



**Universidad  
Rey Juan Carlos**

**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA**

**GRADO EN INGENIERIA INFORMATICA**

**Curso Académico 2017/2018**

**Trabajo Fin de Grado**

**DESARROLLO DE UNA APLICACIÓN PARA LA  
SIMULACIÓN DE UN ROBOT MÓVIL CON  
DIRECCIONAMIENTO DIFERENCIAL**

**Autor:** David Vacas Miguel

**Tutor:** Alberto Herrán González

# Índice

<b>ÍNDICE.....</b>	<b>2</b>
<b>AGRADECIMIENTOS.....</b>	<b>3</b>
<b>RESUMEN.....</b>	<b>4</b>
<b>CAPÍTULO 1 INTRODUCCIÓN .....</b>	<b>5</b>
1. MOTIVACIÓN .....	5
2. OBJETIVOS.....	7
3. ESTADO DEL ARTE.....	8
4. ESTRUCTURA DE LA MEMORIA .....	8
<b>CAPÍTULO 2 ROBÓTICA MÓVIL .....</b>	<b>10</b>
1. ROBÓTICA MÓVIL.....	10
2. VEHÍCULOS CON RUEDAS .....	10
3. MODELO FÍSICO DIRECCIONAMIENTO DIFERENCIAL.....	13
4. NAVEGACIÓN AUTÓNOMA .....	16
<b>CAPÍTULO 3 ENTORNO TECNOLÓGICO .....</b>	<b>19</b>
1. OPENGL .....	19
2. ETAPA GEOMÉTRICA.....	21
3. QT .....	26
4. METODOLOGÍA ÁGIL.....	28
<b>CAPÍTULO 4 DESCRIPCIÓN DE LA APLICACIÓN.....</b>	<b>30</b>
1. ZONA IZQUIERDA: VIEWPORT .....	30
2. ZONA DERECHA: CAMPOS DE ENTRADA DE DATOS .....	32
3. CASOS DE USO.....	36
<b>CONCLUSIONES.....</b>	<b>42</b>
<b>BIBLIOGRAFÍA.....</b>	<b>43</b>

## Agradecimientos

Me gustaría agradecer a la universidad por el conocimiento recibido y en especial a mi tutor, Alberto Herrán, por su ayuda y apoyo durante el desarrollo del proyecto.

# Resumen

En este TFG se ha desarrollado una aplicación con la que poder simular y visualizar el funcionamiento de un robot móvil con direccionamiento diferencial cuando trata de seguir el recorrido marcado por una línea negra sobre un fondo blanco.

Para ello, se ha comenzado estudiando la mecánica de tal sistema a través de las ecuaciones que relacionan el giro de los motores con la posición y orientación del robot. A continuación, se han analizado las diferentes tecnologías disponibles para la implementación de la aplicación, habiéndose elegido Qt y OpenGL. Dicha aplicación, tiene implementadas tanto la simulación del comportamiento del robot como su visualización sobre el circuito, además de la interacción con el usuario final mediante inputs en la aplicación. Finalmente, se han implementado opciones adicionales para la mejora de la experiencia del usuario y de la aplicación, como son los distintos tipos de visualización, la posibilidad de realizar zoom, ...

# Capítulo 1 Introducción

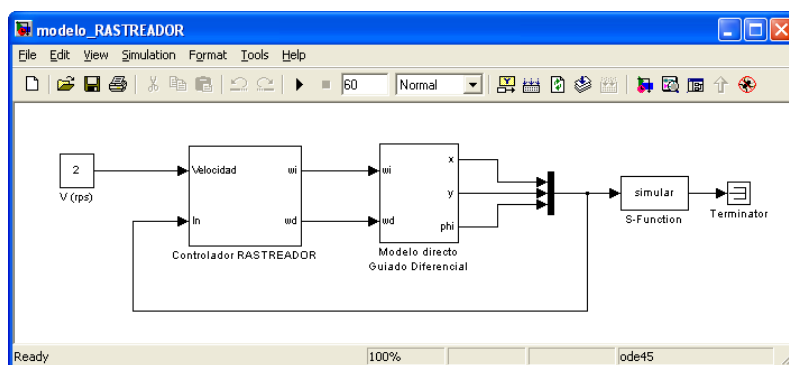
## 1. Motivación

Este trabajo nace motivado por un intento de mejorar una de las herramientas que se utilizaban en la antigua asignatura “Robótica” de la titulación “Ingeniería Técnica en Informática de Sistemas” impartida en el antiguo Centro de Estudios Superiores Felipe Segundo, que actualmente constituye el Campus de Aranjuez de la Universidad Rey Juan Carlos.

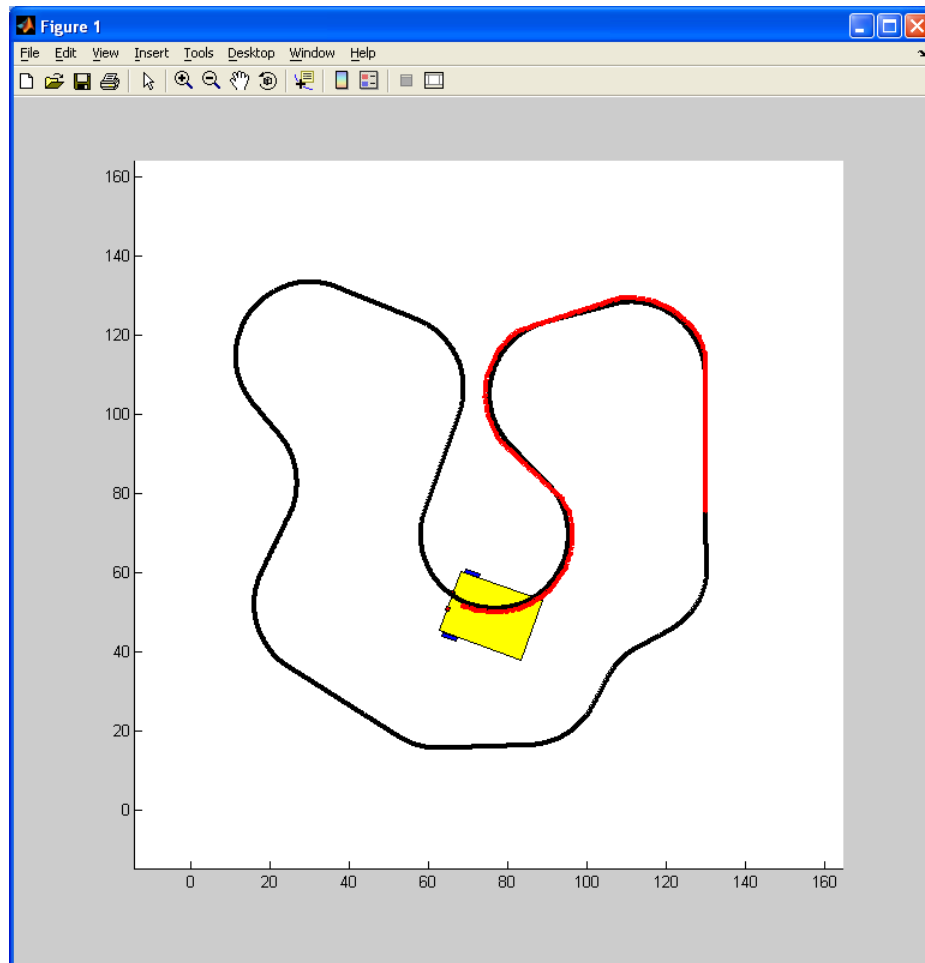
En esta asignatura los alumnos, en grupos, construyen robots de distintos tipos: sigue líneas, velocistas, etc. Al final del curso el profesor realiza una competición en la que se usan los robots contruidos para ver cuál es el más rápido realizando diferentes tareas. En concreto, para el robot sigue-líneas, se cronometra el tiempo que tarda cada robot en recorrer el circuito. Este tiempo depende de ciertos parámetros que los alumnos eligen a la hora de crear el robot.

El objetivo de este proyecto es desarrollar una aplicación que permita simular y visualizar la dinámica de dicho robot bajo diferentes configuraciones (parámetros, circuitos, etc) sirviendo de banco de pruebas con el que analizar el comportamiento del mismo antes de su construcción real.

Actualmente se dispone de un desarrollo previo en MATLAB-Simulink. En la figura 1 se puede observar el esquema de dicho desarrollo, en el que se puede ver los diferentes bloques que se utilizan en la aplicación, estos bloques son: el responsable de controlar la velocidad de las ruedas que decide si alguna de las ruedas debe pararse y el responsable de los cálculos de la posición, orientación y pintado del robot. La figura 2 muestra el programa en ejecución.



**Figura 1.** Modelo del desarrollo hecho en MATLAB-Simulink.



**Figura 2.** Vista de la simulación desarrollada en MATLAB-Simulink.

Sin embargo, este tiene varios inconvenientes:

- a) Se debe tener instalado MATLAB-Simulink para su uso.
- b) No se permite cambiar los parámetros del robot de una forma sencilla ya que estos se encuentran almacenados en un vector cuya estructura se debe conocer. Esto se puede ver en la figura 3, estos números indican el largo del robot, su ancho, la separación entre sensores, ... respectivamente.
- c) No ofrece todas las funcionalidades requeridas (estas se especifican más adelante en el punto 2 Objetivos).

```

>> robot

robot =

    22    16     3     2    -2     3     2     3     1
  
```

**Figura 3.** Vector con los parámetros del robot.

Por ello, se ha desarrollado una aplicación (integrando las tecnologías C++, Qt y OpenGL) de fácil uso para el alumno, y cuyo único requisito para su uso es el de tener instaladas en el sistema las librerías necesarias para hacerlo funcionar.

