se desea testear. Idealmente, este módulo se encargaría también de la generación del robot y posibles objetos adicionales a partir de la información recabada. Sin embargo, la API remota de V-REP para Matlab aún carece de las funciones específicas para la creación de cuerpos geométricos y sensores. Por este motivo, se hace uso de la función genérica *simxCallScriptFunction* que permite llamar a una determinada función de un script asociado a un objeto de V-REP.

Asimismo, este módulo se encarga del control del robot simulado. Gracias a las funciones de la API remota, que a este respecto sí se encuentran ya implementadas, es posible obtener la información de los sensores del robot y aplicar la conveniente consigna a los motores.

Módulo de V-REP El módulo de V-REP es el encargado de llevar a cabo y mostrar en pantalla la simulación del robot diseñado por el usuario en el entorno que ha estimado conveniente. Además, como se ha explicado previamente, la creación de objetos solo puede aún llevarse a cabo a través de la interfaz del propio V-REP o, de forma programática, a través de la API regular. Por ello, es preciso que cada escena de V-REP en la que se desee llevar a cabo una simulación cuente con un script en Lua que contenga las funciones necesarias para la creación de las diferentes partes del robot y a las que se llamará a través de *simxCallScriptFunction* desde la aplicación de Matlab.

2.2. Comunicación entre los módulos

La comunicación entre ambos módulos se establece, como se ha mencionado, gracias a la API remota con la que cuenta V-REP. A través de un

socket, se produce la interacción entre las dos aplicaciones, siendo V-REP el servidor y la aplicación desarrollada en Matlab, el cliente. La API remota se activa por parte de V-REP gracias al plugin *v_repExtRemoteApi.dll*, cargado por defecto. Para habilitar la API remota en la aplicación de Matlab, se hace uso de la función *vrep.simxStart*, tras haber cargado la librería *remoteApi.dll*. Para más información a este respecto, se recomienda consultar el manual de usuario de V-REP [1].

La comunicación establecida puede ser síncrona o asíncrona. En este caso se ha optado por el primer tipo, que se activa del lado del cliente con la función *simxSynchronous()* y permite que cada paso de la simulación se ejecute cuando reciba un *trigger* desde la API remota gracias a la función *simxSynchronousTrigger()*. En la figura 2 se ilustra este procedimiento.

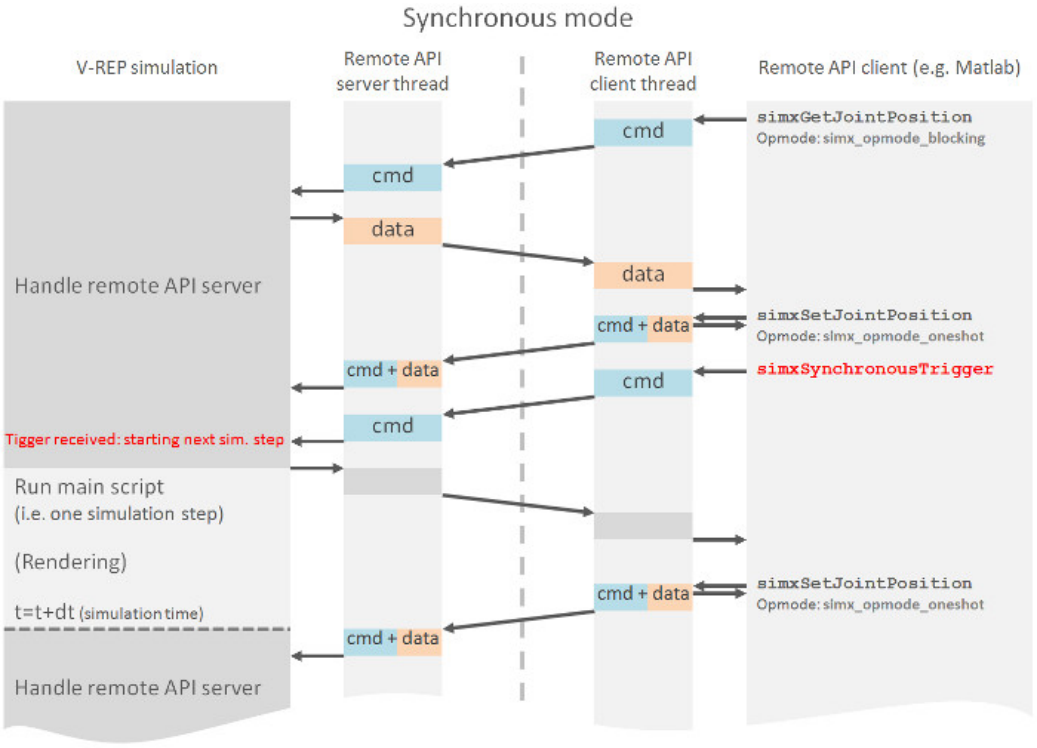

Figura 2: Modo síncrono

Además, la mayoría de las funciones de la API remota exigen como argumentos el identificador *clientID* proporcionado por *simxStart* y un modo de operación (*simx_opmode_oneshot*, *simx_opmode_blocking*, *simx_opmode_streaming*,

simx_opmode_buffer, etc.) que define qué ocurre con la petición y la respuesta asociadas a cada llamada. A este respecto, se ha optado en todos los casos por el modo de operación recomendado por el manual de usuario de V-REP [1] para cada función.

3. Descripción de la interfaz

De la misma forma que el marsupio del koala permite a las crías dar sus primeros pasos en un entorno que de otro modo podría resultar hostil, Koala ofrece al usuario la posibilidad de iniciarse en el diseño y la simulación del comportamiento de robots móviles sin verse abrumado por las posibilidades que a este respecto ofrecen V-REP y otras plataformas similares. La interfaz desarrollada en el entorno GUIDE de Matlab permite ejecutar todas las fases de diseño y simulación del comportamiento de robots móviles de forma sencilla. Su estructura se esquematiza en la figura 3.

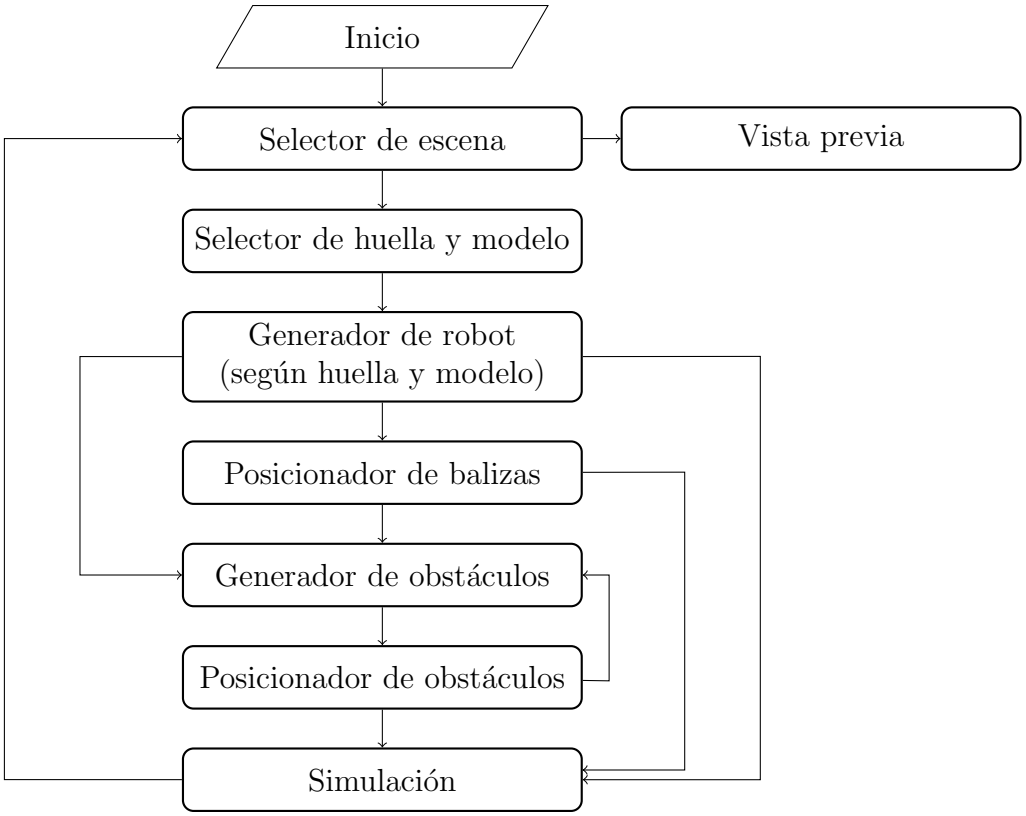

Figura 3: Estructura de la interfaz

Todas las ventanas de la interfaz se posicionan en la esquina superior derecha de la pantalla, de forma que interfieran los menos posible con la ventana de V-REP en la que el usuario observa los resultados del proceso de

creación del robot y la simulación, como se aprecia en la figura 4.

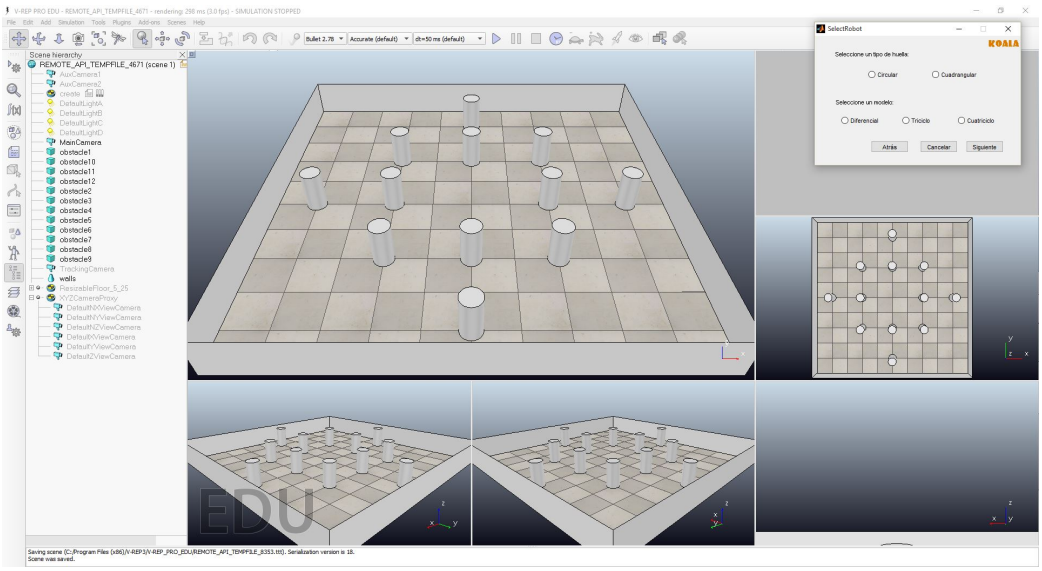

Figura 4: Ventana de la interfaz desarrollada sobre ventana de V-REP

El proceso de creación del robot y generación de posibles obstáculos o balizas puede ser cancelado por el usuario en cualquier momento bien utilizando los botones *Cancelar* dispuestos para tal fin o bien cerrando la ventana, en cuyo caso se solicitará confirmación.

A continuación, se describe cada una de las ventanas que componen la interfaz. Una primera ventana, que se muestra en la figura 5, da la bienvenida al usuario y le permite entablar la conexión con V-REP, que ha de haber sido iniciado previamente.

Seguidamente, en la ventana que se aprecia en la figura 6, se ofrece al usuario la posibilidad de elegir la escena en la que desea llevar a cabo la simulación, pudiendo escoger entre cuatro entornos predefinidos o cargar una escena propia creada previamente con V-REP. En este úlimo caso, recomendado para los usuarios más avanzados, ha de verificarse que la escena contenga un *dummy* con el script en Lua *create* asociado, como ya incluyen las escenas predefinidas, a fin de poder llevar a cabo la creación del robot y posibles obstáculos y/o balizas adicionales de forma satisfactoria.

Figura 5: Ventana de inicio

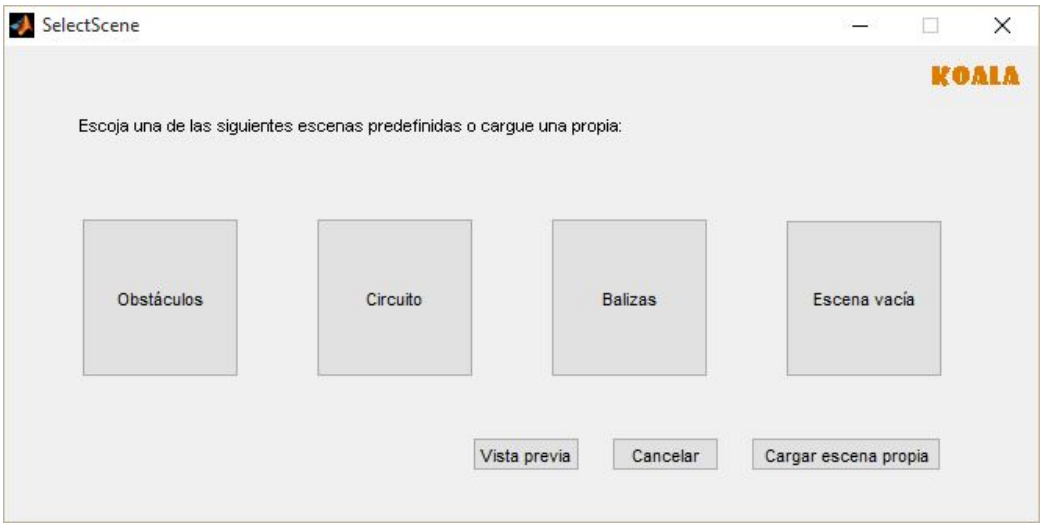

Figura 6: Selector de escenas

Asimismo, a fin de facilitar la elección, se ofrece al usuario una vista previa y descripción del comportamiento que se espera del robot en cada una de las escenas en la ventanta que se muestra en la figura 7

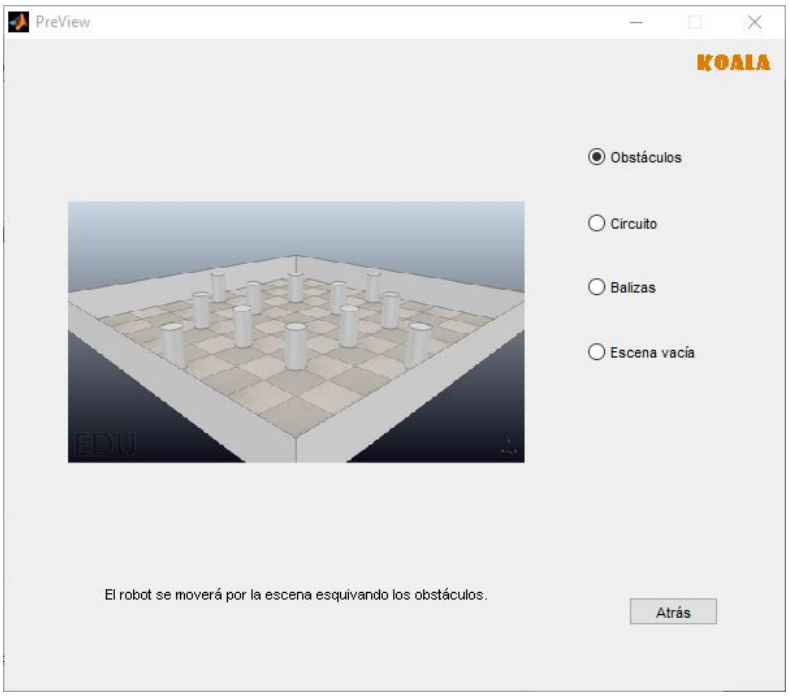

Figura 7: Vista previa de cada una de las escenas

A continuación, el usuario ha de seleccionar en la ventana que se aprecia en la figura 8 el tipo de huella (circular o cuadrangular) y el modelo de robot móvil (diferencial, triciclo o cuatriciclo) que desea crear y simular.

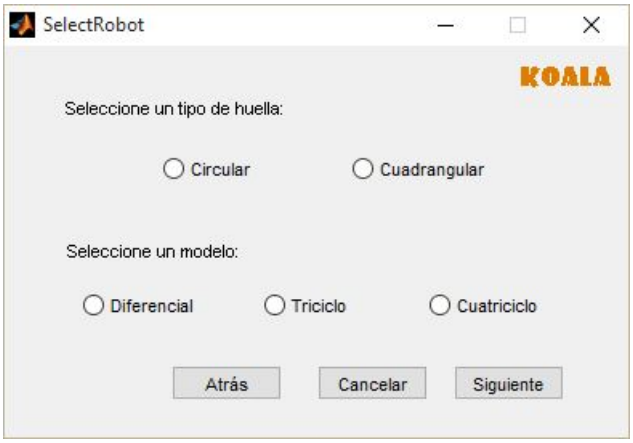

Figura 8: Selector de huella y modelo de robot

En función de la elección del usuario, el proceso de creación del robot continuará con una de las seis ventanas asociadas a cada combinación de huella y modelo. Para cualquier caso, se comprobará que los valores introducidos por el usuario sean numéricos y positivos. De forma adicional, se asegurará que las dimensiones especificadas cumplan con las restricciones geométricas de cada modelo y tipo de huella. En caso de que esto no sea así, un mensaje de error avisará al usuario, indicándole qué restricción no se cumple y le sugerirá cambiar los valores de las dimensiones implicadas. Asimismo, todas las ventanas ofrecen la posibilidad, en caso de equivocación por parte del usuario, de regresar a la ventana de selección de huella y modelo y escoger unas nuevas características.

Robot diferencial con huella circular Se solicita al usuario, tal y como se muestra en la figura 9, el radio y la altura del cuerpo del robot, la distancia entre ejes y el radio y el ancho de las ruedas.

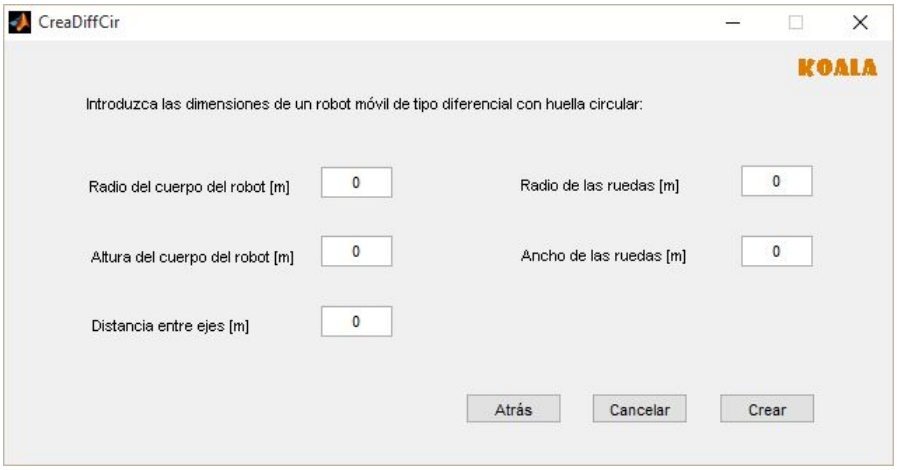

Figura 9: Generador de robot diferencial con huella circular

Robot diferencial con huella cuadrangular Se solicita al usuario, tal y como se aprecia en la ventana de la figura 10, el largo, el ancho y la altura del cuerpo del robot, la distancia entre ejes y el radio y el ancho de las ruedas.

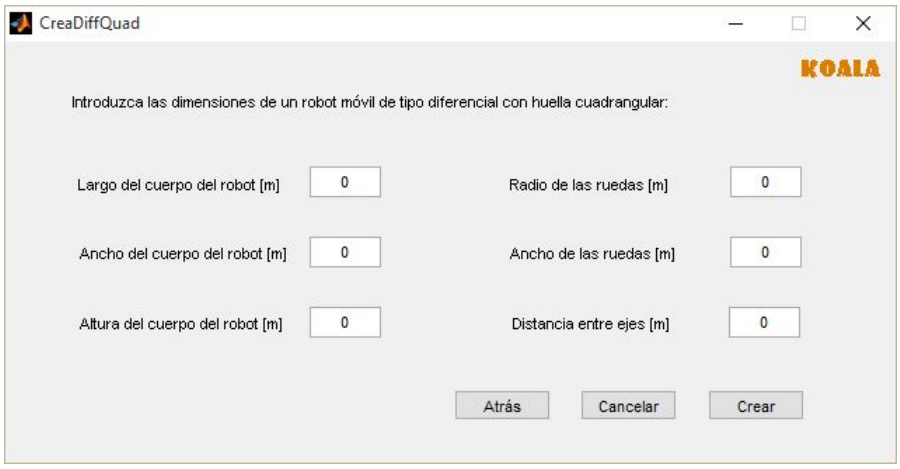

Figura 10: Generador de robot diferencial con huella cuadrangular

Robot de tipo triciclo con huella circular Se solicita al usuario, tal y como se aprecia en la ventana de la figura 11, el radio y la altura del cuerpo del robot, las distancias entre los ejes izquierdo-derecho y delantero-trasero y el radio y el ancho de las ruedas.

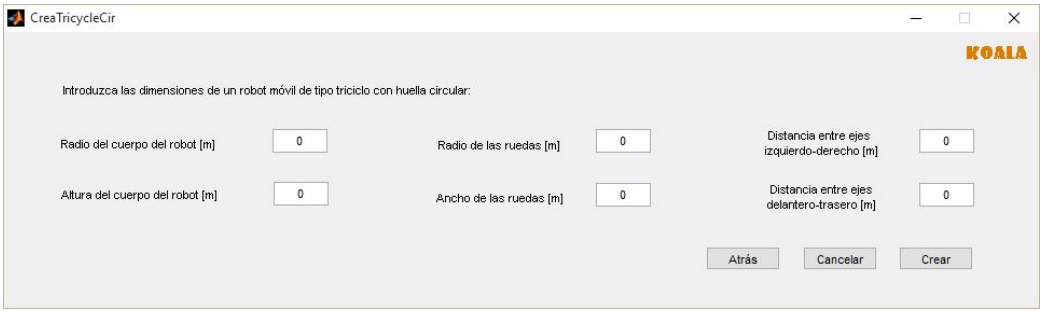

Figura 11: Generador de robot de tipo triciclo con huella circular

Robot de tipo triciclo con huella cuadrangular Se solicita al usuario, tal y como se muestra en la ventana de la figura 12, el largo, el ancho y la altura del cuerpo del robot, las distancias entre los ejes izquierdo-derecho y delantero-trasero y el radio y el ancho de las ruedas.

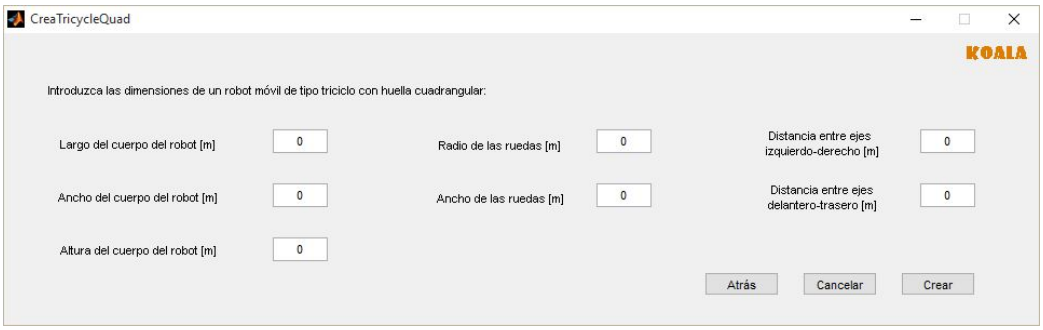

Figura 12: Generador de robot de tipo triciclo con huella cuadrangular

Robot de tipo cuatriciclo con huella circular Se solicita al usuario, tal y como se aprecia en la ventana de la figura 13, el radio y la altura del cuerpo del robot, las distancias entre los ejes izquierdo-derecho y delantero-trasero y el radio y el ancho de las ruedas.

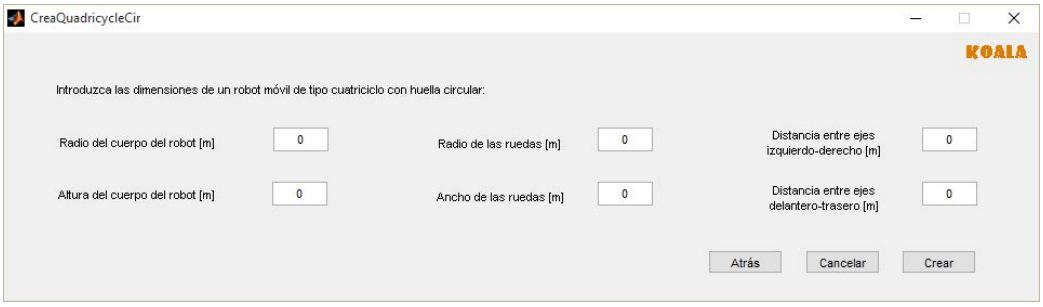

Figura 13: Generador de robot de tipo cuatriciclo con huella circular

Robot de tipo cuatriciclo con huella cuadrangular Se solicita al usuario, tal y como se aprecia en la ventana de la figura 14, el radio y la altura del cuerpo del robot, las distancias entre los ejes izquierdo-derecho y delantero-trasero y el radio y el ancho de las ruedas.

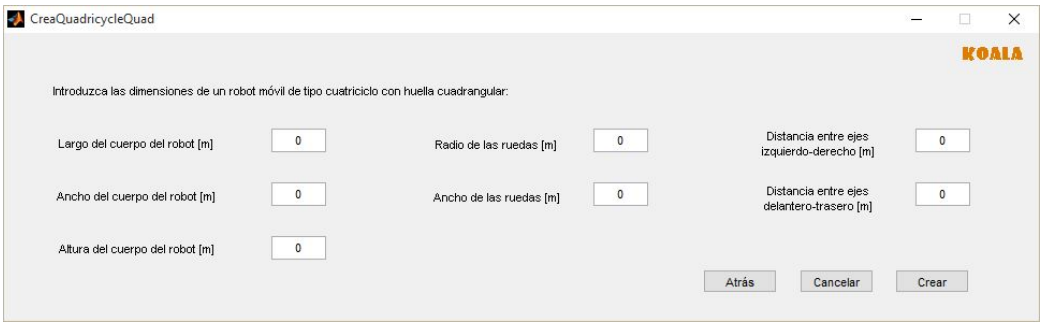

Figura 14: Generador de robot de tipo cuatriciclo con huella cuadrangular

A continuación, se ofrece al usuario la posibilidad de añadir balizas u obstáculos a la escena, como se muestra en la figura 15, (en el caso de haber elegido probar el robot en las escenas *Balizas*, *Escena Vacía* o haber cargado una escena creada previamente) o solo obstáculos (para cualquiera de los casos restantes).

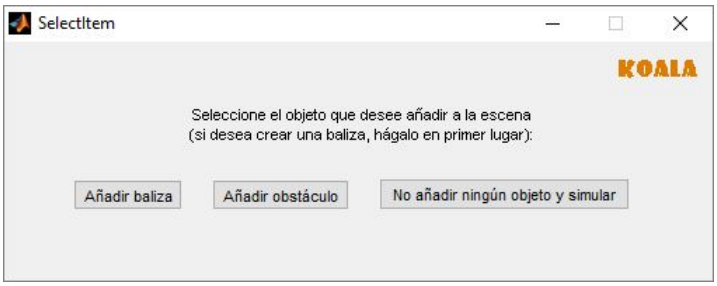

Figura 15: Selector de ítem

Si se elige añadir una baliza, su forma cilindríca y sus dimensiones (radio de la base de 0,125 m y 0,5 m de altura) y, especialmente, su color verde (R=0, G=1, B=0), fundamental para su detección por parte del robot, se consideran fijos.

Por defecto, cada nueva baliza se crea a ras de suelo justo a 1 m enfrente del robot. No obstante, como se muestra en la figura 16, el usuario puede modificar a su conveniencia la posición de la baliza, bien introduciendo las coordenadas cartesianas deseadas o bien, si lo encuentra más intuitivo, arrastrando directamente la baliza hasta la posición que estima conveniente en la ventana de V-REP. En este último caso, haciendo clic en el botón *Actualizar coordenadas* se muestran en la ventana las nuevas coordenadas cartesianas

de la baliza. El usuario puede añadir tantas balizas como desee haciendo clic en *Crear una nueva baliza*.

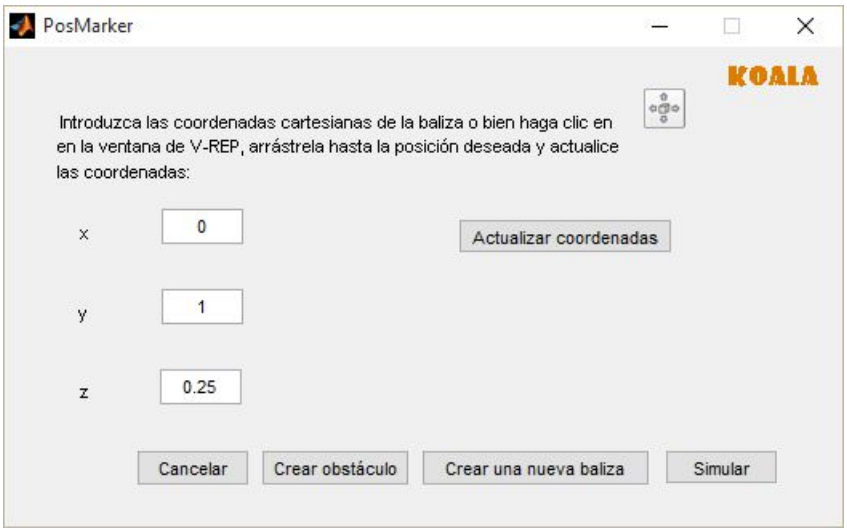

Figura 16: Posicionador de balizas

Si se elige añadir un obstáculo, se pregunta en primer lugar al usuario, como se aprecia en la ventana de la figura 17, por su forma (prismática, cilíndrica o cónica) y sus dimensiones (lado o radio de la base, según el caso y altura).

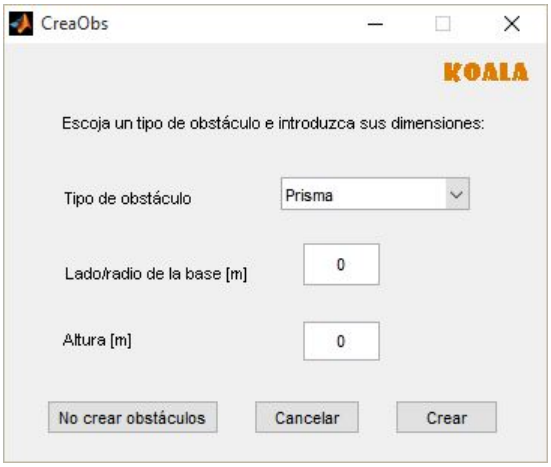

Figura 17: Generador de obstáculos

Al igual que en el caso de las balizas, cada nuevo obstáculo se crea por defecto a ras de suelo justo a 1 m enfrente del robot. No obstante, el usuario puede, nuevamente, introducir las coordenadas cartesianas que estime convenientes para el obstáculo o arrastrarlo en la ventana de V-REP hasta la posición deseada (como se muestra en la figura 18). De nuevo, haciendo clic en en el botón *Actualizar coordenadas* se muestran en la ventana las actuales coordenadas cartesianas del obstáculo. El usuario puede repetir la secuencia de generación y posicionamiento de obstáculos tantas veces como desee.

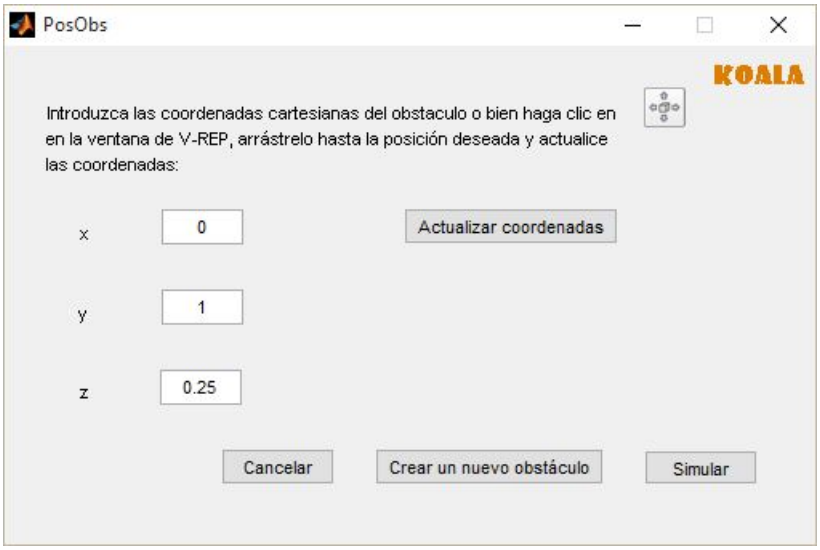

Figura 18: Posicionador de obstáculos

Tras haber añadido las balizas y/o los obstáculos que el usuario haya estimado conveniente, una útlima ventana, que se muestra en la figura 19 permite iniciar, pausar, parar y retomar la simulación, así como seleccionar y cambiar tantas veces como se desee la velocidad lineal del robot dentro de un intervalo de 0 a 1 m/s que se ha estimado razonable para las dimensiones de los recintos en los que tiene lugar el testeo. Al parar la simulación, se ofrece la posibilidad de probar un nuevo robot en una nueva escena, para lo cual el usuario es remitido a la ventana *Selector de escenas*, o finalizar Koala de forma definitiva.

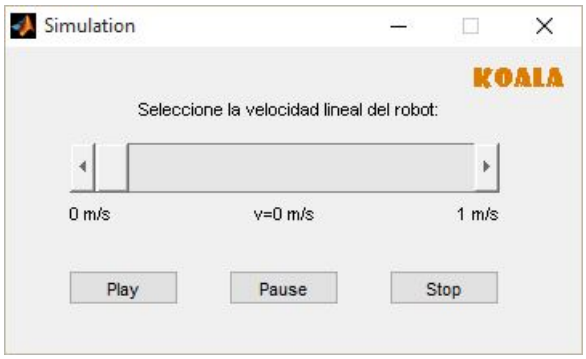

Figura 19: Ventana de simulación

4. Descripción de los robots móviles

En este capítulo se describen los modelos de robots móviles que permite crear Koala.

4.1. Diferencial

Un robot de tipo diferencial cuenta, como se aprecia en la figura 20, con dos ruedas motrices situadas a ambos lados de su cuerpo, así como dos esferas deslizantes ubicadas en las partes frontal y trasera que sirven de punto de apoyo y cuyas dimensiones no influyen, idealmente, en la cinemática del robot.

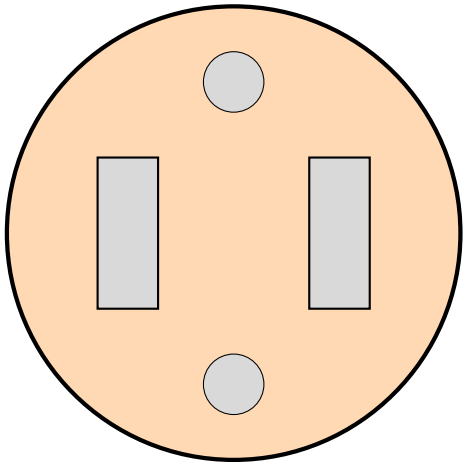

Figura 20: Robot diferencial

El avance, retroceso o giro del robot se produce asignando una determinada velocidad angular a cada uno de los motores de las ruedas. La velocidad lineal del robot viene dada por (1).

$$v = \frac{v_i + v_d}{2} \tag{1}$$

donde v_i y v_d son, respectivamente, las velocidades lineales de las ruedas izquierda y derecha calculadas según (2) y (3) a partir de las velocidades angulares de las ruedas ω_i y ω_d y el radio de las mismas r.

$$v_i = \omega_i * r \tag{2}$$

$$v_d = \omega_d * r \tag{3}$$

La velocidad angular del robot viene dada por (4)

$$\omega = \frac{v_i - v_d}{d} \tag{4}$$

donde d es la distancia entre los ejes.

El estado del robot estará caracterizado por su posición x e y y su orientación α obtenidas a partir de (6), (7) y (5), respectivamente.

$$\dot{\alpha}(t) = \omega(t) \tag{5}$$

$$\dot{x}(t) = v(t) * cos\,[\alpha(t)] \tag{6}$$

$$\dot{y}(t) = v(t) * sen\,[\alpha(t)] \tag{7}$$

En la figura 21, se muestra un robot diferencial con huella circular creado en la escena de simulación de V-REP.

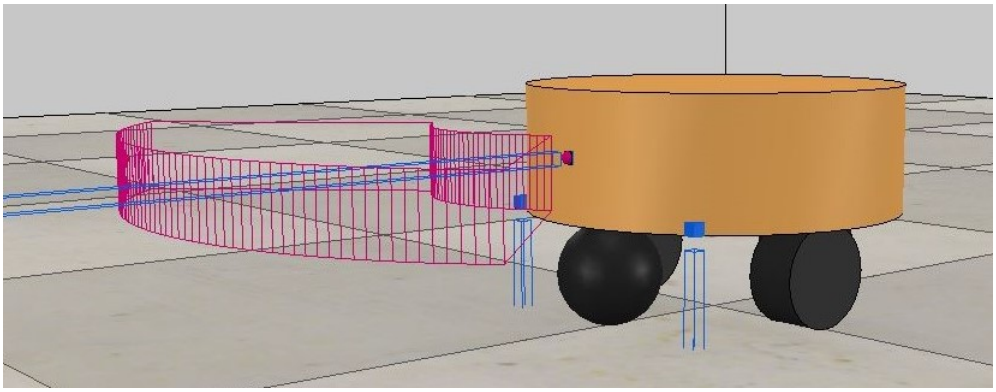

Figura 21: Robot diferencial en la escena de V-REP

4.1.1. Restricciones geométricas para huella circular

Un robot diferencial con huella circular presenta el diseño que se muestra en la figura 22.

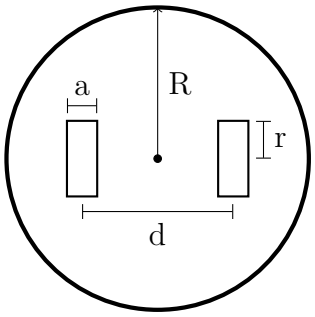

Figura 22: Robot diferencial con huella circular

Las ruedas no han de sobresalir del cuerpo del robot ni tampoco superponerse. Esto se verifica si se cumplen, respectivamente, (8) y (9).

$$\sqrt{\left(\frac{d+a}{2}\right)^2 + r^2} \leq R \tag{8}$$

$$a < d \tag{9}$$

Además, se tiene que en ningún caso la distancia entre ejes puede ser mayor que el diámetro del robot.