## 2. Objetivos

Una vez se han visto las carencias de la aplicación original, los objetivos para subsanarlas, además de otros objetivos adicionales son los siguientes:

- **Eliminar la necesidad de tener software instalado:** con esta aplicación no es necesario disponer de ningún software adicional instalado, más allá de las librerías necesarias para la aplicación, por lo que es más fácil su descarga y utilización inicial.
- **Facilidad a la hora de realizar cambios:** La aplicación debe tener una simple interfaz user-friendly en la que se puedan cambiar fácilmente los parámetros del robot, de forma sencilla y natural. Así los alumnos podrán jugar con el simulador antes de la construcción real del robot con lo que pueden experimentar haciendo cambios en las medidas para poder observar qué cambios se realizan en el movimiento del robot y por lo tanto ir aprendiendo con la visualización de este
- **Realización de pruebas con facilidad:** debido a lo costoso de realizar un robot y la necesidad de crear un circuito por el cual el robot tenga que correr se hace complicado poder probar el robot creado, o diferentes opciones para el mismo.

Con esta aplicación se podrán realizar la cantidad de pruebas que se desee puesto que en unos segundos se podrá tener un robot simulado corriendo por el circuito dado sin necesidad de crear físicamente los mismos.

Por lo tanto, los requisitos que debe tener la aplicación son:

- Facilidad de cambiar los parámetros del robot.
- Visualización de la simulación del robot de forma correcta.
- Integración de la interfaz y de la simulación en la aplicación.
- Interfaz user-friendly, fácil de utilizar para todos.

- Diseño simple de la simulación para poder visualizar mejor los movimientos de tu robot.

### 3. Estado del arte

En cuanto a las herramientas que se pueden encontrar en el mercado para simular un robot de este tipo se pueden encontrar de dos tipos:

- Aplicaciones creadas por una persona que muestran cómo realizarlas o dejan la aplicación en un repositorio público (pocas de estas realizan esto último). Estas aplicaciones suelen tener dos problemas: la dificultad para su descarga y utilización, que no se hallan preparadas para su uso puesto que están en las IDE correspondientes y la interfaz no es fácil de usar, en caso de que haya interfaz y no se tenga que cambiar los parámetros por código.
- Aplicaciones creadas por empresas. En este caso muchas de estas aplicaciones tienen una carencia en la UI, no se puede cambiar de manera sencilla los parámetros del robot. Lo bueno que tenía la herramienta encontrada es que no solo servía para un tipo de robot, sino que incluía en la aplicación varios tipos de robots. Ejemplos de este tipo son: Hemero, una herramienta en Matlab-simulink para la enseñanza de la robótica. El problema que posee esta herramienta es la falta de parte gráfica; RoboWorks, se trata de una aplicación de modelado, simulación y animación de sistemas físicos, sin embargo, se utiliza solo para robots manipuladores<sup>14</sup>. RoboDK, sucede lo mismo que con la anterior, se trata de una aplicación para simular robots, sin embargo, está basada en robots manipuladores<sup>15</sup>.

### 4. Estructura de la memoria

A continuación se describe brevemente la estructura del resto del documento:

En el Capítulo 2, **Robótica móvil**, se comienza explicando qué es un robot móvil, y en concreto, un vehículo con ruedas y sus diferentes configuraciones. A continuación, se describe la configuración seleccionada para la aplicación del direccionamiento diferencial, y se describen las ecuaciones necesarias para simular un robot de este tipo. Finalmente, se incluyen los elementos necesarios para implementar la navegación autónoma a un vehículo de este tipo.



En el Capítulo 3, **Entorno tecnológico**, se habla sobre las diferentes tecnologías y librerías usadas: Qt, OpenGL, freeglut y GLM. De estas se comenta su utilización en el proyecto. También se explica de manera breve el cauce gráfico, y la etapa geométrica con un poco más de detalle. Además, también se describen otras tecnologías menos relevantes para la aplicación, pero de gran importancia para el desarrollo del proyecto como son Trello y GitHub.

En el Capítulo 4, **Descripción de la aplicación**, se expone la aplicación al completo, comenzando por detalles sobre la simulación y visualización del robot, pasando por la descripción de la interfaz y finalizando con casos de uso de la aplicación.

Finalmente, en el Capítulo 5, **Conclusiones**, se presentan tanto las conclusiones del trabajo como su posible ampliación y mejora futura.

# Capítulo 2 Robótica móvil

## 1. Robótica móvil

Los robots móviles son robots que tienen la capacidad de moverse a través de cualquier entorno. Estos normalmente son controlados por software y usan sensores y otros equipos para identificar la zona de su alrededor<sup>1</sup>.

Los robots móviles se pueden diferenciar en dos tipos, autónomos y no autónomos. Los robots móviles autónomos pueden explorar el entorno sin ningún control externo a él, al contrario que los no autónomos, que necesitan algún tipo de sistema de guía para moverse.

## 2. Vehículos con ruedas

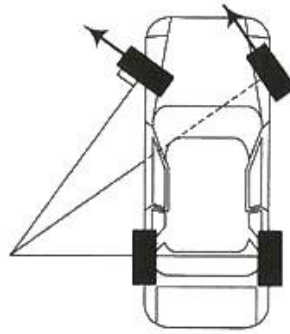
Los vehículos con ruedas son un tipo de robot móvil los cuales proporcionan una solución simple y eficiente para conseguir movilidad sobre terrenos duros y libres de obstáculos, con los que se permite conseguir velocidades más o menos altas.

Su limitación más importante es el deslizamiento al impulsarse, además dependiendo del tipo de terreno, puede aparecer deslizamiento y vibraciones en el mismo.

Otro problema que tienen este tipo de vehículos se halla en que no es posible modificar la estabilidad para adaptarse al terreno, excepto en configuraciones muy especiales, lo que limita los terrenos sobre los que es aceptable el vehículo.

Los vehículos con ruedas emplean distintos tipos de locomoción que les da unas características y propiedades diferentes entre ellos en cuanto a eficiencia energética, dimensiones, maniobrabilidad y carga útil. A continuación, se pasa a mencionar algunas de estas configuraciones:

- **Ackerman:** Es el habitual de los vehículos de cuatro ruedas. Las ruedas delanteras son las que se ocupan del giro, sin embargo, la rueda interior gira un poco más que la exterior lo que hace que se elimine el deslizamiento. El centro de rotación del vehículo se haya en el corte de las prolongaciones de las ruedas delanteras y las ruedas traseras como se muestra en la figura 4.



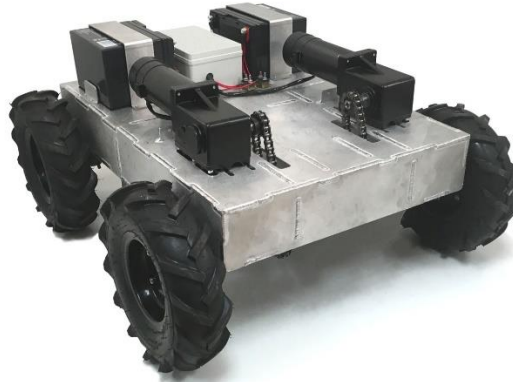
**Figura 4.** Configuración ackerman y centro de rotación.

- **Triciclo:** Se compone por tres ruedas, una delantera central y dos traseras. La rueda delantera actúa tanto para tracción como para direccionamiento, las ruedas traseras son pasivas. Tiene una mayor maniobrabilidad que la configuración anterior, sin embargo, posee una menor estabilidad sobre terrenos difíciles. El centro de gravedad tiende a perderse cuando se desplaza por una pendiente, perdiendo tracción. El centro de rotación se puede calcular igual que en la configuración anterior, es decir, prolongando el eje de la rueda delantera y las traseras y este se encuentra en el punto de corte.



**Figura 5.** Configuración de triciclo.

- **Skid Steer:** Varias ruedas a cada lado del vehículo que actúan de forma simultánea. La dirección del vehículo resulta de combinar las velocidades de las ruedas izquierdas con las de la derecha.



**Figura 6.** Configuración Skid Steer.

- **Pistas de deslizamiento:** Funcionalmente análogo a la configuración Skid Steer. Tanto la impulsión como el direccionamiento se realiza mediante pistas de desplazamiento, estas pistas actúan de forma similar a como lo harían ruedas de gran diámetro. Esta configuración es útil en terrenos irregulares.



**Figura 7.** Pistas de desplazamiento.

- **Síncronas:** Se trata de una configuración en la que todas las ruedas actúan de forma simultánea y, por lo tanto, giran de forma síncrona.
- **Direccionamiento diferencial:** Esta configuración es la que utiliza el robot sigue líneas. En este sistema las ruedas se sitúan de forma paralela y no realizan giros. El direccionamiento diferencial viene dado por la diferencia de velocidades de las ruedas laterales. La tracción se consigue con esas mismas ruedas.

Adicionalmente, existen una o más ruedas para el soporte. En la figura 8 se muestra una imagen de dicho esquema.



**Figura 8:** Robot con direccionamiento diferencial.

### 3. Modelo físico direccionamiento diferencial

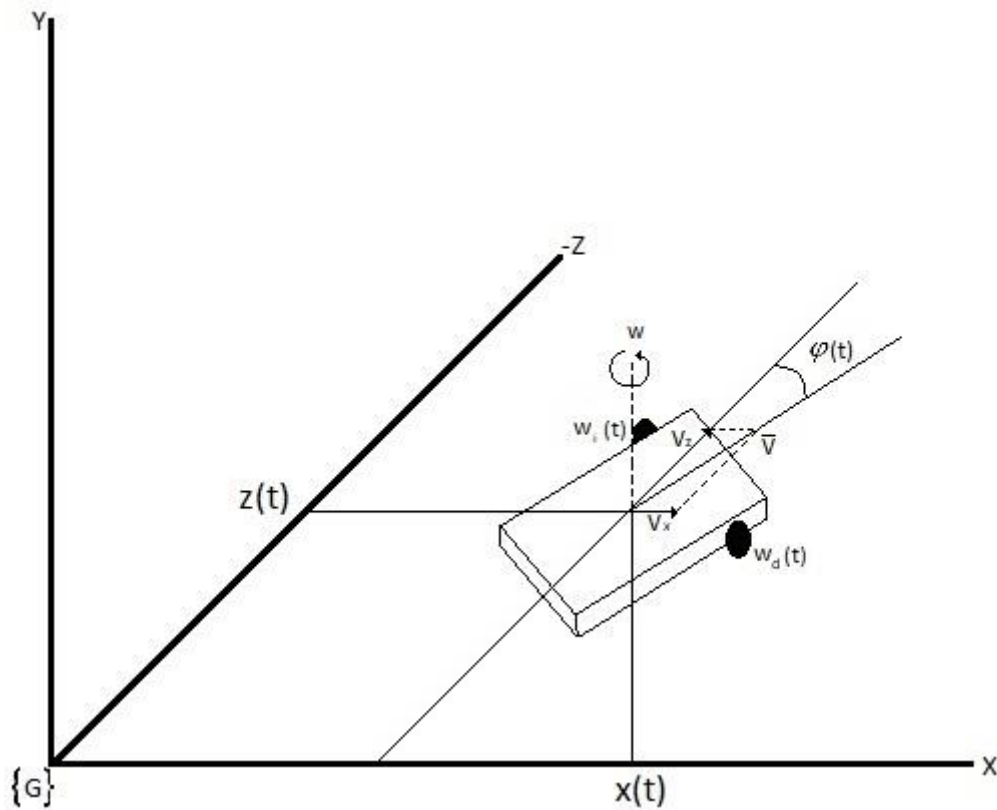
Para poder realizar la simulación gráfica del robot y pintarlo se debe conocer primeramente el estado del sistema. El estado del robot viene dado por su posición y orientación. Puesto que el robot se va a mover, se necesita saber el estado del mismo en los diferentes instantes de tiempo, por lo tanto, se debe definir un modelo de cambios de estado, es decir, una ecuación que a partir del estado actual en  $(t)$  y unas entradas en  $(t)$  se pase al estado en el siguiente instante  $(t+1)$ . En la ecuación X se puede observar la ecuación que define el sistema, siendo  $\bar{s}$  el estado y  $\bar{r}$  las entradas. Asimismo, en esta misma ecuación se muestran las variables que definen el estado del robot en un instante  $(t)$  dado, posición  $(x(t), z(t))$  y su orientación  $\varphi(t)$  en ese instante:

$$\begin{aligned}
 \bar{s}(t+1) &= f(\bar{s}(t), \bar{r}(t)) \\
 \bar{s} &= (x(t), z(t), \varphi(t))
 \end{aligned}
 \tag{1}$$

Para obtener la ecuación de cambio de estado para un robot con direccionamiento diferencial se deben definir las entradas como las velocidades izquierda y derecha. Por lo tanto, la ecuación de las entradas resultante para un robot con direccionamiento diferencial, siendo  $w_i$  la velocidad de la rueda izquierda y  $w_d$  la velocidad de la rueda derecha, es la siguiente:

$$\bar{r} = (w_i(t), w_d(t))
 \tag{2}$$

En la figura 9 se pueden observar los diferentes datos de entrada y del sistema puestos sobre un robot con direccionamiento diferencial.



**Figura 9.** Sistema 3D de un robot con direccionamiento diferencial.

Una vez obtenido esto vamos a realizar el cálculo de las ecuaciones específicas.

A partir de la figura 9, si el vehículo tiene una velocidad de desplazamiento  $v$  y de rotación  $w$ , se obtiene que:

$$\begin{aligned}
 v_x &= v \cdot \sin(\varphi) \\
 v_z &= v \cdot \cos(\varphi) \\
 v_\varphi &= w
 \end{aligned} \tag{3}$$

La derivada de una función puede aproximarse por el cociente incremental:

$$v_x = \frac{dx}{dt} = \frac{x(t+1) - x(t)}{dt} \tag{4}$$

Esto se conoce como derivada discreta hacia adelante, pero también puede aproximarse con el cociente incremental hacia atrás, la aproximación centrada, u otras aproximaciones más complicadas.

El resultado es que, con una ecuación de este tipo, la nueva coordenada  $x$  {en  $t+1$ } se puede calcular a partir de la anterior (en  $t$ ) mediante la ecuación:

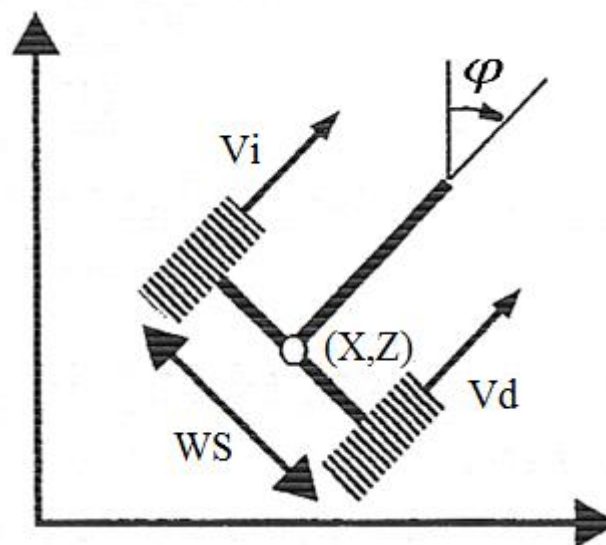
$$x(t+1) = x(t) + v_x \cdot \Delta t \quad (5)$$

Aplicando el mismo resultado al resto de coordenadas del modelo cinemático directo, se obtiene:

$$\begin{aligned} x(t+1) &= x(t) + v_x \cdot \Delta t \\ z(t+1) &= z(t) + v_z \cdot \Delta t \\ \varphi(t+1) &= \varphi(t) + v_\varphi \cdot \Delta t \end{aligned} \quad (6)$$

Por tanto, si se dispone de las coordenadas  $(v_x, v_z, v_\varphi)$  en cada instante de un determinado horizonte temporal, se puede calcular la nueva posición y orientación del robot en dicho horizonte.

Note que para especificar la configuración hay que indicar los valores de las tres variables  $(x, z, \varphi)$ , siendo las variables de control las velocidades de las ruedas laterales.



**Figura 10.** Locomoción mediante guiado diferencial.

Sean  $w_i$  y  $w_d$  las velocidades de giro de las ruedas izquierda y derecha, respectivamente. Si el radio de la rueda es  $WR$ , las velocidades lineales correspondientes son  $v_i = w_i \cdot WR$  y  $v_d = w_d \cdot WR$ . En este caso, la velocidad lineal y velocidad angular correspondientes en el modelo vienen dadas por:

$$\begin{aligned}
 v &= \frac{v_d + v_i}{2} = \frac{(w_d + w_i) \cdot WR}{2} \\
 w &= \frac{v_d - v_i}{WS} = \frac{(w_d - w_i) \cdot WR}{WS}
 \end{aligned} \tag{7}$$

Sustituyendo estas expresiones en las obtenidas a partir de la Figura 10, se obtienen las velocidades de las coordenadas del robot en el sistema  $\{G\}$  a partir de la velocidad de giro de cada rueda:

$$\begin{aligned}
 v_x &= \frac{-(w_d + w_i) \cdot WR}{2} \cdot \sin(\varphi) = -(w_d + w_i) \cdot \frac{WR \cdot \sin(\varphi)}{2} \\
 v_z &= \frac{(w_d + w_i) \cdot WR}{2} \cdot \cos(\varphi) = (w_d + w_i) \cdot \frac{WR \cdot \cos(\varphi)}{2} \\
 v_\varphi &= \frac{(w_d - w_i) \cdot WR}{WS} = (w_d - w_i) \cdot \frac{WR}{WS}
 \end{aligned} \tag{8}$$

Finalmente, utilizando el modelo discreto, se obtiene:

$$\begin{aligned}
 x(t+1) &= x(t) - (w_d + w_i) \cdot \frac{WR \cdot \sin(\varphi(t))}{2} \cdot \Delta t \\
 z(t+1) &= z(t) + (w_d + w_i) \cdot \frac{WR \cdot \cos(\varphi(t))}{2} \cdot \Delta t \\
 \varphi(t+1) &= \varphi(t) + (w_d - w_i) \cdot \frac{WR}{WS} \cdot \Delta t
 \end{aligned} \tag{9}$$

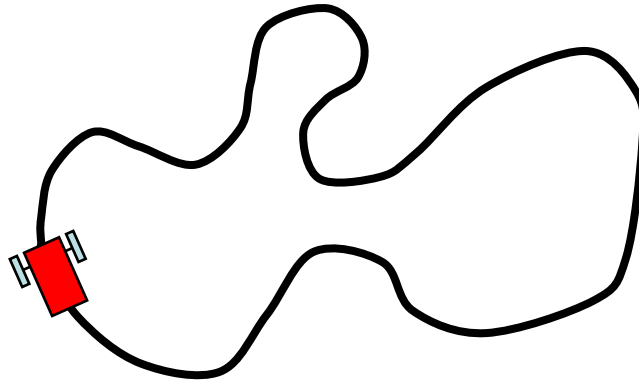
Estas ecuaciones son las que nos sirven para poder calcular el estado del robot en el instante siguiente, es decir, la posición y la orientación en  $(t+1)$ . Las variables necesarias para poder calcular este son: el estado anterior ya sea su posición  $x(t), z(t)$  u orientación  $\varphi(t)$ , el radio  $WR$ , la separación  $WS$  y la velocidad de las ruedas  $w_d, w_i$ .