4.1.2. Restricciones geométricas para huella cuadrangular

Un robot diferencial con huella cuadrangular presenta el diseño que se muestra en la figura 23

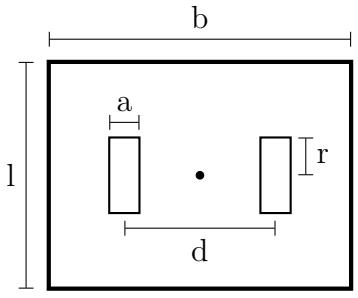

Figura 23: Robot diferencial con huella cuadrangular

Las ruedas no han de sobresalir del cuerpo del robot ni por la partes frontal y trasera ni por los laterales, lo cual se verifica si se cumplen, respectivamente, (10) y (11). Además, se ha de garantizar que las ruedas no se superpongan, para lo cual sigue siendo válida (9).

$$r \leq \frac{l}{2} \tag{10}$$

$$d + a \leq b \tag{11}$$

4.2. Triciclo

El modelo del triciclo surge como una evolución en busca de una mayor estabilidad del modelo de la bicicleta. Un robot de este último tipo, descartado en el desarrollo de Koala por considerarse carente de interés una vez implementado el modelo del triciclo, posee una rueda motriz y una directriz, como se aprecia en la figura 24.

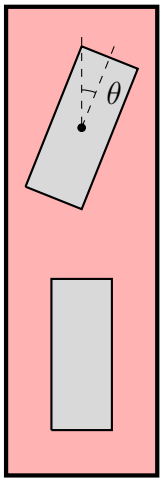

Figura 24: Robot de tipo bicicleta

El modelo del triciclo, que se muestra en la figura 25, añade una segunda rueda motriz, para obtener, de esta manera, los tres puntos de apoyo que garantizan la estabilidad del robot.

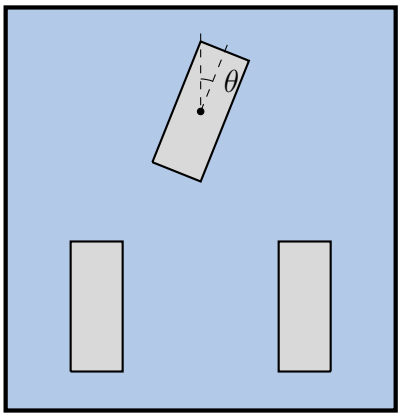

Figura 25: Robot de tipo triciclo

El avance, retroceso o giro de un robot de tipo triciclo se consigue asignando una misma velocidad angular a los motores de ambas ruedas motrices y un ángulo a la rueda directriz. La velocidad lineal del robot coincide con la de las ruedas traseras y su velocidad angular viene dada por (12)

$$\omega = \frac{v * tan(\theta)}{d_{dt}} \tag{12}$$

donde d_{dt} es la distancia entre los ejes delantero y trasero.

El estado del robot estará nuevamente caracterizado por su posición x e y y su orientación α obtenidas, como en el caso de un robot diferencial, a partir de (6), (7) y (5), respectivamente.

En la figura 26, se muestra un robot de tipo triciclo con huella circular creado en la escena de simulación de V-REP.

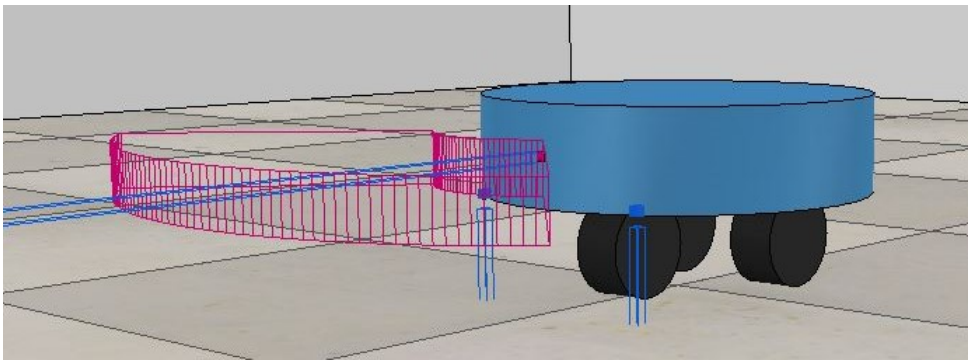

Figura 26: Robot de tipo triciclo en la escena de V-REP

4.2.1. Restricciones geométricas para huella circular

Un robot de tipo triciclo con huella circular presenta el diseño que se muestra en la figura 27.

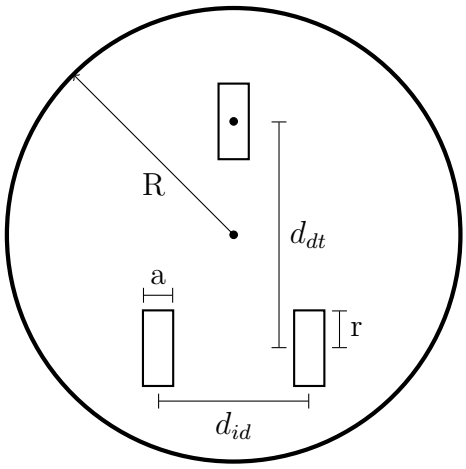

Figura 27: Robot de tipo triciclo con huella circular

Las ruedas no han de sobresalir del cuerpo del robot. Para tal fin, en el caso de las ruedas traseras, ha de cumplirse (13).

$$\sqrt{\left(\frac{d_{id}+a}{2}\right)^2+\left(\frac{d_{dt}}{2}+r\right)^2} \leq R \tag{13}$$

En el caso de la rueda delantera, ha de tenerse en cuenta que esta gira de -$\pi/6$ a $\pi/6$ y no ha de sobresalir del cuerpo del robot en ningún momento. Para ello, se calcula en primer lugar el ángulo $\theta_{dmax} = arctg\left(\frac{a}{2r}\right)$ para el que la distancia de P_1 al centro del robot es máxima. Este ángulo es tal que la diagonal del rectángulo que forma la vista cenital de la rueda es coincidente con el radio del robot, como se aprecia en la figura 28.

Sin embargo, puede que la rueda nunca alcance dicho ángulo durante su giro. Por este motivo, se halla el mínimo entre $\pi/6$ y θ_{dmax}. Teniendo en cuenta que d es estrictamente creciente desde 0 hasta θ_{dmax}, de esta forma se obtiene el ángulo para el que d es máxima dentro del arco de giro de la rueda.

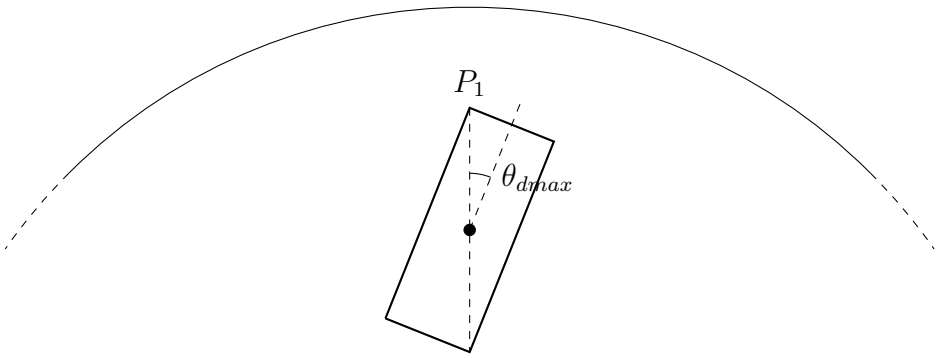

Figura 28: Rueda delantera con ángulo θ_{dmax}

A continuación, se calculan según (14) y (15) las coordenadas cartesianas de P_1 con respecto al centro del cuerpo del robot y se verifica finalmente que la rueda no sobresalga en este el caso más desfavorable dentro de su arco de giro (16).

$$x = -cos\left[arctg\left(\frac{2r}{a}\right) + min\left(\frac{\pi}{6}, \theta_{dmax}\right)\right] * \sqrt{\left(\frac{a}{2}\right)^2 + r^2} \tag{14}$$

$$y = cos\left[arctg\left(\frac{a}{2r}\right) - min\left(\frac{\pi}{6}, \theta_{dmax}\right)\right] * \sqrt{\left(\frac{a}{2}\right)^2 + r^2} + \frac{d_{dt}}{2} \tag{15}$$

$$\sqrt{x^2 + y^2} \leq R \tag{16}$$

En cualquier caso, las distancias entre ejes no han de ser mayores que el diámetro del robot.

Asimismo, las ruedas tampoco han de superponerse. Para ello, se comprueba en primer lugar que las ruedas traseras no se superpongan entre sí, para lo cual ha de cumplirse (17).

$$a < d_{id} \tag{17}$$

Además, se verifica que la rueda delantera no se superponga en su giro con las traseras. Para tal fin, se comprueba que o bien la rueda directriz se encuentre a la suficiente distancia de las ruedas motrices para que sea cual sea su giro nunca lleguen a superponerse, para lo cual ha de cumplirse (18)

o (19), o bien la distancia sea menor pero suficiente para que hasta el ángulo máximo de giro de $\pi/6$ no se produzca superposición entre las ruedas (20).

$$\sqrt{\left(\frac{a}{2}\right)^2 + r^2} < d_{dt} - r \tag{18}$$

$$\sqrt{\left(\frac{a}{2}\right)^2 + r^2} < \frac{d_{id} - a}{2} \tag{19}$$

$$cos\left[arctg\left(\frac{2r}{a}\right) - \frac{\pi}{6}\right] * \sqrt{\left(\frac{a}{2}\right)^2 + r^2} < \frac{d_{id} - a}{2} \tag{20}$$

4.2.2. Restricciones geométricas para huella cuadrangular

Un robot de tipo triciclo con huella cuadrangular presenta el diseño que se muestra en la figura 29.

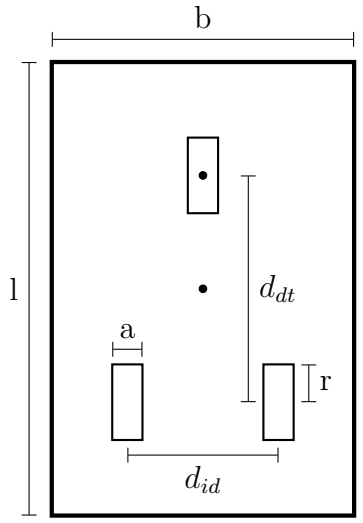

Figura 29: Robot de tipo triciclo con huella cuadrangular

Las ruedas no han de sobresalir del cuerpo del robot. Para ello, en el caso de las ruedas traseras, han de cumplirse tanto (21) como (22) .

$$d_{id} + a \leq b \tag{21}$$

$$\frac{d_{dt}}{2} + r \leq \frac{l}{2} \tag{22}$$

En el caso de la rueda delantera, se comprueba que esta no sobresalga por los laterales del cuerpo del robot (24). Para ello, se calcula en primer lugar según (23) la coordenada x de P_2 con respecto al centro del robot para el mínimo entre $\pi/6$ y el ángulo para el que la coordenada x es máxima $\theta_{xmax} = arctg\left(\frac{2r}{a}\right)$. Este ángulo es tal que la diagonal del rectángulo que forma la vista cenital de la rueda es paralela al ancho del robot, como se aprecia en la figura 30.

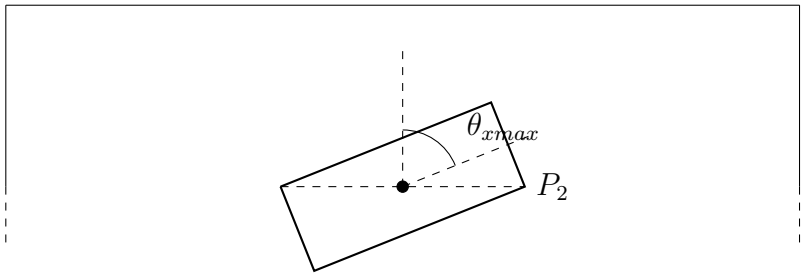

Figura 30: Rueda delantera con ángulo θ_{xmax}

Además, sigue siendo válido lo expuesto en las restricciones para un robot de tipo triciclo de huella circular, según lo cual puede que la rueda no alcance θ_{xmax} en su arco de giro.

$$x = cos\left[arctg\left(\frac{2r}{a}\right) - min\left(\frac{\pi}{6}, \theta_{xmax}\right)\right] * \sqrt{\left(\frac{a}{2}\right)^2 + r^2} \tag{23}$$

$$x \leq \frac{b}{2} \tag{24}$$

A fin de verificar que la rueda delantera no sobresalga por la parte frontal (26), se halla según (25) la coordenada y de P_1 con respecto al centro del robot para el mínimo entre $\pi/6$ y el ángulo para el que la coordenada y es máxima $\theta_{ymax} = arctg\left(\frac{a}{2r}\right)$. Este ángulo es tal que la diagonal del rectángulo que forma la vista cenital de la rueda es paralela al largo del robot, como se aprecia en la figura 31.

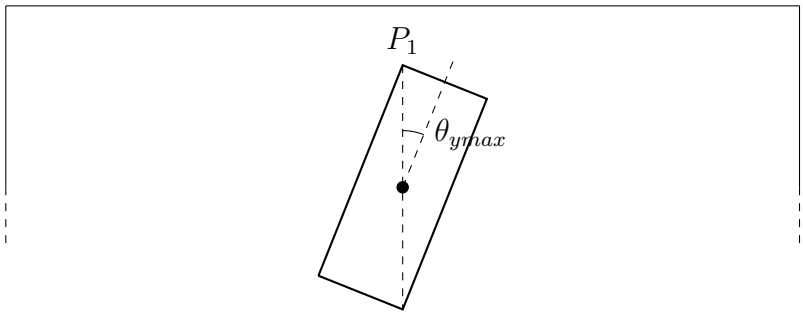

Figura 31: Rueda delantera con ángulo θ_{ymax}

$$y = cos\left[arctg\left(\frac{a}{2r}\right) - min\left(\frac{\pi}{6}, \theta_{ymax}\right)\right] * \sqrt{\left(\frac{a}{2}\right)^2 + r^2} + \frac{d_{dt}}{2} \tag{25}$$

$$y \leq \frac{l}{2} \tag{26}$$

Asimismo, se verifica que las ruedas no se superpongan entre sí, para lo cual sigue siendo válido lo expuesto para el caso de un robot tipo triciclo con huella circular.

4.3. Cuatriciclo

Un robot móvil de tipo cuatriciclo cuenta con dos ruedas motrices y dos ruedas directrices dispuestas como se muestra en la figura 32.

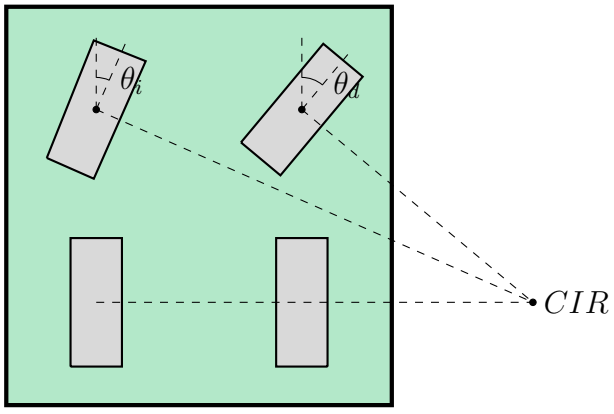

Figura 32: Robot de tipo cuatriciclo

Sus velocidades lineal y angular se obtienen de igual forma que para un robot de tipo triciclo, con la particularidad de que, en este caso, θ no se corresponde al ángulo de ninguna de las dos ruedas directrices. A fin de evitar deslizamiento, el robot ha de poseer un único centro instantáneo de rotación. Para ello, es preciso que la rueda interior (en el caso que se muestra en la figura 32, la derecha) adquiera durante el giro un ángulo ligeramente mayor que la exterior (en este caso, la izquierda). Estos ángulos θ_i y θ_d se calculan, respectivamente, con (27) y (28).

$$\theta_i = arccotg\left[cotg(\theta) - \frac{d_{di}}{2d_{dt}}\right] \tag{27}$$

$$\theta_d = arccotg\left[cotg(\theta) + \frac{d_{di}}{2d_{dt}}\right] \tag{28}$$

donde d_{di} es la distancia entre los ejes derecho e izquierdo y d_{dt}, entre los ejes delantero y trasero.

El estado del robot vuelve a estar caracterizado por su posición x e y y su orientación α obtenidas, como en los dos casos anteriores, a partir de (6), (7) y (5), respectivamente.

En la figura 33, se muestra un robot de tipo cuatriciclo con huella cuadrangular creado en la escena de simulación de V-REP.

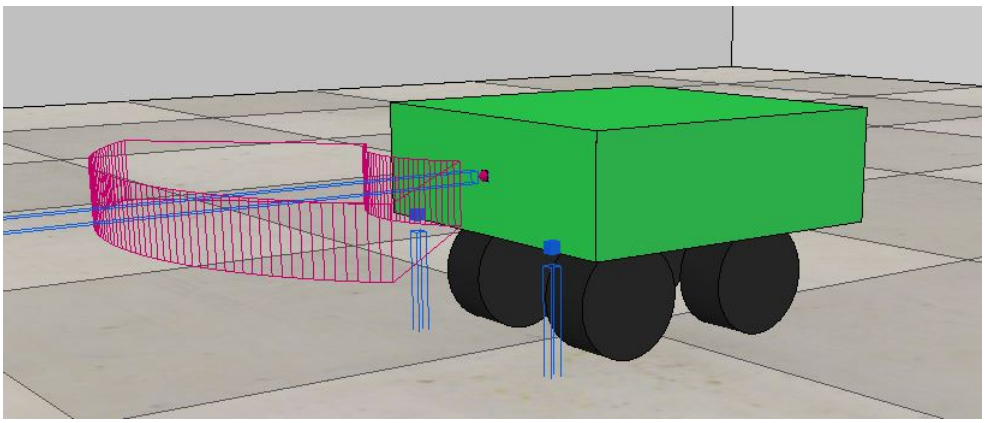

Figura 33: Robot de tipo cuatriciclo en la escena de V-REP

4.3.1. Restricciones geométricas para huella circular

Un robot de tipo cuatriciclo con huella circular presenta el diseño que se muestra en la figura 34.

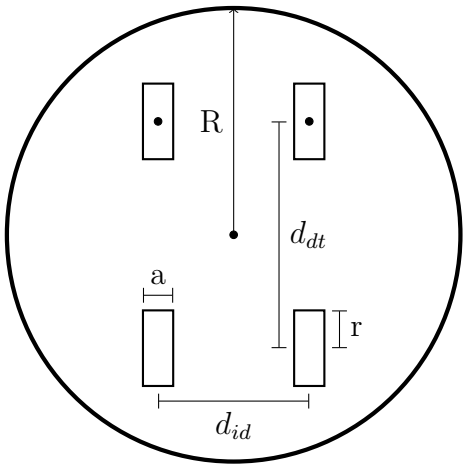

Figura 34: Robot de tipo cuatriciclo con huella circular

Nuevamente, las ruedas no han de sobresalir del cuerpo del robot. En lo que respecta a las ruedas traseras, sigue siendo válido lo expuesto para el caso de un robot de tipo triciclo y huella circular.

En el caso de las ruedas delanteras, se estudia la rueda directriz derecha, siendo los resultados obtenidos válidos también para la rueda directiz izquierda dada la simetría del robot. Ha de tenerse en cuenta que esta gira de $\theta_{d1} = arccotg\left[cotg(\frac{\pi}{6}) + \frac{d_{di}}{2d_{dt}}\right]$ a $\theta_{d2} = arccotg\left[cotg(-\frac{\pi}{6}) + \frac{d_{di}}{2d_{dt}}\right]$ y no ha de sobresalir del cuerpo del robot en ningún momento. Para ello, se calcula en primer lugar el ángulo $\theta_{ddmax} = arctg\left(\frac{d_{di}}{d_{dt}}\right) - arctg\left(\frac{a}{2r}\right)$ para el que la distancia d de P_2 al centro del robot es máxima. Este ángulo es tal que la diagonal del rectángulo que forma la vista cenital de la rueda es coincidente con el radio del robot, como se aprecia en la figura 35.

Sin embargo, nuevamente puede que la rueda nunca alcance dicho ángulo durante su giro. Por este motivo, se halla el mínimo entre θ_{d2} y θ_{ddmax}. Teniendo en cuenta que d es estrictamente creciente desde 0 hasta θ_{ddmax}, de esta forma se obtiene el ángulo para el que d es máxima dentro del arco de giro de la rueda.

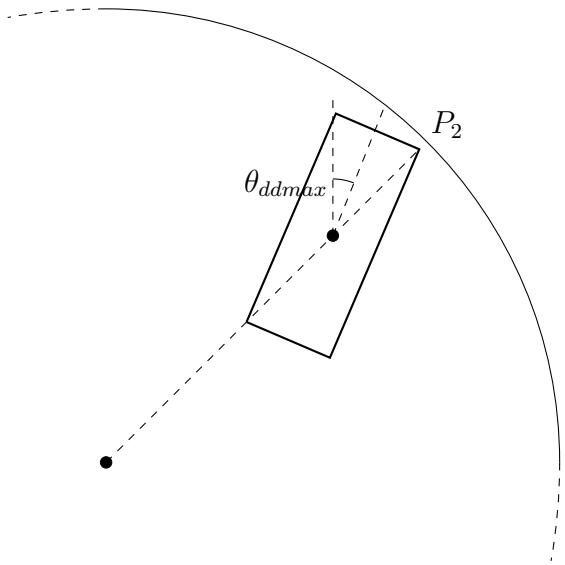

Figura 35: Rueda delantera derecha con ángulo θ_{ddmax}

A continuación, se calculan según (29) y (30) las coordenadas cartesianas de P_2 con respecto al centro del cuerpo del robot y se verifica finalmente que la rueda no sobresalga en este el caso más desfavorable dentro de su arco de giro (31).

$$x = cos\left[arctg\left(\frac{2r}{a}\right) - min\left(-\theta_{d2}, \theta_{ddmax}\right)\right] * \sqrt{\left(\frac{a}{2}\right)^2 + r^2} + \frac{d_{id}}{2} \tag{29}$$

$$y = cos\left[arctg\left(\frac{a}{2r}\right) + min\left(-\theta_{d2}, \theta_{ddmax}\right)\right] * \sqrt{\left(\frac{a}{2}\right)^2 + r^2} + \frac{d_{dt}}{2} \tag{30}$$

$$\sqrt{x^2 + y^2} \leq R \tag{31}$$

Asimismo, las ruedas tampoco han de superponerse. Para ello, se comprueba en primer lugar que las ruedas traseras no se superpongan entre sí, para lo cual sigue siendo válida (17). Además, se verifica que las ruedas delanteras no se superpongan en su giro con las traseras. De nuevo, se estudia la mitad derecha del robot. Como expuesto anteriormente, la rueda delantera derecha gira desde $\theta_{d1} = arccotg\left[cotg(\frac{\pi}{6}) + \frac{d_{di}}{2d_{dt}}\right]$ a $\theta_{d2} =$

$arccotg\left[cotg(-\frac{\pi}{6}) + \frac{d_{di}}{2d_{dt}}\right]$ y no ha de superponerse con la rueda trasera derecha en ningún momento. Para ello, se calcula en primer lugar el ángulo $\theta_{dymin} = arctg\left(\frac{a}{2r}\right)$ para el que la coordenada y de P_3 es mínima, que coincide con el ángulo para el que la coordenada y de P_1 es máxima. Este ángulo es tal que la diagonal del rectángulo que forma la vista cenital de la rueda es paralela a la distancia entre los ejes delantero y trasero, como se muestra en la figura 36

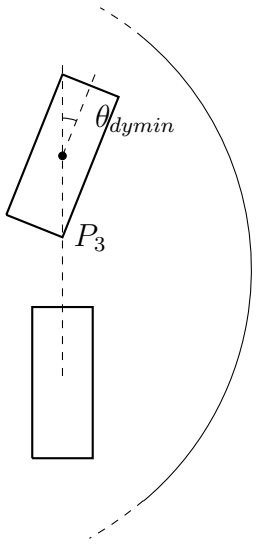

Figura 36: Rueda delantera derecha con ángulo θ_{dymin}

De nuevo, puede que la rueda no alcance dicho ángulo durante su giro, por lo que se halla el mínimo entre θ_{d2} y θ_{dymin} y se calcula la coordenada y de P_3 con respecto al centro del robot según (32).

$$y = -cos\left[arctg\left(\frac{a}{2r}\right) - min\left(-\theta_{d2}, \theta_{dymin}\right)\right] * \sqrt{\left(\frac{a}{2}\right)^2 + r^2} + \frac{d_{dt}}{2} \qquad (32)$$

Finalmente se comprueba que las ruedas no se superpongan con (33).

$$y > \frac{-d_{dt}}{2} + r \qquad (33)$$

Para verificar que las ruedas traseras no se superpongan entre sí sigue siendo válida (17).

4.3.2. Restricciones geométricas para huella cuadrangular

Un robot de tipo cuatriciclo con huella cuadrangular presenta el diseño que se muestra en la figura 37.

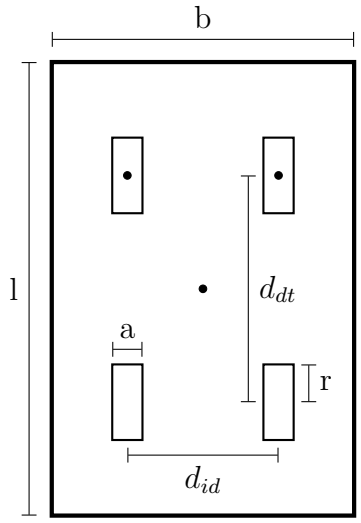

Figura 37: Robot de tipo cuatriciclo con huella cuadrangular

Como para todos los casos anteriores, las ruedas no han de sobresalir del cuerpo del robot. En lo que respecta a las ruedas traseras, sigue siendo válido lo expuesto para un robot de tipo triciclo y huella cuadrangular.

En el caso de las ruedas delanteras, se sigue un razonamiento similar al planteado para el caso de un robot de tipo triciclo y huella cuadrangular particularizado para la rueda directiz derecha, que se muestra en las figuras 38 y 39, siendo los resultados obtenidos nuevamente válidos para la rueda izquierda, dada la simetría del robot.

A fin de verificar que la rueda no sobresalga por el lateral del robot con (24), se halla la coordenada x máxima que puede alcanzar P_2 durante su giro desde θ_{d1} hasta θ_{d2} según (34).

$$x = cos\left[arctg\left(\frac{2r}{a}\right) - min\left(-\theta_{d2}, \theta_{dxmax}\right)\right] * \sqrt{\left(\frac{a}{2}\right)^2 + r^2} + \frac{d_{id}}{2} \qquad (34)$$

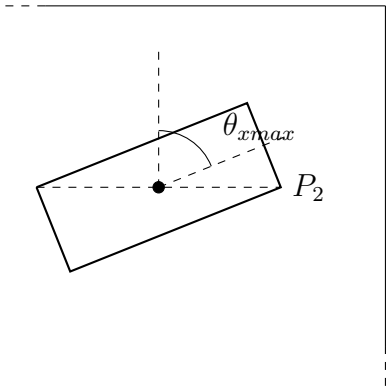

Figura 38: Rueda delantera derecha con ángulo θ_{dxmax}

Para comprobar que la rueda delantera no sobresalga por la parte frontal (26), se halla según (35) la coordenada y máxima de P_1 durante su giro.

$$y = cos\left[arctg\left(\frac{a}{2r}\right) - min\left(-\theta_{d2}, \theta_{dymax}\right)\right] * \sqrt{\left(\frac{a}{2}\right)^2 + r^2} + \frac{d_{dt}}{2} \qquad (35)$$

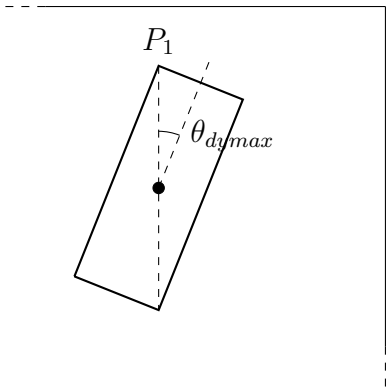

Figura 39: Rueda delantera derecha con ángulo θ_{dymax}

De igual modo, las ruedas tampoco han de superponerse entre sí, para lo cual sigue siendo válido todo lo expuesto para el caso de un robot de tipo cuatriciclo y huella circular.

4.4. Sensores

Sea cual sea el modelo escogido, el robot móvil creado cuenta con los siguientes sensores, vitales para su correcto comportamiento en los experimentos que se detallarán en el siguiente capítulo.

Sensor de proximidad El robot cuenta con un sensor de ultrasonidos, como el que se muestra en la figura 40. Está situado en el centro de la parte frontal del robot y se encarga de detectar cualquier obstáculo situado enfrente del mismo.

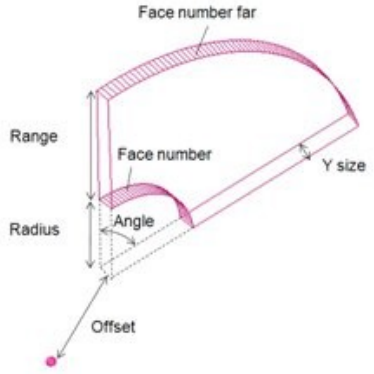

Figura 40: Sensor de proximidad [1]

Es de tipo disco, con un ángulo fijo de $\pi/2$, un radio de 0,1 m y un rango que varía de forma que el sensor cubra todo el diámetro (36) o el ancho (37), según el caso, del cuerpo del robot.

$$rango = \frac{R}{cos\left(\frac{\pi}{4}\right)} \tag{36}$$

$$rango = \frac{b}{2 * cos\left(\frac{\pi}{4}\right)} \tag{37}$$

Se aprovecha el radio de 0,1 m para que, sumado al rango obtenido, actúe de margen de seguridad de que el sensor cubra todo el cuerpo del robot.

Sensores de visión El robot dispone de tres sensores de visión como los que pueden apreciarse en la figura 41.

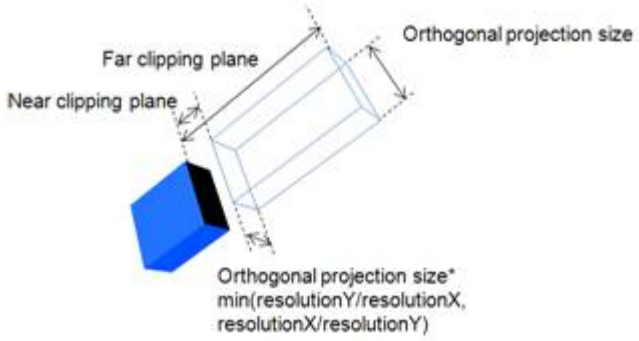

Figura 41: Sensor de visión [1]

Dos de ellos se encargan de detectar el circuito que ha de seguir el robot en uno de los posibles experimentos. Estos están situados en el borde inferior de la parte frontal del robot, a 0'1 m de su eje de simetría y apuntando hacia el suelo. Poseen una proyección ortogonal de 0,01 m x 0,01 m, una resolución de un pixel y un *far clipping plane* de 3 veces el radio de las ruedas que garantiza que el sensor siempre detecte el suelo.

Asimismo, el robot cuenta con un tercer sensor de visión situado en el mismo punto que el sensor de proximidad encargado de detectar las balizas hacia las que ha de dirigirse en otro de los posibles experimentos. Posee una proyección ortogonal de 0,01 m x 0,01 m, una resolución de un pixel y un *far clipping plane* de $\sqrt{5^2 + 5^2}$ m que garantiza que el sensor detecte cualquier baliza situada en la escena de dimensiones 5 m x 5 m.

4.5. Propiedades dinámicas

Masa La masa de cada uno de los robots se obtiene a partir de unos valores de densidad fijos de 0,1 g/cm^3 para el cuerpo y 0,5 g/cm^3 para las ruedas.

Materiales Los robots poseen los materiales predefinidos por V-REP *rest_stack_grasp_material*, en el cuerpo; *wheelMaterial*, en las ruedas y *noFriction-Material*, en las esferas deslizantes en el caso de los robots diferenciales.

4.6. Adecuación del modelo al software de simulación

A fin de obtener una simulación más rápida y estable en V-REP, se opta por usar solamente formas puras tanto en el cuerpo (cilíndrico o prismáti-

co) como en las ruedas (cilíndricas) del robot. De este modo, y aun siendo conscientes de que en la realidad el cuerpo del robot contaría con unas hendiduras en las que encajarían las ruedas, se opta por elevar el cuerpo del robot una altura de $2 * r$, como se aprecia en la figura 42, de forma que las ruedas puedan girar sin colisionar con el cuerpo.

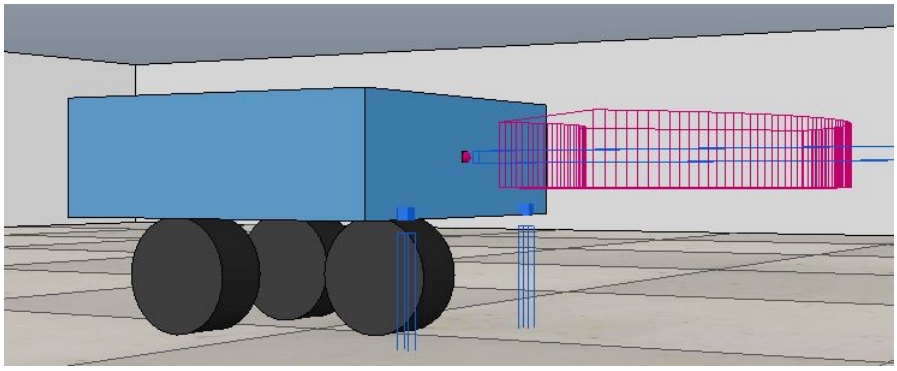

Figura 42: Robot con formas puras

Por otra parte, las ruedas directrices de los modelo triciclo y cuatriciclo cuentan con dos ejes de rotación, uno que permite que la rueda gire libremente y no deslice por el suelo y otro alrededor del cual la rueda adquiere el ángulo deseado para un determinado giro del robot. Sin embargo, la jerarquía de V-REP no permite encadenar dos articulaciones. Por este motivo, se opta por la creación, por cada rueda directriz, de una rueda invisible que sirva de nexo entre ambos ejes, como se aprecia en la jerarquía de la figura 43.

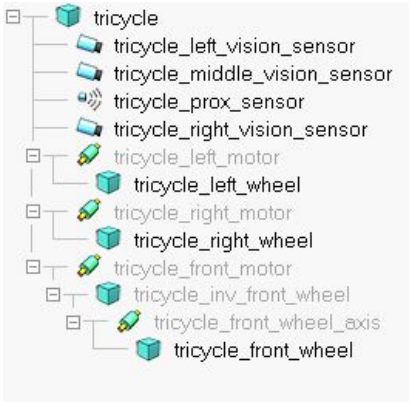

Figura 43: Jerarquía de un robot con rueda directriz

Esta contará con las propiedades *respondable*, *collidable*, *measurable*, *detectable* y *renderable* a 0, a fin de que no interfiera con ningún otro cuerpo durante la simulación y sea, por tanto, inexistente a tales efectos.

5. Descripción de los experimentos

En este capítulo se describen las escenas predefinidas en el software V-REP y el comportamiento que se espera del robot en cada una de ellas. Todas las escenas poseen unas dimensiones de 5 m x 5 m y están delimitadas por paredes de color grisáceo (R=0.95, G=0.95, B=0.95) y 0'5 m de alto.

Dado el alto grado de personalización que permite Koala en lo que respecta al diseño de los robots y aunque se presume que cualquier combinación de parámetros introducida por el usuario y verificada por el programa daría lugar a un robot que se comporte de forma adecuada en cada uno de los experimentos previstos, resulta imposible comprobar todas las posibles combinaciones. Por ello, se facilita para cada experimento tres tablas (una por modelo) con las dimensiones de robots móviles cuyo correcto comportamiento ha sido verificado.

5.1. Cámaras

A fin de poder observar correctamente cualquier detalle de la simulación, se han dispuesto cuatro cámaras adicionales a las que por defecto proporciona V-REP en cualquier escena. Las vistas obtenidas por estas cámaras se han dispuesto en una página 1+5 como la esquematizada en la figura 44.

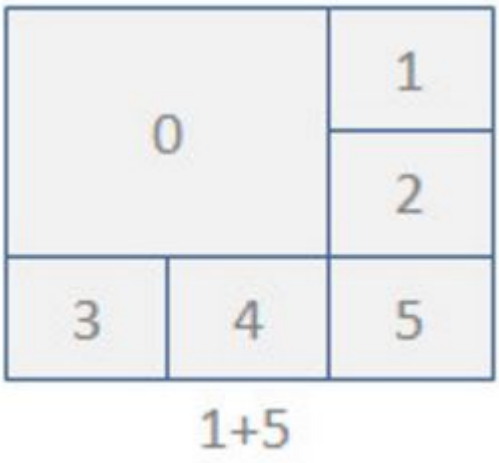

Figura 44: Página 1+5

Se ha asociado la vista de la cámara principal *MainCamera*, situada en

el borde inferior del recinto, al recuadro 0. Los recuadros 3 y 4 muestran las vistas de las cámaras auxiliares 1 y 2, ubicadas, respectivamente en las esquinas superiores izquierda y derecha del escenario. El recuadro 2 muestra la vista cenital del recinto proporcionada por *DefaultZViewCamera* y en el recuadro 5 se aprecia la vista obtenida por *TrackingCamera*. Esta cámara se ofrece a seguir el objeto que el usuario le indique (presumiblemente el propio robot) haciendo clic en su icono y editando sus propiedades. Idealmente, esta acción debería ser realizada de forma programática, pero V-REP aún carece de las funciones necesarias para tal fin. No se ha asociado ninguna vista al recuadro superior derecho de la página, pues sobre él se sitúa la ventana de la interfaz desarrollada en Matlab. En la figura 45, se muestra la página de vistas de uno de los posibles experimentos a modo de ejemplo.