## 4. Navegación autónoma

El robot sigue líneas que se ha implementado realiza su movimiento de manera autónoma, se coloca el robot sobre un fondo blanco con una línea negra que representa el circuito, como se muestra en la figura 11, y este deberá recorrer el circuito sin salirse del mismo. Esto se puede realizar gracias a dos sensores que son

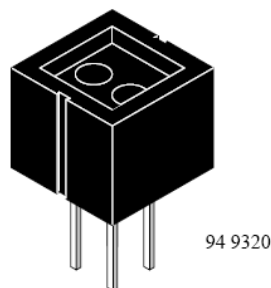


implantados en la parte delantera del robot los cuales son responsables de la detección de la línea del circuito. En función de lo que estos sensores recojan (están sobre el circuito o no) el robot realiza cambios en la velocidad de sus ruedas resultando en un movimiento recto, rotatorio hacia la izquierda o rotatorio hacia la derecha.



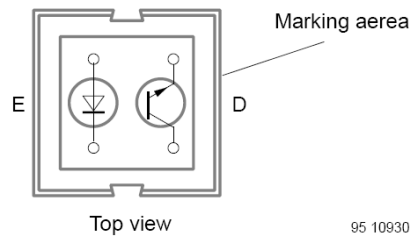
**Figura 11.** Colocación del robot en un circuito negro sobre fondo blanco.

Los sensores que se usan en este tipo de robots son sensores CNY70, los cuales se muestran en la Figura 12.



**Figura 12.** Sensor CNY70.

Estos son sensores ópticos reflexivos de corto alcance basados en un diodo de emisión de luz infrarroja y un receptor formado por un fototransistor que ambos apuntan en la misma dirección, esta estructura de forma simplificada se puede observar en la figura 13. Cuando el sensor se haya sobre una línea negra la luz es absorbida y el fototransistor envía una señal (ya sea alta o baja dependiendo del montaje del sensor), sin embargo, cuando se haya sobre fondo blanco la luz es reflejada y por lo tanto el fototransistor envía la señal contraria a la enviada al estar sobre negro.



**Figura 13.** Estructura simplificada del sensor CNY70.

Para implementar dicho comportamiento en el simulador se ha seguido la siguiente lógica:

$S_i$  -> sensor izquierdo. |  $S_d$  -> sensor derecho. |  $w$  -> velocidad ruedas.

#### Algoritmo sensor

```
if( $S_i$  = circuito)
```

```
     $w_i = 0$ ;
```

```
if( $S_d$  = circuito)
```

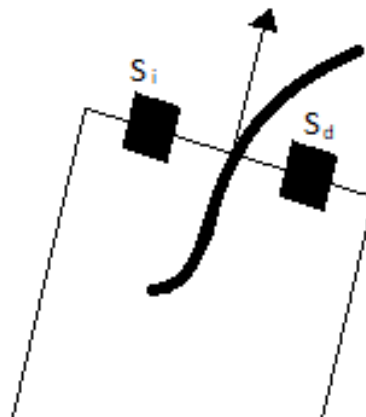
```
     $w_d = 0$ ;
```

#### Ecuacion de estado

```
[...]
```

```
 $w_i = w$ ;
```

```
 $w_d = w$ ;
```



**Figura 14.** Posición sensores sobre robot.

Esta es una de las posibles implementaciones, otras opciones son: la rueda en vez de frenar realiza el giro hacia atrás  $w_i = -w$  o disminuye la velocidad en vez de frenar

$$w_i = w - \Delta w$$

## Capítulo 3 Entorno tecnológico

Para el desarrollo del trabajo se han utilizado diferentes tecnologías:

- Qt para la creación de la interfaz de usuario y la ventana para la simulación del robot.
- OpenGL para implementar gráficos 3D y realizar la simulación del robot.
- Glm para el cálculo de matrices necesarias para la simulación del robot.
- Freeglut para crear modelos 3D.



### 1. OpenGL

Para la implementación de gráficos 3D existen diferentes APIs que se pueden utilizar como pueden ser Direct3D (para Windows), Mantle (para tarjetas AMD), Vulkan (basado en Mantle y multiplataforma) o OpenGL. Se decidió utilizar OpenGL puesto que es la librería más conocida, pública y multiplataforma, además es fácil integrarla en la mayoría de los proyectos.

OpenGL es una API multilenguaje y multiplataforma que se utiliza para el desarrollo de aplicaciones en las que se utilicen gráficos 2D y 3D. Este te proporciona funciones con las cuales podrás realizar imágenes, animaciones, juegos, simulaciones, etc. OpenGL te permite realizar entre otras muchas cosas:

- Construir formas geométricas a partir de las primitivas que este te proporciona.
- Ubicar los objetos en la escena.
- Ubicar el punto desde el que se visualiza la escena.
- Poner color o texturas.
- Crear luces.
- Realizar la rasterización.

OpenGL tiene diferentes versiones que siguen pudiendo ser utilizadas hoy en día, a continuación, se realiza una muy breve explicación de las más relevantes:

- **OpenGL 1.X:** en las diferentes actualizaciones fueron haciendo extensiones al núcleo de la API.
- **OpenGL 2.X:** se incorporó GLSL (OpenGL Shading Language), con el cual se podía programar las etapas de transformación y rasterizado del cauce gráfico.
- **OpenGL 3.X:** en la primera etapa (OpenGL 3.0) se nombra ciertas funciones como obsoletas, que serán marcadas para ser eliminadas en futuras versiones (la mayoría de ellas en la versión 3.1).
- **OpenGL 4.X:** actualmente la última versión de OpenGL (4.6) lanzada en 2017. Se añaden una gran cantidad de funcionalidades.

A pesar de esto OpenGL tiene un problema, no es fácil crear una interfaz con la que un usuario pueda interactuar, por ello y para solucionar este problema se utiliza Qt en este proyecto.

El cauce gráfico es el conjunto de transformaciones y procesados de la imagen que se realiza a los elementos que definen la escena hasta llegar a la imagen resultante final. Actualmente la generación de estas imágenes sigue una serie de pasos ya definidos.

El cauce gráfico se divide en varios pasos o etapas que se conectan entre ellas, es decir la salida de la primera será la entrada de la segunda, y así sucesivamente. En la figura 15 podemos observar las diferentes etapas y como se conectan como hemos citado anteriormente.



**Figura 15.** Etapas del cauce gráfico.

Las tres etapas principales del cauce gráfico son: transformación/geométrica, rasterizado y sombreado.

## 2. Etapa Geométrica

En esta etapa se transforman las coordenadas de los vértices de los objetos de su sistema local a su proyección en 2D. Esto se realiza mediante cálculos de matrices en los que una vez se han realizado todos los cálculos se consiguen las coordenadas en la proyección. Para 3D, las matrices que se utilizan para los cálculos son matrices 4x4. Para obtener las coordenadas donde se encuentra cada punto del objeto dentro de la escena es necesario calcular la matriz model-view-projection (MVP), como su nombre indica, esta matriz viene dada por la multiplicación de tres matrices distintas las cuales realizan cálculos para la obtención de diferentes funcionalidades, es decir, la ecuación que se implementa en esta etapa es

$$v' = M_{projection} \cdot M_{view} \cdot M_{model} \cdot v \quad (10)$$

donde  $v'$  es el vértice proyectado visto desde la cámara y  $v$  el vértice del modelo del objeto a visualizar<sup>3</sup>.

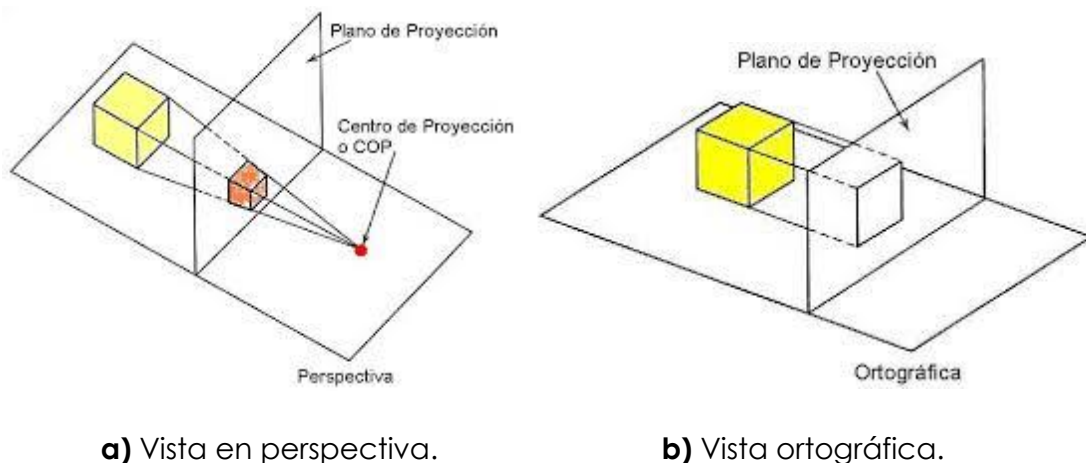
Para la creación de los modelos 3D se utiliza la librería freeglut. Estos modelos son los que contienen las coordenadas de cada uno de los vértices del mismo.