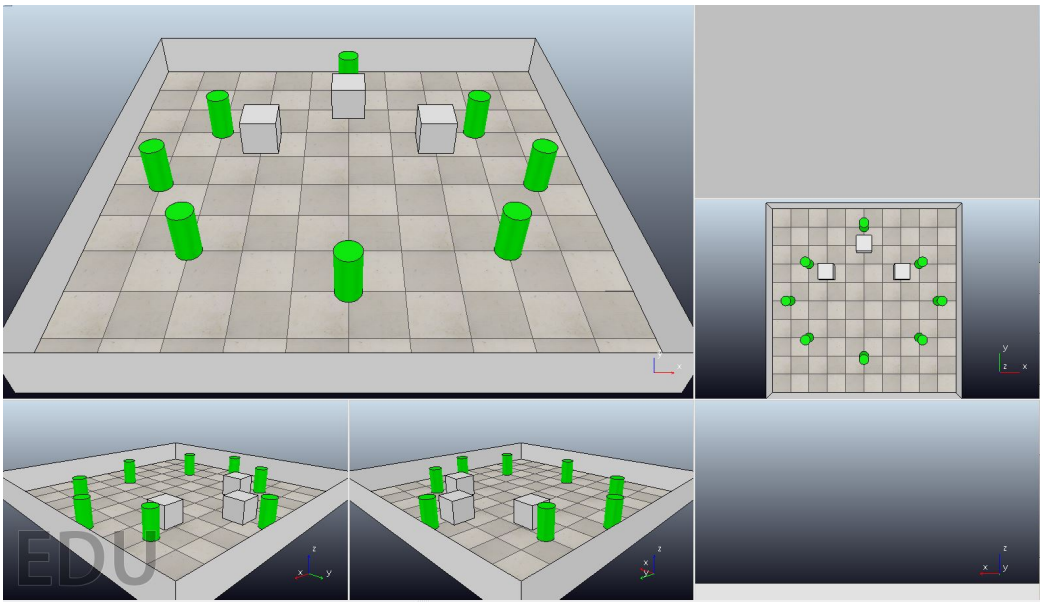

Figura 45: Página de vistas

5.2. Obstáculos

Esta escena cuenta, como se aprecia en la figura 46, con doce obstáculos cilíndricos de color grisáceo (R=0.95, G=0.95, B=0.95), base de 0,125 m de radio y 0,5 m de altura. El usuario puede añadir obstáculos con la forma y dimensiones deseadas en la posición que estime conveniente.

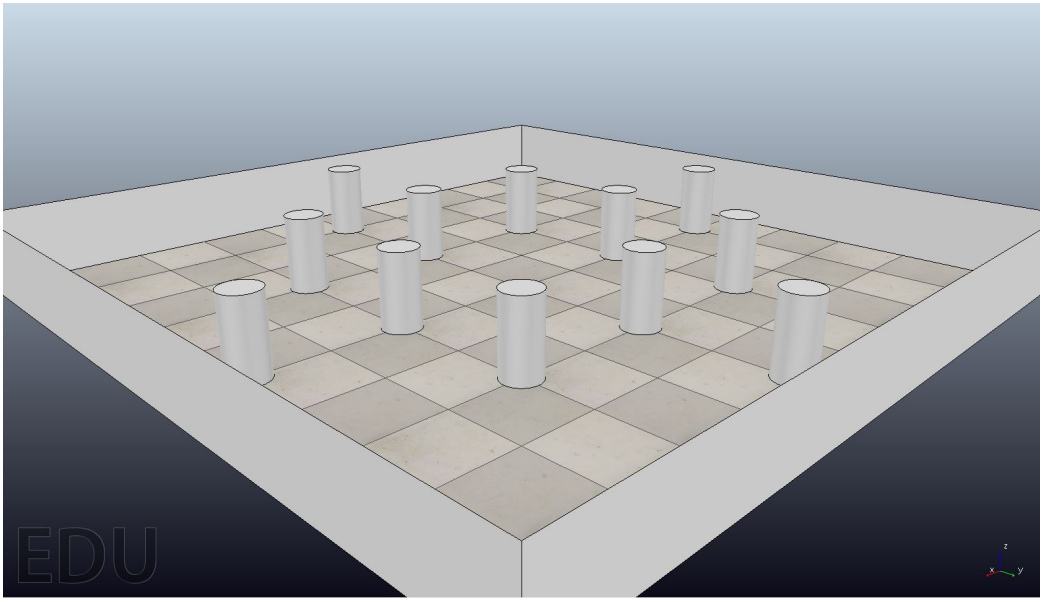

Figura 46: Escena con obstáculos

El robot ha de moverse libremente por ella con la velocidad lineal indicada por el usuario esquivando los obstáculos. Para ello, avanzará en línea recta hasta que el sensor de proximidad situado en la parte frontal del mismo detecte un obstáculo. En este momento, el robot retrocederá girando ligeramente durante un determinado intervalo de tiempo y posteriormente realizará una maniobra de recuperación avanzando y girando ligeramente hacia el sentido contrario del retroceso. Se desprende que los sensores de visión no resultan necesarios para el control del robot en esta escena. Se opta, pues, por ocultarlos a fin de obtener una simulación lo más representativa posible.

El comportamiento descrito se consigue de distintas maneras según el tipo de robot móvil simulado.

Diferencial En el caso de un robot de tipo diferencial el retroceso y ligero giro del robot se consigue asignando al motor izquierdo una velocidad angular de -4/5 ω y al motor derecho, de -6/5 ω. El instante de tiempo en el que el robot ha de finalizar la maniobra de retroceso depende de la velocidad lineal deseada y se obtiene según

$$t_b = t + \frac{50}{v} \tag{38}$$

donde t es el instante de tiempo actual; 50, una constante obtenida empíricamente y v, la velocidad lineal del robot deseada.

Tras el retroceso, el robot ejecuta la maniobra de recuperación asignando al motor izquierdo una velocidad angular de 6/5 ω y al motor derecho, de 4/5 ω. El instante de tiempo en el que el robot ha de finalizar la maniobra de recuperación depende también de la velocidad lineal deseada y se obtiene según

$$t_s = t_b + \frac{50}{v} \tag{39}$$

Finalmente, el avance normal del robot se consigue asignando a ambos motores una velocidad angular de ω. En cualquiera de los tres casos (retroceso, recuperación o avance normal) se consigue que la velocidad lineal del robot móvil sea la deseada por el usuario.

En el cuadro 1, se recogen las dimensiones de un robot con huella circular cuyo correcto comportamiento ha sido comprobado para una velocidad lineal de 0,33 m/s.

Dimensiones	[m]
Radio del cuerpo del robot	0.15
Altura del cuerpo del robot	0.1
Distancia entre ejes	0.2
Radio de las ruedas	0.04
Ancho de las ruedas	0.04

Cuadro 1: Robot diferencial con huella circular verificado en escena con obstáculos

En la figura 47, se muestra un instante de la simulación llevada a cabo con dicho robot.

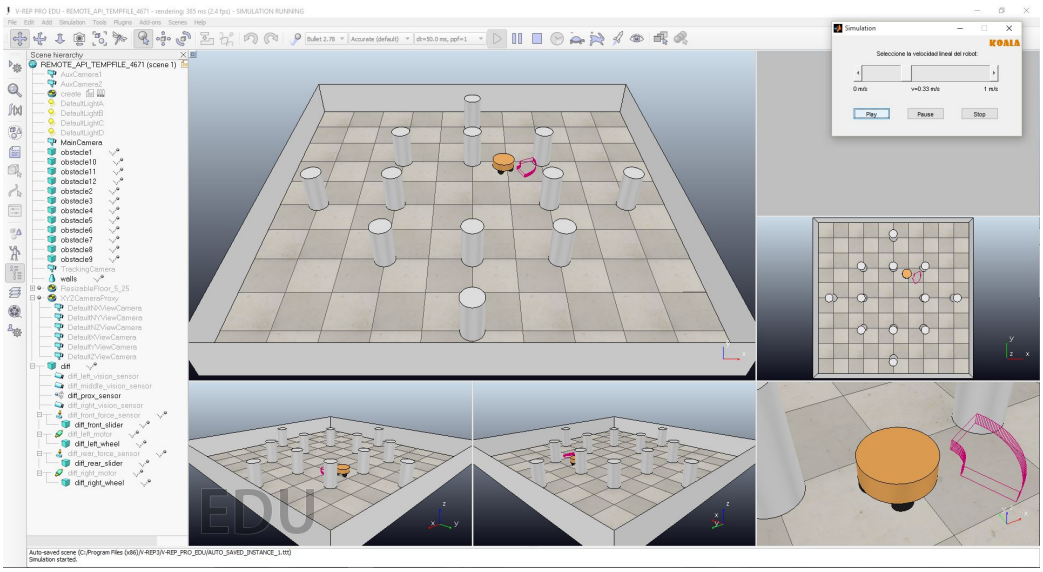

Figura 47: Simulación de robot diferencial en escena con obstáculos

Triciclo En el caso de un robot de tipo triciclo, el retroceso se consigue aisgnando a los motores de ambas ruedas motrices una velocidad angular de -ω y un ángulo de $\pi/6$ a la rueda directriz.

El instante de tiempo en el que el robot ha de finalizar la maniobra de retroceso se calcula de forma análoga al caso de un robot de tipo diferencial pero con una constante mayor, dada la menor maniobrabilidad de los robots tipo triciclo.

$$t_b = t + \frac{200}{v} \tag{40}$$

Tras el retroceso, el robot ejecuta la maniobra de recuperación asignando a los motores de ambas ruedas motrices una velocidad angular de ω y un ángulo de -$\pi/6$ a la rueda directriz. El instante de tiempo en el que el robot ha de finalizar la maniobra de recuperación se obtiene igualmente como

$$t_s = t_b + \frac{200}{v} \tag{41}$$

Finalmente, el avance normal del robot se consigue asignando a los motores de las ruedas motrices una velocidad angular de ω y un ángulo de 0 a la rueda directriz. En cualquiera de las tres maniobras se consigue, de nuevo, que la velocidad lineal del robot móvil sea la deseada por el usuario.

En el cuadro 2, se recogen las dimensiones de un robot con huella cuadrangular cuyo correcto comportamiento ha sido comprobado para una velocidad lineal de 0,33 m/s.

Dimensiones	[m]
Largo del cuerpo del robot	0.3
Ancho del cuerpo del robot	0.3
Altura del cuerpo del robot	0.1
Radio de las ruedas	0.05
Ancho de las ruedas	0.05
Distancia entre ejes izquierdo-derecho	0.18
Distancia entre ejes delantero-trasero	0.18

Cuadro 2: Robot de tipo triciclo con huella cuadrangular verificado en escena con obstáculos

Cuatriciclo En el caso de un robot de tipo cuatriciclo, el retroceso se consigue asignando a los motores de ambas ruedas motrices una velocidad angular de -ω , un ángulo θ_i a la rueda directriz izquierda y un ángulo θ_d a la rueda directriz derecha, siendo θ_i y θ_d los ángulos que permiten un giro del robot de $\pi/6$ sin que se produzca deslizamiento en las ruedas.

Tras el retroceso, el robot ejecuta la maniobra de recuperación asignando a los motores de ambas ruedas motrices una velocidad angular de ω , un ángulo de θ'_i a la rueda directriz izquierda y un ángulo θ'_d a la rueda directriz derecha, a fin de conseguir un giro de -$\pi/6$ sin que se produzca deslizamiento en las ruedas. Los instantes de tiempo en los que el robot ha de finalizar las maniobras de retroceso y recuperación se calculan de igual forma que para un robot de tipo triciclo, ya que ambos modelos comparten una maniobrabilidad más reducida.

Finalmente, el avance normal del robot se consigue asignando a los motores de las ruedas directrices una velocidad angular de ω y un ángulo de 0 a las rueda directrices. Nuevamente, se garantiza que, en cualquiera de las tres maniobras, la velocidad lineal del robot móvil sea la deseada por el usuario.

En el cuadro 3, se recogen las dimensiones de un robot con huella circular cuyo correcto comportamiento ha sido comprobado para una velocidad lineal de 0,33 m/s.

Dimensiones	[m]
Radio del cuerpo del robot	0.2
Altura del cuerpo del robot	0.1
Radio de las ruedas	0.05
Ancho de las ruedas	0.05
Distancia entre ejes izquierdo-derecho	0.18
Distancia entre ejes delantero-trasero	0.18

Cuadro 3: Robot de tipo cuatriciclo con huella circular verificado en escena con obstáculos

5.3. Circuito

Esta escena cuenta, como se aprecia en la figura 48, con un circuito de color negro y 0,15 m de ancho dispuesto en el suelo. El usuario puede añadir obstáculos con la forma y dimensiones deseadas en la posición que estime conveniente.

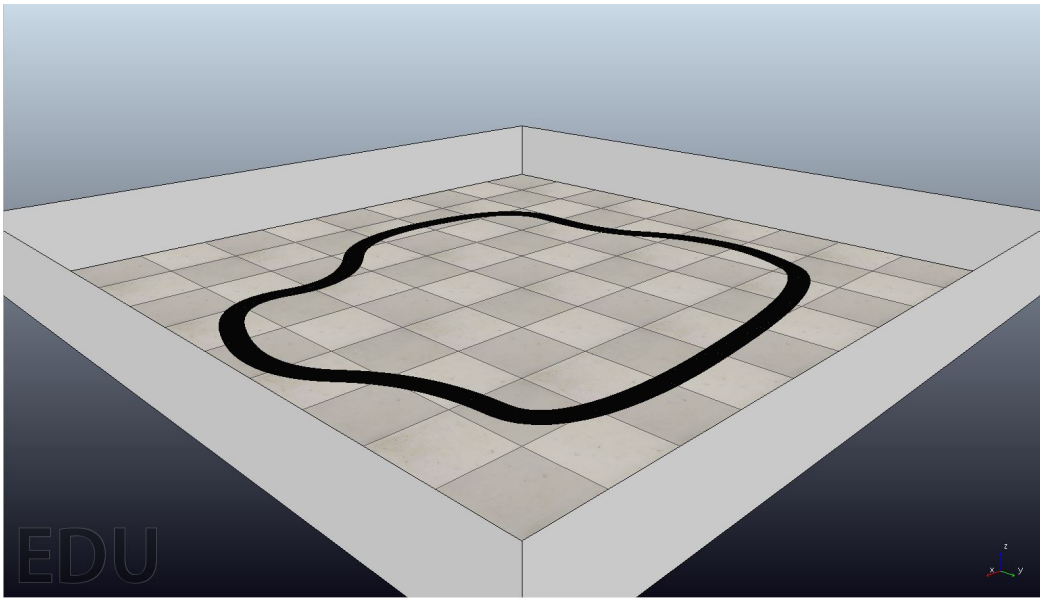

Figura 48: Escena con circuito

El robot ha de incorporarse al circuito a partir de su posición inicial en el centro de la escena y seguirlo indefinidamente. Para ello, el robot avanzará

en línea recta hasta que uno de sus dos sensores de vision situados en la parte frontal del mismo detecte el circuito.

En este momento, el robot girará en el sentido del sensor en cuestión hasta que, presumiblemente, los dos sensores detecten el circuito. En dicho instante el robot girará en el sentido contrario al que lo venía haciendo hasta dicho instante hasta que, de nuevo, solo uno de los dos sensores detecte circuito. En tal caso, el robot volverá a girar en el sentido del sensor en cuestión.

Una vez que el robot se haya incorporado al circuito, este comportamiento seguirá siendo válido. El robot girará a la izquierda si dicho sensor detecta circuito (consecuencia de una curva a la izquierda) y a la derecha si sucede lo contrario. Se considera que un sensor detecta el circuito, de color negro, cuando el valor medio de intensidad obtenido es inferior a 0,3.

Asimismo, el robot continuará siendo capaz de esquivar posibles obstáculos que el usuario añada a la escena, retrociendo y girando ligeramente y recuperándose tal y como hacía en la escena anterior. Se desprende que el sensor de visión central no resulta necesario para el control del robot en esta escena. Se opta, pues, por ocultarlo a fin de obtener una simulación lo más representativa posible.

El comportamiento descrito se consigue de distintas maneras según el tipo de robot móvil simulado.

Diferencial En el caso de un robot de tipo diferencial, el giro a la izquierda se consigue asignando al motor izquierdo una velocidad angular de 4/5 ω y al motor derecho, de 6/5 ω. De forma análoga, el robot gira a la derecha asignando al motor izquierdo una velocidad angular de 6/5 ω y de 4/5 ω al motor derecho.

Las maniobras de retroceso y recuperación al detectar un obstáculo se llevan a cabo de igual forma que en la escena anterior, si bien en esta ocasión los ligeros giros durante estas maniobras no son fijos. Durante el retroceso, se asigna una velocidad angular de de -4/5 ω al motor izquierdo y de -6/5 ω al motor derecho si el robot giró por última vez a la derecha y de -6/5 ω al motor izquierdo y de -4/5 ω al motor derecho, si lo hizo a la izquierda. En la maniobra de recuperación, el robot gira ligeramente en sentido contrario al que lo hiciera durante el retroceso. Los instantes de tiempo en los que el robot ha de finalizar sendas maniobras se calculan de igual forma que para la escena con obstáculos. En cualquier caso, se garantiza que la velocidad lineal del robot sea la indicada por el usuario.

En el cuadro 4, se recogen las dimensiones de un robot con huella cuadrangular cuyo correcto comportamiento ha sido comprobado para una velocidad lineal de 0,33 m/s.

Dimensiones	[m]
Largo del cuerpo del robot	0.4
Ancho del cuerpo del robot	0.3
Altura del cuerpo del robot	0.1
Radio de las ruedas	0.05
Ancho de las ruedas	0.05
Distancia entre ejes	0.2

Cuadro 4: Robot diferencial con huella cuadrangular verificado en escena con circuito

En la figura 49, se aprecia un instante de la simulación llevada a cabo con dicho robot.

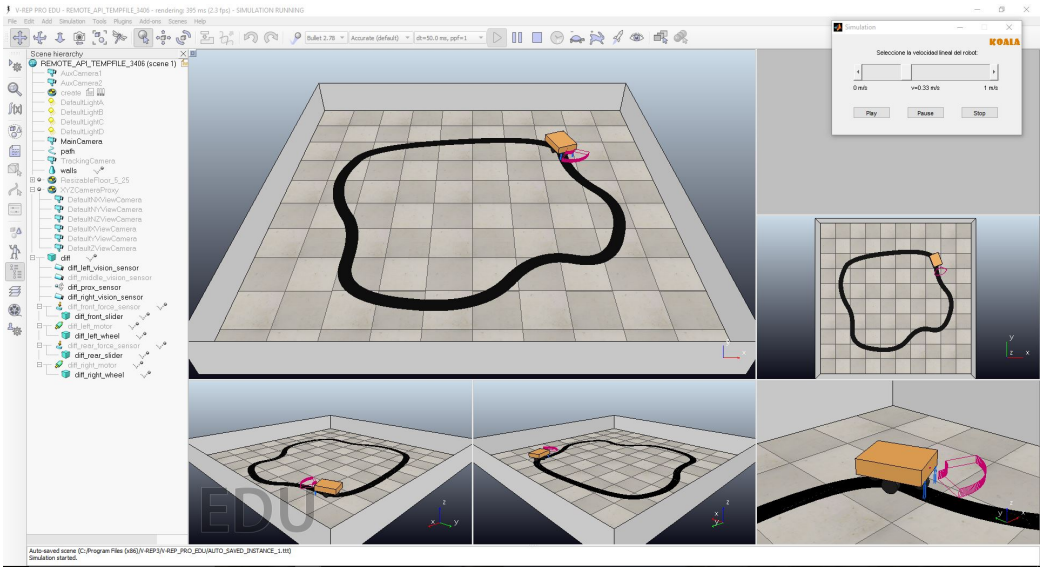

Figura 49: Simulación de robot diferencial en escena con circuito

Triciclo En el caso de un robot de tipo triciclo, el giro a la izquierda se consigue manteniendo la velocidad angular de ω a los motores de ambas

ruedas motrices y asignando un ángulo de $\pi/6$ a la rueda directriz. De forma análoga, el robot gira a la derecha manteniendo la velocidad angular de las ruedas motrices y asignando a la rueda directiz un ángulo de -$\pi/6$.

Las maniobras de retroceso y recuperación se llevan a cabo de forma similar al caso de la escena con obstáculos, si bien en esta ocasión los ángulos de la rueda directriz no son fijos. Durante el retroceso, este será igual a -$\pi/6$ si el robot giró por última vez a la izquierda y a $\pi/6$, si lo hizo a la derecha. Por su parte durante la maniobra de recuperación, el ángulo de la rueda directriz será de $\pi/6$ si el último giro realizado por el robot tuvo lugar hacia la izquierda y de -$\pi/6$, si fue hacia la derecha. Los instantes de tiempo en los que el robot ha de finalizar sendas maniobras se calculan de igual forma que para la escena con obstáculos. En cualquier caso, se consigue, de nuevo, que la velocidad lineal del robot móvil sea la deseada por el usuario.

En el cuadro 5, se recogen las dimensiones de un robot con huella circular cuyo correcto comportamiento ha sido comprobado para una velocidad lineal de 0,25 m/s.

Dimensiones	[m]
Radio del cuerpo del robot	0.2
Altura del cuerpo del robot	0.1
Radio de las ruedas	0.05
Ancho de las ruedas	0.05
Distancia entre ejes izquierdo-derecho	0.15
Distancia entre ejes delantero-trasero	0.15

Cuadro 5: Robot de tipo triciclo con huella circular verificado en escena con circuito

Cuatriciclo En el caso de un robot de tipo cuatriciclo, el giro a la izquierda se consigue manteniendo la velocidad angular de ω a los motores de ambas ruedas motrices y asignando un ángulo de θ_i a la rueda directriz izquierda y un ángulo de θ_d a la rueda directriz derecha, a fin de conseguir un giro de $\pi/6$ sin que se produzca deslizamiento en las ruedas. De forma análoga, el robot gira a la derecha manteniendo la velocidad angular de las ruedas motrices y asignando un ángulo de θ'_i a la rueda directriz izquierda y un ángulo de θ'_d a la rueda directriz derecha, consiguiendo un giro de -$\pi/6$ sin que se produzca deslizamiento en las ruedas

Las maniobras de retroceso y recuperación se llevan a cabo de forma similar al caso de la escena con obstáculos, si bien en esta ocasión los ángulos de giro no son fijos. Durante el retroceso, este será igual a -$\pi/6$ si el robot giró por última vez a la izquierda y a $\pi/6$, si lo hizo a la derecha. Por su parte durante la maniobra de recuperación, el ángulo será de $\pi/6$ si el último giro realizado por el robot tuvo lugar hacia la izquierda y de -$\pi/6$, si fue hacia la derecha. Los instantes de tiempo en los que el robot ha de finalizar sendas maniobras se calculan de igual forma que para la escena con obstáculos. Nuevamente, se consigue que, en todo momento la velocidad lineal del robot sea la deseada por el usuario.

En el cuadro 6, se recogen las dimensiones de un robot con huella cuadrangular cuyo correcto comportamiento ha sido comprobado para una velocidad lineal de 0,25 m/s.

Dimensiones	[m]
Largo del cuerpo del robot	0.3
Ancho del cuerpo del robot	0.3
Altura del cuerpo del robot	0.1
Radio de las ruedas	0.05
Ancho de las ruedas	0.05
Distancia entre ejes izquierdo-derecho	0.15
Distancia entre ejes delantero-trasero	0.15

Cuadro 6: Robot de tipo cuatriciclo con huella cuadrangular verificado en escena con circuito

5.4. Balizas

Esta escena cuenta, como se distingue en la figura 50, con ocho balizas cilíndricas de color verde (R=0, G=1, B=0), base de 0,125 m de radio y 0,5 m de altura y tres obstáculos cúbicos, de color grisáceo (R=0.95, G=0.95, B=0.95) y 0,4 m de lado, situados entre la posición inicial del robot en el centro de la escena y tres de las balizas. El usuario puede añadir balizas y obstáculos, estos últimos con la forma y dimensiones deseadas, en la posición que estime conveniente.

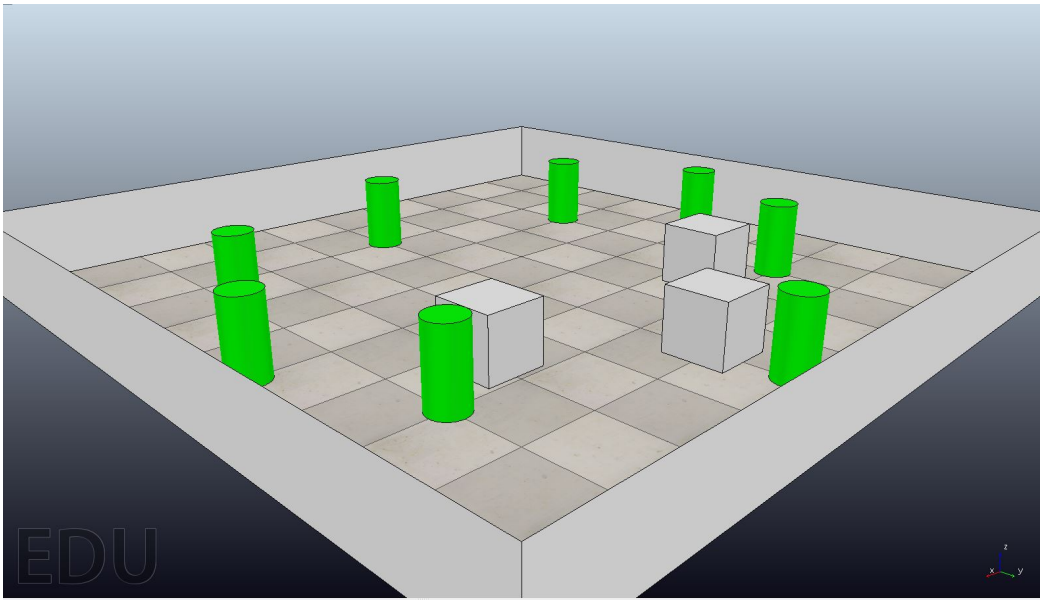

Figura 50: Escena con balizas

El robot ha de girar hasta que el sensor de visión situado en la parte frontal del mismo detecte un objeto de color verde. En dicho momento, avanzará en línea recta hasta que el sensor de proximidad detecte la baliza. El robot retrocederá entonces girando ligeramente y comenzará la búsqueda de un nuevo objetivo. Se considera que el sensor detecta una baliza, de color verde, cuando los valores de R y B obtenidos sean inferiores a 0.2 y el valor de G, superior a 0.8.

Asimismo, el robot seguirá siendo capaz de esquivar posibles obstáculos que el usuario añada a la escena. Se desprende que los sensores de visión izquierdo y derecho no resultan necesarios para el control del robot en esta escena. Se opta, pues, por ocultarlos a fin de obtener una simulación lo más representativa posible.

El comportamiento descrito se consigue de distiintas maneras según el tipo de robot móvil simulado.

Diferencial En el caso de un robot de tipo diferencial, el giro sobre sí mismo en busca de una baliza se consigue asignado al motor izquierdo una velocidad angular de 1/30 ω y al motor derecho, de -1/30 ω. Estas han de ser tan reducidas para impedir que el robot pase por alto alguna baliza. Las

maniobras de avance y retroceso se ejecutan de igual forma que en las escenas anteriores.

En el cuadro 7, se recogen las dimensiones de un robot con huella circular cuyo correcto comportamiento ha sido comprobado para una velocidad lineal de 0,33 m/s.

Dimensiones	[m]
Radio del cuerpo del robot	0.15
Altura del cuerpo del robot	0.1
Distancia entre ejes	0.18
Radio de las ruedas	0.04
Ancho de las ruedas	0.04

Cuadro 7: Robot diferencial con huella circular verificado en escena con balizas

En la figura 51, se muestra un instante de la simulación llevada a cabo con dicho robot.

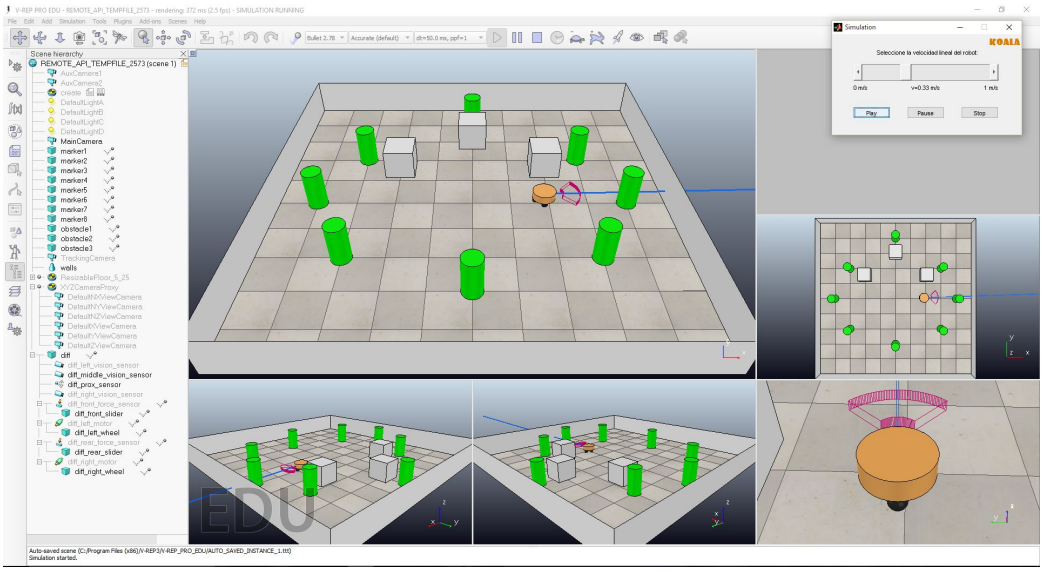

Figura 51: Simulación de robot diferencial en escena con balizas

Triciclo En el caso de un robot de tipo triciclo, el giro en busca de una baliza se consigue asignado a los motores de ambas ruedas motrices una velocidad angular de ω y un ángulo de $\pi/6$ a la rueda directriz. Las maniobras de avance y retroceso se ejecutan de igual forma que en las escenas anteriores.

En el cuadro 8, se recogen las dimensiones de un robot con huella cuadrangular cuyo correcto comportamiento ha sido comprobado para una velocidad lineal de 0,33 m/s.

Dimensiones	[m]
Largo del cuerpo del robot	0.3
Ancho del cuerpo del robot	0.3
Altura del cuerpo del robot	0.1
Radio de las ruedas	0.05
Ancho de las ruedas	0.05
Distancia entre ejes izquierdo-derecho	0.18
Distancia entre ejes delantero-trasero	0.18

Cuadro 8: Robot de tipo triciclo con huella cuadrangular verificado en escena con balizas

Cuatriciclo En el caso de un robot de tipo cuatriciclo, el giro en busca de una baliza se consigue asignando a los motores de ambas ruedas motrices una velocidad angular de ω, un ángulo de θ_i a la rueda directriz izquierda y un ángulo θ_d a la rueda directriz derecha, a fin de conseguir un giro de $\pi/6$ sin que se produzca deslizamiento en las ruedas. Las maniobras de avance y retroceso se ejecutan de igual forma que en las escenas anteriores.

En el cuadro 9, se recogen las dimensiones de un robot con huella circular cuyo correcto comportamiento ha sido comprobado para una velocidad lineal de 0,33 m/s.

Dimensiones	[m]
Radio del cuerpo del robot	0.2
Altura del cuerpo del robot	0.1
Radio de las ruedas	0.05
Ancho de las ruedas	0.05
Distancia entre ejes izquierdo-derecho	0.18
Distancia entre ejes delantero-trasero	0.18

Cuadro 9: Robot de tipo cuatriciclo con huella circular verificado en escena con balizas

5.5. Escena vacía

Esta escena, vacía inicialmente como se muestra en la figura 52, permite añadir en las posiciones indicadas por el usuario obstáculos con la forma y dimensiones deseadas y balizas.

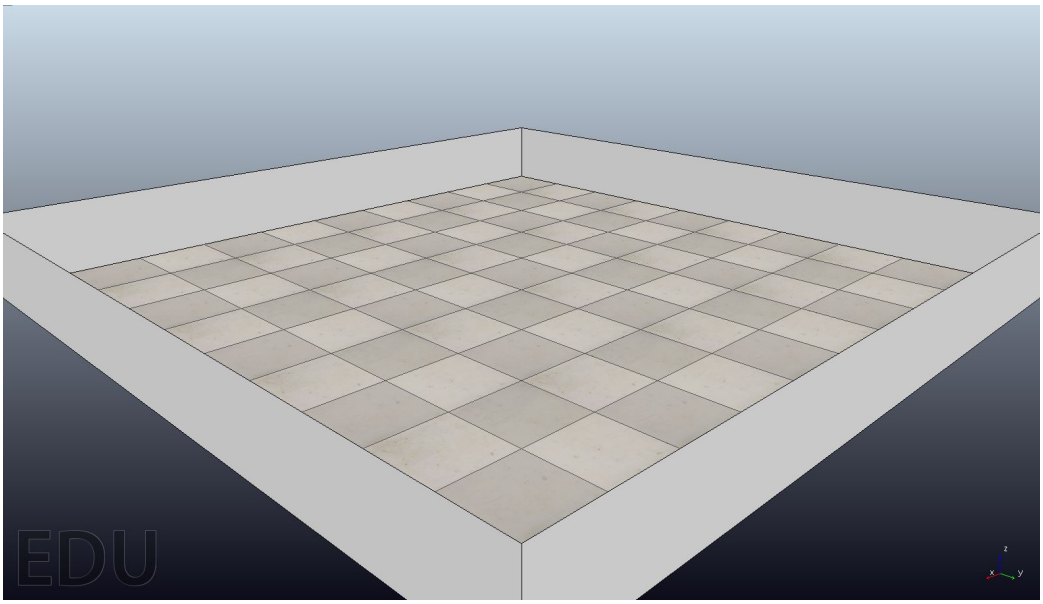

Figura 52: Escena vacía

El usuario eligirá qué tipo de comportamiento (dirigirse hacia las balizas esquivando los obstáculos, o simplemente evitar estos útlimos) desea que

muestre el robot.

5.6. Escena creada por el usuario

El usuario eligirá qué tipo de comportamiento (seguir un circuito esquivando los obstáculos, dirigirse hacia las balizas esquivando los obstáculos, o simplemente evitar estos útlimos) desea que muestre el robot en una escena creada previamente.

6. Conclusiones y líneas abiertas

6.1. Conclusiones

De acuerdo a los objetivos del Trabajo expuestos al comienzo de esta memoria, se ha desarrollado una plataforma que permite el diseño y la simulación de robots móviles. Su correcto funcionamiento se admite probado por los experimentos descritos en el capítulo anterior.

Se considera que la elección de integrar en la plataforma el software de experimentación V-REP, cuyo funcionamiento se desconocía antes de iniciar este Trabajo, se ha antojado acertada. Sus extensas funcionalidades han permitido desarrollar posibilidades que se hubieran visto restringidas de haber limitado el trabajo realizado a herramientas software que resultaban familiares como era el caso de Matlab.

En lo que respecta a las posibles aplicaciones del trabajo realizado, se ha buscado en todo momento que la interfaz desarrollada fuera clara y accesible a cualquier usuario sin experiencia en el manejo de sofware avanzado. Además, V-REP permite ofrecer simulaciones visualmente muy atractivas. Por este motivo, se aprecia la utilidad que la plataforma podría tener en entornos universitarios o, incluso, en niveles educativos inferiores a fin de despertar la curiosidad por la robótica en potenciales ingenieros.

6.2. Líneas abiertas

Actualmente, Koala permite probar las tres configuraciones de robots móviles más extendidas en unos escenarios predefinidos que se han considerado los más adecuados para testear el funcionamiento básico del robot. Sin embargo, la estructura de la plataforma permitiría añadir en un futuro de forma sencilla cualquier otra configuración del robot o escenario que se estimen oportunos.

No obstante, resulta necesario tener en cuenta que cualquier añadido al trabajo realizado vendría condicionado por la existencia o no de funciones para tal fin en las API (preferiblemente, la remota) que ofrece V-REP. A este respecto, se recomendaría modificar, en cuanto se desarrollen las funciones de la API remota adecuadas, el método de creación del robot y demás objetos en la escena para que esta tuviera lugar desde la propia aplicación de Matlab. De esta forma, no sería ya necearia la inclusión del script *create* en cada escena. Se aconsejaría, además asociar la cámara *TrackingCamera* al robot

de forma programática en cuanto se desarrolle una función de la API que lo permita.

Asimismo, durante el desarrollo del trabajo, se consideró que el robot dibujara su trayectoria en el suelo del escenario. Dada la inexistencia de funciones para tal fin en la API remota, se realizó una tentativa con funciones de la API regular llamadas desde la aplicación de Matlab que resultó ineficiente en términos de velocidad de simulación. Nuevamente, se recomendaría no desistir de esta idea y analizar las posibilidades que a este respecto se ofrezcan en un futuro.

Por otra parte, ya a más largo plazo, se propondría la construcción física de un robot (controlado por el mismo código desarrollado en Matlab exportado a algún tipo de sistema embebido) y un escenario de las características de los simulados que permitiera corroborar los resultados de la simulación en un entorno real.

Referencias

[1] Coppelia Robotics, *V-REP User Manual. Version 3.3.1*, 2016
Disponible en: http://www.coppeliarobotics.com/helpFiles/

[2] G. Dudek, M. Jenkin, *Computational Principles of Mobile Robotics*, 2000

[3] A. Ollero Baturone, *Robótica:manipuladores y robots móviles*, 1991

[4] R. Siegwart, *Introduction to Autonomous Mobile Robots*, 2004

A. Código de Matlab

A.1. Start.m

```
function varargout = Start(varargin)

% Comienzo del código de inicialización - NO EDITAR
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Start_OpeningFcn, ...
                   'gui_OutputFcn',  @Start_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:})
        ;
else
    gui_mainfcn(gui_State, varargin{:});
end
% Fin del código de inicialización - NO EDITAR


%--- Se ejecuta justo antes de que Start se haga visible.
function Start_OpeningFcn(hObject, eventdata, handles, varargin)
global next
next=0;

% Se posiciona la ventana en la esquina superior derecha de la
   pantalla.
scrsz=get(0,'ScreenSize');
pos_act=get(gcf,'Position');
xr=scrsz(3)-pos_act(3);
xp=round(xr/2);
yr=scrsz(4)-pos_act(4);
yp=round(yr/2);
set(gcf,'position',[xp,yp,pos_act(3),pos_act(4)]);

handles.output = hObject;
guidata(hObject, handles);

```

```
40
41 %--- Las salidas de esta función se devuelven por la línea de
       comandos.
42 function varargout = Start_OutputFcn(hObject, eventdata, handles
       )
43 varargout{1} = handles.output;
44
45
46 %--- Función que se ejecuta al pulsar el botón Connect.
47 function Connect_Callback(hObject, eventdata, handles)
48     global vrep
49     global clientID
50     global next
51
52     vrep=remApi('remoteApi');
53         vrep.simxFinish(-1);
54         clientID=vrep.simxStart('127.0.0.1',19997,true,true
               ,5000,5);
55     if (clientID~=-1)
56         next=1;
57         close Start
58         SelectScene
59     else
60       errordlg('No ha sido posible conectar con el servidor.
             Compruebe que V-REP se encuentre abierto.','ERROR')
61     end
62
63
64 %--- Función que se ejecuta cuando se intenta cerrar la ventana
       .
65 function figure1_CloseRequestFcn(hObject, eventdata, handles)
66 global next
67
68 % Si la petición de cierre de la ventana no es consecuencia del
       normal tránsito del programa de una ventana a otra
69 % sino que deriva del clic por parte del usuario en el icono de
       cierre, se exige una confirmación.
70 if next==0
71     opc=questdlg('¿Desea salir del programa?','Salir','Sí','No',
           'No');
72     if strcmp(opc,'No')
73         return
74     end
75 end
76 next=0;
```

```
77 delete(hObject);
```

A.2. SelectScene.m

```
function varargout = SelectScene(varargin)
% Comienzo del código de inicialización - NO EDITAR
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
    'gui_Singleton',  gui_Singleton, ...
    'gui_OpeningFcn', @SelectScene_OpeningFcn, ...
    'gui_OutputFcn',  @SelectScene_OutputFcn, ...
    'gui_LayoutFcn',  [] , ...
    'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:})
        ;
else
    gui_mainfcn(gui_State, varargin{:});
end
% Fin del código de inicialización - NO EDITAR


%--- Se ejecuta justo antes de que SelectScene se haga visible.
function SelectScene_OpeningFcn(hObject, eventdata, handles,
    varargin)
% Se posiciona la ventana en la esquina superior derecha de la
    pantalla.
scrsz=get(0,'ScreenSize');
pos_act=get(gcf,'Position');
xr=scrsz(3)-pos_act(3);
xp=round(xr/2);
yr=scrsz(4)-pos_act(4);
yp=round(yr/2);
set(gcf,'position',[xp,yp,pos_act(3),pos_act(4)]);

handles.output = hObject;
guidata(hObject, handles);


%--- Las salidas de esta función se devuelven por la línea de
    comandos.
function varargout = SelectScene_OutputFcn(hObject, eventdata,
    handles)
```

```
39 varargout{1} = handles.output;
40
41
42 %--- Función que se ejecuta al pulsar el botón 'Obstacles'
       encargada de abrir la escena asociada en V-REP.
43 function Obstacles_Callback(hObject, eventdata, handles)
44 global vrep
45 global clientID
46 global next
47 global num_obs
48 global scene
49
50 scene=1;
51 num_obs=12;
52
53 file_name='Obstacles.ttt';
54 path=pwd;
55 vrep.simxLoadScene(clientID,[path,'\',file_name],1,vrep.
       simx_opmode_blocking);
56 vrep.simxSynchronous(clientID,true);
57
58 next=1;
59 close SelectScene
60 SelectRobot
61
62
63 %--- Función que se ejecuta al pulsar el botón 'Circuit'
       encargada de abrir la escena asociada en V-REP.
64
65 function Circuit_Callback(hObject, eventdata, handles)
66 global vrep
67 global clientID
68 global next
69 global num_obs
70 global scene
71
72 scene=2;
73 num_obs=0;
74
75 file_name='Circuit.ttt';
76 path=pwd;
77 vrep.simxLoadScene(clientID,[path,'\',file_name],1,vrep.
       simx_opmode_blocking);
78 vrep.simxSynchronous(clientID,true);
79
```

```
80  next=1;
81  close SelectScene
82  SelectRobot
83
84  %--- Función que se ejecuta al pulsar el botón 'Markers'
        encargada de
85  % abrir la escena asociada en V-REP.
86  function Markers_Callback(hObject, eventdata, handles)
87  global vrep
88  global clientID
89  global next
90  global num_obs
91  global num_marker
92  global scene
93
94  scene=3;
95  num_obs=3;
96  num_marker=8;
97
98  file_name='Markers.ttt';
99  path=pwd;
100 vrep.simxLoadScene(clientID,[path,'\',file_name],1,vrep.
        simx_opmode_blocking);
101 vrep.simxSynchronous(clientID,true);
102
103 next=1;
104 close SelectScene
105 SelectRobot
106
107 %--- Función que se ejecuta al pulsar el botón 'Empty'
        encargada de
108 % abrir la escena asociada en V-REP.
109 function Empty_Callback(hObject, eventdata, handles)
110 global vrep
111 global clientID
112 global next
113 global num_obs
114 global num_marker
115 global scene
116
117 scene=4;
118 num_obs=0;
119 num_marker=0;
120
121 file_name='Empty.ttt';
```

```
122 path=pwd;
123 vrep.simxLoadScene(clientID,[path,'\',file_name],1,vrep.
        simx_opmode_blocking);
124 vrep.simxSynchronous(clientID,true);
125
126 next=1;
127 close SelectScene
128 SelectRobot
129
130
131 %--- Función que se ejecuta al pulsar el botón 'Load' encargada
        de
132 % abrir en V-REP la escena elegida por el usuario y comprobar
        que contenga
133 % el dummy 'create'.
134 function Load_Callback(hObject, eventdata, handles)
135 global vrep
136 global clientID
137 global next
138 global scene
139
140 scene=5;
141
142 [file_name,path]=uigetfile('*.ttt','Seleccione una escena');
143 vrep.simxLoadScene(clientID,[path,'\',file_name],1,vrep.
        simx_opmode_blocking);
144 ret=vrep.simxGetObjectHandle(clientID,'create',vrep.
        simx_opmode_blocking);
145
146 if(ret~=0)
147     errordlg('La escena seleccionada no es válida. Compruebe que
            contiene el dummmy create con el script en Lua que puede
            encontrar en esta misma carpeta.','ERROR')
148     vrep.simxCloseScene(clientID,vrep.simx_opmode_blocking);
149 else
150     vrep.simxSynchronous(clientID,true);
151
152     next=1;
153     close SelectScene
154     SelectRobot
155 end
156
157
158 %--- Función que se ejecuta al pulsar el botón Cancel.
159 function Cancel_Callback(hObject, eventdata, handles)
```

```
160 global vrep
161 global clientID
162
163 vrep.simxFinish(clientID);
164 close SelectScene
165
166
167 %--- Función que se ejecuta al pulsar el botón PreviousView.
168 function PreviousView_Callback(hObject, eventdata, handles)
169 PreView
170
171
172 %--- Función que se ejecuta cuando se intenta cerrar la ventana
    .
173 function figure1_CloseRequestFcn(hObject, eventdata, handles)
174 global next
175
176 % Si la petición de cierre de la ventana no es consecuencia del
    normal tránsito del programa de una ventana a otra
177 % sino que deriva del clic por parte del usuario en el icono de
    cierre, se exige una confirmación.
178 if next==0
179     opc=questdlg('¿Desea salir del programa?','Salir','Sí','No',
        'No');
180     if strcmp(opc,'No')
181         return
182     end
183 end
184 next=0;
185 delete(hObject);
```

A.3. PreView.m

```
function varargout = PreView(varargin)
% Comienzo del código de inicialización - NO EDITAR
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
    'gui_Singleton',  gui_Singleton, ...
    'gui_OpeningFcn', @PreView_OpeningFcn, ...
    'gui_OutputFcn',  @PreView_OutputFcn, ...
    'gui_LayoutFcn',  [] , ...
    'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:})
        ;
else
    gui_mainfcn(gui_State, varargin{:});
end
% Fin del código de inicialización - NO EDITAR


%--- Se ejecuta justo antes de que PreView se haga visible.
function PreView_OpeningFcn(hObject, eventdata, handles,
    varargin)
% Se posiciona la ventana en la esquina superior derecha de la
    pantalla.
scrsz=get(0,'ScreenSize');
pos_act=get(gcf,'Position');
xr=scrsz(3)-pos_act(3);
xp=round(xr/2);
yr=scrsz(4)-pos_act(4);
yp=round(yr/2);
set(gcf,'position',[xp,yp,pos_act(3),pos_act(4)]);

% Por defecto, se muestra la vista previa y la descripción de la
    escena
% 'Obstacles' al abrir la ventana.
axes(handles.axes1)
path=pwd;
image=imread([path,'\Obstacles.jpg']);
image=imresize(image,[250,NaN]);
axis off
```

```
40 imshow(image)
41 set(handles.Description,'string','El robot se moverá por la
       escena esquivando los obstáculos.');
42
43 handles.output = hObject;
44 guidata(hObject, handles);
45
46
47 %--- Las salidas de esta función se devuelven por la línea de
       comandos.
48 function varargout = PreView_OutputFcn(hObject, eventdata,
       handles)
49 varargout{1} = handles.output;
50
51
52 %--- Función que se ejecuta al pulsar el botón Back.
53 function Back_Callback(hObject, eventdata, handles)
54 global next
55
56 next=1;
57 close PreView
58
59
60 %--- Funnción que se ejecuta cuando cambia la selección el el
       panel Scene encargada de mostrar la vista previa y la
       descripción de la escena elegida.
61 function Scene_SelectionChangeFcn(hObject, eventdata, handles)
62 if(hObject==handles.Obstacles)
63     file_name='Obstacles.jpg';
64     set(handles.Description,'string','El robot se moverá por la
           escena esquivando los obstáculos.');
65
66 elseif(hObject==handles.Circuit)
67     file_name='Circuit.jpg';
68     set(handles.Description,'string','El robot seguirá el
           circuito y esquivará los obstáculos que podrá añadir
           posteriormente.');
69
70 elseif(hObject==handles.Markers)
71     file_name='Markers.jpg';
72     set(handles.Description,'string','El robot se dirigirá hacia
            las balizas de color verde y esquivará los obstáculos.')
           ;
73
74 elseif(hObject==handles.Empty)
```

```
75      file_name='Empty.jpg';
76      set(handles.Description,'string','El robot se moverá por la
            escena según el comportamiento que especifique
            posteriormente.');
77  end
78
79  axes(handles.axes1)
80  path=pwd;
81  image=imread([path,'\',file_name]);
82  image=imresize(image,[250,NaN]);
83  axis off
84  imshow(image)
85
86  %--- Función que se ejecuta cuando se intenta cerrar la ventana
        .
87  function figure1_CloseRequestFcn(hObject, eventdata, handles)
88  global next
89
90  %Si la petición de cierre de la ventana no es consecuencia del
        normal tránsito del programa de una ventana a otra
91  %sino que deriva del clic por parte del usuario en el icono de
        cierre, se exige una confirmación.
92  if next==0
93      opc=questdlg('¿Desea salir del programa?','Salir','Sí','No',
            'No');
94      if strcmp(opc,'No')
95          return
96      end
97  end
98  next=0;
99  delete(hObject);
```

A.4. SelectRobot.m

```
function varargout = SelectRobot(varargin)
% Comienzo del código de inicialización - NO EDITAR
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
    'gui_Singleton',  gui_Singleton, ...
    'gui_OpeningFcn', @SelectRobot_OpeningFcn, ...
    'gui_OutputFcn',  @SelectRobot_OutputFcn, ...
    'gui_LayoutFcn',  [] , ...
    'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:})
        ;
else
    gui_mainfcn(gui_State, varargin{:});
end
% Fin del código de inicialización - NO EDITAR


%--- Se ejecuta justo antes de que SelectRobot se haga visible.
function SelectRobot_OpeningFcn(hObject, eventdata, handles,
    varargin)
global footprint
global model

% Se inicializan las variables asociadas a la huella y el modelo
    .
footprint=0;
model=0;

% Se posiciona la ventana en la esquina superior derecha de la
    pantalla.
scrsz=get(0,'ScreenSize');
pos_act=get(gcf,'Position');
xr=scrsz(3)-pos_act(3);
xp=round(xr/2);
yr=scrsz(4)-pos_act(4);
yp=round(yr/2);
set(gcf,'position',[xp,yp,pos_act(3),pos_act(4)]);

```

```
40 handles.output = hObject;
41 guidata(hObject, handles);
42
43
44 %--- Las salidas de esta función se devuelven por la línea de
       comandos.
45 function varargout = SelectRobot_OutputFcn(hObject, eventdata,
       handles)
46 varargout{1} = handles.output;
47
48
49 %--- Funnción que se ejecuta cuando cambia la selección en el
       panel
50 % Footprint.
51 function Footprint_SelectionChangeFcn(hObject, eventdata,
       handles)
52 global footprint
53
54 if(hObject==handles.Circular)
55     footprint=1;
56
57 elseif(hObject==handles.Quadrangular)
58     footprint=2;
59 end
60
61 guidata(hObject,handles);
62
63
64 %--- Funnción que se ejecuta cuando cambia la selección en el
       panel
65 % Model.
66 function Model_SelectionChangeFcn(hObject, eventdata, handles)
67 global model
68
69 if(hObject==handles.Differential)
70     model=1;
71
72 elseif(hObject==handles.Tricycle)
73     model=2;
74
75 elseif(hObject==handles.Quadricycle)
76     model=3;
77
78 end
79
```

```
80  guidata(hObject,handles);
81
82
83  %--- Función que se ejecuta al pulsar el botón Back.
84  function Back_Callback(hObject, eventdata, handles)
85  global vrep
86  global clientID
87  global next
88
89  vrep.simxCloseScene(clientID,vrep.simx_opmode_blocking);
90  next=1;
91  close SelectRobot
92  SelectScene
93
94
95  %--- Función que se ejecuta al pulsar el botón Cancel.
96  function Cancel_Callback(hObject, eventdata, handles)
97  global vrep
98  global clientID
99
100 vrep.simxCloseScene(clientID,vrep.simx_opmode_blocking);
101 vrep.simxGetPingTime(clientID);
102 vrep.simxFinish(clientID);
103 close SelectRobot
104
105
106 %--- Función que se ejecuta al pulsar el botón Next encargada
        de transitar a la ventana generadora de robot correspondiente
        .
107 function Next_Callback(hObject, eventdata, handles)
108 global next
109 global footprint
110 global model
111
112 switch model
113     case 1
114         switch footprint
115             case 1
116                 next=1;
117                 close SelectRobot
118                 CreaDiffCir
119             case 2
120                 next=1;
121                 close SelectRobot
122                 CreaDiffQuad
```

```
123             otherwise
124                 errordlg('Seleccione un tipo de huella.','ERROR'
                        )
125         end
126
127     case 2
128         switch footprint
129             case 1
130                 next=1;
131                 close SelectRobot
132                 CreaTricycleCir
133             case 2
134                 next=1;
135                 close SelectRobot
136                 CreaTricycleQuad
137             otherwise
138                 errordlg('Seleccione un tipo de huella.','ERROR'
                        )
139         end
140
141     case 3
142         switch footprint
143             case 1
144                 next=1;
145                 close SelectRobot
146                 CreaQuadricycleCir
147             case 2
148                 next=1;
149                 close SelectRobot
150                 CreaQuadricycleQuad
151             otherwise
152                 errordlg('Seleccione un tipo de huella.','ERROR'
                        )
153         end
154     otherwise
155      errordlg('Seleccione un modelo.','ERROR')
156
157 end
158
159
160 %--- Función que se ejecuta cuando se intenta cerrar la ventana
        .
161 function figure1_CloseRequestFcn(hObject, eventdata, handles)
162 global next
163
```

```
164 %Si la petición de cierre de la ventana no es consecuencia del
    normal tránsito del programa de una ventana a otra
165 %sino que deriva del clic por parte del usuario en el icono de
    cierre, se exige una confirmación.
166 if next==0
167     opc=questdlg('¿Desea salir del programa?','Salir','Sí','No',
        'No');
168     if strcmp(opc,'No')
169         return
170     end
171 end
172 next=0;
173 delete(hObject);
```

A.5. CreaDiffCir.m

```
function varargout = CreaDiffCir(varargin)
% Comienzo del código de inicialización - NO EDITAR
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @CreaDiffCir_OpeningFcn, ...
                   'gui_OutputFcn',  @CreaDiffCir_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% Fin del código de inicialización - NO EDITAR


%--- Se ejecuta justo antes de que CreaDiffCir se haga visible.
function CreaDiffCir_OpeningFcn(hObject, eventdata, handles, varargin)
% Se posiciona la ventana en la esquina superior derecha de la pantalla.
scrsz=get(0,'ScreenSize');
pos_act=get(gcf,'Position');
xr=scrsz(3)-pos_act(3);
xp=round(xr/2);
yr=scrsz(4)-pos_act(4);
yp=round(yr/2);
set(gcf,'position',[xp,yp,pos_act(3),pos_act(4)]);

handles.output = hObject;
guidata(hObject, handles);


%--- Las salidas de esta función se devuelven por la línea de comandos.
function varargout = CreaDiffCir_OutputFcn(hObject, eventdata,
```

```
    handles)
varargout{1} = handles.output;

%--- Función que lee el valor del radio del cuerpo del robot y
    verifica
%que sea un número positivo.
function BodyRad_Callback(hObject, eventdata, handles)
body_rad=str2double(get(hObject,'string'));
if (isnan(body_rad)||sign(body_rad)~=1)
      errordlg('El radio del cuerpo del robot ha de ser un número
          positivo.','ERROR')
      body_rad=0;
      set(hObject,'string',body_rad);
end
handles.BodyRad=body_rad;
guidata(hObject,handles);


%--- Función que lee el valor de la altura del cuerpo del robot
    y verifica
%que sea un número positivo.
function BodyHeight_Callback(hObject, eventdata, handles)
body_height=str2double(get(hObject,'string'));
if (isnan(body_height)||sign(body_height)~=1)
      errordlg('La altura del cuerpo del robot ha de ser un
          número positivo.','ERROR')
      body_height=0;
      set(hObject,'string',body_height);
end
handles.BodyHeight=body_height;
guidata(hObject,handles);

%--- Función que lee el valor del radio de las ruedas y
    verifica
%que sea un número positivo.
function WheelsRad_Callback(hObject, eventdata, handles)
global wheels_rad
wheels_rad=str2double(get(hObject,'string'));
if (isnan(wheels_rad)||sign(wheels_rad)~=1)
      errordlg('El radio de las ruedas ha de ser un número
          positivo.','ERROR')
      wheels_rad=0;
      set(hObject,'string',wheels_rad);
end
handles.WheelsRad=wheels_rad;
```

```
77 guidata(hObject,handles);
78
79
80 %--- Función que lee el valor del ancho de las ruedas y
       verifica
81 %que sea un número positivo.
82 function WheelsWidth_Callback(hObject, eventdata, handles)
83 wheels_width=str2double(get(hObject,'string'));
84 if (isnan(wheels_width)||sign(wheels_width)~=1)
85       errordlg('El ancho de las ruedas ha de ser un número
             positivo.','ERROR')
86       wheels_width=0;
87       set(hObject,'string',wheels_width);
88 end
89 handles.WheelsWidth=wheels_width;
90 guidata(hObject,handles);
91
92
93 %--- Función que lee el valor de la distancia entre ejes y
       verifica
94 %que sea un número positivo.
95 function AxisDist_Callback(hObject, eventdata, handles)
96 axis_dist=str2double(get(hObject,'string'));
97 if (isnan(axis_dist)||sign(axis_dist)~=1)
98       errordlg('La distancia entre ejes ha de ser un número
             positivo.','ERROR')
99       axis_dist=0;
100      set(hObject,'string',axis_dist);
101 end
102 handles.AxisDist=axis_dist;
103 guidata(hObject,handles);
104
105
106 %--- Función que se ejecuta al pulsar el botón Create.
107 function Create_Callback(hObject, eventdata, handles)
108 global vrep
109 global clientID
110 global scene
111 global next
112
113 %Se realizan las comprobaciones necesarias en lo que respecta a
       las
114 %restricciones geométricas del robot.
115 if(handles.AxisDist>2*handles.BodyRad)
116     errordlg('La distancia entre ejes no puede ser en ningún
```

```
        caso mayor que el diámetro del cuerpo del robot.','ERROR'
        )
117
118 elseif(sqrt((handles.AxisDist/2+handles.WheelsWidth/2)^2+(
    handles.WheelsRad)^2)>handles.BodyRad)
119     errordlg('Las ruedas sobresalen del cuerpo del robot. Pruebe
         a reducir la distancia entre ejes, el radio o el ancho
        de las ruedas o aumente el radio del cuerpo del robot.','
        ERROR')
120
121 elseif(handles.AxisDist<=handles.WheelsWidth)
122     errordlg('Las ruedas se superponen. Pruebe a aumentar la
        distancia entre ejes o reduzca el ancho de las ruedas.','
        ERROR')
123
124 else
125     %Se calculan las dimensiones, la masa y la posición del
        cuerpo del
126     %robot a partir de los datos introducidos por el usuario.
127     body_size=[2*handles.BodyRad,2*handles.BodyRad,handles.
        BodyHeight];
128     body_density=100;
129     body_mass=body_density*pi*handles.BodyRad^2*handles.
        BodyHeight;
130     body_pos=[0,0,2*handles.WheelsRad+handles.BodyHeight/2];
131
132     %Se calculan las dimensiones, la masa y la posición de las
        ruedas
133     %a partir de los datos introducidos por el usuario.
134     wheels_size=[2*handles.WheelsRad,2*handles.WheelsRad,handles
        .WheelsWidth];
135     wheels_density=500;
136     wheels_mass=wheels_density*pi*handles.WheelsRad^2*handles.
        WheelsWidth;
137     left_wheel_pos=[-handles.AxisDist/2,0,handles.WheelsRad];
138     right_wheel_pos=[handles.AxisDist/2,0,handles.WheelsRad];
139
140     %Se calculan el rango y la posición del sensor de
        proximidad
141     %a partir de los datos introducidos por el usuario.
142     prox_sensor_range=handles.BodyRad/cos(pi/4);
143     prox_sensor_pos=[0,handles.BodyRad,2*handles.WheelsRad+
        handles.BodyHeight/2];
144
145     %Se calcula la posición de los sensores de fuerza
```

```
146     %a partir de los datos introducidos por el usuario.
147     front_force_sensor_pos=[0,handles.BodyRad-handles.WheelsRad
            ,2*handles.WheelsRad];
148     back_force_sensor_pos=[0,-handles.BodyRad+handles.WheelsRad
            ,2*handles.WheelsRad];
149
150     %Se calculan las dimensiones, la masa y la posición de las
            esferas
151     %deslizantes a partir de los datos introducidos por el
            usuario.
152     sliders_size=[2*handles.WheelsRad,2*handles.WheelsRad,2*
            handles.WheelsRad];
153     sliders_mass=0.01;
154     front_slider_pos=[0,handles.BodyRad-handles.WheelsRad,
            handles.WheelsRad];
155     back_slider_pos=[0,-handles.BodyRad+handles.WheelsRad,
            handles.WheelsRad];
156
157     %Se calculan el 'far clipping plane' y la posición de los
            sensores de
158     %visión a partir de los datos introducidos por el usuario.
159     far_clipping_plane=3*handles.WheelsRad;
160     left_vision_sensor_pos=[-0.1,sqrt(handles.BodyRad^2-0.1^2)
            +0.005,2*handles.WheelsRad];
161     right_vision_sensor_pos=[0.1,sqrt(handles.BodyRad^2-0.1^2)
            +0.005,2*handles.WheelsRad];
162     middle_vision_sensor_pos=[0,handles.BodyRad,2*handles.
            WheelsRad+handles.BodyHeight/2];
163
164     vrep.simxStartSimulation(clientID,vrep.simx_opmode_oneshot);
165
166     %Se llama a la función 'CreateDiff' del script Lua 'create'
167     %encargada de crear el robot de modelo diferencial en la
            escena.
168     vrep.simxCallScriptFunction(clientID,'create',vrep.
            sim_scripttype_childscript,'CreateDiff_function',2,...
169         [body_size,body_mass,body_pos,wheels_size,wheels_mass,
                left_wheel_pos,right_wheel_pos,prox_sensor_range,...
170         prox_sensor_pos,front_force_sensor_pos,
                back_force_sensor_pos,sliders_size,sliders_mass,
                front_slider_pos,...
171         back_slider_pos,far_clipping_plane,
                left_vision_sensor_pos,right_vision_sensor_pos,
                middle_vision_sensor_pos],...
172         [],[],vrep.simx_opmode_blocking);
```

```
173
174     %Se cierra la ventana y se continúa con la de generación de
175     %obstáculos si la escena en la que se trabaja es 'Obstacles'
176     %'Circuit' o con la selección del ítem a añadir en cualquier otro caso.
177     next=1;
178     close CreaDiffCir
179     if(scene==1 || scene==2)
180         CreaObs
181     else
182         SelectItem
183     end
184 end
185
186
187 %--- Función que se ejecuta al pulsar el botón Back.
188 function Back_Callback(hObject, eventdata, handles)
189 close CreaDiffCir
190 SelectRobot
191
192
193 %--- Función que se ejecuta al pulsar el botón Cancel.
194 function Cancel_Callback(hObject, eventdata, handles)
195 global vrep
196 global clientID
197
198 vrep.simxCloseScene(clientID,vrep.simx_opmode_blocking);
199 vrep.simxGetPingTime(clientID);
200 vrep.simxFinish(clientID);
201 close CreaDiffCir
202
203
204 %--- Función que se ejecuta cuando se intenta cerrar la ventana.
205 function figure1_CloseRequestFcn(hObject, eventdata, handles)
206 global next
207
208 %Si la petición de cierre de la ventana no es consecuencia del normal tránsito del programa de una ventana a otra
209 %sino que deriva del clic por parte del usuario en el icono de cierre, se exige una confirmación.
210 if next==0
211     opc=questdlg('¿Desea salir del programa?','Salir','Sí','No','No');
```

```
212         if strcmp(opc,'No')
213             return
214         end
215     end
216     next=0;
217     delete(hObject);
```

A.6. CreaDiffQuad.m

```
function varargout = CreaDiffQuad(varargin)
% Comienzo del código de inicialización - NO EDITAR
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @CreaDiffQuad_OpeningFcn, ...
                   'gui_OutputFcn',  @CreaDiffQuad_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% Fin del código de inicialización - NO EDITAR


%--- Se ejecuta justo antes de que CreaDiffQuad se haga visible.
function CreaDiffQuad_OpeningFcn(hObject, eventdata, handles, varargin)
% Se posiciona la ventana en la esquina superior derecha de la pantalla.
scrsz=get(0,'ScreenSize');
pos_act=get(gcf,'Position');
xr=scrsz(3)-pos_act(3);
xp=round(xr/2);
yr=scrsz(4)-pos_act(4);
yp=round(yr/2);
set(gcf,'position',[xp,yp,pos_act(3),pos_act(4)]);

handles.output = hObject;
guidata(hObject, handles);


%--- Las salidas de esta función se devuelven por la línea de
```

```
    comandos.
38 function varargout = CreaDiffQuad_OutputFcn(hObject, eventdata,
    handles)
39 varargout{1} = handles.output;
40
41
42 %--- Función que lee el valor del largo del cuerpo del robot y
    verifica
43 %que sea un número positivo.
44 function BodyLength_Callback(hObject, eventdata, handles)
45 body_length=str2double(get(hObject,'string'));
46 if (isnan(body_length)||sign(body_length)~=1)
47       errordlg('El largo del cuerpo del robot ha de ser un número
             positivo.','ERROR')
48       body_length=0;
49       set(hObject,'string',body_length);
50 end
51 handles.BodyLength=body_length;
52 guidata(hObject,handles);
53
54
55 %--- Función que lee el valor del ancho del cuerpo del robot y
    verifica
56 %que sea un número positivo.
57 function BodyWidth_Callback(hObject, eventdata, handles)
58 body_width=str2double(get(hObject,'string'));
59 if (isnan(body_width)||sign(body_width)~=1)
60       errordlg('El ancho del cuerpo del robot ha de ser un número
             positivo.','ERROR')
61       body_width=0;
62       set(hObject,'string',body_width);
63 end
64 handles.BodyWidth=body_width;
65 guidata(hObject,handles);
66
67
68 %--- Función que lee el valor de la altura del cuerpo del robot
     y verifica
69 %que sea un número positivo.
70 function BodyHeight_Callback(hObject, eventdata, handles)
71 body_height=str2double(get(hObject,'string'));
72 if (isnan(body_height)||sign(body_height)~=1)
73       errordlg('La altura del cuerpo del robot ha de ser un
            número positivo.','ERROR')
74       body_height=0;
```

```
75        set(hObject,'string',body_height);
76    end
77    handles.BodyHeight=body_height;
78    guidata(hObject,handles);
79
80
81    %--- Función que lee el valor del radio de las ruedas y
          verifica
82    %que sea un número positivo.
83    function WheelsRad_Callback(hObject, eventdata, handles)
84    global wheels_rad
85    wheels_rad=str2double(get(hObject,'string'));
86    if (isnan(wheels_rad)||sign(wheels_rad)~=1)
87        errordlg('El radio de las ruedas ha de ser un número
              positivo.','ERROR')
88        wheels_rad=0;
89        set(hObject,'string',wheels_rad);
90    end
91    handles.WheelsRad=wheels_rad;
92    guidata(hObject,handles);
93
94
95    %--- Función que lee el valor del ancho de las ruedas y
          verifica
96    %que sea un número positivo.
97    function WheelsWidth_Callback(hObject, eventdata, handles)
98    wheels_width=str2double(get(hObject,'string'));
99    if (isnan(wheels_width)||sign(wheels_width)~=1)
100       errordlg('El ancho de las ruedas ha de ser un número
              positivo.','ERROR')
101       wheels_width=0;
102       set(hObject,'string',wheels_width);
103   end
104   handles.WheelsWidth=wheels_width;
105   guidata(hObject,handles);
106
107
108   %--- Función que lee el valor de la distancia entre ejes y
          verifica
109   %que sea un número positivo.
110   function AxisDist_Callback(hObject, eventdata, handles)
111   axis_dist=str2double(get(hObject,'string'));
112   if (isnan(axis_dist)||sign(axis_dist)~=1)
113       errordlg('La distancia entre ejes ha de ser un número
              positivo.','ERROR')
```

```
114         axis_dist=0;
115         set(hObject,'string',axis_dist);
116 end
117 handles.AxisDist=axis_dist;
118 guidata(hObject,handles);
119
120
121 %--- Función que se ejecuta al pulsar el botón Create.
122 function Create_Callback(hObject, eventdata, handles)
123 global vrep
124 global clientID
125 global scene
126 global next
127
128 %Se realizan las comprobaciones necesarias en lo que respecta a
        las
129 %restricciones geométricas del robot.
130
131 if(handles.AxisDist+handles.WheelsWidth>handles.BodyWidth)
132     errordlg('Las ruedas sobresalen por los laterales del cuerpo
          del robot. Pruebe a reducir la distancia entre ejes o el
          ancho de las ruedas o aumente el ancho del cuerpo del
         robot.','ERROR')
133
134 elseif(handles.WheelsRad>handles.BodyLength/2)
135     errordlg('Las ruedas sobresalen por las partes delantera y
         trasera del cuerpo del robot. Pruebe a reducir el radio
         de las ruedas o aumente el largo del cuerpo del robot.','
         ERROR')
136
137 elseif(handles.AxisDist<=handles.WheelsWidth)
138     errordlg('Las ruedas se superponen. Pruebe a aumentar la
         distancia entre ejes o reduzca el ancho de las ruedas.','
         ERROR')
139
140 else
141     %Se calculan las dimensiones, la masa y la posición del
         cuerpo del
142     %robot a partir de los datos introducidos por el usuario.
143     body_size=[handles.BodyWidth,handles.BodyLength,handles.
         BodyHeight];
144     body_density=100;
145     body_mass=body_density*handles.BodyLength*handles.BodyWidth*
         handles.BodyHeight;
146     body_pos=[0,0,2*handles.WheelsRad+handles.BodyHeight/2];
```

```
147
148     %Se calculan las dimensiones, la masa y la posición de las
            ruedas
149     %a partir de los datos introducidos por el usuario.
150     wheels_size=[2*handles.WheelsRad,2*handles.WheelsRad,handles
            .WheelsWidth];
151     wheels_density=50;
152     wheels_mass=wheels_density*pi*handles.WheelsRad^2*handles.
            WheelsWidth;
153     left_wheel_pos=[-handles.AxisDist/2,0,handles.WheelsRad];
154     right_wheel_pos=[handles.AxisDist/2,0,handles.WheelsRad];
155
156     %Se calculan el rango y la posición del sensor de
            proximidad
157     %a partir de los datos introducidos por el usuario.
158     prox_sensor_range=(handles.BodyWidth/2)/cos(pi/4);
159     prox_sensor_pos=[0,handles.BodyLength/2,2*handles.WheelsRad+
            handles.BodyHeight/2];
160
161     %Se calcula la posición de los sensores de fuerza
162     %a partir de los datos introducidos por el usuario.
163     front_force_sensor_pos=[0,handles.BodyLength/2-handles.
            WheelsRad,2*handles.WheelsRad];
164     back_force_sensor_pos=[0,-handles.BodyLength/2+handles.
            WheelsRad,2*handles.WheelsRad];
165
166     %Se calculan las dimensiones, la masa y la posición de las
            esferas
167     %deslizantes a partir de los datos introducidos por el
            usuario.
168     sliders_size=[2*handles.WheelsRad,2*handles.WheelsRad,2*
            handles.WheelsRad];
169     sliders_mass=0.01;
170     front_slider_pos=[0,handles.BodyLength/2-handles.WheelsRad,
            handles.WheelsRad];
171     back_slider_pos=[0,-handles.BodyLength/2+handles.WheelsRad,
            handles.WheelsRad];
172
173     %Se calculan el 'far clipping plane' y la posición de los
            sensores de
174     %visión a partir de los datos introducidos por el usuario.
175     far_clipping_plane=3*handles.WheelsRad;
176     left_vision_sensor_pos=[-0.1,handles.BodyLength/2+0.005,2*
            handles.WheelsRad];
177     right_vision_sensor_pos=[0.1,handles.BodyLength/2+0.005,2*
```

```
        handles.WheelsRad];
178     middle_vision_sensor_pos=[0,handles.BodyLength/2,2*handles.
        WheelsRad+handles.BodyHeight/2];
179
180     vrep.simxStartSimulation(clientID,vrep.simx_opmode_oneshot);
181
182     %Se llama a la función 'CreateDiff' del script Lua 'create'
183     %encargada de crear el robot de modelo diferencial en la
        escena.
184     vrep.simxCallScriptFunction(clientID,'create',vrep.
        sim_scripttype_childscript,'CreateDiff_function',0,...
185         [body_size,body_mass,body_pos,wheels_size,wheels_mass,
            left_wheel_pos,right_wheel_pos,prox_sensor_range,...
186         prox_sensor_pos,front_force_sensor_pos,
            back_force_sensor_pos,sliders_size,sliders_mass,
            front_slider_pos,...
187         back_slider_pos,far_clipping_plane,
            left_vision_sensor_pos,right_vision_sensor_pos,
            middle_vision_sensor_pos],...
188         [],[],vrep.simx_opmode_blocking);
189
190     %Se cierra la ventana y se continúa con la de generación de
191     %obstáculos si la escena en la que se trabaja es 'Obstacles
        ' o
192     %'Circuit' o con la selección del ítem a añadir en
        cualquier otro caso.
193     next=1;
194     close CreaDiffQuad
195      if(scene==1 || scene==2)
196         CreaObs
197     else
198         SelectItem
199     end
200 end
201
202
203 %--- Función que se ejecuta al pulsar el botón Back.
204 function Back_Callback(hObject, eventdata, handles)
205 global next
206 next=1;
207 close CreaDiffQuad
208 SelectRobot
209
210
211 %--- Función que se ejecuta al pulsar el botón Cancel.
```

```
212 function Cancel_Callback(hObject, eventdata, handles)
213 global vrep
214 global clientID
215
216 vrep.simxCloseScene(clientID,vrep.simx_opmode_blocking);
217 vrep.simxGetPingTime(clientID);
218 vrep.simxFinish(clientID);
219 close CreaDiffQuad
220
221
222 %--- Función que se ejecuta cuando se intenta cerrar la ventana
    .
223 function figure1_CloseRequestFcn(hObject, eventdata, handles)
224 global next
225
226 %Si la petición de cierre de la ventana no es consecuencia del
    normal tránsito del programa de una ventana a otra
227 %sino que deriva del clic por parte del usuario en el icono de
    cierre, se exige una confirmación.
228 if next==0
229     opc=questdlg('¿Desea salir del programa?','Salir','Sí','No',
        'No');
230     if strcmp(opc,'No')
231         return
232     end
233 end
234 next=0;
235 delete(hObject);
```