En lo referido a la matriz MVP, OpenGL no ofrece todo el control que se desea, por ello se ha utilizado la librería GLM para realizar los cálculos pertinentes sobre esta matriz. Se ha optado por la utilización de GLM como librería para el cálculo de la matriz MVP puesto que es la librería recomendada por OpenGL y por lo tanto es la más apta para este funcionamiento. A continuación, se pasa a explicar las diferentes

matrices por separado y las funciones utilizadas de la librería GLM en cada una de ellas.

- **Projection (P):** Esta matriz define como se realiza la proyección, en perspectiva u ortográfica.

La diferencia principal entre estos dos tipos de vistas (proyección y ortográfica) se basa en que la vista en perspectiva mantiene las dimensiones reales de los objetos si nos acercamos o nos alejamos de ellos, por lo que se acerca más a la vista de una persona. En las figuras 16 a) y 16 b) podemos observar como dos cámaras, una de cada tipo de vista, generan el plano de proyección a partir de los puntos de los objetos del espacio.



**Figura 16.** Tipos de proyección.

Esto se debe a que en una vista ortográfica los puntos de los objetos se proyectan de forma perpendicular al plano de proyección, mientras que en la vista en perspectiva los puntos se dirigen al punto central de la cámara o centro de proyección.

Las funciones de GLM que se utilizan en el cálculo de esta matriz son

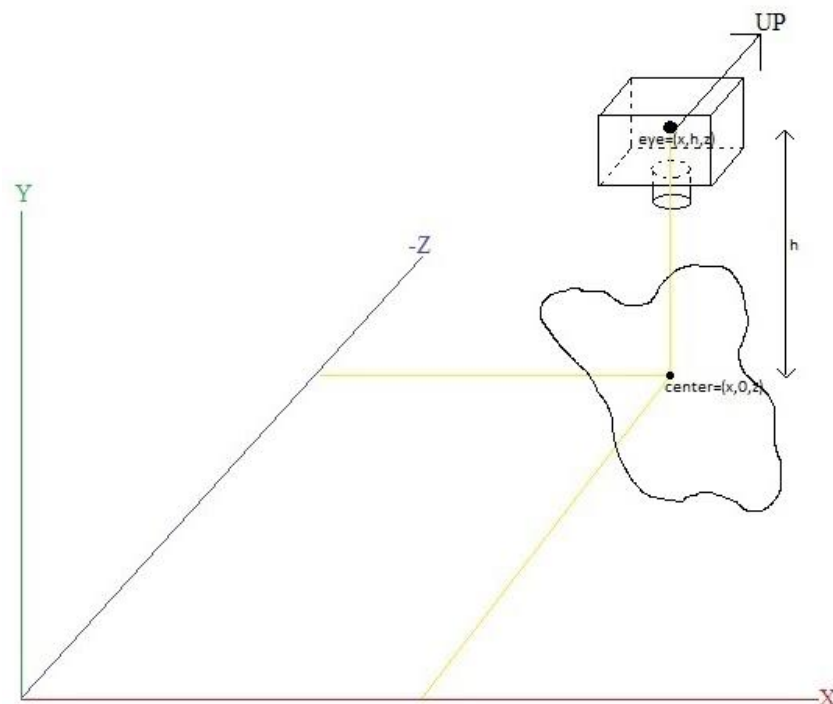
***perspective(fov, aspectRatio, near, far)*** para el cálculo de la matriz en perspectiva donde ***fov*** indica el ángulo de visión de la cámara, el ***aspectRatio*** es la proporción entre su ancho y altura, ***near*** es la distancia desde la que empieza a ver y ***far*** es la distancia hasta donde puede ver<sup>11</sup>.

***ortho(left, right, bottom, top, near, far)*** para el cálculo de la matriz en ortográfica donde ***left***, ***right***, ***bottom*** y ***top*** configuran la visión de la cámara indicando el límite de la cámara en la izquierda, derecha, abajo y arriba respectivamente.

- **View(V):** Esta matriz hace las funciones de cámara, necesitando para poder usar esta matriz la posición, el punto hacia el que mira y la orientación de la cámara.

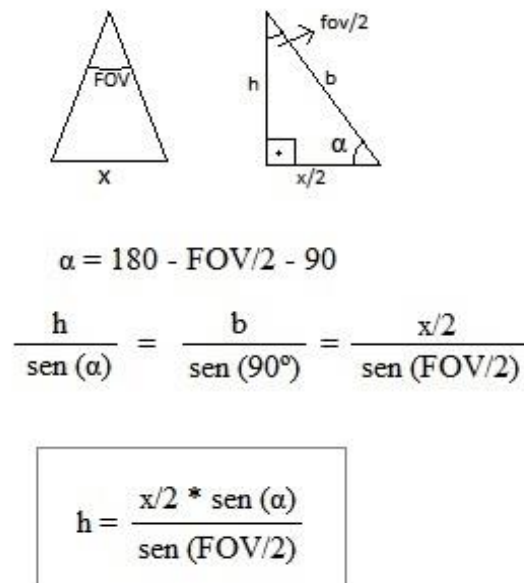
La función de GLM que se utiliza para esta matriz es ***lookAt(eye, center, up)*** que posiciona la cámara según los parámetros de entrada que son ***eye*** la posición de la cámara, ***center*** el punto hacia donde mira y ***up*** la orientación, como se muestra en la figura 17.

A continuación, se explica el algoritmo diseñado para posicionar la cámara de forma automática para un circuito dado. Para calcular ***center*** se buscan los puntos más externos del circuito en los dos ejes y con ellos se calcula el punto central del circuito.



**Figura 17.** Posición y orientación de la cámara.

Una vez determinado **center**, se calcula en que eje (X,Z) se tiene mayor dimensión. Sabiendo que eje posee una mayor distancia y el **fov** de la cámara podemos calcular mediante trigonometría a la altura ( $h$ ) a la que se debe posicionar la cámara para poder ver todo el circuito. En la figura 18 se pueden observar los cálculos trigonométricos realizados asumiendo que el eje de mayor longitud es X. Por lo tanto, **eye**, se sitúa en **center** levantado  $h$ .



**Figura 18.** Trigonometría utilizada para el cálculo de la altura de la cámara.

Con esto la cámara se situará automáticamente en el centro de cualquier circuito que se introduzca y a una distancia a la cual se pueda ver el circuito completamente.

Una vez tenemos la posición de la cámara, dependiendo de que opción haya elegido el usuario se hacen los cálculos para la proyección en perspectiva u ortográfica, esto implica que se cambia también la posición de la cámara obtenida anteriormente. En caso de que se opte por una visualización en ortográfica la cámara se situara justo encima del circuito, sin embargo, en caso de que se opte por una visualización en perspectiva se colocara la cámara a  $45^\circ$  respecto a la posición inicial (justo encima del circuito) para que sea posible observar los modelos 3D y la perspectiva de la imagen. El cálculo de eye en perspectiva se realiza colocando una de las dos distancias en un



eje en ambos, es decir,  $eye = (x, h, x)$  ó  $eye = (z, h, z)$ . Lo que crea un ángulo de 45 grados respecto al plano XZ.

Como añadido, se ha implementado la funcionalidad de zoom en ambas proyecciones el cual se puede utilizar con la rueda del ratón.

- **Model (M):** Esta matriz realiza una transformación de la posición en el modelo a la posición global. Normalmente es una combinación de tres posibles movimientos: trasladar, escalar y rotar. Cada uno de estos movimientos vienen dados por matrices, las cuales se multiplican unas sobre otras, comenzando por la matriz identidad, dando así una única matriz que será la matriz *model*. En esta aplicación, para el cálculo de la matriz *model*, se implementa la ecuación

$$M = T \cdot R_y \quad (11)$$

es decir, se realizan primero rotaciones respecto al eje y ( $R_y$ ) y después translaciones ( $T$ ).

Las matrices específicas que se utilizan en este proyecto se pueden observar en las ecuaciones 12 y 13. En la ecuación 12 se puede ver la matriz de rotación en Y donde  $\varphi$  representa el ángulo que rota el robot.

$$R_y = \begin{bmatrix} \cos \varphi & 0 & \sin \varphi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \varphi & 0 & \cos \varphi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (12)$$

En la ecuación 13 se puede observar la matriz de translación en X y Z que son los ejes donde se puede mover el robot.

$$T = \begin{bmatrix} 1 & 0 & 0 & X \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & Z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (13)$$

En ambas matrices simplemente se debe sustituir las variables  $X$ ,  $Y$  y  $\varphi$  con los valores correspondientes y se obtienen las matrices que se utilizan en la matriz model.