A.7. CreaTricycleCir.m

```
function varargout = CreaTricycleCir(varargin)
% Comienzo del código de inicialización - NO EDITAR
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
    'gui_Singleton',  gui_Singleton, ...
    'gui_OpeningFcn', @CreaTricycleCir_OpeningFcn, ...
    'gui_OutputFcn',  @CreaTricycleCir_OutputFcn, ...
    'gui_LayoutFcn',  [] , ...
    'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:})
        ;
else
    gui_mainfcn(gui_State, varargin{:});
end
% Fin del código de inicialización - NO EDITAR


%--- Se ejecuta justo antes de que CreaTricycleCir se haga
    visible.
function CreaTricycleCir_OpeningFcn(hObject, eventdata, handles,
    varargin)
% Se posiciona la ventana en la esquina superior derecha de la
    pantalla.
scrsz=get(0,'ScreenSize');
pos_act=get(gcf,'Position');
xr=scrsz(3)-pos_act(3);
xp=round(xr/2);
yr=scrsz(4)-pos_act(4);
yp=round(yr/2);
set(gcf,'position',[xp,yp,pos_act(3),pos_act(4)]);

handles.output = hObject;
guidata(hObject, handles);


%--- Las salidas de esta función se devuelven por la línea de
    comandos.
function varargout = CreaTricycleCir_OutputFcn(hObject,
```

```
    eventdata, handles)
varargout{1} = handles.output;


%--- Función que lee el valor del radio del cuerpo del robot y
    verifica que sea un número positivo.
function BodyRad_Callback(hObject, eventdata, handles)
body_rad=str2double(get(hObject,'string'));
if (isnan(body_rad)||sign(body_rad)~=1)
    errordlg('El radio del cuerpo del robot ha de ser un número
        positivo.','ERROR')
    body_rad=0;
    set(hObject,'string',body_rad);
end
handles.BodyRad=body_rad;
guidata(hObject,handles);


%--- Función que lee el valor de la altura del cuerpo del robot
    y verifica que sea un número positivo.
function BodyHeight_Callback(hObject, eventdata, handles)
body_height=str2double(get(hObject,'string'));
if (isnan(body_height)||sign(body_height)~=1)
    errordlg('El alto del cuerpo del robot ha de ser un número
        positivo.','ERROR')
    body_height=0;
    set(hObject,'string',body_height);
end
handles.BodyHeight=body_height;
guidata(hObject,handles);


%--- Función que lee el valor del radio de las ruedas y
    verifica
%que sea un número positivo.
function WheelsRad_Callback(hObject, eventdata, handles)
global wheels_rad
wheels_rad=str2double(get(hObject,'string'));
if (isnan(wheels_rad)||sign(wheels_rad)~=1)
    errordlg('El radio de las ruedas ha de ser un número
        positivo.','ERROR')
    wheels_rad=0;
    set(hObject,'string',wheels_rad);
end
handles.WheelsRad=wheels_rad;
```

```
77 guidata(hObject,handles);
78
79
80 %--- Función que lee el valor del ancho de las ruedas y
       verifica
81 %que sea un número positivo.
82 function WheelsWidth_Callback(hObject, eventdata, handles)
83 wheels_width=str2double(get(hObject,'string'));
84 if (isnan(wheels_width)||sign(wheels_width)~=1)
85     errordlg('El ancho de las ruedas ha de ser un número
           positivo.','ERROR')
86     wheels_width=0;
87     set(hObject,'string',wheels_width);
88 end
89 handles.WheelsWidth=wheels_width;
90 guidata(hObject,handles);
91
92
93 %--- Función que lee el valor de la distancia entre los ejes
       izquierdo-derecho y verifica
94 %que sea un número positivo.
95 function AxisDistLR_Callback(hObject, eventdata, handles)
96 axis_dist_lr=str2double(get(hObject,'string'));
97 if (isnan(axis_dist_lr)||sign(axis_dist_lr)~=1)
98     errordlg('La distancia entre los ejes izquierdo y derecho ha
           de ser un número positivo.','ERROR')
99     axis_dist_lr=0;
100    set(hObject,'string',axis_dist_lr);
101 end
102 handles.AxisDistLR=axis_dist_lr;
103 guidata(hObject,handles);
104
105
106 %--- Función que lee el valor de la distancia entre los ejes
       delantero-trasero y verifica
107 %que sea un número positivo.
108 function AxisDistFR_Callback(hObject, eventdata, handles)
109 axis_dist_fr=str2double(get(hObject,'string'));
110 if (isnan(axis_dist_fr)||sign(axis_dist_fr)~=1)
111     errordlg('La distancia entre los ejes delantero y trasero ha
           de ser un número positivo.','ERROR')
112     axis_dist_fr=0;
113     set(hObject,'string',axis_dist_fr);
114 end
115 handles.AxisDistFR=axis_dist_fr;
```

```
116 guidata(hObject,handles);
117
118
119 %--- Función que se ejecuta al pulsar el botón Create.
120 function Create_Callback(hObject, eventdata, handles)
121 global vrep
122 global clientID
123 global scene
124 global next
125
126 %Se realizan las comprobaciones necesarias en lo que respecta a
        las
127 %restricciones geométricas del robot.
128 if(handles.AxisDistLR>2*handles.BodyRad)
129     errordlg('La distancia entre los ejes izquierdo y derecho no
            puede ser en ningún caso mayor que el diámetro del
           cuerpo del robot.','ERROR')
130
131 elseif(handles.AxisDistFR>2*handles.BodyRad)
132     errordlg('La distancia entre los ejes delantero y trasero no
            puede ser en ningún caso mayor que el diámetro del
           cuerpo del robot.','ERROR')
133
134 elseif(sqrt((handles.AxisDistLR/2+handles.WheelsWidth/2)^2+(
        handles.AxisDistFR/2+handles.WheelsRad)^2)>handles.BodyRad)
135     errordlg('Las ruedas traseras sobresalen del cuerpo del
           robot. Pruebe a reducir las distancias entre ejes, el
           radio o el ancho de las ruedas o aumente el radio del
           cuerpo del robot.','ERROR')
136
137 elseif(sqrt(cos(atan(handles.WheelsRad/(handles.WheelsWidth/2))+
        min(pi/6,atan((handles.WheelsWidth/2)/handles.WheelsRad)))...
138         *sqrt((handles.WheelsWidth/2)^2+handles.WheelsRad^2))
                ^2+...
139         (cos(atan((handles.WheelsWidth/2)/handles.WheelsRad)-min
                (pi/6,atan((handles.WheelsWidth/2)/handles.WheelsRad)
                ))...
140         *sqrt((handles.WheelsWidth/2)^2+handles.WheelsRad^2)+
                handles.AxisDistFR/2)^2>handles.BodyRad)
141     errordlg('La rueda delantera sobresale al girar del cuerpo
           del robot. Pruebe a reducir la distancia entre los ejes
           delantero y trasero, el radio o el ancho de las ruedas o
           aumente el radio del cuerpo del robot.','ERROR')
142
143 elseif(handles.AxisDistLR<=handles.WheelsWidth)
```

```
144     errordlg('Las ruedas traseras se superponen. Pruebe a
            aumentar la distancia entre los ejes izquierdo y derecho
            o reduzca el ancho de las ruedas.','ERROR')
145
146 elseif(handles.AxisDistFR-handles.WheelsRad<=...
147         sqrt((handles.WheelsWidth/2)^2+handles.WheelsRad^2)
                &&...
148         handles.AxisDistLR/2-handles.WheelsWidth/2<=...
149         sqrt((handles.WheelsWidth/2)^2+handles.WheelsRad^2)
                &&...
150         handles.AxisDistLR/2-handles.WheelsWidth/2<=...
151         cos(atan(handles.WheelsRad/(handles.WheelsWidth/2))-pi
                /6)*sqrt((handles.WheelsWidth/2)^2+handles.WheelsRad
                ^2))
152     errordlg('La rueda delantera se superpone al girar con las
            traseras. Pruebe a aumentar las distancias entre ejes o
            reduzca el radio o el ancho de las ruedas.','ERROR')
153
154 else
155     %Se calculan las dimensiones, la masa y la posición del
            cuerpo del
156     %robot a partir de los datos introducidos por el usuario.
157     body_size=[2*handles.BodyRad,2*handles.BodyRad,handles.
            BodyHeight];
158     body_density=100;
159     body_mass=body_density*pi*handles.BodyRad^2*handles.
            BodyHeight;
160     body_pos=[0,0,2*handles.WheelsRad+handles.BodyHeight/2];
161
162     %Se calculan las dimensiones, la masa y la posición de las
            ruedas
163     %a partir de los datos introducidos por el usuario.
164     wheels_size=[2*handles.WheelsRad,2*handles.WheelsRad,handles
            .WheelsWidth];
165     wheels_density=500;
166     wheels_mass=wheels_density*pi*handles.WheelsRad^2*handles.
            WheelsWidth;
167     left_wheel_pos=[-handles.AxisDistLR/2,-handles.AxisDistFR/2,
            handles.WheelsRad];
168     right_wheel_pos=[handles.AxisDistLR/2,-handles.AxisDistFR/2,
            handles.WheelsRad];
169     orient_wheel_pos=[0,handles.AxisDistFR/2,handles.WheelsRad];
170
171     %Se calculan el rango y la posición del sensor de
            proximidad
```

```
172     %a partir de los datos introducidos por el usuario.
173     prox_sensor_range=handles.BodyRad/cos(pi/4);
174     prox_sensor_pos=[0,handles.BodyRad,2*handles.WheelsRad+
            handles.BodyHeight/2];
175
176     %Se calculan el 'far clipping plane' y la posición de los
            sensores de
177     %visión a partir de los datos introducidos por el usuario.
178     far_clipping_plane=3*handles.WheelsRad;
179     left_vision_sensor_pos=[-0.1,sqrt(handles.BodyRad^2-0.1^2)
            +0.005,2*handles.WheelsRad];
180     right_vision_sensor_pos=[0.1,sqrt(handles.BodyRad^2-0.1^2)
            +0.005,2*handles.WheelsRad];
181     middle_vision_sensor_pos=[0,handles.BodyRad,2*handles.
            WheelsRad+handles.BodyHeight/2];
182
183     vrep.simxStartSimulation(clientID,vrep.simx_opmode_oneshot);
184
185     %Se llama a la función 'CreateTricycle' del script Lua '
            create'
186     %encargada de crear el robot de modelo triciclo en la
            escena.
187     vrep.simxCallScriptFunction(clientID,'create',vrep.
            sim_scripttype_childscript,'CreateTricycle_function'
            ,2,...
188         [body_size,body_mass,body_pos,wheels_size,wheels_mass,
                left_wheel_pos,right_wheel_pos,orient_wheel_pos,...
189         prox_sensor_range,prox_sensor_pos,far_clipping_plane,
                left_vision_sensor_pos,right_vision_sensor_pos,...
190         middle_vision_sensor_pos],[],[],vrep.
                simx_opmode_blocking);
191
192     %Se cierra la ventana y se continúa con la de generación de
193     %obstáculos si la escena en la que se trabaja es 'Obstacles
            ' o
194     %'Circuit' o con la selección del ítem a añadir en
            cualquier otro caso.
195     next=1;
196     close CreaTricycleCir
197     if(scene==1 || scene==2)
198         CreaObs
199     else
200         SelectItem
201     end
202 end
```

```
203
204 %--- Función que se ejecuta al pulsar el botón 'Back'.
205 function Back_Callback(hObject, eventdata, handles)
206 global next
207
208 next=1;
209 close CreaTricycleCir
210 SelectRobot
211
212
213 %--- Función que se ejecuta al pulsar el botón 'Cancel'.
214 function Cancel_Callback(hObject, eventdata, handles)
215 global vrep
216 global clientID
217
218 vrep.simxCloseScene(clientID,vrep.simx_opmode_blocking);
219 vrep.simxGetPingTime(clientID);
220 vrep.simxFinish(clientID);
221 close CreaTricycleCir
222
223 %--- Función que se ejecuta cuando se intenta cerrar la ventana
    .
224 function figure1_CloseRequestFcn(hObject, eventdata, handles)
225 global next
226
227 %Si la petición de cierre de la ventana no es consecuencia del
    normal tránsito del programa de una ventana a otra
228 %sino que deriva del clic por parte del usuario en el icono de
    cierre, se exige una confirmación.
229 if next==0
230     opc=questdlg('¿Desea salir del programa?','Salir','Sí','No',
        'No');
231     if strcmp(opc,'No')
232         return
233     end
234 end
235 next=0;
236 delete(hObject);
```

A.8. CreaTricycleQuad.m

```
function varargout = CreaTricycleQuad(varargin)
% Comienzo del código de inicialización - NO EDITAR
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
    'gui_Singleton',  gui_Singleton, ...
    'gui_OpeningFcn', @CreaTricycleQuad_OpeningFcn, ...
    'gui_OutputFcn',  @CreaTricycleQuad_OutputFcn, ...
    'gui_LayoutFcn',  [] , ...
    'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:})
        ;
else
    gui_mainfcn(gui_State, varargin{:});
end
% Fin del código de inicialización - NO EDITAR


%--- Se ejecuta justo antes de que CreaTricycleQuad se haga
    visible.
function CreaTricycleQuad_OpeningFcn(hObject, eventdata, handles
    , varargin)
% Se posiciona la ventana en la esquina superior derecha de la
    pantalla.
scrsz=get(0,'ScreenSize');
pos_act=get(gcf,'Position');
xr=scrsz(3)-pos_act(3);
xp=round(xr/2);
yr=scrsz(4)-pos_act(4);
yp=round(yr/2);
set(gcf,'position',[xp,yp,pos_act(3),pos_act(4)]);

handles.output = hObject;
guidata(hObject, handles);


%--- Las salidas de esta función se devuelven por la línea de
    comandos.
function varargout = CreaTricycleQuad_OutputFcn(hObject,
```

```
    eventdata, handles)
varargout{1} = handles.output;


%--- Función que lee el valor del largo del cuerpo del robot y
    verifica
%que sea un número positivo.
function BodyLength_Callback(hObject, eventdata, handles)
body_length=str2double(get(hObject,'string'));
if (isnan(body_length)||sign(body_length)~=1)
    errordlg('El largo del cuerpo del robot ha de ser un número
        positivo.','ERROR')
    body_length=0;
    set(hObject,'string',body_length);
end
handles.BodyLength=body_length;
guidata(hObject,handles);


%--- Función que lee el valor del ancho del cuerpo del robot y
    verifica
%que sea un número positivo.
function BodyWidth_Callback(hObject, eventdata, handles)
body_width=str2double(get(hObject,'string'));
if (isnan(body_width)||sign(body_width)~=1)
    errordlg('El ancho del cuerpo del robot ha de ser un número
        positivo.','ERROR')
    body_width=0;
    set(hObject,'string',body_width);
end
handles.BodyWidth=body_width;
guidata(hObject,handles);


%--- Función que lee el valor de la altura del cuerpo del robot
    y verifica
%que sea un número positivo.
function BodyHeight_Callback(hObject, eventdata, handles)
body_height=str2double(get(hObject,'string'));
if (isnan(body_height)||sign(body_height)~=1)
    errordlg('El alto del cuerpo del robot ha de ser un número
        positivo.','ERROR')
    body_height=0;
    set(hObject,'string',body_height);
end
```

```
77  handles.BodyHeight=body_height;
78  guidata(hObject,handles);
79
80
81  %--- Función que lee el valor del radio de las ruedas y
        verifica
82  %que sea un número positivo.
83  function WheelsRad_Callback(hObject, eventdata, handles)
84  global wheels_rad
85
86  wheels_rad=str2double(get(hObject,'string'));
87  if (isnan(wheels_rad)||sign(wheels_rad)~=1)
88      errordlg('El radio de las ruedas ha de ser un número
            positivo.','ERROR')
89      wheels_rad=0;
90      set(hObject,'string',wheels_rad);
91  end
92  handles.WheelsRad=wheels_rad;
93  guidata(hObject,handles);
94
95
96  %--- Función que lee el valor del ancho de las ruedas y
        verifica
97  %que sea un número positivo.
98  function WheelsWidth_Callback(hObject, eventdata, handles)
99  wheels_width=str2double(get(hObject,'string'));
100 if (isnan(wheels_width)||sign(wheels_width)~=1)
101     errordlg('El ancho de las ruedas ha de ser un número
            positivo.','ERROR')
102     wheels_width=0;
103     set(hObject,'string',wheels_width);
104 end
105 handles.WheelsWidth=wheels_width;
106 guidata(hObject,handles);
107
108
109 %--- Función que lee el valor de la distancia entre los ejes
        izquierdo-derecho y verifica
110 %que sea un número positivo.
111 function AxisDistLR_Callback(hObject, eventdata, handles)
112 axis_dist_lr=str2double(get(hObject,'string'));
113 if (isnan(axis_dist_lr)||sign(axis_dist_lr)~=1)
114     errordlg('La distancia entre los ejes izquierdo y derecho ha
            de ser un número positivo.','ERROR')
115     axis_dist_lr=0;
```

```
116     set(hObject,'string',axis_dist_lr);
117 end
118 handles.AxisDistLR=axis_dist_lr;
119 guidata(hObject,handles);
120
121
122 %--- Función que lee el valor de la distancia entre los ejes
    delantero-trasero y verifica
123 %que sea un número positivo.
124 function AxisDistFR_Callback(hObject, eventdata, handles)
125 axis_dist_fr=str2double(get(hObject,'string'));
126 if (isnan(axis_dist_fr)||sign(axis_dist_fr)~=1)
127     errordlg('La distancia entre los ejes delantero y trasero ha
        de ser un número positivo.','ERROR')
128     axis_dist_fr=0;
129     set(hObject,'string',axis_dist_fr);
130 end
131 handles.AxisDistFR=axis_dist_fr;
132 guidata(hObject,handles);
133
134
135 %--- Función que se ejecuta al pulsar el botón Create.
136 function Create_Callback(hObject, eventdata, handles)
137 global vrep
138 global clientID
139 global scene
140 global next
141
142 %Se realizan las comprobaciones necesarias en lo que respecta a
    las
143 %restricciones de diseño del robot.
144
145 if((handles.AxisDistLR+handles.WheelsWidth)>handles.BodyWidth)
146     errordlg('Las ruedas traseras sobresalen por los laterales
        del cuerpo del robot. Pruebe a reducir la distancias
        entre los ejes izquierdo y derecho o el ancho de las
        ruedas o aumente el ancho del cuerpo del robot.','ERROR')
147
148 elseif((handles.AxisDistFR/2+handles.WheelsRad)>handles.
    BodyLength/2)
149     errordlg('Las ruedas traseras sobresalen por la parte
        posterior del cuerpo del robot. Pruebe a reducir la
        distancia entre los ejes delantero y trasero o el radio
        de las ruedas o aumente el largo del cuerpo del robot.','
        ERROR')
```

```
150
151 elseif(cos(atan((handles.WheelsRad)/(handles.WheelsWidth/2))-min
    (pi/6,atan(handles.WheelsRad/(handles.WheelsWidth/2)))) ...
152         *sqrt((handles.WheelsWidth/2)^2+handles.WheelsRad^2)>
                handles.BodyWidth/2)
153     errordlg('La rueda delantera sobresale al girar por los
            laterales del cuerpo del robot. Pruebe a reducir el radio
            o el ancho de las ruedas o aumente el ancho del cuerpo
            del robot.','ERROR')
154
155 elseif(cos(atan((handles.WheelsWidth/2)/handles.WheelsRad)-min(
    pi/6,atan((handles.WheelsWidth/2)/handles.WheelsRad))) ...
156         *sqrt((handles.WheelsWidth/2)^2+handles.WheelsRad^2)+
                handles.AxisDistFR/2>handles.BodyLength/2)
157     errordlg('La rueda delantera sobresale al girar por la parte
            frontal del cuerpo del robot. Pruebe a reducir la
            distancia entre los ejes delantero y trasero, el radio o
            el ancho de las ruedas o aumente el largo del cuerpo del
            robot.','ERROR')
158
159 elseif(handles.AxisDistLR<=handles.WheelsWidth)
160     errordlg('Las ruedas traseras se superponen. Pruebe a
            aumentar la distancia entre los ejes izquierdo y derecho
            o reduzca el ancho de las ruedas.','ERROR')
161
162 elseif(handles.AxisDistFR-handles.WheelsRad<=...
163         sqrt((handles.WheelsWidth/2)^2+handles.WheelsRad^2)
                &&...
164         handles.AxisDistLR/2-handles.WheelsWidth/2<=...
165         sqrt((handles.WheelsWidth/2)^2+handles.WheelsRad^2)
                &&...
166         handles.AxisDistLR/2-handles.WheelsWidth/2<=...
167         cos(atan(handles.WheelsRad/(handles.WheelsWidth/2))-pi
                /6)*sqrt((handles.WheelsWidth/2)^2+handles.WheelsRad
                ^2))
168     errordlg('La rueda delantera se superpone al girar con las
            traseras. Pruebe a aumentar las distancias entre ejes o
            reduzca el radio o el ancho de las ruedas.','ERROR')
169
170
171 else
172     %Se calculan las dimensiones, la masa y la posición del
            cuerpo del
173     %robot a partir de los datos introducidos por el usuario.
174     body_size=[handles.BodyWidth,handles.BodyLength,handles.
```

```
        BodyHeight];
175     body_density=100;
176     body_mass=body_density*handles.BodyLength*handles.BodyWidth*
            handles.BodyHeight;
177     body_pos=[0,0,2*handles.WheelsRad+handles.BodyHeight/2];
178
179     %Se calculan las dimensiones, la masa y la posición de las
            ruedas
180     %a partir de los datos introducidos por el usuario.
181     wheels_size=[2*handles.WheelsRad,2*handles.WheelsRad,handles
            .WheelsWidth];
182     wheels_density=500;
183     wheels_mass=wheels_density*pi*handles.WheelsRad^2*handles.
            WheelsWidth;
184     left_wheel_pos=[-handles.AxisDistLR/2,-handles.AxisDistFR/2,
            handles.WheelsRad];
185     right_wheel_pos=[handles.AxisDistLR/2,-handles.AxisDistFR/2,
            handles.WheelsRad];
186     orient_wheel_pos=[0,handles.AxisDistFR/2,handles.WheelsRad];
187
188     %Se calculan el rango y la posición del sensor de
            proximidad
189     %a partir de los datos introducidos por el usuario.
190     prox_sensor_range=(handles.BodyWidth/2)/cos(pi/4);
191     prox_sensor_pos=[0,handles.BodyLength/2,2*handles.WheelsRad+
            handles.BodyHeight/2];
192
193     %Se calculan el 'far clipping plane' y la posición de los
            sensores de
194     %visión a partir de los datos introducidos por el usuario.
195     far_clipping_plane=3*handles.WheelsRad;
196     left_vision_sensor_pos=[-0.1,handles.BodyLength/2+0.005,2*
            handles.WheelsRad];
197     right_vision_sensor_pos=[0.1,handles.BodyLength/2+0.005,2*
            handles.WheelsRad];
198     middle_vision_sensor_pos=[0,handles.BodyLength/2,2*handles.
            WheelsRad+handles.BodyHeight/2];
199
200     vrep.simxStartSimulation(clientID,vrep.simx_opmode_oneshot);
201
202     %Se llama a la función 'CreateTricycle' del script Lua '
            create'
203     %encargada de crear el robot de modelo triciclo en la
            escena.
204     vrep.simxCallScriptFunction(clientID,'create',vrep.
```

```
        sim_scripttype_childscript,'CreateTricycle_function'
        ,0,...
205     [body_size,body_mass,body_pos,wheels_size,wheels_mass,
            left_wheel_pos,right_wheel_pos,orient_wheel_pos,...
206     prox_sensor_range,prox_sensor_pos,far_clipping_plane,
            left_vision_sensor_pos,right_vision_sensor_pos,...
207     middle_vision_sensor_pos],[],[],vrep.
            simx_opmode_blocking);
208
209     %Se cierra la ventana y se continúa con la de generación de
210     %obstáculos si la escena en la que se trabaja es 'Obstacles
            ' o
211     %'Circuit' o con la selección del ítem a añadir en
            cualquier otro caso.
212     next=1;
213     close CreaTricycleQuad
214     if(scene==1 || scene==2)
215         CreaObs
216     else
217         SelectItem
218     end
219 end
220
221
222 %--- Función que se ejecuta al pulsar el botón Back.
223 function Back_Callback(hObject, eventdata, handles)
224 global next
225
226 next=1;
227 close CreaTricycleQuad
228 SelectRobot
229
230
231 %--- Función que se ejecuta al pulsar el botón Cancel.
232 function Cancel_Callback(hObject, eventdata, handles)
233 global vrep
234 global clientID
235
236 vrep.simxCloseScene(clientID,vrep.simx_opmode_blocking);
237 vrep.simxGetPingTime(clientID);
238 vrep.simxFinish(clientID);
239 close CreaTricycleQuad
240
241 %--- Función que se ejecuta cuando se intenta cerrar la ventana
        .
```

```
242 function figure1_CloseRequestFcn(hObject, eventdata, handles)
243 global next
244
245 %Si la petición de cierre de la ventana no es consecuencia del
       normal tránsito del programa de una ventana a otra
246 %sino que deriva del clic por parte del usuario en el icono de
       cierre, se exige una confirmación.
247 if next==0
248     opc=questdlg('¿Desea salir del programa?','Salir','Sí','No',
          'No');
249     if strcmp(opc,'No')
250         return
251     end
252 end
253 next=0;
254 delete(hObject);
```

A.9. CreaQuadricycleCir.m

```
function varargout = CreaQuadricycleCir(varargin)
% Comienzo del código de inicialización - NO EDITAR
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn',
                      @CreaQuadricycleCir_OpeningFcn, ...
                   'gui_OutputFcn',
                      @CreaQuadricycleCir_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:})
        ;
else
    gui_mainfcn(gui_State, varargin{:});
end
% Fin del código de inicialización - NO EDITAR


%--- Se ejecuta justo antes de que CreaQuadricycleCir se haga
    visible.
function CreaQuadricycleCir_OpeningFcn(hObject, eventdata,
    handles, varargin)
% Se posiciona la ventana en la esquina superior derecha de la
    pantalla.
scrsz=get(0,'ScreenSize');
pos_act=get(gcf,'Position');
xr=scrsz(3)-pos_act(3);
xp=round(xr/2);
yr=scrsz(4)-pos_act(4);
yp=round(yr/2);
set(gcf,'position',[xp,yp,pos_act(3),pos_act(4)]);

handles.output = hObject;
guidata(hObject, handles);


%--- Las salidas de esta función se devuelven por la línea de
```

```
    comandos.
38 function varargout = CreaQuadricycleCir_OutputFcn(hObject,
    eventdata, handles)
39 varargout{1} = handles.output;
40
41
42 %--- Función que lee el valor del radio del cuerpo del robot y
    verifica
43 %que sea un número positivo.
44 function BodyRad_Callback(hObject, eventdata, handles)
45 body_rad=str2double(get(hObject,'string'));
46 if (isnan(body_rad)||sign(body_rad)~=1)
47       errordlg('El radio del cuerpo del robot ha de ser un número
             positivo.','ERROR')
48       body_rad=0;
49       set(hObject,'string',body_rad);
50 end
51 handles.BodyRad=body_rad;
52 guidata(hObject,handles);
53
54
55 %--- Función que lee el valor de la altura del cuerpo del robot
     y verifica
56 %que sea un número positivo.
57 function BodyHeight_Callback(hObject, eventdata, handles)
58 body_height=str2double(get(hObject,'string'));
59 if (isnan(body_height)||sign(body_height)~=1)
60       errordlg('El alto del cuerpo del robot ha de ser un número
            positivo.','ERROR')
61       body_height=0;
62       set(hObject,'string',body_height);
63 end
64 handles.BodyHeight=body_height;
65 guidata(hObject,handles);
66
67
68 %--- Función que lee el valor del radio de las ruedas y
    verifica
69 %que sea un número positivo.
70 function WheelsRad_Callback(hObject, eventdata, handles)
71 global wheels_rad
72 wheels_rad=str2double(get(hObject,'string'));
73 if (isnan(wheels_rad)||sign(wheels_rad)~=1)
74       errordlg('El radio de las ruedas ha de ser un número
            positivo.','ERROR')
```

```
75        wheels_rad=0;
76        set(hObject,'string',wheels_rad);
77    end
78    handles.WheelsRad=wheels_rad;
79    guidata(hObject,handles);
80
81
82    %--- Función que lee el valor del ancho de las ruedas y
          verifica
83    %que sea un número positivo.
84    function WheelsWidth_Callback(hObject, eventdata, handles)
85    wheels_width=str2double(get(hObject,'string'));
86    if (isnan(wheels_width)||sign(wheels_width)~=1)
87        errordlg('El ancho de las ruedas ha de ser un número
              positivo.','ERROR')
88        wheels_width=0;
89        set(hObject,'string',wheels_width);
90    end
91    handles.WheelsWidth=wheels_width;
92    guidata(hObject,handles);
93
94
95    %--- Función que lee el valor de la distancia entre los ejes
          izquierdo-derecho y verifica
96    %que sea un número positivo.
97    function AxisDistLR_Callback(hObject, eventdata, handles)
98    global axis_dist_lr
99    axis_dist_lr=str2double(get(hObject,'string'));
100   if (isnan(axis_dist_lr)||sign(axis_dist_lr)~=1)
101       errordlg('La distancia entre los ejes izquierdo y derecho
              ha de ser un número positivo.','ERROR')
102       axis_dist_lr=0;
103       set(hObject,'string',axis_dist_lr);
104   end
105   handles.AxisDistLR=axis_dist_lr;
106   guidata(hObject,handles);
107
108
109   %--- Función que lee el valor de la distancia entre los ejes
          delantero-trasero y verifica
110   %que sea un número positivo.
111   function AxisDistFR_Callback(hObject, eventdata, handles)
112   global axis_dist_fr
113   axis_dist_fr=str2double(get(hObject,'string'));
114   if (isnan(axis_dist_fr)||sign(axis_dist_fr)~=1)
```

```
115         errordlg('La distancia entre los ejes delantero y trasero
                ha de ser un número positivo.','ERROR')
116         axis_dist_fr=0;
117         set(hObject,'string',axis_dist_fr);
118 end
119 handles.AxisDistFR=axis_dist_fr;
120 guidata(hObject,handles);
121
122
123 %--- Función que se ejecuta al pulsar el botón Create.
124 function Create_Callback(hObject, eventdata, handles)
125 global vrep
126 global clientID
127 global scene
128 global next
129
130 %Se calcula el ángulo máximo que se asignará a la rueda
        directriz derecha para
131 %conseguir un giro de pi/6 del robot.
132 theta=-pi/6;
133 theta_right_max=acot(cot(theta)+(handles.AxisDistLR/2)/handles.
        AxisDistFR);
134
135 %Se realizan las comprobaciones necesarias en lo que respecta a
        las
136 %restricciones geométricas del robot.
137 if(handles.AxisDistLR>2*handles.BodyRad)
138     errordlg('La distancia entre los ejes izquierdo y derecho no
            puede ser en ningún caso mayor que el diámetro del
            cuerpo del robot.','ERROR')
139
140 elseif(handles.AxisDistFR>2*handles.BodyRad)
141     errordlg('La distancia entre los ejes delantero y trasero no
            puede ser en ningún caso mayor que el diámetro del
            cuerpo del robot.','ERROR')
142
143 elseif((handles.AxisDistLR/2+handles.WheelsWidth/2)^2+(handles.
        AxisDistFR/2+handles.WheelsRad)^2>handles.BodyRad^2)
144     errordlg('Las ruedas traseras sobresalen del cuerpo del
            robot. Pruebe a reducir las distancias entre ejes, el
            radio o el ancho de las ruedas o aumente el radio del
            cuerpo del robot.','ERROR')
145
146 elseif(sqrt(cos(atan(handles.WheelsRad/(handles.WheelsWidth/2)))-
        min(-theta_right_max,atan(handles.AxisDistLR/handles.
```

```
    AxisDistFR)-atan((handles.WheelsWidth/2)/handles.WheelsRad)))
    ...
147         *sqrt((handles.WheelsWidth/2)^2+handles.WheelsRad^2)+
                handles.AxisDistLR/2)^2+...
148         (cos(atan((handles.WheelsWidth/2)/handles.WheelsRad)+min
                (-theta_right_max,atan(handles.AxisDistLR/handles.
                AxisDistFR)-atan((handles.WheelsWidth/2)/handles.
                WheelsRad)))...
149         *sqrt((handles.WheelsWidth/2)^2+handles.WheelsRad^2)+
                handles.AxisDistFR/2)^2>handles.BodyRad)
150     errordlg('Las ruedas delanteras sobresalen al girar del
            cuerpo del robot. Pruebe a reducir las distancias entre
            ejes, el radio o el ancho de las ruedas o aumente el
            radio del cuerpo del robot.','ERROR')
151
152 elseif(handles.AxisDistLR<=handles.WheelsWidth)
153     errordlg('Las ruedas traseras se superponen. Pruebe a
            aumentar la distancia entre los ejes izquierdo y derecho
            o reduzca el ancho de las ruedas.','ERROR')
154
155 elseif(-cos(atan((handles.WheelsWidth/2)/handles.WheelsRad)-min
    (-theta_right_max,atan((handles.WheelsWidth/2)/handles.
    WheelsRad)))...
156         *sqrt((handles.WheelsWidth/2)^2+handles.WheelsRad^2)+
                handles.AxisDistFR/2<=-handles.AxisDistFR/2+handles.
                WheelsRad)
157     errordlg('Las ruedas delanteras se superponen al girar con
            las traseras. Pruebe a aumentar la distancias entre ejes
            o reduzca el radio o el ancho de las ruedas.','ERROR')
158
159 else
160     %Se calculan las dimensiones, la masa y la posición del
            cuerpo del
161     %robot a partir de los datos introducidos por el usuario.
162     body_size=[2*handles.BodyRad,2*handles.BodyRad,handles.
            BodyHeight];
163     body_density=100;
164     body_mass=body_density*pi*handles.BodyRad^2*handles.
            BodyHeight;
165     body_pos=[0,0,2*handles.WheelsRad+handles.BodyHeight/2];
166
167     %Se calculan las dimensiones, la masa y la posición de las
            ruedas
168     %a partir de los datos introducidos por el usuario.
169     wheels_size=[2*handles.WheelsRad,2*handles.WheelsRad,handles
```

```
        .WheelsWidth];
170     wheels_density=500;
171     wheels_mass=wheels_density*pi*handles.WheelsRad^2*handles.
            WheelsWidth;
172     rear_left_wheel_pos=[-handles.AxisDistLR/2,-handles.
            AxisDistFR/2,handles.WheelsRad];
173     rear_right_wheel_pos=[handles.AxisDistLR/2,-handles.
            AxisDistFR/2,handles.WheelsRad];
174     front_left_wheel_pos=[-handles.AxisDistLR/2,handles.
            AxisDistFR/2,handles.WheelsRad];
175     front_right_wheel_pos=[handles.AxisDistLR/2,handles.
            AxisDistFR/2,handles.WheelsRad];
176
177     %Se calculan el rango y la posición del sensor de
            proximidad
178     %a partir de los datos introducidos por el usuario.
179     prox_sensor_range=handles.BodyRad/cos(pi/4);
180     prox_sensor_pos=[0,handles.BodyRad,2*handles.WheelsRad+
            handles.BodyHeight/2];
181
182     %Se calculan el 'far clipping plane' y la posición de los
            sensores de
183     %visión a partir de los datos introducidos por el usuario.
184     far_clipping_plane=3*handles.WheelsRad;
185     left_vision_sensor_pos=[-0.1,sqrt(handles.BodyRad^2-0.1^2)
            +0.005,2*handles.WheelsRad];
186     right_vision_sensor_pos=[0.1,sqrt(handles.BodyRad^2-0.1^2)
            +0.005,2*handles.WheelsRad];
187     middle_vision_sensor_pos=[0,handles.BodyRad,2*handles.
            WheelsRad+handles.BodyHeight/2];
188
189     vrep.simxStartSimulation(clientID,vrep.simx_opmode_oneshot);
190
191     %Se llama a la función 'CreateQuadricycle' del script Lua '
            create'
192     %encargada de crear el robot de modelo cuatriciclo en la
            escena.
193     vrep.simxCallScriptFunction(clientID,'create',vrep.
            sim_scripttype_childscript,'CreateQuadricycle_function'
            ,2,...
194         [body_size,body_mass,body_pos,wheels_size,wheels_mass,
                rear_left_wheel_pos,rear_right_wheel_pos,...
195         front_left_wheel_pos,front_right_wheel_pos,
                prox_sensor_range,prox_sensor_pos,far_clipping_plane
                ,...
```

```
196         left_vision_sensor_pos,right_vision_sensor_pos,
                middle_vision_sensor_pos],[],[],vrep.
                simx_opmode_blocking);
197
198     %Se cierra la ventana y se continúa con la de generación de
199     %obstáculos si la escena en la que se trabaja es 'Obstacles
            ' o
200     %'Circuit' o con la selección del ítem a añadir en
            cualquier otro caso.
201     next=1;
202     close CreaQuadricycleCir
203     if(scene==1 || scene==2)
204         CreaObs
205     else
206         SelectItem
207     end
208 end
209
210
211 %--- Función que se ejecuta al pulsar el botón Back.
212 function Back_Callback(hObject, eventdata, handles)
213 global next
214 next=1;
215 close CreaQuadricycleCir
216 SelectRobot
217
218
219 %--- Función que se ejecuta al pulsar el botón Cancel.
220 function Cancel_Callback(hObject, eventdata, handles)
221 global vrep
222 global clientID
223
224 vrep.simxCloseScene(clientID,vrep.simx_opmode_blocking);
225 vrep.simxGetPingTime(clientID);
226 vrep.simxFinish(clientID);
227 close CreaQuadricycleCir
228
229
230 %--- Función que se ejecuta cuando se intenta cerrar la ventana
        .
231 function figure1_CloseRequestFcn(hObject, eventdata, handles)
232 global next
233
234 %Si la petición de cierre de la ventana no es consecuencia del
        normal tránsito del programa de una ventana a otra
```

```
235 %sino que deriva del clic por parte del usuario en el icono de
    cierre, se exige una confirmación.
236 if next==0
237     opc=questdlg('¿Desea salir del programa?','Salir','Sí','No',
        'No');
238     if strcmp(opc,'No')
239         return
240     end
241 end
242 next=0;
243 delete(hObject);
```