Las funciones de GLM son:

***translate(m, v)***, donde ***m*** es la matriz anterior y ***v*** es el vector de tres posiciones (x,y,z) que indica cuanto y hacia donde se mueve;

***rotate(m, angle, axis)*** donde ***m*** es la matriz anterior, ***angle*** es el ángulo que se rota y ***axis*** es un vector de tres posiciones que indica en que eje rota.

### 3. Qt

OpenGL no puede crear ventanas y no tiene una forma sencilla de interactuar con el usuario por lo que es necesario cubrir estas necesidades con otro software. Entre las diferentes alternativas se encuentran freeglut, sdl, Qt, etc... Se ha optado por Qt puesto que tiene una interacción con el usuario más sencilla y es fácil incluir en un proyecto OpenGL.

Qt es un framework de desarrollo de aplicaciones multiplataforma para PC que se suele utilizar en gran parte para programas que utilicen interfaz gráfica.

Internamente se utiliza C++ con alguna extensión para funciones como Signals y Slots, por lo tanto, se utiliza orientación a objetos<sup>8</sup>. Puesto que se utiliza C++, en los proyectos se encontrarán archivos de 4 tipos:

- .pro: Solo habrá un archivo .pro en el proyecto. Este archivo contiene toda la información necesaria para realizar la build de la aplicación.
- .h: Estos archivos incluyen la declaración de variables y las cabeceras de las funciones de la clase correspondiente, tanto pública como privada.
- .cpp: Son los archivos en los cuales se sitúa el código fuente de la clase.
- .ui: Este archivo es el correspondiente a la interfaz gráfica.

Para el desarrollo de la interfaz gráfica se puede escribir en C++ utilizando el módulo Widget, además de esto, Qt tiene una herramienta gráfica llamada Qt Designer que es un generador de código basado en Widgets. En la figura 19 se puede observar Qt Designer con widgets colocados en una aplicación.

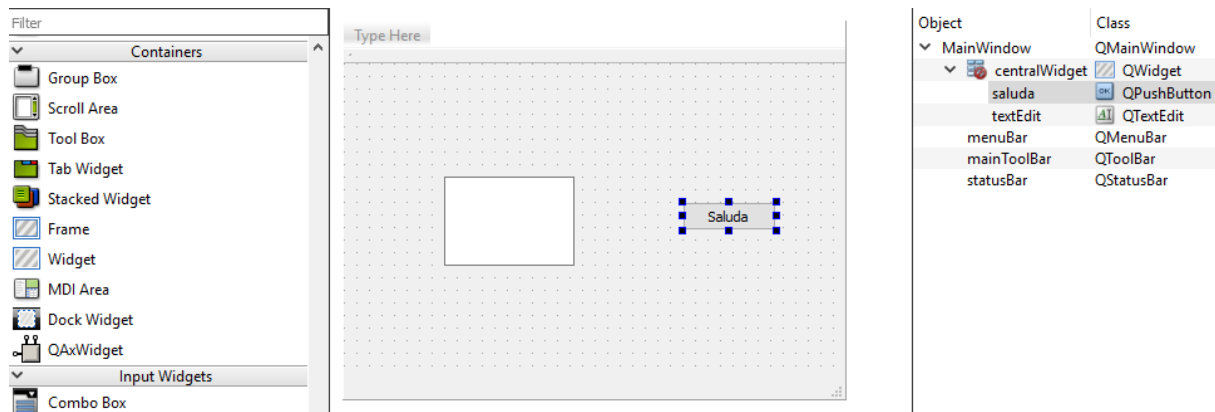


Figura 19. Qt Designer.

La integración de estas herramientas (OpenGL y Qt) se realiza de forma muy sencilla puesto que Qt tiene la API de OpenGL y por lo tanto solo hace falta decirle al IDE que estás utilizando que vas a utilizar esas librerías incluyendo lo siguiente **"#include <QOpenGLWidget>"**. Además, en caso de que se esté utilizando como IDE Qt Creator se debe ir al fichero .ui y en el widget en el que se muestra la simulación, promoverle a la clase que tenga el código de la simulación. En caso de utilizar librerías externas se debe incluir en el fichero .pro la palabra reservada **LIBS** seguido del path donde se encuentre la librería: **LIBS += pathDeMiLibreria** y en la clase en la que se utiliza realizar el include correspondiente.

Las funciones básicas y de mayor importancia que nos proporciona OpenGL con Qt son las siguientes:

- **initializeGL():** Esta función se ejecuta una sola vez y antes que las otras dos funciones. Por lo tanto, se utiliza para inicializar y configurar todo lo necesario para la utilización de OpenGL u otros.
- **paintGL():** Esta función es la que renderiza la escena de OpenGL y se llama siempre que el widget necesite ser actualizado. Además, este método será en el cual se modificará la posición de la cámara y la posición del robot, es decir, las matrices *view* y *model*. Gracias a los cambios en esta última matriz se realiza la animación de movimiento del robot.
- **resizeGL(int w, int h):** En este método se debe configurar el viewport (los parámetros de entrada *w* y *h* son el ancho y el alto respectivamente de la zona donde se podrá visualizar el código desarrollado en OpenGL), el tipo de vista (perspectiva u ortográfica), etc. Es llamado por primera vez cuando el widget

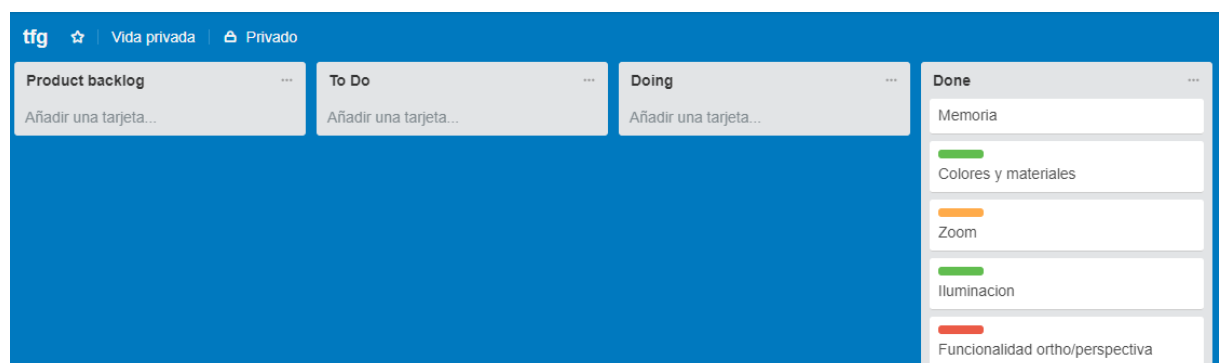
se crea (siempre después de ***initializeGL***) y siempre que el widget sea reescalado. Este método es el responsable de crear la matriz projection.

En cuanto a las funciones relacionadas con la GUI se trata de las funciones típicas que podrías encontrarte en otros software que incluyen interacción con el usuario, por ejemplo la función que se ejecuta al pulsar un botón es ***on\_nombreedetuboton\_clicked()***. Además cada tipo de widget tiene métodos específicos, por ejemplo un campo de texto tiene el método ***value()*** que devuelve el texto introducido en ese campo, otro ejemplo se trata del widget checkbox el cual tiene el método ***isChecked()*** que devuelve si el checkbox está con un tick.

## 4. Metodología ágil

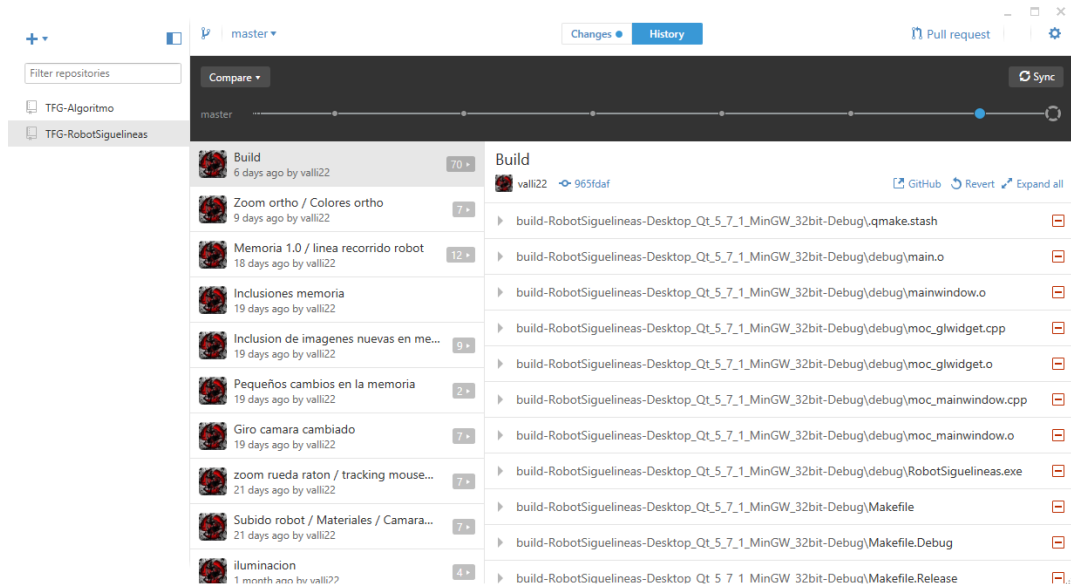
Durante el desarrollo de la aplicación se han utilizado herramientas utilizadas habitualmente en proyectos en los que se aplica metodología ágil. Estas herramientas han sido:

- **Trello** como tablero de tareas, este se divide en las columnas habituales (Product backlog, To Do, Doing, Done). A su vez cada tarea tiene asignada una dificultad representada mediante colores. Las tareas no tienen asignadas personas puesto que hay una sola persona encargada de este tablero. A pesar de no ser relevante para la organización de un equipo y la división de tareas, puesto que solo se trata de una persona, ha resultado muy útil para no perder la visión de proyecto. Todo esto se puede ver en la figura 20.



**Figura 20:** Trello del proyecto Simulación de un robot siguelineas.

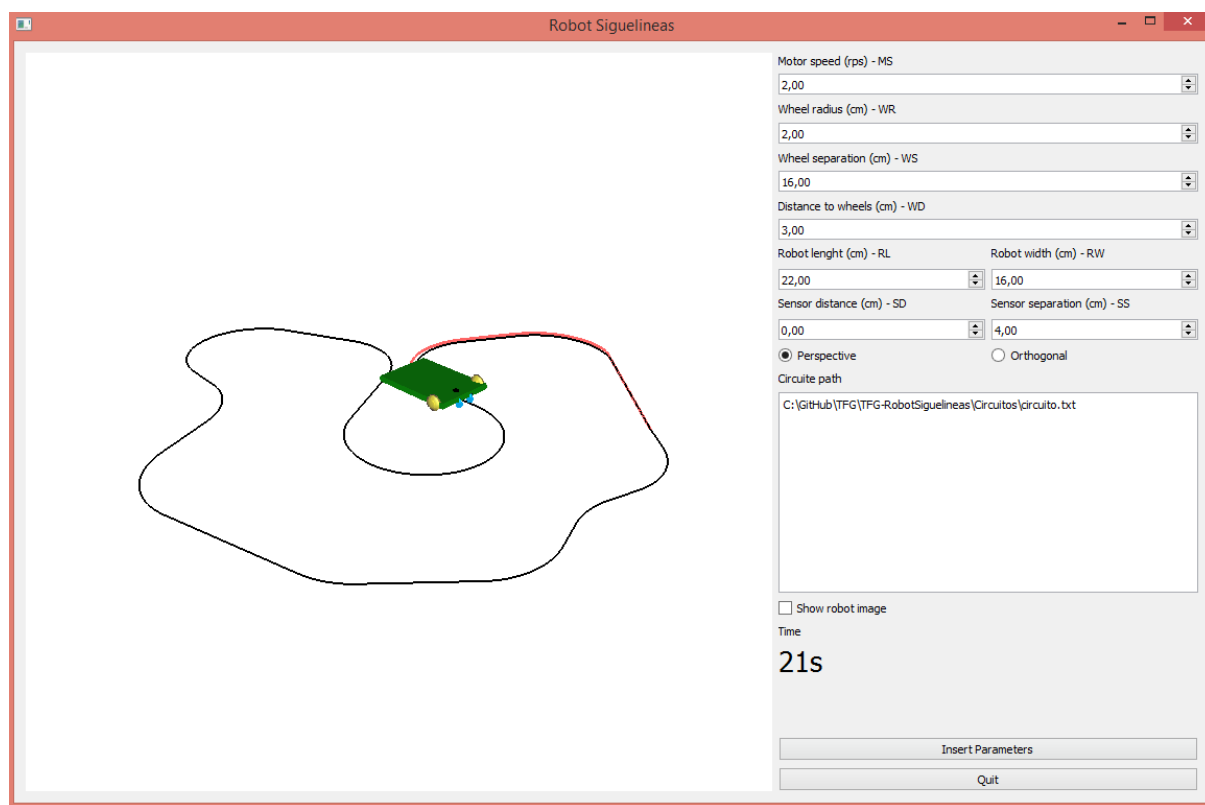
- **Git** como repositorio y control de versiones (utilizado concretamente Github). Sobre este repositorio se ha ido subiendo los diferentes incrementos de funcionalidad de la aplicación de forma periódica y que gracias a él se ha podido realizar un control de versiones. Se puede observar el repositorio desde la aplicación de Windows en la figura 21.



**Figura 21:** Repositorio en Github desde la aplicación de escritorio de Windows.

## Capítulo 4 Descripción de la aplicación

La figura 22 muestra el aspecto final de la aplicación desarrollada, la cual se puede dividir en dos zonas: izquierda y derecha. En la zona de la izquierda se muestra la simulación del robot en 3D, mientras que en la zona de la derecha se sitúan los controles de usuario.

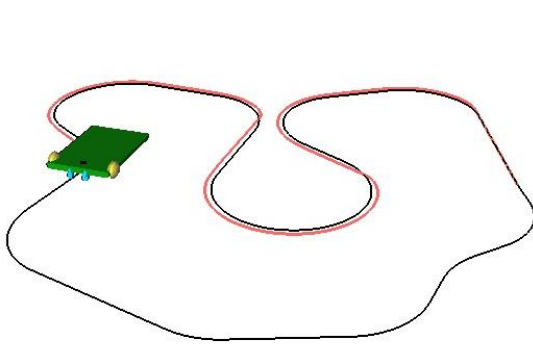


**Figura 22.** Vista completa de la aplicación.

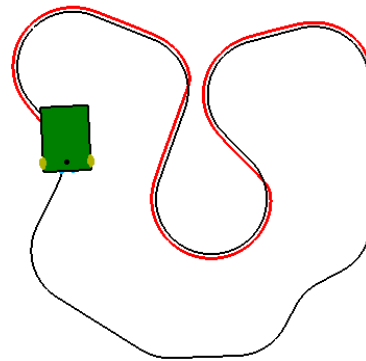
### 1. Zona izquierda: Viewport

En la parte izquierda de la aplicación se puede observar una pantalla en blanco, el viewport, sobre la que muestra la simulación del robot una vez ingresados los datos de este.

En esta zona se puede observar la simulación de la aplicación, en función de lo escogido en la zona derecha. La figura 23 muestra el resultado en 2 casos con diferentes modelos de proyección: a) perspectiva, y b) ortográfica.



a) Vista en perspectiva.

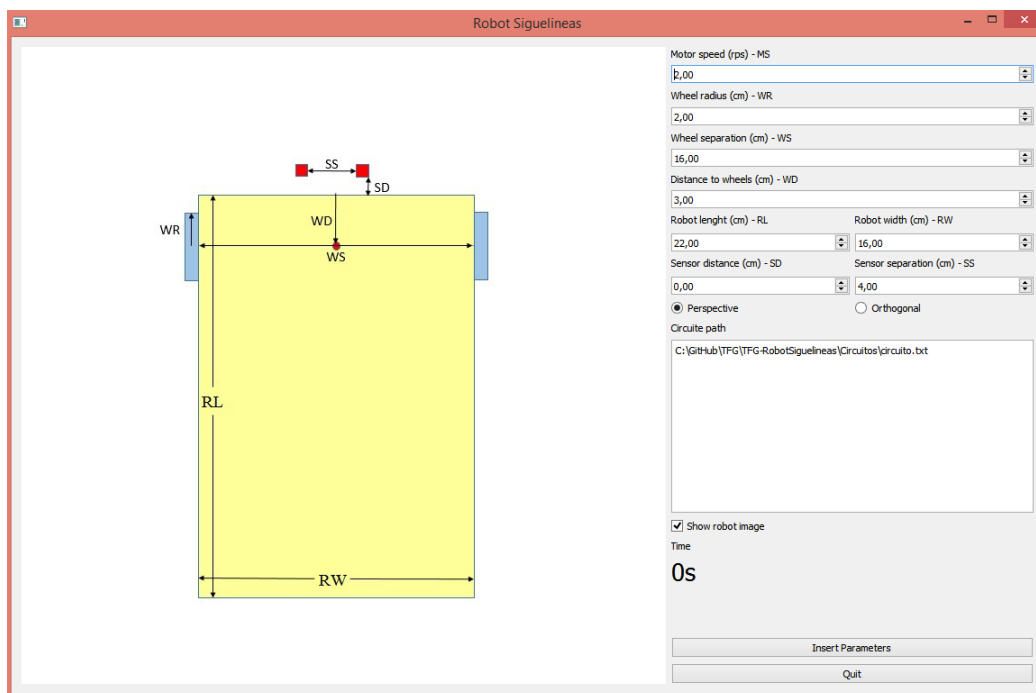


b) Vista en ortográfica.

**Figura 23.** Viewport.

El viewport muestra la simulación del robot. En todo momento el recorrido que el robot ha llevado durante esa ejecución mediante una línea roja, con lo que se da un mejor feedback al usuario.

Esta zona también se aprovecha para mostrar todos los parámetros geométricos del robot sobre una imagen de este como se muestra en la figura 24. Esta imagen se puede activar y desactivar en todo momento desde la zona derecha de la aplicación. Se explica con mayor detenimiento en el siguiente apartado.