A.10. CreaQuadricycleQuad.m

```
function varargout = CreaQuadricycleQuad(varargin)
% Comienzo del código de inicialización - NO EDITAR
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn',
                      @CreaQuadricycleQuad_OpeningFcn, ...
                   'gui_OutputFcn',
                      @CreaQuadricycleQuad_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:})
        ;
else
    gui_mainfcn(gui_State, varargin{:});
end
% Fin del código de inicialización - NO EDITAR


%--- Se ejecuta justo antes de que CreaQuadricycleQuad se haga
    visible.
function CreaQuadricycleQuad_OpeningFcn(hObject, eventdata,
    handles, varargin)
% Se posiciona la ventana en la esquina superior derecha de la
    pantalla.
scrsz=get(0,'ScreenSize');
pos_act=get(gcf,'Position');
xr=scrsz(3)-pos_act(3);
xp=round(xr/2);
yr=scrsz(4)-pos_act(4);
yp=round(yr/2);
set(gcf,'position',[xp,yp,pos_act(3),pos_act(4)]);

handles.output = hObject;
guidata(hObject, handles);


%--- Las salidas de esta función se devuelven por la línea de
```

```
    comandos.
38 function varargout = CreaQuadricycleQuad_OutputFcn(hObject,
    eventdata, handles)
39 varargout{1} = handles.output;
40
41
42 %--- Función que lee el valor del largo del cuerpo del robot y
    verifica
43 %que sea un número positivo.
44 function BodyLength_Callback(hObject, eventdata, handles)
45 body_length=str2double(get(hObject,'string'));
46 if (isnan(body_length)||sign(body_length)~=1)
47       errordlg('El largo del cuerpo del robot ha de ser un número
             positivo.','ERROR')
48       body_length=0;
49       set(hObject,'string',body_length);
50 end
51 handles.BodyLength=body_length;
52 guidata(hObject,handles);
53
54
55 %--- Función que lee el valor del ancho del cuerpo del robot y
    verifica
56 %que sea un número positivo.
57 function BodyWidth_Callback(hObject, eventdata, handles)
58 body_width=str2double(get(hObject,'string'));
59 if (isnan(body_width)||sign(body_width)~=1)
60       errordlg('El ancho del cuerpo del robot ha de ser un número
             positivo.','ERROR')
61       body_width=0;
62       set(hObject,'string',body_width);
63 end
64 handles.BodyWidth=body_width;
65 guidata(hObject,handles);
66
67
68 %--- Función que lee el valor de la altura del cuerpo del robot
     y verifica
69 %que sea un número positivo.
70 function BodyHeight_Callback(hObject, eventdata, handles)
71 body_height=str2double(get(hObject,'string'));
72 if (isnan(body_height)||sign(body_height)~=1)
73       errordlg('El alto del cuerpo del robot ha de ser un número
            positivo.','ERROR')
74       body_height=0;
```

```
75        set(hObject,'string',body_height);
76    end
77    handles.BodyHeight=body_height;
78    guidata(hObject,handles);
79
80
81    %--- Función que lee el valor del radio de las ruedas y
          verifica
82    %que sea un número positivo.
83    function WheelsRad_Callback(hObject, eventdata, handles)
84    global wheels_rad
85    wheels_rad=str2double(get(hObject,'string'));
86    if (isnan(wheels_rad)||sign(wheels_rad)~=1)
87        errordlg('El radio de las ruedas ha de ser un número
              positivo.','ERROR')
88        wheels_rad=0;
89        set(hObject,'string',wheels_rad);
90    end
91    handles.WheelsRad=wheels_rad;
92    guidata(hObject,handles);
93
94
95    %--- Función que lee el valor del ancho de las ruedas y
          verifica
96    %que sea un número positivo.
97    function WheelsWidth_Callback(hObject, eventdata, handles)
98    wheels_width=str2double(get(hObject,'string'));
99    if (isnan(wheels_width)||sign(wheels_width)~=1)
100       errordlg('El ancho de las ruedas ha de ser un número
              positivo.','ERROR')
101       wheels_width=0;
102       set(hObject,'string',wheels_width);
103   end
104   handles.WheelsWidth=wheels_width;
105   guidata(hObject,handles);
106
107
108   %--- Función que lee el valor de la distancia entre los ejes
          izquierdo-derecho y verifica
109   %que sea un número positivo.
110   function AxisDistLR_Callback(hObject, eventdata, handles)
111   global axis_dist_lr
112   axis_dist_lr=str2double(get(hObject,'string'));
113   if (isnan(axis_dist_lr)||sign(axis_dist_lr)~=1)
114       errordlg('La distancia entre los ejes izquierdo y derecho
```

```
            ha de ser un número positivo.','ERROR')
115         axis_dist_lr=0;
116         set(hObject,'string',axis_dist_lr);
117 end
118 handles.AxisDistLR=axis_dist_lr;
119 guidata(hObject,handles);
120
121
122 %--- Función que lee el valor de la distancia entre los ejes
        delantero-trasero y verifica
123 %que sea un número positivo.
124 function AxisDistFR_Callback(hObject, eventdata, handles)
125 global axis_dist_fr
126 axis_dist_fr=str2double(get(hObject,'string'));
127 if (isnan(axis_dist_fr)||sign(axis_dist_fr)~=1)
128         errordlg('La distancia entre los ejes delantero y trasero
            ha de ser un número positivo.','ERROR')
129         axis_dist_fr=0;
130         set(hObject,'string',axis_dist_fr);
131 end
132 handles.AxisDistFR=axis_dist_fr;
133 guidata(hObject,handles);
134
135
136 %--- Función que se ejecuta al pulsar el botón Create.
137 function Create_Callback(hObject, eventdata, handles)
138 global vrep
139 global clientID
140 global scene
141 global next
142
143 %Se calcula el ángulos que se asignará a la rueda directriz
        derecha para
144 %conseguir un giro de pi/6 del robot.
145 theta=-pi/6;
146 theta_right_max=acot(cot(theta)+(handles.AxisDistLR/2)/handles.
        AxisDistFR);
147
148 %Se realizan las comprobaciones necesarias en lo que respecta a
        las
149 %restricciones geométricas del robot.
150 if(handles.AxisDistLR>handles.BodyWidth)
151     errordlg('La distancia entre los ejes izquierdo y derecho no
            puede ser en ningún caso mayor que el ancho del cuerpo
            del robot.','ERROR')
```

```
152
153 elseif(handles.AxisDistFR>2*handles.BodyLength)
154     errordlg('La distancia entre los ejes delantero y trasero no
            puede ser en ningún caso mayor que el largo del cuerpo
        del robot.','ERROR')
155
156 elseif((handles.AxisDistLR/2+handles.WheelsWidth/2)>handles.
    BodyWidth/2)
157     errordlg('Las ruedas traseras sobresalen por los laterales
        del cuerpo del robot. Pruebe a reducir la distancias
        entre los ejes izquierdo y derecho o el ancho de las
        ruedas o aumente el ancho del cuerpo del robot.','ERROR')
158
159 elseif((handles.AxisDistFR/2+handles.WheelsRad)>handles.
    BodyLength/2)
160     errordlg('Las ruedas traseras sobresalen por la parte
        posterior del cuerpo del robot. Pruebe a reducir la
        distancia entre los ejes delantero y trasero o el radio
        de las ruedas o aumente el largo del cuerpo del robot.','
        ERROR')
161
162 elseif(cos(atan(handles.WheelsRad/(handles.WheelsWidth/2))-min(-
    theta_right_max,atan(handles.WheelsRad/(handles.WheelsWidth
    /2))))...
163         *sqrt((handles.WheelsWidth/2)^2+handles.WheelsRad^2)+
            handles.AxisDistLR/2>handles.BodyWidth/2)
164     errordlg('Las ruedas delanteras sobresalen al girar por los
        laterales del cuerpo del robot. Pruebe a reducir la
        distancia entre los ejes izquierdo y derecho, el radio o
        el ancho de las ruedas o aumente el ancho del cuerpo del
        robot.','ERROR')
165
166 elseif(cos(atan((handles.WheelsWidth/2)/handles.WheelsRad)-min(-
    theta_right_max,atan((handles.WheelsWidth/2)/handles.
    WheelsRad)))...
167         *sqrt((handles.WheelsWidth/2)^2+handles.WheelsRad^2)+
            handles.AxisDistFR/2>handles.BodyLength/2)
168     errordlg('Las rueda delanteras sobresalen al girar por la
        parte frontal del cuerpo del robot. Pruebe a reducir la
        distancia entre los ejes delantero y trasero, el radio o
        el ancho de las ruedas o aumente el largo del cuerpo del
        robot.','ERROR')
169
170 elseif(handles.AxisDistLR<=handles.WheelsWidth)
171     errordlg('Las ruedas traseras se superponen. Pruebe a
```

```
        aumentar la distancia entre los ejes izquierdo y derecho
        o reduzca el ancho de las ruedas.','ERROR')
172
173 elseif(-cos(atan((handles.WheelsWidth/2)/handles.WheelsRad)-min
    (-theta_right_max,atan((handles.WheelsWidth/2)/handles.
    WheelsRad)))...
174         *sqrt((handles.WheelsWidth/2)^2+handles.WheelsRad^2)+
                handles.AxisDistFR/2<=-handles.AxisDistFR/2+handles.
                WheelsRad)
175     errordlg('Las ruedas delanteras se superponen al girar con
        las traseras. Pruebe a aumentar la distancias entre ejes
        o reduzca el radio o el ancho de las ruedas.','ERROR')
176
177
178
179 else
180     %Se calculan las dimensiones, la masa y la posición del
        cuerpo del
181     %robot a partir de los datos introducidos por el usuario.
182     body_size=[handles.BodyWidth,handles.BodyLength,handles.
        BodyHeight];
183     body_density=100;
184     body_mass=body_density*handles.BodyLength*handles.BodyWidth*
        handles.BodyHeight;
185     body_pos=[0,0,2*handles.WheelsRad+handles.BodyHeight/2];
186
187     %Se calculan las dimensiones, la masa y la posición de las
        ruedas
188     %a partir de los datos introducidos por el usuario.
189     wheels_size=[2*handles.WheelsRad,2*handles.WheelsRad,handles
        .WheelsWidth];
190     wheels_density=500;
191     wheels_mass=wheels_density*pi*handles.WheelsRad^2*handles.
        WheelsWidth;
192     rear_left_wheel_pos=[-handles.AxisDistLR/2,-handles.
        AxisDistFR/2,handles.WheelsRad];
193     rear_right_wheel_pos=[handles.AxisDistLR/2,-handles.
        AxisDistFR/2,handles.WheelsRad];
194     front_left_wheel_pos=[-handles.AxisDistLR/2,handles.
        AxisDistFR/2,handles.WheelsRad];
195     front_right_wheel_pos=[handles.AxisDistLR/2,handles.
        AxisDistFR/2,handles.WheelsRad];
196
197     %Se calculan el rango y la posición del sensor de
        proximidad
```

```
198     %a partir de los datos introducidos por el usuario.
199     prox_sensor_range=(handles.BodyWidth/2)/cos(pi/4);
200     prox_sensor_pos=[0,handles.BodyLength/2,2*handles.WheelsRad+
            handles.BodyHeight/2];
201
202     %Se calculan el 'far clipping plane' y la posición de los
            sensores de
203     %visión a partir de los datos introducidos por el usuario.
204     far_clipping_plane=3*handles.WheelsRad;
205     left_vision_sensor_pos=[-0.1,handles.BodyLength/2+0.005,2*
            handles.WheelsRad];
206     right_vision_sensor_pos=[0.1,handles.BodyLength/2+0.005,2*
            handles.WheelsRad];
207     middle_vision_sensor_pos=[0,handles.BodyLength/2,2*handles.
            WheelsRad+handles.BodyHeight/2];
208
209     vrep.simxStartSimulation(clientID,vrep.simx_opmode_oneshot);
210
211     %Se llama a la función 'CreateQuadricycle' del script Lua '
            create'
212     %encargada de crear el robot de modelo cuatriciclo en la
            escena.
213     vrep.simxCallScriptFunction(clientID,'create',vrep.
            sim_scripttype_childscript,'CreateQuadricycle_function'
            ,0,...
214         [body_size,body_mass,body_pos,wheels_size,wheels_mass,
                rear_left_wheel_pos,rear_right_wheel_pos,...
215         front_left_wheel_pos,front_right_wheel_pos,
                prox_sensor_range,prox_sensor_pos,far_clipping_plane
                ,...
216         left_vision_sensor_pos,right_vision_sensor_pos,
                middle_vision_sensor_pos],[],[],vrep.
                simx_opmode_blocking);
217
218     %Se cierra la ventana y se continúa con la de generación de
219     %obstáculos si la escena en la que se trabaja es 'Obstacles
            ' o
220     %'Circuit' o con la selección del ítem a añadir en
            cualquier otro caso.
221     next=1;
222     close CreaQuadricycleQuad
223      if(scene==1 || scene==2)
224         CreaObs
225     else
226         SelectItem
```

```
227     end
228 end
229
230
231 %--- Función que se ejecuta al pulsar el botón Back.
232 function Back_Callback(hObject, eventdata, handles)
233 global next
234 next=1;
235 close CreaQuadricycleQuad
236 SelectRobot
237
238
239 %--- Función que se ejecuta al pulsar el botón Cancel.
240 function Cancel_Callback(hObject, eventdata, handles)
241 global vrep
242 global clientID
243
244 vrep.simxCloseScene(clientID,vrep.simx_opmode_blocking);
245 vrep.simxGetPingTime(clientID);
246 vrep.simxFinish(clientID);
247 close CreaQuadricycleQuad
248
249
250 %--- Función que se ejecuta cuando se intenta cerrar la ventana
    .
251 function figure1_CloseRequestFcn(hObject, eventdata, handles)
252 global next
253
254 %Si la petición de cierre de la ventana no es consecuencia del
    normal tránsito del programa de una ventana a otra
255 %sino que deriva del clic por parte del usuario en el icono de
    cierre, se exige una confirmación.
256 if next==0
257     opc=questdlg('¿Desea salir del programa?','Salir','Sí','No',
        'No');
258     if strcmp(opc,'No')
259         return
260     end
261 end
262 next=0;
263 delete(hObject);
```

A.11. SelectItem.m

```
function varargout = SelectItem(varargin)
% Comienzo del código de inicialización - NO EDITAR
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @SelectItem_OpeningFcn, ...
                   'gui_OutputFcn',  @SelectItem_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:})
        ;
else
    gui_mainfcn(gui_State, varargin{:});
end
% Fin del código de inicialización - NO EDITAR


%--- Se ejecuta justo antes de que SelectItem se haga visible.
function SelectItem_OpeningFcn(hObject, eventdata, handles,
    varargin)
% Se posiciona la ventana en la esquina superior derecha de la
    pantalla.
scrsz=get(0,'ScreenSize');
pos_act=get(gcf,'Position');
xr=scrsz(3)-pos_act(3);
xp=round(xr/2);
yr=scrsz(4)-pos_act(4);
yp=round(yr/2);
set(gcf,'position',[xp,yp,pos_act(3),pos_act(4)]);

handles.output = hObject;
guidata(hObject, handles);


%--- Las salidas de esta función se devuelven por la línea de
    comandos.
function varargout = SelectItem_OutputFcn(hObject, eventdata,
    handles)
```

```
39 varargout{1} = handles.output;
40
41
42 %--- Función que se ejecuta al pulsar el botón Marker.
43 function Marker_Callback(hObject, eventdata, handles)
44 global next
45
46 next=1;
47 close SelectItem
48 PosMarker
49
50
51 %--- Función que se ejecuta al pulsar el botón Obstacle.
52 function Obstacle_Callback(hObject, eventdata, handles)
53 global next
54
55 next=1;
56 close SelectItem
57 CreaObs
58
59
60 %--- Función que se ejecuta al pulsar el botón Simulate.
61 function Simulate_Callback(hObject, eventdata, handles)
62 global next
63
64 next=1;
65 close SelectItem
66 Simulation
67
68
69 %--- Función que se ejecuta cuando se intenta cerrar la ventana
       .
70 function figure1_CloseRequestFcn(hObject, eventdata, handles)
71 global next
72
73 %Si la petición de cierre de la ventana no es consecuencia del
       normal tránsito del programa de una ventana a otra
74 %sino que deriva del clic por parte del usuario en el icono de
       cierre, se exige una confirmación.
75 if next==0
76     opc=questdlg('¿Desea salir del programa?','Salir','Sí','No',
           'No');
77     if strcmp(opc,'No')
78         return
79     end
```

```
80 end
81 next=0;
82 delete(hObject);
```

A.12. PosMarker.m

```
function varargout = PosMarker(varargin)
% Comienzo del código de inicialización - NO EDITAR
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @PosMarker_OpeningFcn, ...
                   'gui_OutputFcn',  @PosMarker_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:})
        ;
else
    gui_mainfcn(gui_State, varargin{:});
end
% Fin del código de inicialización - NO EDITAR


%--- Se ejecuta justo antes de que PosMarker se haga visible.
function PosMarker_OpeningFcn(hObject, eventdata, handles,
    varargin)
global vrep
global clientID
global num_marker
global marker

% Se posiciona la ventana en la esquina superior derecha de la
    pantalla.
scrsz=get(0,'ScreenSize');
pos_act=get(gcf,'Position');
xr=scrsz(3)-pos_act(3);
xp=round(xr/2);
yr=scrsz(4)-pos_act(4);
yp=round(yr/2);
set(gcf,'position',[xp,yp,pos_act(3),pos_act(4)]);

% Se muestra la imagen del icono asociado al desplazamiento de
    un objeto en
% V-REP.
```

```
40 axes(handles.axes1)
41 path=pwd;
42 image=imread([path,'\objectShiftButton.jpg']);
43 image=imresize(image,[25,NaN]);
44 axis off
45 imshow(image)
46
47 num_marker=num_marker+1;
48
49 %Se determina el nombre de la nueva baliza y su posición por
       defecto.
50 marker_name=strcat('marker',num2str(num_marker));
51 marker_pos=[0,1,0.25];
52
53 %Se llama a la función 'CreateMarker' del script Lua 'create'
54 %encargada de crear una baliza en la escena. Esta devuelve el
       identificador
55 %de la baliza que se añade al vector marker.
56 [~, retInts , ~, ~,]=vrep.simxCallScriptFunction(clientID,'
       create',vrep.sim_scripttype_childscript,...
57     'CreateMarker_function',[],marker_pos,marker_name,[],vrep.
           simx_opmode_blocking);
58 marker(num_marker)=retInts(1);
59
60 %Se fuerzan las coordenadas cartesianas de la baliza los
       valores por
61 %defecto.
62 set(handles.XPos,'string',0);
63 set(handles.YPos,'string',1);
64 set(handles.ZPos,'string',0.25);
65
66 handles.output = hObject;
67 guidata(hObject, handles);
68
69
70 %--- Las salidas de esta función se devuelven por la línea de
       comandos.
71 function varargout = PosMarker_OutputFcn(hObject, eventdata,
       handles)
72 varargout{1} = handles.output;
73
74
75 %--- Función que actualiza el valor de la coordenada x del
       obstáculo.
76 function XPos_Callback(hObject, eventdata, handles)
```

```
77  global vrep
78  global clientID
79  global marker
80  global num_marker
81
82  x_pos=str2double(get(hObject,'string'));
83  y_pos=str2double(get(handles.YPos,'string'));
84  z_pos=str2double(get(handles.ZPos,'string'));
85  guidata(hObject,handles);
86  marker_pos=[x_pos,y_pos,z_pos];
87
88  %Se llama a la función 'Position' del script Lua 'create'
89  %encargada de posicionar un objeto en la escena.
90  vrep.simxCallScriptFunction(clientID,'create',vrep.
        sim_scripttype_childscript,...
91  'Position_function',marker(num_marker),marker_pos,[],[],vrep.
        simx_opmode_blocking);
92
93
94
95  %--- Función que actualiza el valor de la coordenada y del
        obstáculo.
96  function YPos_Callback(hObject, eventdata, handles)
97  global vrep
98  global clientID
99  global marker
100 global num_marker
101
102 x_pos=str2double(get(handles.XPos,'string'));
103 y_pos=str2double(get(hObject,'string'));
104 z_pos=str2double(get(handles.ZPos,'string'));
105 guidata(hObject,handles);
106 marker_pos=[x_pos,y_pos,z_pos];
107
108 %Se llama a la función 'Position' del script Lua 'create'
109 %encargada de posicionar un objeto en la escena.
110 vrep.simxCallScriptFunction(clientID,'create',vrep.
        sim_scripttype_childscript,...
111 'Position_function',marker(num_marker),marker_pos,[],[],vrep.
        simx_opmode_blocking);
112
113
114
115 %--- Función que actualiza el valor de la coordenada z del
        obstáculo.
```

```
116 function ZPos_Callback(hObject, eventdata, handles)
117 global vrep
118 global clientID
119 global marker
120 global num_marker
121
122 x_pos=str2double(get(handles.XPos,'string'));
123 y_pos=str2double(get(handles.YPos,'string'));
124 z_pos=str2double(get(hObject,'string'));
125 guidata(hObject,handles);
126 marker_pos=[x_pos,y_pos,z_pos];
127
128 %Se llama a la función 'Position' del script Lua 'create'
129 %encargada de posicionar un objeto en la escena.
130 vrep.simxCallScriptFunction(clientID,'create',vrep.
    sim_scripttype_childscript,...
131 'Position_function',marker(num_marker),marker_pos,[],[],vrep.
    simx_opmode_blocking);
132
133
134
135 %--- Función que se ejecuta al pulsar el botón NewMarker.
136 function NewMarker_Callback(hObject, eventdata, handles)
137 global vrep
138 global clientID
139 global marker
140 global num_marker
141
142 num_marker=num_marker+1;
143
144 %Se determina el nombre de la nueva baliza y su posición por
    defecto.
145 marker_name=strcat('marker',num2str(num_marker));
146 marker_pos=[0,1,0.25];
147
148 %Se llama a la función 'CreateMarker' del script Lua 'create'
149 %encargada de crear una baliza en la escena. Esta devuelve el
    identificador
150 %de la baliza que se añade al vector marker.
151 [~, retInts, ~, ~,]=vrep.simxCallScriptFunction(clientID,'
    create',vrep.sim_scripttype_childscript,...
152     'CreateMarker_function',[],marker_pos,marker_name,[],vrep.
        simx_opmode_blocking);
153 marker(num_marker)=retInts(1);
154
```

```
155 %Se fuerzan las coordenadas cartesianas de la baliza los
        valores por
156 %defecto.
157 set(handles.XPos,'string',marker_pos(1));
158 set(handles.YPos,'string',marker_pos(2));
159 set(handles.ZPos,'string',marker_pos(3));
160
161 %--- Función que se ejecuta al pulsar el botón Simulate.
162 function Simulate_Callback(hObject, eventdata, handles)
163 global next
164
165 next=1;
166 close PosMarker
167 Simulation;
168
169
170 %--- Función que se ejecuta al pulsar el botón Obstacle.
171 function Obstacle_Callback(hObject, eventdata, handles)
172 global next
173
174 next=1;
175 close PosMarker
176 CreaObs
177
178 %--- Función que se ejecuta al pulsar el botón Update encargada
        de
179 %actualizar el valor de las coordenadas de la baliza tras
        haberla arrastrado por la ventana de V-REP.
180 function Update_Callback(hObject, eventdata, handles)
181 global vrep
182 global clientID
183 global marker
184 global num_marker
185
186 %Se realiza una primera lectura de la posición de la baliza
187 %cuyo resultado, por motivos del funcionamiento de la
188 %API, es desechado.
189 [~,marker_pos]=vrep.simxGetObjectPosition(clientID,marker(
        num_marker),-1,vrep.simx_opmode_streaming);
190
191 vrep.simxGetPingTime(clientID);
192 vrep.simxSynchronousTrigger(clientID);
193
194 [~,marker_pos]=vrep.simxGetObjectPosition(clientID,marker(
        num_marker),-1,vrep.simx_opmode_buffer);
```

```
195
196 set(handles.XPos,'string',marker_pos(1));
197 set(handles.YPos,'string',marker_pos(2));
198 set(handles.ZPos,'string',marker_pos(3));
199
200 guidata(hObject, handles);
201
202
203 %--- Función que se ejecuta al pulsar el botón Cancel.
204 function Cancel_Callback(hObject, eventdata, handles)
205 global vrep
206 global clientID
207
208 vrep.simxStopSimulation(clientID,vrep.simx_opmode_oneshot);
209 vrep.simxGetPingTime(clientID);
210 vrep.simxFinish(clientID);
211 close PosMarker
212
213 %--- Función que se ejecuta cuando se intenta cerrar la ventana
    .
214 function figure1_CloseRequestFcn(hObject, eventdata, handles)
215 global next
216
217 % Si la petición de cierre de la ventana no es consecuencia del
    normal tránsito del programa de una ventana a otra
218 % sino que deriva del clic por parte del usuario en el icono de
    cierre, se exige una confirmación.
219 if next==0
220     opc=questdlg('¿Desea salir del programa?','Salir','Sí','No',
        'No');
221     if strcmp(opc,'No')
222         return
223     end
224 end
225 next=0;
226 delete(hObject);
```

A.13. CreaObs.m

```
function varargout = CreaObs(varargin)
% Comienzo del código de inicialización - NO EDITAR
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @CreaObs_OpeningFcn, ...
                   'gui_OutputFcn',  @CreaObs_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:})
        ;
else
    gui_mainfcn(gui_State, varargin{:});
end
% Fin del código de inicialización - NO EDITAR


%--- Se ejecuta justo antes de que CreaObs se haga visible.
function CreaObs_OpeningFcn(hObject, eventdata, handles,
    varargin)
% Se posiciona la ventana en la esquina superior derecha de la
    pantalla.
scrsz=get(0,'ScreenSize');
pos_act=get(gcf,'Position');
xr=scrsz(3)-pos_act(3);
xp=round(xr/2);
yr=scrsz(4)-pos_act(4);
yp=round(yr/2);
set(gcf,'position',[xp,yp,pos_act(3),pos_act(4)]);

handles.output = hObject;
guidata(hObject, handles);


%--- Las salidas de esta función se devuelven por la línea de
    comandos.
function varargout = CreaObs_OutputFcn(hObject, eventdata,
    handles)
```

```
39 varargout{1} = handles.output;
40
41
42 %--- Función que se ejecuta al cambiar la selección de Type.
43 function Type_Callback(hObject, eventdata, handles)
44
45
46 %--- Función que lee el valor del lado o radio, según
       corresponda, de la base del obstáculo y verifica
47 %que sea un número positivo.
48 function SideRad_Callback(hObject, eventdata, handles)
49 side_rad=str2double(get(hObject,'string'));
50 if (isnan(side_rad)||sign(side_rad)~=1)
51       errordlg('El lado/radio del obstáculo ha de ser un número
             positivo.','ERROR')
52       side_rad=0;
53       set(hObject,'string',side_rad);
54 end
55 handles.SideRad=side_rad;
56 guidata(hObject,handles);
57
58
59 %--- Función que lee el valor de la altura del obstáculo y
       verifica
60 %que sea un número positivo.
61 function Height_Callback(hObject, eventdata, handles)
62 height=str2double(get(hObject,'string'));
63 if (isnan(height)||sign(height)~=1)
64       errordlg('La altura del obstáculo ha de ser un número
             positivo.','ERROR')
65       height=0;
66       set(hObject,'string',height);
67 end
68 handles.Height=height;
69 guidata(hObject,handles);
70
71
72 %--- Función que se ejecuta al pulsar el botón Create.
73 function Create_Callback(hObject, eventdata, handles)
74 global vrep
75 global clientID
76 global next
77 global obstacle
78 global obs_pos
79 global num_obs
```

```
80
81 num_obs=num_obs+1;
82
83 %Se determina el tipo de obstáculo, sus dimensiones y su
       posición por
84 %defecto a partir de los datos introducidos por el usuario. Se
       define
85 %asimismo su masa y su nombre.
86 sel=get(handles.Type,'value');
87 switch sel
88     case 1
89         obs_type=0;
90         obs_size=[handles.SideRad,handles.SideRad,handles.Height
               ];
91     case 2
92         obs_type=2;
93         obs_size=[2*handles.SideRad,2*handles.SideRad,handles.
               Height];
94     case 3
95         obs_type=3;
96         obs_size=[2*handles.SideRad,2*handles.SideRad,handles.
               Height];
97
98 end
99 obs_pos=[0.0,1.0,handles.Height/2];
100 obs_mass=1.0;
101 obs_name=strcat('obstacle',num2str(num_obs));
102
103 %Se llama a la función 'CreateObstacle' del script Lua 'create'
104 %encargada de crear un obstáculo en la escena. Esta devuelve el
105 %identificador del obstáculo que se añade al vector obstacle.
106 [~, retInts , ~, ~,]=vrep.simxCallScriptFunction(clientID,'
       create',vrep.sim_scripttype_childscript,...
107     'CreateObstacle_function',obs_type,[obs_size,obs_pos,
           obs_mass],obs_name,[],vrep.simx_opmode_blocking);
108
109 obstacle(num_obs)=retInts(1);
110
111 vrep.simxSynchronousTrigger(clientID);
112 next=1;
113 close CreaObs
114 PosObs;
115
116
117 %--- Función que se ejecuta al pulsar el botón Cancel.
```

```
118 function Cancel_Callback(hObject, eventdata, handles)
119 global vrep
120 global clientID
121
122 vrep.simxStopSimulation(clientID,vrep.simx_opmode_oneshot);
123 vrep.simxCloseScene(clientID,vrep.simx_opmode_blocking);
124 vrep.simxGetPingTime(clientID);
125 vrep.simxFinish(clientID);
126 close CreaObs
127
128
129 %--- Función que se ejecuta al pulsar el botón Skip.
130 function Skip_Callback(hObject, eventdata, handles)
131 global next
132
133 next=1;
134 close CreaObs
135 Simulation
136
137 %--- Función que se ejecuta cuando se intenta cerrar la ventana
    .
138 function figure1_CloseRequestFcn(hObject, eventdata, handles)
139 global next
140
141 %Si la petición de cierre de la ventana no es consecuencia del
    normal tránsito del programa de una ventana a otra
142 %sino que deriva del clic por parte del usuario en el icono de
    cierre, se exige una confirmación.
143 if next==0
144     opc=questdlg('¿Desea salir del programa?','Salir','Sí','No',
        'No');
145     if strcmp(opc,'No')
146         return
147     end
148 end
149 next=0;
150 delete(hObject);
```

A.14. PosObs.m

```
function varargout = PosObs(varargin)
% Comienzo del código de inicialización - NO EDITAR
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @PosObs_OpeningFcn, ...
                   'gui_OutputFcn',  @PosObs_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:})
        ;
else
    gui_mainfcn(gui_State, varargin{:});
end
% Fin del código de inicialización - NO EDITAR


%--- Se ejecuta justo antes de que PosObs se haga visible.
function PosObs_OpeningFcn(hObject, eventdata, handles, varargin
    )
global obs_pos
% Se posiciona la ventana en la esquina superior derecha de la
    pantalla.
scrsz=get(0,'ScreenSize');
pos_act=get(gcf,'Position');
xr=scrsz(3)-pos_act(3);
xp=round(xr/2);
yr=scrsz(4)-pos_act(4);
yp=round(yr/2);
set(gcf,'position',[xp,yp,pos_act(3),pos_act(4)]);

% Se inicializan las coordenadas cartesianas del obstáculo a los
    valores por
% defecto determinados en la ventana anterior.
set(handles.XPos,'string',obs_pos(1));
set(handles.YPos,'string',obs_pos(2));
set(handles.ZPos,'string',obs_pos(3));

```

```
40 %Se muestra la imagen del icono asociado al desplazamiento de
       un objeto en
41 %V-REP.
42 axes(handles.axes1)
43 path=pwd;
44 image=imread([path,'\objectShiftButton.jpg']);
45 image=imresize(image,[25,NaN]);
46 axis off
47 imshow(image)
48
49 handles.output = hObject;
50 guidata(hObject, handles);
51
52
53 %--- Las salidas de esta función se devuelven por la línea de
       comandos.
54 function varargout = PosObs_OutputFcn(hObject, eventdata,
       handles)
55 varargout{1} = handles.output;
56
57
58 %--- Función que actualiza el valor de la coordenada x del
       obstáculo.
59 function XPos_Callback(hObject, eventdata, handles)
60 global vrep
61 global clientID
62 global obstacle
63 global num_obs
64
65 x_pos=str2double(get(hObject,'string'));
66 y_pos=str2double(get(handles.YPos,'string'));
67 z_pos=str2double(get(handles.ZPos,'string'));
68 guidata(hObject,handles);
69 obs_pos=[x_pos,y_pos,z_pos];
70
71 %Se llama a la función 'Position' del script Lua 'create'
72 %encargada de posicionar un objeto en la escena.
73 vrep.simxCallScriptFunction(clientID,'create',vrep.
       sim_scripttype_childscript,...
74 'Position_function',obstacle(num_obs),obs_pos,[],[],vrep.
       simx_opmode_blocking);
75
76
77
78 %--- Función que actualiza el valor de la coordenada y del
```

```
    obstáculo.
79  function YPos_Callback(hObject, eventdata, handles)
80  global vrep
81  global clientID
82  global obstacle
83  global num_obs
84
85  x_pos=str2double(get(handles.XPos,'string'));
86  y_pos=str2double(get(hObject,'string'));
87  z_pos=str2double(get(handles.ZPos,'string'));
88  guidata(hObject,handles);
89  obs_pos=[x_pos,y_pos,z_pos];
90
91  %Se llama a la función 'Position' del script Lua 'create'
92  %encargada de posicionar un objeto en la escena.
93  vrep.simxCallScriptFunction(clientID,'create',vrep.
        sim_scripttype_childscript,...
94  'Position_function',obstacle(num_obs),obs_pos,[],[],vrep.
        simx_opmode_blocking);
95
96
97
98  %--- Función que actualiza el valor de la coordenada z del
        obstáculo.
99  function ZPos_Callback(hObject, eventdata, handles)
100 global vrep
101 global clientID
102 global obstacle
103 global num_obs
104
105 x_pos=str2double(get(handles.XPos,'string'));
106 y_pos=str2double(get(handles.YPos,'string'));
107 z_pos=str2double(get(hObject,'string'));
108 guidata(hObject,handles);
109 obs_pos=[x_pos,y_pos,z_pos];
110
111 %Se llama a la función 'Position' del script Lua 'create'
112 %encargada de posicionar un objeto en la escena.
113 vrep.simxCallScriptFunction(clientID,'create',vrep.
        sim_scripttype_childscript,...
114 'Position_function',obstacle(num_obs),obs_pos,[],[],vrep.
        simx_opmode_blocking);
115
116
117 %--- Función que se ejecuta al pulsar el botón NewObstacle.
```

```
118 function NewObstacle_Callback(hObject, eventdata, handles)
119 global next
120
121 next=1;
122 close PosObs
123 CreaObs;
124
125
126 %--- Función que se ejecuta al pulsar el botón Simulate.
127 function Simulate_Callback(hObject, eventdata, handles)
128 global next
129
130 next=1;
131 close PosObs
132 Simulation;
133
134
135 %--- Función que se ejecuta al pulsar el botón Update encargada
        de actualizar el valor de las coordenadas del obstáculo tras
        haberlo arrastrado por la ventana de V-REP.
136 function Update_Callback(hObject, eventdata, handles)
137 global vrep
138 global clientID
139 global obstacle
140 global num_obs
141
142 %Se realiza una primera lectura de la posición del obstáculo
143 %cuyo resultado, por motivos del funcionamiento de la
144 %API, es desechado.
145 [~,obs_pos]=vrep.simxGetObjectPosition(clientID,obstacle(num_obs
        ),-1,vrep.simx_opmode_streaming);
146
147 vrep.simxSynchronousTrigger(clientID);
148 vrep.simxGetPingTime(clientID);
149
150 [~,obs_pos]=vrep.simxGetObjectPosition(clientID,obstacle(num_obs
        ),-1,vrep.simx_opmode_buffer);
151
152 set(handles.XPos,'string',obs_pos(1));
153 set(handles.YPos,'string',obs_pos(2));
154 set(handles.ZPos,'string',obs_pos(3));
155
156 guidata(hObject, handles);
157
158
```

```
159 %--- Función que se ejecuta al pulsar el botón Cancel.
160 function Cancel_Callback(hObject, eventdata, handles)
161 global vrep
162 global clientID
163
164 vrep.simxStopSimulation(clientID,vrep.simx_opmode_oneshot);
165 vrep.simxGetPingTime(clientID);
166 vrep.simxFinish(clientID);
167 close PosObs
168
169
170 %--- Función que se ejecuta cuando se intenta cerrar la ventana
    .
171 function figure1_CloseRequestFcn(hObject, eventdata, handles)
172 global next
173
174 % Si la petición de cierre de la ventana no es consecuencia del
    normal tránsito del programa de una ventana a otra
175 % sino que deriva del clic por parte del usuario en el icono de
    cierre, se exige una confirmación.
176 if next==0
177     opc=questdlg('¿Desea salir del programa?','Salir','Sí','No',
        'No');
178     if strcmp(opc,'No')
179         return
180     end
181 end
182 next=0;
183 delete(hObject);
```

A.15. Simulation.m

```
function varargout = Simulation(varargin)
% Comienzo del código de inicialización - NO EDITAR
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
    'gui_Singleton',  gui_Singleton, ...
    'gui_OpeningFcn', @Simulation_OpeningFcn, ...
    'gui_OutputFcn',  @Simulation_OutputFcn, ...
    'gui_LayoutFcn',  [] , ...
    'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:})
        ;
else
    gui_mainfcn(gui_State, varargin{:});
end
% Fin del código de inicialización - NO EDITAR


%--- Se ejecuta justo antes de que Simulation se haga visible.
function Simulation_OpeningFcn(hObject, eventdata, handles,
    varargin)
% Se posiciona la ventana en la esquina superior derecha de la
    pantalla.
scrsz=get(0,'ScreenSize');
pos_act=get(gcf,'Position');
xr=scrsz(3)-pos_act(3);
xp=round(xr/2);
yr=scrsz(4)-pos_act(4);
yp=round(yr/2);
set(gcf,'position',[xp,yp,pos_act(3),pos_act(4)]);

handles.output = hObject;
guidata(hObject, handles);


%--- Las salidas de esta función se devuelven por la línea de
    comandos.
function varargout = Simulation_OutputFcn(hObject, eventdata,
    handles)
```

```
39 varargout{1} = handles.output;
40
41
42 %--- Función que se ejecuta al pulsar el botón 'PlaySim'
       encargada de
43 % iniciar o retomar la simulación y del control del robot
       durante la misma.
44 function PlaySim_Callback(hObject, eventdata, handles)
45 global vrep
46 global clientID
47 global model
48 global scene
49 global w
50 global axis_dist_fr
51 global axis_dist_lr
52 global i
53
54 %Se inicializa la variable asociada al bucle de control.
55 i=1;
56
57 %Se inicializan las variables que ayudarán a definir las
       maniobras del
58 %robot.
59 back_until_time=0;
60 steer_until_time=0;
61 forward=0; %se pondrá a 1 cuando el robot se dirija en línea
       recta hacia una baliza
62 steer=1; %variará a 1 o -1 según el sentido de giro del robot
       deseado
63 in=0; %se pondrá a 1 cuando el robot haya alcanzado el circuito
64
65 %Si se ha escogido realizar la simulación en una escena vacía o
       en una
66 %cargada por el usuario, se pregunta por el comportamiento del
       robot
67 %deseado.
68 if (scene==4||scene==5)
69     opc=questdlg('Seleccione el comportamiento deseado del robot
           :','Comportamiento','Evitar obstáculos','Seguir circuito
           y evitar obstáculos','Dirigirse a las balizas y evitar
           obstáculos','Evitar obstáculos');
70     if strcmp(opc,'Evitar obstáculos')
71         scene=1;
72     elseif strcmp(opc,'Seguir circuito y evitar obstáculos')
73         scene=2;
```

```
74      elseif strcmp(opc,'Dirigirse a las balizas y evitar
            obstáculos')
75          scene=3;
76      end
77  end
78
79  vrep.simxStartSimulation(clientID,vrep.simx_opmode_oneshot);
80
81  %Se llama a la función 'Hide' del script Lua 'create'
82  %encargada de ocultar los sensores que no intervengan en el
        comportamiento del robot en la escena elegidaa.
83  vrep.simxCallScriptFunction(clientID,'create',vrep.
        sim_scripttype_childscript,'Hide_function',scene,...
84      [],[],[],vrep.simx_opmode_blocking);
85
86  %Se define el comportamiento del robot según la escena y el
        modelo
87  %elegidos.
88  switch scene
89      case 1
90          switch model
91              case 1
92                  %--- Robot de tipo diferencial en escena con
                        obstáculos
93
94                  %Se obtienen los identificadores de los motores
                        y el sensor
95                  %de proximidad.
96                  [~,left_motor]=vrep.simxGetObjectHandle(clientID
                        ,'diff_left_motor',vrep.simx_opmode_blocking)
                        ;
97                  [~,right_motor]=vrep.simxGetObjectHandle(
                        clientID,'diff_right_motor',vrep.
                        simx_opmode_blocking);
98                  [~,prox_sensor]=vrep.simxGetObjectHandle(
                        clientID,'diff_prox_sensor',vrep.
                        simx_opmode_blocking);
99
100                 %Se realiza una primera lectura del sensor de
                        proximidad
101                 %cuyo resultado, por motivos del funcionamiento
                        de la
102                 %API, es desechado.
103                 [~,prox_detState,~,~,~]=vrep.
                        simxReadProximitySensor(clientID,prox_sensor,
```

```
                vrep.simx_opmode_streaming);
104
105             vrep.simxSynchronousTrigger(clientID);
106             vrep.simxGetPingTime(clientID);
107
108             %Se inicia el bucle de control del robot.
109             while(i==1)
110                 [~,prox_detState,~,~,~]=vrep.
                        simxReadProximitySensor(clientID,
                        prox_sensor,vrep.simx_opmode_buffer);
111
112                 current_time=vrep.simxGetLastCmdTime(
                        clientID);
113
114                 if (prox_detState==1)
115                     back_until_time=current_time+50/handles.
                            LinVel;
116                 end
117
118                 if(current_time<back_until_time)
119                     wl=-4*w/5;
120                     wr=-6*w/5;
121                     steer_until_time=back_until_time+50/
                            handles.LinVel;
122
123                 elseif(current_time<steer_until_time)
124                     wl=6*w/5;
125                     wr=4*w/5;
126
127                 else
128                     wl=w;
129                     wr=w;
130
131                 end
132                 vrep.simxSetJointTargetVelocity(clientID,
                        left_motor,wl,vrep.simx_opmode_oneshot);
133                 vrep.simxSetJointTargetVelocity(clientID,
                        right_motor,wr,vrep.simx_opmode_oneshot);
134
135                 vrep.simxSynchronousTrigger(clientID);
136
137                 pause(0.01)
138             end
139
140         case 2
```

```
141                 %--- Robot de tipo triciclo en escena con
                        obstáculos
142
143                 %Se obtienen los identificadores de los motores
                        y el sensor
144                 %de proximidad.
145                 [~,left_motor]=vrep.simxGetObjectHandle(clientID
                        ,'tricycle_left_motor',vrep.
                        simx_opmode_blocking);
146                 [~,right_motor]=vrep.simxGetObjectHandle(
                        clientID,'tricycle_right_motor',vrep.
                        simx_opmode_blocking);
147                 [~,front_motor]=vrep.simxGetObjectHandle(
                        clientID,'tricycle_front_motor',vrep.
                        simx_opmode_blocking);
148                 [~,prox_sensor]=vrep.simxGetObjectHandle(
                        clientID,'tricycle_prox_sensor',vrep.
                        simx_opmode_blocking);
149
150                 %Se realiza una primera lectura del sensor de
                        proximidad
151                 %cuyo resultado, por motivos del funcionamiento
                        de la
152                 %API, es desechado.
153                 [~,prox_detState,~,~,~]=vrep.
                        simxReadProximitySensor(clientID,prox_sensor,
                        vrep.simx_opmode_streaming);
154
155                 vrep.simxSynchronousTrigger(clientID);
156                 vrep.simxGetPingTime(clientID);
157
158                 %Se inicia el bucle de control del robot.
159                 while(i==1)
160                     [~,prox_detState,~,~,~]=vrep.
                            simxReadProximitySensor(clientID,
                            prox_sensor,vrep.simx_opmode_buffer);
161
162                     current_time=vrep.simxGetLastCmdTime(
                            clientID);
163
164                     if (prox_detState==1)
165                         back_until_time=current_time+200/handles
                                .LinVel;
166                     end
167
```

```
168                     if(current_time<back_until_time)
169                         wl=-w;
170                         wr=-w;
171                         theta=pi/6;
172                         steer_until_time=back_until_time+200/
                                handles.LinVel;
173
174                     elseif(current_time<steer_until_time)
175                         wl=w;
176                         wr=w;
177                         theta=-pi/6;
178
179                     else
180                         wl=w;
181                         wr=w;
182                         theta=0;
183                     end
184
185                     vrep.simxSetJointTargetVelocity(clientID,
                            left_motor,wl,vrep.simx_opmode_oneshot);
186                     vrep.simxSetJointTargetVelocity(clientID,
                            right_motor,wr,vrep.simx_opmode_oneshot);
187                     vrep.simxSetJointTargetPosition(clientID,
                            front_motor,theta,vrep.
                            simx_opmode_oneshot);
188
189                     vrep.simxSynchronousTrigger(clientID);
190                     pause(0.01)
191                 end
192
193             case 3
194                 %--- Robot de tipo cuatriciclo en escena con
                        obstáculos
195
196                 %Se obtienen los identificadores de los motores
                        y el sensor
197                 %de proximidad.
198                 [~,rear_left_motor]=vrep.simxGetObjectHandle(
                        clientID,'quadricycle_rear_left_motor',vrep.
                        simx_opmode_blocking);
199                 [~,rear_right_motor]=vrep.simxGetObjectHandle(
                        clientID,'quadricycle_rear_right_motor',vrep.
                        simx_opmode_blocking);
200                 [~,front_left_motor]=vrep.simxGetObjectHandle(
                        clientID,'quadricycle_front_left_motor',vrep.
```

```
201                 simx_opmode_blocking);
                    [~,front_right_motor]=vrep.simxGetObjectHandle(
                        clientID,'quadricycle_front_right_motor',vrep
                        .simx_opmode_blocking);
202                 [~,prox_sensor]=vrep.simxGetObjectHandle(
                        clientID,'quadricycle_prox_sensor',vrep.
                        simx_opmode_blocking);
203
204                 %Se realiza una primera lectura del sensor de
                        proximidad
205                 %cuyo resultado, por motivos del funcionamiento
                         de la
206                 %API, es desechado.
207                 [~,prox_detState,~,~,~]=vrep.
                        simxReadProximitySensor(clientID,prox_sensor,
                        vrep.simx_opmode_streaming);
208
209                 vrep.simxSynchronousTrigger(clientID);
210                 vrep.simxGetPingTime(clientID);
211
212                 %Se inicia el bucle de control del robot.
213                 while(i==1)
214                     [~,prox_detState,~,~,~]=vrep.
                            simxReadProximitySensor(clientID,
                            prox_sensor,vrep.simx_opmode_buffer);
215
216                     current_time=vrep.simxGetLastCmdTime(
                            clientID);
217
218                     if (prox_detState==1)
219                         back_until_time=current_time+200/handles
                                .LinVel;
220                     end
221
222                     if(current_time<back_until_time)
223                         wl=-w;
224                         wr=-w;
225                         theta=pi/6;
226                         steer_until_time=back_until_time+200/
                                handles.LinVel;
227
228                     elseif(current_time<steer_until_time)
229                         wl=w;
230                         wr=w;
231                         theta=-pi/6;
```

```
232
233                         else
234                             wl=w;
235                             wr=w;
236                             theta=0;
237                         end
238
239                         theta_left=acot(cot(theta)-(axis_dist_lr/2)/
                                axis_dist_fr);
240                         theta_right=acot(cot(theta)+(axis_dist_lr/2)
                                /axis_dist_fr);
241
242                         vrep.simxSetJointTargetVelocity(clientID,
                                rear_left_motor,wl,vrep.
                                simx_opmode_oneshot);
243                         vrep.simxSetJointTargetVelocity(clientID,
                                rear_right_motor,wr,vrep.
                                simx_opmode_oneshot);
244                         vrep.simxSetJointTargetPosition(clientID,
                                front_left_motor,theta_left,vrep.
                                simx_opmode_oneshot);
245                         vrep.simxSetJointTargetPosition(clientID,
                                front_right_motor,theta_right,vrep.
                                simx_opmode_oneshot);
246
247                         vrep.simxSynchronousTrigger(clientID);
248                         pause(0.01)
249                     end
250             end
251
252         case 2
253             switch model
254                 case 1
255                     %--- Robot de tipo diferencial en escena con
                            circuito
256
257                     %Se obtienen los identificadores de los motores
                            y los sensores de proximidad y visión
                            izquierdo y derecho.
258                     [~,left_motor]=vrep.simxGetObjectHandle(clientID
                            ,'diff_left_motor',vrep.simx_opmode_blocking)
                            ;
259                     [~,right_motor]=vrep.simxGetObjectHandle(
                            clientID,'diff_right_motor',vrep.
                            simx_opmode_blocking);
```

```
260                 [~,prox_sensor]=vrep.simxGetObjectHandle(
                        clientID,'diff_prox_sensor',vrep.
                        simx_opmode_blocking);
261                 [~,left_vision_sensor]=vrep.simxGetObjectHandle(
                        clientID,'diff_left_vision_sensor',vrep.
                        simx_opmode_blocking);
262                 [~,right_vision_sensor]=vrep.simxGetObjectHandle
                        (clientID,'diff_right_vision_sensor',vrep.
                        simx_opmode_blocking);
263
264                 %Se realiza una primera lectura de los sensores
265                 %cuyo resultado, por motivos del funcionamiento
                        de la
266                 %API, es desechado.
267                 [~,prox_detState,~,~,~]=vrep.
                        simxReadProximitySensor(clientID,prox_sensor,
                        vrep.simx_opmode_streaming);
268                 [~,~,left_vision_auxData,~]=vrep.
                        simxReadVisionSensor(clientID,
                        left_vision_sensor,vrep.simx_opmode_streaming
                        );
269                 [~,~,right_vision_auxData,~]=vrep.
                        simxReadVisionSensor(clientID,
                        right_vision_sensor,vrep.
                        simx_opmode_streaming);
270
271                 vrep.simxSynchronousTrigger(clientID);
272                 vrep.simxGetPingTime(clientID);
273
274                 %Se inicia el bucle de control del robot.
275                 while(i==1)
276                     [~,prox_detState,~,~,~]=vrep.
                            simxReadProximitySensor(clientID,
                            prox_sensor,vrep.simx_opmode_buffer);
277                     [~,~,left_vision_auxData,~]=vrep.
                            simxReadVisionSensor(clientID,
                            left_vision_sensor,vrep.
                            simx_opmode_buffer);
278                     [~,~,right_vision_auxData,~]=vrep.
                            simxReadVisionSensor(clientID,
                            right_vision_sensor,vrep.
                            simx_opmode_buffer);
279
280                     left_vision_read=(left_vision_auxData(11)
                            <0.3);
```

```
281                 right_vision_read=(right_vision_auxData(11)
                        <0.3);
282
283                 current_time=vrep.simxGetLastCmdTime(
                        clientID);
284
285
286                 if (prox_detState==1)
287                     back_until_time=current_time+50/handles.
                            LinVel;
288                 end
289
290                 if(current_time<back_until_time)
291                     if (steer==1)
292                         wl=-6*w/5;
293                         wr=-4*w/5;
294                     else
295                         wl=-4*w/5;
296                         wr=-6*w/5;
297                     end
298                     steer_until_time=back_until_time+50/
                            handles.LinVel;
299
300                 elseif(current_time<steer_until_time)
301                     if (steer==1)
302                         wl=4*w/5;
303                         wr=6*w/5;
304                     else
305                         wl=6*w/5;
306                         wr=4*w/5;
307                     end
308                 elseif(left_vision_read==1 &&
                        right_vision_read==0)
309                     steer=1;
310                     wl=4*w/5;
311                     wr=6*w/5;
312
313                 elseif(left_vision_read==0 &&
                        right_vision_read==1)
314                     steer=-1;
315                     wl=6*w/5;
316                     wr=4*w/5;
317
318                 elseif(left_vision_read==1 &&
                        right_vision_read==1)
```

```
319                         in=1;
320                         if (steer==1)
321                             wl=6*w/5;
322                             wr=4*w/5;
323                         else
324                             wl=4*w/5;
325                             wr=6*w/5;
326                         end
327
328                     elseif(in==0)
329                         wl=w;
330                         wr=w;
331                     end
332
333                     vrep.simxSetJointTargetVelocity(clientID,
                            left_motor,wl,vrep.simx_opmode_oneshot);
334                     vrep.simxSetJointTargetVelocity(clientID,
                            right_motor,wr,vrep.simx_opmode_oneshot);
335                     vrep.simxSynchronousTrigger(clientID);
336                     pause(0.01)
337                 end
338
339             case 2
340                 %--- Robot de tipo triciclo en escena con
                        circuito
341
342                 %Se obtienen los identificadores de los motores
                        y los
343                 %sensores de proximidad y visión izquierdo y
                        derecho.
344                 [~,left_motor]=vrep.simxGetObjectHandle(clientID
                        ,'tricycle_left_motor',vrep.
                        simx_opmode_blocking);
345                 [~,right_motor]=vrep.simxGetObjectHandle(
                        clientID,'tricycle_right_motor',vrep.
                        simx_opmode_blocking);
346                 [~,front_motor]=vrep.simxGetObjectHandle(
                        clientID,'tricycle_front_motor',vrep.
                        simx_opmode_blocking);
347                 [~,prox_sensor]=vrep.simxGetObjectHandle(
                        clientID,'tricycle_prox_sensor',vrep.
                        simx_opmode_blocking);
348                 [~,left_vision_sensor]=vrep.simxGetObjectHandle(
                        clientID,'tricycle_left_vision_sensor',vrep.
                        simx_opmode_blocking);
```

```
349                 [~,right_vision_sensor]=vrep.simxGetObjectHandle
                        (clientID,'tricycle_right_vision_sensor',vrep
                        .simx_opmode_blocking);
350
351                 %Se realiza una primera lectura de los sensores
352                 %cuyo resultado, por motivos del funcionamiento
                        de la
353                 %API, es desechado.
354                 [~,prox_detState,~,~,~]=vrep.
                        simxReadProximitySensor(clientID,prox_sensor,
                        vrep.simx_opmode_streaming);
355                 [~,~,left_vision_auxData,~]=vrep.
                        simxReadVisionSensor(clientID,
                        left_vision_sensor,vrep.simx_opmode_streaming
                        );
356                 [~,~,right_vision_auxData,~]=vrep.
                        simxReadVisionSensor(clientID,
                        right_vision_sensor,vrep.
                        simx_opmode_streaming);
357
358                 vrep.simxSynchronousTrigger(clientID);
359                 vrep.simxGetPingTime(clientID);
360
361                 %Se inicia el bucle de control del robot.
362                 while(i==1)
363                     [~,prox_detState,~,~,~]=vrep.
                            simxReadProximitySensor(clientID,
                            prox_sensor,vrep.simx_opmode_buffer);
364                     [~,~,left_vision_auxData,~]=vrep.
                            simxReadVisionSensor(clientID,
                            left_vision_sensor,vrep.
                            simx_opmode_buffer);
365                     [~,~,right_vision_auxData,~]=vrep.
                            simxReadVisionSensor(clientID,
                            right_vision_sensor,vrep.
                            simx_opmode_buffer);
366
367                     left_vision_read=(left_vision_auxData(11)
                            <0.3);
368                     right_vision_read=(right_vision_auxData(11)
                            <0.3);
369
370                     current_time=vrep.simxGetLastCmdTime(
                            clientID);
371
```

```
372                 if (prox_detState==1)
373                     back_until_time=current_time+200/handles
                            .LinVel;
374                 end
375
376                 if(current_time<back_until_time)
377                     wl=-w;
378                     wr=-w;
379                     theta=-steer*pi/6;
380                     steer_until_time=back_until_time+200/
                            handles.LinVel;
381
382                 elseif(current_time<steer_until_time)
383                     wl=w;
384                     wr=w;
385                     theta=steer*pi/6;
386
387                 elseif(left_vision_read==1 &&
                        right_vision_read==0)
388                     wl=w;
389                     wr=w;
390                     steer=1;
391                     theta=steer*pi/6;
392
393                 elseif(left_vision_read==0 &&
                        right_vision_read==1)
394                     wl=w;
395                     wr=w;
396                     steer=-1;
397                     theta=steer*pi/6;
398
399                 elseif(left_vision_read==1 &&
                        right_vision_read==1)
400                     in=1;
401                     wl=w;
402                     wr=w;
403                     theta=-steer*pi/6;
404
405                 elseif(in==0)
406                     wl=w;
407                     wr=w;
408                     theta=0;
409                 end
410
411                 vrep.simxSetJointTargetVelocity(clientID,
```

```
                    left_motor,wl,vrep.simx_opmode_oneshot);
412                 vrep.simxSetJointTargetVelocity(clientID,
                        right_motor,wr,vrep.simx_opmode_oneshot);
413                 vrep.simxSetJointTargetPosition(clientID,
                        front_motor,theta,vrep.
                        simx_opmode_oneshot);
414
415                 vrep.simxSynchronousTrigger(clientID);
416                 pause(0.01)
417             end
418
419         case 3
420             %--- Robot de tipo cuatriciclo en escena con
                    circuito
421
422             %Se obtienen los identificadores de los motores
                    y los
423             %sensores de proximidad y visión izquierdo y
                    derecho.
424             [~,rear_left_motor]=vrep.simxGetObjectHandle(
                    clientID,'quadricycle_rear_left_motor',vrep.
                    simx_opmode_blocking);
425             [~,rear_right_motor]=vrep.simxGetObjectHandle(
                    clientID,'quadricycle_rear_right_motor',vrep.
                    simx_opmode_blocking);
426             [~,front_left_motor]=vrep.simxGetObjectHandle(
                    clientID,'quadricycle_front_left_motor',vrep.
                    simx_opmode_blocking);
427             [~,front_right_motor]=vrep.simxGetObjectHandle(
                    clientID,'quadricycle_front_right_motor',vrep
                    .simx_opmode_blocking);
428             [~,prox_sensor]=vrep.simxGetObjectHandle(
                    clientID,'quadricycle_prox_sensor',vrep.
                    simx_opmode_blocking);
429             [~,left_vision_sensor]=vrep.simxGetObjectHandle(
                    clientID,'quadricycle_left_vision_sensor',
                    vrep.simx_opmode_blocking);
430             [~,right_vision_sensor]=vrep.simxGetObjectHandle
                    (clientID,'quadricycle_right_vision_sensor',
                    vrep.simx_opmode_blocking);
431
432             %Se realiza una primera lectura de los sensores
433             %cuyo resultado, por motivos del funcionamiento
                    de la
434             %API, es desechado.
```

```
435                 [~,prox_detState,~,~,~]=vrep.
                        simxReadProximitySensor(clientID,prox_sensor,
                        vrep.simx_opmode_streaming);
436                 [~,~,left_vision_auxData,~]=vrep.
                        simxReadVisionSensor(clientID,
                        left_vision_sensor,vrep.simx_opmode_streaming
                        );
437                 [~,~,right_vision_auxData,~]=vrep.
                        simxReadVisionSensor(clientID,
                        right_vision_sensor,vrep.
                        simx_opmode_streaming);
438
439                 vrep.simxSynchronousTrigger(clientID);
440                 vrep.simxGetPingTime(clientID);
441
442                 %Se inicia el bucle de control del robot.
443                 while(i==1)
444                     [~,prox_detState,~,~,~]=vrep.
                            simxReadProximitySensor(clientID,
                            prox_sensor,vrep.simx_opmode_buffer);
445                     [~,~,left_vision_auxData,~]=vrep.
                            simxReadVisionSensor(clientID,
                            left_vision_sensor,vrep.
                            simx_opmode_buffer);
446                     [~,~,right_vision_auxData,~]=vrep.
                            simxReadVisionSensor(clientID,
                            right_vision_sensor,vrep.
                            simx_opmode_buffer);
447
448                     left_vision_read=(left_vision_auxData(11)
                            <0.3);
449                     right_vision_read=(right_vision_auxData(11)
                            <0.3);
450
451                     current_time=vrep.simxGetLastCmdTime(
                            clientID);
452
453                     if (prox_detState==1)
454                         back_until_time=current_time+200/handles
                                .LinVel;
455                     end
456
457                     if(current_time<back_until_time)
458                         wl=-w;
459                         wr=-w;
```

```
460                     theta=-steer*pi/6;
461                     steer_until_time=back_until_time+200/
                            handles.LinVel;
462
463                 elseif(current_time<steer_until_time)
464                     wl=w;
465                     wr=w;
466                     theta=steer*pi/6;
467
468                 elseif(left_vision_read==1 &&
                        right_vision_read==0)
469                     wl=w;
470                     wr=w;
471                     steer=1;
472                     theta=steer*pi/6;
473
474                 elseif(left_vision_read==0 &&
                        right_vision_read==1)
475                     wl=w;
476                     wr=w;
477                     steer=-1;
478                     theta=steer*pi/6;
479
480                 elseif(left_vision_read==1 &&
                        right_vision_read==1)
481                     in=1;
482                     wl=w;
483                     wr=w;
484                     theta=-steer*pi/6;
485
486                 elseif(in==0)
487                     wl=w;
488                     wr=w;
489                     theta=0;
490                 end
491
492                 theta_left=acot(cot(theta)-(axis_dist_lr/2)/
                        axis_dist_fr);
493                 theta_right=acot(cot(theta)+(axis_dist_lr/2)
                        /axis_dist_fr);
494
495                 vrep.simxSetJointTargetVelocity(clientID,
                        rear_left_motor,wl,vrep.
                        simx_opmode_oneshot);
496                 vrep.simxSetJointTargetVelocity(clientID,
```

```
                        rear_right_motor,wr,vrep.
                            simx_opmode_oneshot);
497                     vrep.simxSetJointTargetPosition(clientID,
                            front_left_motor,theta_left,vrep.
                            simx_opmode_oneshot);
498                     vrep.simxSetJointTargetPosition(clientID,
                            front_right_motor,theta_right,vrep.
                            simx_opmode_oneshot);
499
500                     vrep.simxSynchronousTrigger(clientID);
501                     pause(0.01)
502                 end
503         end
504
505     case 3
506         switch model
507             case 1
508                 %--- Robot de tipo diferencial en escena con
                        balizas
509
510                 %Se obtienen los identificadores de los motores
                        y el sensor
511                 %de visión central.
512                 [~,left_motor]=vrep.simxGetObjectHandle(clientID
                        ,'diff_left_motor',vrep.simx_opmode_blocking)
                        ;
513                 [~,right_motor]=vrep.simxGetObjectHandle(
                        clientID,'diff_right_motor',vrep.
                        simx_opmode_blocking);
514                 [~,prox_sensor]=vrep.simxGetObjectHandle(
                        clientID,'diff_prox_sensor',vrep.
                        simx_opmode_blocking);
515                 [~,middle_vision_sensor]=vrep.
                        simxGetObjectHandle(clientID,'
                        diff_middle_vision_sensor',vrep.
                        simx_opmode_blocking);
516
517                 %Se realiza una primera lectura del sensor de
                        proximidad y
518                 %el de visión central cuyo resultado, por
                        motivos del funcionamiento de la
519                 %API, es desechado.
520                 [~,prox_detState,~,~,~]=vrep.
                        simxReadProximitySensor(clientID,prox_sensor,
                        vrep.simx_opmode_streaming);
```

```
521                 [~,~,middle_vision_auxData,~]=vrep.
                        simxReadVisionSensor(clientID,
                        middle_vision_sensor,vrep.
                        simx_opmode_streaming);
522
523                 vrep.simxSynchronousTrigger(clientID);
524                 vrep.simxGetPingTime(clientID);
525
526                 %Se inicia el bucle de control del robot.
527                 while(i==1)
528                     [~,prox_detState,~,~,~]=vrep.
                            simxReadProximitySensor(clientID,
                            prox_sensor,vrep.simx_opmode_buffer);
529                     [~,~,middle_vision_auxData,~]=vrep.
                            simxReadVisionSensor(clientID,
                            middle_vision_sensor,vrep.
                            simx_opmode_buffer);
530
531                     middle_vision_read=(middle_vision_auxData(7)
                            <0.2 && middle_vision_auxData(8)>0.8 &&
                            middle_vision_auxData(9)<0.2);
532
533                     current_time=vrep.simxGetLastCmdTime(
                            clientID);
534
535                     if (prox_detState==1)
536                         back_until_time=current_time+50/handles.
                                LinVel;
537
538                     elseif (middle_vision_read==1)
539                         forward=1;
540                     end
541
542                     if(current_time<back_until_time)
543                         forward=0;
544                         wl=-4*w/5;
545                         wr=-6*w/5;
546
547                     elseif(forward==1)
548                         wl=w;
549                         wr=w;
550
551                     else
552                         wl=w/30;
553                         wr=-w/30;
```

```
554                     end
555
556                     vrep.simxSetJointTargetVelocity(clientID,
                            left_motor,wl,vrep.simx_opmode_oneshot);
557                     vrep.simxSetJointTargetVelocity(clientID,
                            right_motor,wr,vrep.simx_opmode_oneshot);
558
559                     vrep.simxSynchronousTrigger(clientID);
560                     pause(0.01)
561                 end
562
563             case 2
564                 %--- Robot de tipo triciclo en escena con
                        balizas
565
566                 %Se obtienen los identificadores de los motores
                        y el sensor
567                 %de visión central.
568                 [~,left_motor]=vrep.simxGetObjectHandle(clientID
                        ,'tricycle_left_motor',vrep.
                        simx_opmode_blocking);
569                 [~,right_motor]=vrep.simxGetObjectHandle(
                        clientID,'tricycle_right_motor',vrep.
                        simx_opmode_blocking);
570                 [~,front_motor]=vrep.simxGetObjectHandle(
                        clientID,'tricycle_front_motor',vrep.
                        simx_opmode_blocking);
571                 [~,prox_sensor]=vrep.simxGetObjectHandle(
                        clientID,'tricycle_prox_sensor',vrep.
                        simx_opmode_blocking);
572                 [~,middle_vision_sensor]=vrep.
                        simxGetObjectHandle(clientID,'
                        tricycle_middle_vision_sensor',vrep.
                        simx_opmode_blocking);
573
574                 %Se realiza una primera lectura del sensor de
                        proximidad y
575                 %el de visión central cuyo resultado, por
                        motivos del funcionamiento de la
576                 %API, es desechado.
577                 [~,prox_detState,~,~,~]=vrep.
                        simxReadProximitySensor(clientID,prox_sensor,
                        vrep.simx_opmode_streaming);
578                 [~,~,middle_vision_auxData,~]=vrep.
                        simxReadVisionSensor(clientID,
```

```
                middle_vision_sensor,vrep.
                simx_opmode_streaming);
579
580             vrep.simxSynchronousTrigger(clientID);
581             vrep.simxGetPingTime(clientID);
582
583             %Se inicia el bucle de control del robot.
584             while(i==1)
585                 [~,prox_detState,~,~,~]=vrep.
                        simxReadProximitySensor(clientID,
                        prox_sensor,vrep.simx_opmode_buffer);
586                 [~,~,middle_vision_auxData,~]=vrep.
                        simxReadVisionSensor(clientID,
                        middle_vision_sensor,vrep.
                        simx_opmode_buffer);
587
588                 middle_vision_read=(middle_vision_auxData(7)
                        <0.2 && middle_vision_auxData(8)>0.8 &&
                        middle_vision_auxData(9)<0.2);
589
590                 current_time=vrep.simxGetLastCmdTime(
                        clientID);
591
592                 if (prox_detState==1)
593                     back_until_time=current_time+150/handles
                            .LinVel;
594
595                 elseif (middle_vision_read==1)
596                     forward=1;
597                 end
598
599                 if(current_time<back_until_time)
600                     forward=0;
601                     wl=-w;
602                     wr=-w;
603                     theta=-pi/6;
604
605                 elseif(forward==1)
606                     wl=w;
607                     wr=w;
608                     theta=0;
609
610                 else
611                     wl=w;
612                     wr=w;
```

```
613                         theta=pi/6;
614                     end
615
616                     vrep.simxSetJointTargetVelocity(clientID,
                            left_motor,wl,vrep.simx_opmode_oneshot);
617                     vrep.simxSetJointTargetVelocity(clientID,
                            right_motor,wr,vrep.simx_opmode_oneshot);
618                     vrep.simxSetJointTargetPosition(clientID,
                            front_motor,theta,vrep.
                            simx_opmode_oneshot);
619
620                     vrep.simxSynchronousTrigger(clientID);
621                     pause(0.01)
622                 end
623
624             case 3
625                 %--- Robot de tipo cuatriciclo en escena con
                        balizas
626
627                 %Se obtienen los identificadores de los motores
                        y el sensor
628                 %de visión central.
629                 [~,rear_left_motor]=vrep.simxGetObjectHandle(
                        clientID,'quadricycle_rear_left_motor',vrep.
                        simx_opmode_blocking);
630                 [~,rear_right_motor]=vrep.simxGetObjectHandle(
                        clientID,'quadricycle_rear_right_motor',vrep.
                        simx_opmode_blocking);
631                 [~,front_left_motor]=vrep.simxGetObjectHandle(
                        clientID,'quadricycle_front_left_motor',vrep.
                        simx_opmode_blocking);
632                 [~,front_right_motor]=vrep.simxGetObjectHandle(
                        clientID,'quadricycle_front_right_motor',vrep
                        .simx_opmode_blocking);
633                 [~,prox_sensor]=vrep.simxGetObjectHandle(
                        clientID,'quadricycle_prox_sensor',vrep.
                        simx_opmode_blocking);
634                 [~,middle_vision_sensor]=vrep.
                        simxGetObjectHandle(clientID,'
                        quadricycle_middle_vision_sensor',vrep.
                        simx_opmode_blocking);
635
636                 %Se realiza una primera lectura del sensor de
                        proximidad y
637                 %el de visión central cuyo resultado, por
```

```
                    motivos del funcionamiento de la
638                 %API, es desechado.
639                 [~,prox_detState,~,~,~]=vrep.
                        simxReadProximitySensor(clientID,prox_sensor,
                        vrep.simx_opmode_streaming);
640                 [~,~,middle_vision_auxData,~]=vrep.
                        simxReadVisionSensor(clientID,
                        middle_vision_sensor,vrep.
                        simx_opmode_streaming);
641
642                 vrep.simxSynchronousTrigger(clientID);
643                 vrep.simxGetPingTime(clientID);
644
645                 %Se inicia el bucle de control del robot.
646                 while(i==1)
647                     [~,prox_detState,~,~,~]=vrep.
                            simxReadProximitySensor(clientID,
                            prox_sensor,vrep.simx_opmode_buffer);
648                     [~,~,middle_vision_auxData,~]=vrep.
                            simxReadVisionSensor(clientID,
                            middle_vision_sensor,vrep.
                            simx_opmode_buffer);
649
650                     middle_vision_read=(middle_vision_auxData(7)
                            <0.2 && middle_vision_auxData(8)>0.8 &&
                            middle_vision_auxData(9)<0.2);
651
652                     current_time=vrep.simxGetLastCmdTime(
                            clientID);
653
654                     if (prox_detState==1)
655                         back_until_time=current_time+150/handles
                                .LinVel;
656
657                     elseif (middle_vision_read==1)
658                         forward=1;
659                     end
660
661                     if(current_time<back_until_time)
662                         forward=0;
663                         wl=-w;
664                         wr=-w;
665                         theta=-pi/6;
666
667                     elseif(forward==1)
```

```
668                             wl=w;
669                             wr=w;
670                             theta=0;
671
672                         else
673                             wl=w;
674                             wr=w;
675                             theta=pi/6;
676                         end
677
678                         theta_left=acot(cot(theta)-(axis_dist_lr/2)/
                                axis_dist_fr);
679                         theta_right=acot(cot(theta)+(axis_dist_lr/2)
                                /axis_dist_fr);
680
681                         vrep.simxSetJointTargetVelocity(clientID,
                                rear_left_motor,wl,vrep.
                                simx_opmode_oneshot);
682                         vrep.simxSetJointTargetVelocity(clientID,
                                rear_right_motor,wr,vrep.
                                simx_opmode_oneshot);
683                         vrep.simxSetJointTargetPosition(clientID,
                                front_left_motor,theta_left,vrep.
                                simx_opmode_oneshot);
684                         vrep.simxSetJointTargetPosition(clientID,
                                front_right_motor,theta_right,vrep.
                                simx_opmode_oneshot);
685
686                         vrep.simxSynchronousTrigger(clientID);
687                         pause(0.01)
688                     end
689             end
690 end
691
692 %--- Función que se ejecuta al pulsar el botón 'PauseSim'
        encargada de
693 %pausar la simulación.
694 function PauseSim_Callback(hObject, eventdata, handles)
695 global vrep
696 global clientID
697 global i
698
699 i=0;
700 vrep.simxPauseSimulation(clientID,vrep.simx_opmode_oneshot);
701
```

```
702 %--- Función que se ejecuta al pulsar el botón 'StopSim'
        encargada de
703 % parar la simulación y finalizar la conexión con V-REP si no se
        desea realizar un nuevo testeo.
704 function StopSim_Callback(hObject, eventdata, handles)
705 global vrep
706 global clientID
707 global i
708 global next
709
710 i=0;
711 vrep.simxStopSimulation(clientID,vrep.simx_opmode_oneshot);
712
713 opc=questdlg('¿Desea realizar una nueva simulación?','Salir','Sí
        ','No','No');
714 if strcmp(opc,'No')
715     vrep.simxGetPingTime(clientID);
716     vrep.simxFinish(clientID);
717     next=1;
718     close Simulation
719 else
720     next=1;
721     close Simulation
722     SelectScene
723 end
724
725 %--- %--- Función que lee el valor de LinVel y obtiene a
        partir de él un
726 % valor de velocidad angular de referencia para las ruedas.
727 function LinVel_Callback(hObject, eventdata, handles)
728 global scene
729 global wheels_rad
730 global w
731
732 lin_vel=get(hObject,'value');
733 handles.LinVel=lin_vel;
734 if(scene==2 && lin_vel>0.5)
735     warndlg('Velocidades lineales muy altas pueden impedir que
            el robot siga correctamente el circuito.','ADVERTENCIA')
736 end
737 guidata(hObject,handles);
738 disp_lin_vel=round(100*lin_vel)/100;
739 set(handles.DispLinVel,'string',['v=',num2str(disp_lin_vel),' m/
        s']);
740 guidata(hObject,handles);
```

```
741
742 w=handles.LinVel/wheels_rad;
743
744 %--- Función que se ejecuta cuando se intenta cerrar la ventana
    .
745 function figure1_CloseRequestFcn(hObject, eventdata, handles)
746 global next
747
748 %Si la petición de cierre de la ventana no es consecuencia del
    normal tránsito del programa de una ventana a otra
749 %sino que deriva del clic por parte del usuario en el icono de
    cierre, se exige una confirmación.
750 if next==0
751     opc=questdlg('¿Desea salir del programa?','Salir','Sí','No',
        'No');
752     if strcmp(opc,'No')
753         return
754     end
755 end
756 next=0;
757 delete(hObject);
```

B. Código Lua

B.1. create

```
1  --Se definen los materiales de las ruedas y las esferas
       deslizantes
2  if (sim_call_type==sim_childscriptcall_initialization) then
3  wheelMaterial=simGetMaterialId('wheelMaterial')
4  noFrictionMaterial=simGetMaterialId('noFrictionMaterial')
5  end
6
7  --Función que crea un robot diferencial en la escena a partir de
       los parámetros recibidos desde Matlab. Para una información
8  --detallada acerca de los argumentos de cada función API,
       consultar el listado 'Regular API functions'.
9
10 CreateDiff_function=function(inInts,inFloats,inStrings,inBuffer)
11     diff=simCreatePureShape(inInts[1],01110,{inFloats[1],
           inFloats[2],inFloats[3]},inFloats[4],NULL)
12     simSetObjectPosition(diff,-1,{inFloats[5],inFloats[6],
           inFloats[7]})
13     simSetObjectName(diff,'diff')
14     simSetObjectSpecialProperty(diff,
           sim_objectspecialproperty_measurable+
           sim_objectspecialproperty_collidable+
           sim_objectspecialproperty_renderable+
           sim_objectspecialproperty_detectable_all)
15     simSetObjectInt32Parameter(diff,sim_shapeintparam_static,0)
16     simSetObjectInt32Parameter(diff,
           sim_shapeintparam_respondable,1)
17     simSetShapeColor(diff,NULL,
           sim_colorcomponent_ambient_diffuse,{1.0,0.71,0.37})
18
19     left_wheel=simCreatePureShape(2,01110,{inFloats[8],inFloats
           [9],inFloats[10]},inFloats[11],NULL)
20     simSetObjectPosition(left_wheel,-1,{inFloats[12],inFloats
           [13],inFloats[14]})
21     simSetObjectOrientation(left_wheel,-1,{0,-math.pi/2,0})
22     simSetObjectName(left_wheel,'diff_left_wheel')
23     simSetObjectSpecialProperty(left_wheel,
           sim_objectspecialproperty_measurable+
           sim_objectspecialproperty_collidable+
           sim_objectspecialproperty_renderable+
           sim_objectspecialproperty_detectable_all)
24     simSetObjectInt32Parameter(left_wheel,
```

```
        sim_shapeintparam_static,0)
25      simSetObjectInt32Parameter(left_wheel,
            sim_shapeintparam_respondable,1)
26      simSetShapeColor(left_wheel,NULL,
            sim_colorcomponent_ambient_diffuse,{0.15,0.15,0.15})
27      simSetShapeMaterial(left_wheel,wheelMaterial)
28
29      right_wheel=simCreatePureShape(2,01110,{inFloats[8],inFloats
            [9],inFloats[10]},inFloats[11],NULL)
30      simSetObjectPosition(right_wheel,-1,{inFloats[15],inFloats
            [16],inFloats[17]})
31      simSetObjectOrientation(right_wheel,-1,{0,-math.pi/2,0})
32      simSetObjectName(right_wheel,'diff_right_wheel')
33      simSetObjectSpecialProperty(right_wheel,
            sim_objectspecialproperty_measurable+
            sim_objectspecialproperty_collidable+
            sim_objectspecialproperty_renderable+
            sim_objectspecialproperty_detectable_all)
34      simSetObjectInt32Parameter(right_wheel,
            sim_shapeintparam_static,0)
35      simSetObjectInt32Parameter(right_wheel,
            sim_shapeintparam_respondable,1)
36      simSetShapeColor(right_wheel,NULL,
            sim_colorcomponent_ambient_diffuse,{0.15,0.15,0.15})
37      simSetShapeMaterial(right_wheel,wheelMaterial)
38
39      left_motor=simCreateJoint(sim_joint_revolute_subtype,
            sim_jointmode_force,0,NULL,NULL,NULL)
40      simSetObjectPosition(left_motor,-1,{inFloats[12],inFloats
            [13],inFloats[14]})
41      simSetObjectOrientation(left_motor,-1,{0,-math.pi/2,0})
42      simSetObjectName(left_motor,'diff_left_motor')
43      simSetObjectInt32Parameter(left_motor,
            sim_jointintparam_motor_enabled,1)
44      simSetObjectInt32Parameter(left_motor,
            sim_jointintparam_velocity_lock,1)
45      simSetObjectInt32Parameter(left_motor,
            sim_objintparam_visibility_layer,1024)
46
47      right_motor=simCreateJoint(sim_joint_revolute_subtype,
            sim_jointmode_force,0,NULL,NULL,NULL)
48      simSetObjectPosition(right_motor,-1,{inFloats[15],inFloats
            [16],inFloats[17]})
49      simSetObjectOrientation(right_motor,-1,{0,-math.pi/2,0})
50      simSetObjectName(right_motor,'diff_right_motor')
```

```
51      simSetObjectInt32Parameter(right_motor,
            sim_jointintparam_motor_enabled,1)
52      simSetObjectInt32Parameter(right_motor,
            sim_jointintparam_velocity_lock,1)
53      simSetObjectInt32Parameter(right_motor,
            sim_objintparam_visibility_layer,1024)
54
55      prox_sensor=simCreateProximitySensor(
            sim_proximitysensor_disc_subtype,
            sim_objectspecialproperty_detectable_ultrasonic
            ,0000000000,{16,32,0,0,0,0,0,0},{0.0,inFloats[18],0.0,0
            .05,0.0,0.0,0.0,0.1,0.0,math.pi/2,0.0,0.0,0.01,0.0,0.0},
            NULL)
56      simSetObjectPosition(prox_sensor,-1,{inFloats[19],inFloats
            [20],inFloats[21]})
57      simSetObjectOrientation(prox_sensor,-1,{-math.pi/2,0,0})
58      simSetObjectName(prox_sensor,'diff_prox_sensor')
59
60      front_force_sensor=simCreateForceSensor(00,{0,1,10,0,0},{0
            .01,100.0,10.0,0.0,0.0},NULL)
61      simSetObjectPosition(front_force_sensor,-1,{inFloats[22],
            inFloats[23],inFloats[24]})
62      simSetObjectName(front_force_sensor,'diff_front_force_sensor
            ')
63      simSetObjectInt32Parameter(front_force_sensor,
            sim_objintparam_visibility_layer,1024)
64
65      rear_force_sensor=simCreateForceSensor(00,{0,1,10,0,0},{0.01
            ,100.0,10.0,0.0,0.0},NULL)
66      simSetObjectPosition(rear_force_sensor,-1,{inFloats[25],
            inFloats[26],inFloats[27]})
67      simSetObjectName(rear_force_sensor,'diff_rear_force_sensor')
68      simSetObjectInt32Parameter(rear_force_sensor,
            sim_objintparam_visibility_layer,1024)
69
70      front_slider=simCreatePureShape(1,01110,{inFloats[28],
            inFloats[29],inFloats[30]},inFloats[31],NULL)
71      simSetObjectPosition(front_slider,-1,{inFloats[32],inFloats
            [33],inFloats[34]})
72      simSetObjectOrientation(front_slider,-1,{0,-math.pi/2,0})
73      simSetObjectName(front_slider,'diff_front_slider')
74      simSetObjectSpecialProperty(front_slider,
            sim_objectspecialproperty_measurable+
            sim_objectspecialproperty_collidable+
            sim_objectspecialproperty_renderable+
```

```
        sim_objectspecialproperty_detectable_all)
75      simSetObjectInt32Parameter(front_slider,
            sim_shapeintparam_static,0)
76      simSetObjectInt32Parameter(front_slider,
            sim_shapeintparam_respondable,1)
77      simSetShapeColor(front_slider,NULL,
            sim_colorcomponent_ambient_diffuse,{0.15,0.15,0.15})
78      simSetShapeMaterial(front_slider,noFrictionMaterial)
79
80      rear_slider=simCreatePureShape(1,01110,{inFloats[28],
            inFloats[29],inFloats[30]},inFloats[31],NULL)
81      simSetObjectPosition(rear_slider,-1,{inFloats[35],inFloats
            [36],inFloats[37]})
82      simSetObjectOrientation(rear_slider,-1,{0,-math.pi/2,0})
83      simSetObjectName(rear_slider,'diff_rear_slider')
84      simSetObjectSpecialProperty(rear_slider,
            sim_objectspecialproperty_measurable+
            sim_objectspecialproperty_collidable+
            sim_objectspecialproperty_renderable+
            sim_objectspecialproperty_detectable_all)
85      simSetObjectInt32Parameter(rear_slider,
            sim_shapeintparam_static,0)
86      simSetObjectInt32Parameter(rear_slider,
            sim_shapeintparam_respondable,1)
87      simSetShapeColor(rear_slider,NULL,
            sim_colorcomponent_ambient_diffuse,{0.15,0.15,0.15})
88      simSetShapeMaterial(front_slider,noFrictionMaterial)
89
90      left_vision_sensor=simCreateVisionSensor
            (00000000,{1,1,0,0},{0.01,inFloats[38],0.01,0.01,0.01,0
            .01,0.0,0.0,0.0,0.0,0.0},NULL)
91      simSetObjectPosition(left_vision_sensor,-1,{inFloats[39],
            inFloats[40],inFloats[41]})
92      simSetObjectName(left_vision_sensor,'diff_left_vision_sensor
            ')
93      simSetObjectOrientation(left_vision_sensor,-1,{math.pi,0,0})
94
95      right_vision_sensor=simCreateVisionSensor
            (00000000,{1,1,0,0},{0.01,inFloats[38],0.01,0.01,0.01,0
            .01,0.0,0.0,0.0,0.0,0.0},NULL)
96      simSetObjectPosition(right_vision_sensor,-1,{inFloats[42],
            inFloats[43],inFloats[44]})
97      simSetObjectName(right_vision_sensor,'
            diff_right_vision_sensor')
98      simSetObjectOrientation(right_vision_sensor,-1,{math.pi
```

```
        ,0,0})
99
100     middle_vision_sensor=simCreateVisionSensor
            (00000000,{1,1,0,0},{0.01,math.sqrt(2*5.00^2),0.01,0.01,0
            .01,0.01,0.0,0.0,0.0,0.0,0.0},NULL)
101     simSetObjectPosition(middle_vision_sensor,-1,{inFloats[45],
            inFloats[46],inFloats[47]})
102     simSetObjectName(middle_vision_sensor,'
            diff_middle_vision_sensor')
103     simSetObjectOrientation(middle_vision_sensor,-1,{-math.pi
            /2,0,0})
104
105     simSetObjectParent(left_wheel,left_motor,true)
106     simSetObjectParent(right_wheel,right_motor,true)
107     simSetObjectParent(front_slider,front_force_sensor,true)
108     simSetObjectParent(rear_slider,rear_force_sensor,true)
109     simSetObjectParent(left_motor,diff,true)
110     simSetObjectParent(right_motor,diff,true)
111     simSetObjectParent(prox_sensor,diff,true)
112     simSetObjectParent(front_force_sensor,diff,true)
113     simSetObjectParent(rear_force_sensor,diff,true)
114     simSetObjectParent(left_vision_sensor,diff,true)
115     simSetObjectParent(right_vision_sensor,diff,true)
116     simSetObjectParent(middle_vision_sensor,diff,true)
117
118     return {},{},{},''
119 end
120
121
122
123 --Función que crea un robot de tipo triciclo en la escena a
        partir de los parámetros recibidos desde Matlab. Para una
        información
124 --detallada acerca de los argumentos de cada función API,
        consultar el listado 'Regular API functions'.
125 CreateTricycle_function=function(inInts,inFloats,inStrings,
        inBuffer)
126     tricycle=simCreatePureShape(inInts[1],01110,{inFloats[1],
            inFloats[2],inFloats[3]},inFloats[4],NULL)
127     simSetObjectPosition(tricycle,-1,{inFloats[5],inFloats[6],
            inFloats[7]})
128     simSetObjectName(tricycle,'tricycle')
129     simSetObjectSpecialProperty(tricycle,
            sim_objectspecialproperty_measurable+
            sim_objectspecialproperty_collidable+
```

```
            sim_objectspecialproperty_renderable+
            sim_objectspecialproperty_detectable_all)
130         simSetObjectInt32Parameter(tricycle,sim_shapeintparam_static
            ,0)
131         simSetObjectInt32Parameter(tricycle,
            sim_shapeintparam_respondable,1)
132         simSetShapeColor(tricycle,NULL,
            sim_colorcomponent_ambient_diffuse,{0.30,0.61,0.84})
133
134         left_wheel=simCreatePureShape(2,01110,{inFloats[8],inFloats
            [9],inFloats[10]},inFloats[11],NULL)
135         simSetObjectPosition(left_wheel,-1,{inFloats[12],inFloats
            [13],inFloats[14]})
136         simSetObjectOrientation(left_wheel,-1,{0,-math.pi/2,0})
137         simSetObjectName(left_wheel,'tricycle_left_wheel')
138         simSetObjectSpecialProperty(left_wheel,
            sim_objectspecialproperty_measurable+
            sim_objectspecialproperty_collidable+
            sim_objectspecialproperty_renderable+
            sim_objectspecialproperty_detectable_all)
139         simSetObjectInt32Parameter(left_wheel,
            sim_shapeintparam_static,0)
140         simSetObjectInt32Parameter(left_wheel,
            sim_shapeintparam_respondable,1)
141         simSetShapeColor(left_wheel,NULL,
            sim_colorcomponent_ambient_diffuse,{0.15,0.15,0.15})
142         simSetShapeMaterial(left_wheel,wheelMaterial)
143
144         right_wheel=simCreatePureShape(2,01110,{inFloats[8],inFloats
            [9],inFloats[10]},inFloats[11],NULL)
145         simSetObjectPosition(right_wheel,-1,{inFloats[15],inFloats
            [16],inFloats[17]})
146         simSetObjectOrientation(right_wheel,-1,{0,-math.pi/2,0})
147         simSetObjectName(right_wheel,'tricycle_right_wheel')
148         simSetObjectSpecialProperty(right_wheel,
            sim_objectspecialproperty_measurable+
            sim_objectspecialproperty_collidable+
            sim_objectspecialproperty_renderable+
            sim_objectspecialproperty_detectable_all)
149         simSetObjectInt32Parameter(right_wheel,
            sim_shapeintparam_static,0)
150         simSetObjectInt32Parameter(right_wheel,
            sim_shapeintparam_respondable,1)
151         simSetShapeColor(right_wheel,NULL,
            sim_colorcomponent_ambient_diffuse,{0.15,0.15,0.15})
```

```
152     simSetShapeMaterial(right_wheel,wheelMaterial)
153
154     front_wheel=simCreatePureShape(2,01110,{inFloats[8],inFloats
            [9],inFloats[10]},inFloats[11],NULL)
155     simSetObjectPosition(front_wheel,-1,{inFloats[18],inFloats
            [19],inFloats[20]})
156     simSetObjectOrientation(front_wheel,-1,{0,-math.pi/2,0})
157     simSetObjectName(front_wheel,'tricycle_front_wheel')
158     simSetObjectSpecialProperty(front_wheel,
            sim_objectspecialproperty_measurable+
            sim_objectspecialproperty_collidable+
            sim_objectspecialproperty_renderable+
            sim_objectspecialproperty_detectable_all)
159     simSetObjectInt32Parameter(front_wheel,
            sim_shapeintparam_static,0)
160     simSetObjectInt32Parameter(front_wheel,
            sim_shapeintparam_respondable,1)
161     simSetShapeColor(front_wheel,NULL,
            sim_colorcomponent_ambient_diffuse,{0.15,0.15,0.15})
162     simSetShapeMaterial(front_wheel,wheelMaterial)
163
164     inv_front_wheel=simCreatePureShape(2,01110,{inFloats[8],
            inFloats[9],inFloats[10]},inFloats[11],NULL)
165     simSetObjectPosition(inv_front_wheel,-1,{inFloats[18],
            inFloats[19],inFloats[20]})
166     simSetObjectOrientation(inv_front_wheel,-1,{0,-math.pi/2,0})
167     simSetObjectName(inv_front_wheel,'tricycle_inv_front_wheel')
168     simSetObjectInt32Parameter(inv_front_wheel,
            sim_shapeintparam_static,0)
169     simSetObjectInt32Parameter(inv_front_wheel,
            sim_shapeintparam_respondable,0)
170     simSetObjectInt32Parameter(inv_front_wheel,
            sim_objintparam_visibility_layer,1024)
171     simSetShapeMaterial(inv_front_wheel,wheelMaterial)
172
173     left_motor=simCreateJoint(sim_joint_revolute_subtype,
            sim_jointmode_force,0,NULL,NULL,NULL)
174     simSetObjectPosition(left_motor,-1,{inFloats[12],inFloats
            [13],inFloats[14]})
175     simSetObjectOrientation(left_motor,-1,{0,-math.pi/2,0})
176     simSetObjectName(left_motor,'tricycle_left_motor')
177     simSetObjectInt32Parameter(left_motor,
            sim_jointintparam_motor_enabled,1)
178     simSetObjectInt32Parameter(left_motor,
            sim_jointintparam_velocity_lock,1)
```

```
179     simSetObjectInt32Parameter(left_motor,
            sim_objintparam_visibility_layer,1024)
180
181     right_motor=simCreateJoint(sim_joint_revolute_subtype,
            sim_jointmode_force,0,NULL,NULL,NULL)
182     simSetObjectPosition(right_motor,-1,{inFloats[15],inFloats
            [16],inFloats[17]})
183     simSetObjectOrientation(right_motor,-1,{0,-math.pi/2,0})
184     simSetObjectName(right_motor,'tricycle_right_motor')
185     simSetObjectInt32Parameter(right_motor,
            sim_jointintparam_motor_enabled,1)
186     simSetObjectInt32Parameter(right_motor,
            sim_jointintparam_velocity_lock,1)
187     simSetObjectInt32Parameter(right_motor,
            sim_objintparam_visibility_layer,1024)
188
189     front_motor=simCreateJoint(sim_joint_revolute_subtype,
            sim_jointmode_force,0,NULL,NULL,NULL)
190     simSetObjectPosition(front_motor,-1,{inFloats[18],inFloats
            [19],inFloats[20]})
191     simSetObjectOrientation(front_motor,-1,{0,0,0})
192     simSetObjectName(front_motor,'tricycle_front_motor')
193     simSetObjectInt32Parameter(front_motor,
            sim_jointintparam_motor_enabled,1)
194     simSetObjectInt32Parameter(front_motor,
            sim_jointintparam_ctrl_enabled,1)
195     simSetObjectInt32Parameter(front_motor,
            sim_objintparam_visibility_layer,1024)
196
197     front_wheel_axis=simCreateJoint(sim_joint_revolute_subtype,
            sim_jointmode_force,0,NULL,NULL,NULL)
198     simSetObjectPosition(front_wheel_axis,-1,{inFloats[18],
            inFloats[19],inFloats[20]})
199     simSetObjectOrientation(front_wheel_axis,-1,{0,-math.pi
            /2,0})
200     simSetObjectName(front_wheel_axis,'tricycle_front_wheel_axis
            ')
201     simSetObjectInt32Parameter(front_wheel_axis,
            sim_objintparam_visibility_layer,1024)
202
203     prox_sensor=simCreateProximitySensor(
            sim_proximitysensor_disc_subtype,
            sim_objectspecialproperty_detectable_ultrasonic
            ,0000000000,{16,32,0,0,0,0,0,0},{0.0,inFloats[21],0.0,0
            .05,0.0,0.0,0.0,0.1,0.0,math.pi/2,0.0,0.0,0.01,0.0,0.0},
```

```
        NULL)
204     simSetObjectPosition(prox_sensor,-1,{inFloats[22],inFloats
            [23],inFloats[24]})
205     simSetObjectOrientation(prox_sensor,-1,{-math.pi/2,0,0})
206     simSetObjectName(prox_sensor,'tricycle_prox_sensor')
207
208     left_vision_sensor=simCreateVisionSensor
            (00000000,{1,1,0,0},{0.01,inFloats[25],0.01,0.01,0.01,0
            .01,0.0,0.0,0.0,0.0,0.0},NULL)
209     simSetObjectPosition(left_vision_sensor,-1,{inFloats[26],
            inFloats[27],inFloats[28]})
210     simSetObjectName(left_vision_sensor,'
            tricycle_left_vision_sensor')
211     simSetObjectOrientation(left_vision_sensor,-1,{math.pi,0,0})
212
213     right_vision_sensor=simCreateVisionSensor
            (00000000,{1,1,0,0},{0.01,inFloats[25],0.01,0.01,0.01,0
            .01,0.0,0.0,0.0,0.0,0.0},NULL)
214     simSetObjectPosition(right_vision_sensor,-1,{inFloats[29],
            inFloats[30],inFloats[31]})
215     simSetObjectName(right_vision_sensor,'
            tricycle_right_vision_sensor')
216     simSetObjectOrientation(right_vision_sensor,-1,{math.pi
            ,0,0})
217
218     middle_vision_sensor=simCreateVisionSensor
            (00000000,{1,1,0,0},{0.01,math.sqrt(2*5.00^2),0.01,0.01,0
            .01,0.01,0.0,0.0,0.0,0.0,0.0},NULL)
219     simSetObjectPosition(middle_vision_sensor,-1,{inFloats[32],
            inFloats[33],inFloats[34]})
220     simSetObjectName(middle_vision_sensor,'
            tricycle_middle_vision_sensor')
221     simSetObjectOrientation(middle_vision_sensor,-1,{-math.pi
            /2,0,0})
222
223
224     simSetObjectParent(left_wheel,left_motor,true)
225     simSetObjectParent(right_wheel,right_motor,true)
226     simSetObjectParent(front_wheel,front_wheel_axis,true)
227     simSetObjectParent(front_wheel_axis,inv_front_wheel,true)
228     simSetObjectParent(inv_front_wheel,front_motor,true)
229     simSetObjectParent(left_motor,tricycle,true)
230     simSetObjectParent(right_motor,tricycle,true)
231     simSetObjectParent(front_motor,tricycle,true)
232     simSetObjectParent(prox_sensor,tricycle,true)
```

```
233     simSetObjectParent(left_vision_sensor,tricycle,true)
234     simSetObjectParent(right_vision_sensor,tricycle,true)
235     simSetObjectParent(middle_vision_sensor,tricycle,true)
236
237
238     return {},{},{},''
239 end
240
241
242 --Función que crea un robot de tipo cuatriciclo en la escena a
        partir de los parámetros recibidos desde Matlab. Para una
        información
243 --detallada acerca de los argumentos de cada función API,
        consultar el listado 'Regular API functions'.
244 CreateQuadricycle_function=function(inInts,inFloats,inStrings,
        inBuffer)
245     quadricycle=simCreatePureShape(inInts[1],01110,{inFloats[1],
            inFloats[2],inFloats[3]},inFloats[4],NULL)
246     simSetObjectPosition(quadricycle,-1,{inFloats[5],inFloats
            [6],inFloats[7]})
247     simSetObjectName(quadricycle,'quadricycle')
248     simSetObjectSpecialProperty(quadricycle,
            sim_objectspecialproperty_measurable+
            sim_objectspecialproperty_collidable+
            sim_objectspecialproperty_renderable+
            sim_objectspecialproperty_detectable_all)
249     simSetObjectInt32Parameter(quadricycle,
            sim_shapeintparam_static,0)
250     simSetObjectInt32Parameter(quadricycle,
            sim_shapeintparam_respondable,1)
251     simSetShapeColor(quadricycle,NULL,
            sim_colorcomponent_ambient_diffuse,{0.19,0.88,0.34})
252
253     rear_left_wheel=simCreatePureShape(2,01110,{inFloats[8],
            inFloats[9],inFloats[10]},inFloats[11],NULL)
254     simSetObjectPosition(rear_left_wheel,-1,{inFloats[12],
            inFloats[13],inFloats[14]})
255     simSetObjectOrientation(rear_left_wheel,-1,{0,-math.pi/2,0})
256     simSetObjectName(rear_left_wheel,'
            quadricycle_rear_left_wheel')
257     simSetObjectSpecialProperty(rear_left_wheel,
            sim_objectspecialproperty_measurable+
            sim_objectspecialproperty_collidable+
            sim_objectspecialproperty_renderable+
            sim_objectspecialproperty_detectable_all)
```

```
258     simSetObjectInt32Parameter(rear_left_wheel,
            sim_shapeintparam_static,0)
259     simSetObjectInt32Parameter(rear_left_wheel,
            sim_shapeintparam_respondable,1)
260     simSetShapeColor(rear_left_wheel,NULL,
            sim_colorcomponent_ambient_diffuse,{0.15,0.15,0.15})
261     simSetShapeMaterial(rear_left_wheel,wheelMaterial)
262
263     rear_right_wheel=simCreatePureShape(2,01110,{inFloats[8],
            inFloats[9],inFloats[10]},inFloats[11],NULL)
264     simSetObjectPosition(rear_right_wheel,-1,{inFloats[15],
            inFloats[16],inFloats[17]})
265     simSetObjectOrientation(rear_right_wheel,-1,{0,-math.pi
            /2,0})
266     simSetObjectName(rear_right_wheel,'
            quadricycle_rear_right_wheel')
267     simSetObjectSpecialProperty(rear_right_wheel,
            sim_objectspecialproperty_measurable+
            sim_objectspecialproperty_collidable+
            sim_objectspecialproperty_renderable+
            sim_objectspecialproperty_detectable_all)
268     simSetObjectInt32Parameter(rear_right_wheel,
            sim_shapeintparam_static,0)
269     simSetObjectInt32Parameter(rear_right_wheel,
            sim_shapeintparam_respondable,1)
270     simSetShapeColor(rear_right_wheel,NULL,
            sim_colorcomponent_ambient_diffuse,{0.15,0.15,0.15})
271     simSetShapeMaterial(rear_right_wheel,wheelMaterial)
272
273     front_left_wheel=simCreatePureShape(2,01110,{inFloats[8],
            inFloats[9],inFloats[10]},inFloats[11],NULL)
274     simSetObjectPosition(front_left_wheel,-1,{inFloats[18],
            inFloats[19],inFloats[20]})
275     simSetObjectOrientation(front_left_wheel,-1,{0,-math.pi
            /2,0})
276     simSetObjectName(front_left_wheel,'
            quadricycle_front_left_wheel')
277     simSetObjectSpecialProperty(front_left_wheel,
            sim_objectspecialproperty_measurable+
            sim_objectspecialproperty_collidable+
            sim_objectspecialproperty_renderable+
            sim_objectspecialproperty_detectable_all)
278     simSetObjectInt32Parameter(front_left_wheel,
            sim_shapeintparam_static,0)
279     simSetObjectInt32Parameter(front_left_wheel,
```

```
        sim_shapeintparam_respondable,1)
280     simSetShapeColor(front_left_wheel,NULL,
        sim_colorcomponent_ambient_diffuse,{0.15,0.15,0.15})
281     simSetShapeMaterial(front_left_wheel,wheelMaterial)
282
283     inv_front_left_wheel=simCreatePureShape(2,01110,{inFloats
        [8],inFloats[9],inFloats[10]},inFloats[11],NULL)
284     simSetObjectPosition(inv_front_left_wheel,-1,{inFloats[18],
        inFloats[19],inFloats[20]})
285     simSetObjectOrientation(inv_front_left_wheel,-1,{0,-math.pi
        /2,0})
286     simSetObjectName(inv_front_left_wheel,'
        quadricycle_inv_front_left_wheel')
287     simSetObjectInt32Parameter(inv_front_left_wheel,
        sim_shapeintparam_static,0)
288     simSetObjectInt32Parameter(inv_front_left_wheel,
        sim_shapeintparam_respondable,0)
289     simSetObjectInt32Parameter(inv_front_left_wheel,
        sim_objintparam_visibility_layer,1024)
290     simSetShapeMaterial(inv_front_left_wheel,wheelMaterial)
291
292     front_right_wheel=simCreatePureShape(2,01110,{inFloats[8],
        inFloats[9],inFloats[10]},inFloats[11],NULL)
293     simSetObjectPosition(front_right_wheel,-1,{inFloats[21],
        inFloats[22],inFloats[23]})
294     simSetObjectOrientation(front_right_wheel,-1,{0,-math.pi
        /2,0})
295     simSetObjectName(front_right_wheel,'
        quadricycle_front_right_wheel')
296     simSetObjectSpecialProperty(front_right_wheel,
        sim_objectspecialproperty_measurable+
        sim_objectspecialproperty_collidable+
        sim_objectspecialproperty_renderable+
        sim_objectspecialproperty_detectable_all)
297     simSetObjectInt32Parameter(front_right_wheel,
        sim_shapeintparam_static,0)
298     simSetObjectInt32Parameter(front_right_wheel,
        sim_shapeintparam_respondable,1)
299     simSetShapeColor(front_right_wheel,NULL,
        sim_colorcomponent_ambient_diffuse,{0.15,0.15,0.15})
300     simSetShapeMaterial(front_right_wheel,wheelMaterial)
301
302     inv_front_right_wheel=simCreatePureShape(2,01110,{inFloats
        [8],inFloats[9],inFloats[10]},inFloats[11],NULL)
303     simSetObjectPosition(inv_front_right_wheel,-1,{inFloats[21],
```

```
        inFloats[22],inFloats[23]})
304     simSetObjectOrientation(inv_front_right_wheel,-1,{0,-math.pi
        /2,0})
305     simSetObjectName(inv_front_right_wheel,'
        quadricycle_inv_rear_right_wheel')
306     simSetObjectInt32Parameter(inv_front_right_wheel,
        sim_shapeintparam_static,0)
307     simSetObjectInt32Parameter(inv_front_right_wheel,
        sim_shapeintparam_respondable,0)
308     simSetObjectInt32Parameter(inv_front_right_wheel,
        sim_objintparam_visibility_layer,1024)
309     simSetShapeMaterial(inv_front_right_wheel,wheelMaterial)
310
311     rear_left_motor=simCreateJoint(sim_joint_revolute_subtype,
        sim_jointmode_force,0,NULL,NULL,NULL)
312     simSetObjectPosition(rear_left_motor,-1,{inFloats[12],
        inFloats[13],inFloats[14]})
313     simSetObjectOrientation(rear_left_motor,-1,{0,-math.pi/2,0})
314     simSetObjectName(rear_left_motor,'
        quadricycle_rear_left_motor')
315     simSetObjectInt32Parameter(rear_left_motor,
        sim_jointintparam_motor_enabled,1)
316     simSetObjectInt32Parameter(rear_left_motor,
        sim_jointintparam_velocity_lock,1)
317     simSetObjectInt32Parameter(rear_left_motor,
        sim_objintparam_visibility_layer,1024)
318
319     rear_right_motor=simCreateJoint(sim_joint_revolute_subtype,
        sim_jointmode_force,0,NULL,NULL,NULL)
320     simSetObjectPosition(rear_right_motor,-1,{inFloats[15],
        inFloats[16],inFloats[17]})
321     simSetObjectOrientation(rear_right_motor,-1,{0,-math.pi
        /2,0})
322     simSetObjectName(rear_right_motor,'
        quadricycle_rear_right_motor')
323     simSetObjectInt32Parameter(rear_right_motor,
        sim_jointintparam_motor_enabled,1)
324     simSetObjectInt32Parameter(rear_right_motor,
        sim_jointintparam_velocity_lock,1)
325     simSetObjectInt32Parameter(rear_right_motor,
        sim_objintparam_visibility_layer,1024)
326
327     front_left_motor=simCreateJoint(sim_joint_revolute_subtype,
        sim_jointmode_force,0,NULL,NULL,NULL)
328     simSetObjectPosition(front_left_motor,-1,{inFloats[18],
```

```
        inFloats[19],inFloats[20]})
329     simSetObjectOrientation(front_left_motor,-1,{0,0,0})
330     simSetObjectName(front_left_motor,'
            quadricycle_front_left_motor')
331     simSetObjectInt32Parameter(front_left_motor,
            sim_jointintparam_motor_enabled,1)
332     simSetObjectInt32Parameter(front_left_motor,
            sim_jointintparam_ctrl_enabled,1)
333     simSetObjectInt32Parameter(front_left_motor,
            sim_objintparam_visibility_layer,1024)
334
335     front_left_wheel_axis=simCreateJoint(
            sim_joint_revolute_subtype,sim_jointmode_force,0,NULL,
            NULL,NULL)
336     simSetObjectPosition(front_left_wheel_axis,-1,{inFloats[18],
            inFloats[19],inFloats[20]})
337     simSetObjectOrientation(front_left_wheel_axis,-1,{0,-math.pi
            /2,0})
338     simSetObjectName(front_left_wheel_axis,'
            quadricycle_front_left_wheel_axis')
339     simSetObjectInt32Parameter(front_left_wheel_axis,
            sim_objintparam_visibility_layer,1024)
340
341     front_right_motor=simCreateJoint(sim_joint_revolute_subtype,
            sim_jointmode_force,0,NULL,NULL,NULL)
342     simSetObjectPosition(front_right_motor,-1,{inFloats[21],
            inFloats[22],inFloats[23]})
343     simSetObjectOrientation(front_right_motor,-1,{0,0,0})
344     simSetObjectName(front_right_motor,'
            quadricycle_front_right_motor')
345     simSetObjectInt32Parameter(front_right_motor,
            sim_jointintparam_motor_enabled,1)
346     simSetObjectInt32Parameter(front_right_motor,
            sim_jointintparam_ctrl_enabled,1)
347     simSetObjectInt32Parameter(front_right_motor,
            sim_objintparam_visibility_layer,1024)
348
349     front_right_wheel_axis=simCreateJoint(
            sim_joint_revolute_subtype,sim_jointmode_force,0,NULL,
            NULL,NULL)
350     simSetObjectPosition(front_right_wheel_axis,-1,{inFloats
            [21],inFloats[22],inFloats[23]})
351     simSetObjectOrientation(front_right_wheel_axis,-1,{0,-
            math.pi/2,0})
352     simSetObjectName(front_right_wheel_axis,'
```

```
        quadricycle_front_right_wheel_axis')
353     simSetObjectInt32Parameter(front_right_wheel_axis,
            sim_objintparam_visibility_layer,1024)
354
355     prox_sensor=simCreateProximitySensor(
            sim_proximitysensor_disc_subtype,
            sim_objectspecialproperty_detectable_ultrasonic
            ,0000000000,{16,32,0,0,0,0,0,0},{0.0,inFloats[24],0.0,0
            .05,0.0,0.0,0.0,0.1,0.0,math.pi/2,0.0,0.0,0.01,0.0,0.0},
            NULL)
356     simSetObjectPosition(prox_sensor,-1,{inFloats[25],inFloats
            [26],inFloats[27]})
357     simSetObjectOrientation(prox_sensor,-1,{-math.pi/2,0,0})
358     simSetObjectName(prox_sensor,'quadricycle_prox_sensor')
359
360     left_vision_sensor=simCreateVisionSensor
            (00000000,{1,1,0,0},{0.01,inFloats[28],0.01,0.01,0.01,0
            .01,0.0,0.0,0.0,0.0,0.0},NULL)
361     simSetObjectPosition(left_vision_sensor,-1,{inFloats[29],
            inFloats[30],inFloats[31]})
362     simSetObjectName(left_vision_sensor,'
            quadricycle_left_vision_sensor')
363     simSetObjectOrientation(left_vision_sensor,-1,{math.pi,0,0})
364
365     right_vision_sensor=simCreateVisionSensor
            (00000000,{1,1,0,0},{0.01,inFloats[28],0.01,0.01,0.01,0
            .01,0.0,0.0,0.0,0.0,0.0},NULL)
366     simSetObjectPosition(right_vision_sensor,-1,{inFloats[32],
            inFloats[33],inFloats[34]})
367     simSetObjectName(right_vision_sensor,'
            quadricycle_right_vision_sensor')
368     simSetObjectOrientation(right_vision_sensor,-1,{math.pi
            ,0,0})
369
370     middle_vision_sensor=simCreateVisionSensor
            (00000000,{1,1,0,0},{0.01,math.sqrt(2*5.00^2),0.01,0.01,0
            .01,0.01,0.0,0.0,0.0,0.0,0.0},NULL)
371     simSetObjectPosition(middle_vision_sensor,-1,{inFloats[35],
            inFloats[36],inFloats[37]})
372     simSetObjectName(middle_vision_sensor,'
            quadricycle_middle_vision_sensor')
373     simSetObjectOrientation(middle_vision_sensor,-1,{-math.pi
            /2,0,0})
374
375     simSetObjectParent(rear_left_wheel,rear_left_motor,true)
```

```
376     simSetObjectParent(rear_right_wheel,rear_right_motor,true)
377     simSetObjectParent(front_left_wheel,front_left_wheel_axis,
            true)
378     simSetObjectParent(front_left_wheel_axis,
            inv_front_left_wheel,true)
379     simSetObjectParent(inv_front_left_wheel,front_left_motor,
            true)
380     simSetObjectParent(front_right_wheel,front_right_wheel_axis,
            true)
381     simSetObjectParent(front_right_wheel_axis,
            inv_front_right_wheel,true)
382     simSetObjectParent(inv_front_right_wheel,front_right_motor,
            true)
383     simSetObjectParent(rear_left_motor,quadricycle,true)
384     simSetObjectParent(rear_right_motor,quadricycle,true)
385     simSetObjectParent(front_left_motor,quadricycle,true)
386     simSetObjectParent(front_right_motor,quadricycle,true)
387     simSetObjectParent(prox_sensor,quadricycle,true)
388     simSetObjectParent(left_vision_sensor,quadricycle,true)
389     simSetObjectParent(right_vision_sensor,quadricycle,true)
390     simSetObjectParent(middle_vision_sensor,quadricycle,true)
391
392
393     return {},{},{},''
394 end
395
396
397 --Función que crea un obstáculo en la escena a partir de los
        parámetros recibidos desde Matlab. Para una información
398 --detallada acerca de los argumentos de cada función API,
        consultar el listado 'Regular API functions'.
399 CreateObstacle_function=function(inInts,inFloats,inStrings,
        inBuffer)
400
401     obstacle=simCreatePureShape(inInts[1],01110,{inFloats[1],
            inFloats[2],inFloats[3]},inFloats[7],NULL)
402     simSetObjectPosition(obstacle,-1,{inFloats[4],inFloats[5],
            inFloats[6]})
403     simSetObjectName(obstacle,inStrings[1])
404     simSetObjectSpecialProperty(obstacle,
            sim_objectspecialproperty_measurable+
            sim_objectspecialproperty_collidable+
            sim_objectspecialproperty_renderable+
            sim_objectspecialproperty_detectable_all)
405     simSetObjectInt32Parameter(obstacle,
```

```
        sim_shapeintparam_respondable,1)
406     simSetShapeColor(obstacle,NULL,
            sim_colorcomponent_ambient_diffuse,{0.95,0.95,0.95})
407     return {obstacle},{},{},''
408 end
409
410
411 --Función que crea una baliza en la escena a partir de los
        parámetros recibidos desde Matlab. Para una información
412 --detallada acerca de los argumentos de cada función API,
        consultar el listado 'Regular API functions'.
413 CreateMarker_function=function(inInts,inFloats,inStrings,
        inBuffer)
414
415     marker=simCreatePureShape(2,01110,{0.25,0.25,0.5},1.0,NULL)
416     simSetObjectPosition(marker,-1,{inFloats[1],inFloats[2],
            inFloats[3]})
417     simSetObjectName(marker,inStrings[1])
418     simSetObjectSpecialProperty(marker,
            sim_objectspecialproperty_measurable+
            sim_objectspecialproperty_collidable+
            sim_objectspecialproperty_renderable+
            sim_objectspecialproperty_detectable_all)
419     simSetObjectInt32Parameter(marker,
            sim_shapeintparam_respondable,1)
420     simSetShapeColor(marker,NULL,
            sim_colorcomponent_ambient_diffuse,{0.0,1.0,0.0})
421     return {marker},{},{},''
422 end
423
424
425 --Función que posiciona un obstáculo o baliza en la escena a
        partir de los parámetros recibidos desde Matlab. Para una
        información
426 --detallada acerca de los argumentos de la función API,
        consultar el listado 'Regular API functions'.
427 Position_function=function(inInts,inFloats,inStrings,inBuffer)
428     simSetObjectPosition(inInts[1],-1,{inFloats[1],inFloats[2],
            inFloats[3]})
429     return {},{},{},''
430 end
431
432
433 --Función que oculta los sensores que no se utilizan en la
        escena a partir de los parámetros recibidos desde Matlab.
```

```
    Para una información
--detallada acerca de los argumentos de cada función API,
    consultar el listado 'Regular API functions'.
Hide_function=function(inInts,inFloats,inStrings,inBuffer)
    if (inInts[1]==1) then
        simSetObjectInt32Parameter(left_vision_sensor,
            sim_objintparam_visibility_layer,1024)
        simSetObjectInt32Parameter(right_vision_sensor,
            sim_objintparam_visibility_layer,1024)
        simSetObjectInt32Parameter(middle_vision_sensor,
            sim_objintparam_visibility_layer,1024)
    elseif (inInts[1]==2) then
        simSetObjectInt32Parameter(middle_vision_sensor,
            sim_objintparam_visibility_layer,1024)
    elseif (inInts[1]==3) then
        simSetObjectInt32Parameter(left_vision_sensor,
            sim_objintparam_visibility_layer,1024)
        simSetObjectInt32Parameter(right_vision_sensor,
            sim_objintparam_visibility_layer,1024)
    end
    return {},{},{},''
end
```