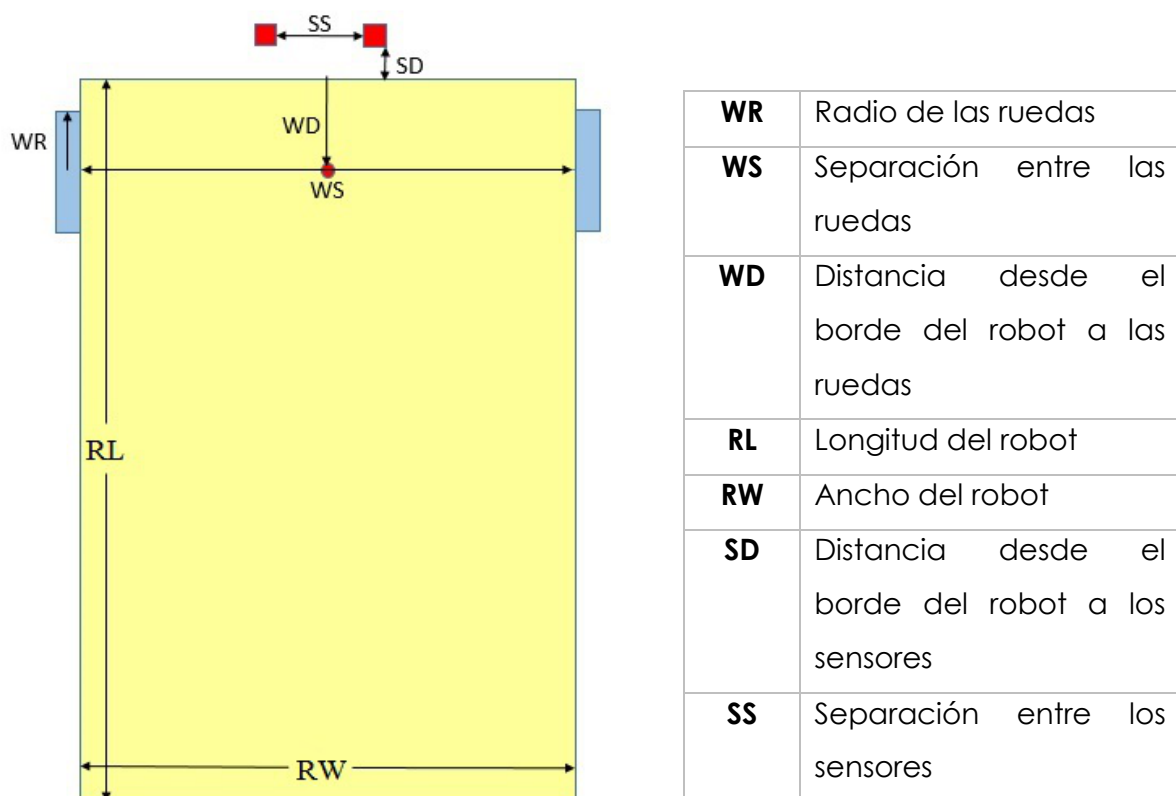


**Figura 24.** Aplicación con imagen de datos geométricos activa.

## 2. Zona derecha: Campos de entrada de datos

La zona derecha de la aplicación se centra en la recogida de datos del usuario.

Para un mejor entendimiento de los widgets de interacción con el usuario, se ha añadido abreviaturas a cada uno de ellos y la posibilidad de superponer una imagen con las referencias visuales sobre el robot, más tarde se explica en más detalle. En la figura 25, se muestra esta imagen y una tabla de abreviaturas que explican el significado de cada una de ellas.



**Figura 25.** Imagen de referencia sobre datos geométricos del robot y tabla de abreviaturas.

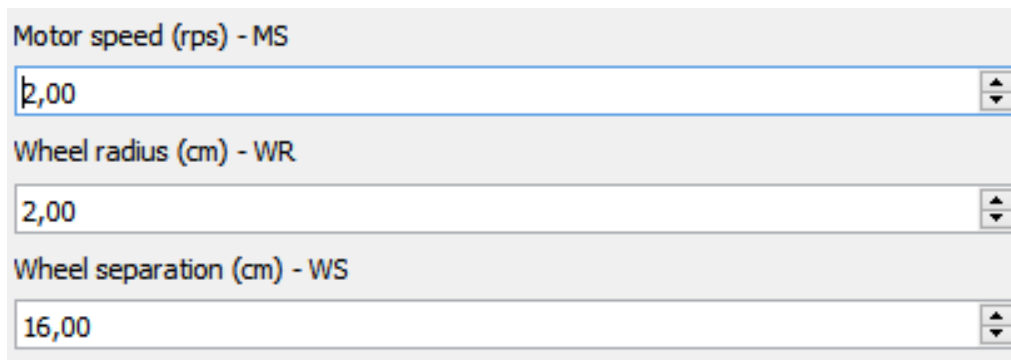
Para comenzar se decidió que la parte donde se realiza la simulación ocupara cerca del 75% de la aplicación puesto que es la parte más importante, el lugar donde más tiempo se está observando y donde más detalles hay que fijarse, la comentada en el punto anterior.

Una vez entrada en la parte de interacción con el usuario se coloca a la derecha por similitud a la mayor parte de aplicaciones encontradas y por lo tanto que le resulte



más natural al usuario el acceso a esta. Además, todas las labels están en inglés puesto que este es el idioma más hablado y por tanto dota a la aplicación de una mayor usabilidad.

A continuación, se opta por incluir los inputs en grupos de conceptos similares, como el grupo de las ruedas, el de los sensores y el del tamaño del robot. En el caso de las ruedas se utilizan tres inputs con sus respectivos labels en diferentes niveles, como se observa en la figura 26, puesto que la unión de los 3 parámetros de las ruedas en un mismo nivel no resultaba fácil de visualizar.

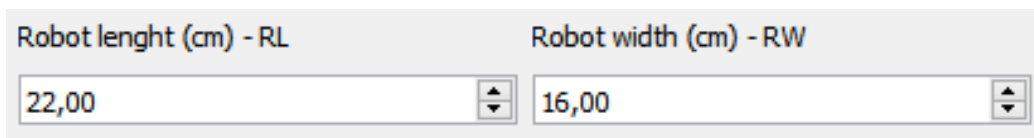


The image shows three vertically stacked input fields. The first is labeled 'Motor speed (rps) - MS' and contains the value '2,00'. The second is labeled 'Wheel radius (cm) - WR' and contains the value '2,00'. The third is labeled 'Wheel separation (cm) - WS' and contains the value '16,00'. Each input field has a small up/down arrow icon on its right side.

**Figura 26.** Paquete de ruedas en tres niveles.

Después de las ruedas viene un parámetro que comparten las ruedas y el robot, que es la distancia a la que se sitúan las ruedas desde el extremo superior del robot. Por lo tanto, este parámetro debe estar situado entre los parámetros de las ruedas y los del robot.

Al contrario que para las ruedas, para el tamaño del robot y los parámetros de los sensores, se agrupan 2 en un mismo nivel puesto que así se asimila a simple vista que esos parámetros están relacionados, como se ve en la figura 27.

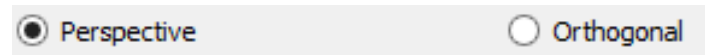


The image shows two input fields side-by-side. The left one is labeled 'Robot length (cm) - RL' and contains the value '22,00'. The right one is labeled 'Robot width (cm) - RW' and contains the value '16,00'. Both input fields have a small up/down arrow icon on their right side.

**Figura 27.** Paquete de dimensiones del robot en un nivel

Hasta aquí están los parámetros de entrada necesarios para la simulación del robot (a excepción del circuito). Por tanto, ahora deben entrar las opciones de la aplicación que no son directamente relevantes para la simulación de la aplicación,

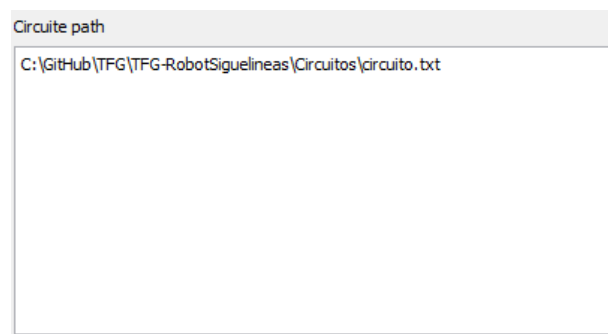
en primer lugar, se puede encontrar dos radio buttons que deciden si el usuario desea la vista perspectiva u ortográfica, como se puede observar en la figura 28.



**Figura 28.** Radio buttons para elegir tipo de proyección.

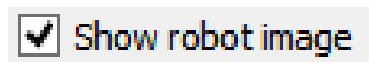
Esta opción está colocada aquí puesto que a pesar de que todavía queda introducir el circuito, si se pusiera detrás del input para el circuito esta opción sería poco visible y podría ignorarse, mientras que viniendo de opciones de tamaño reducido se ve claramente. En la figura 23-a) se puede observar cómo se visualiza con vista en perspectiva, la vista ortográfica se puede observar en la figura 23-b).

El último input se trata de un cuadro de texto editable en el que se debe introducir el path del circuito que se desea utilizar, ver figura 28.



**Figura 28.** Campo de texto donde se debe introducir el path del circuito.

A continuación, se sitúa un checkbox que activa y desactiva la imagen de referencia que tiene las indicaciones y las medidas del robot para que el usuario sepa exactamente qué es lo que está modificando en todo momento. Se puede observar la imagen que se da como referencia al usuario en la figura 25 y el checkbox que la activa en la figura 29.



**Figura 29.** Checkbox que muestra o esconde la imagen de referencia del robot.

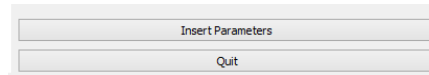
Después, hay una zona en la que se muestran los segundos pasados desde que empezó la simulación como se muestra en la figura 30, con un tamaño de letra que hace que sobresalga sobre el resto de la interfaz haciendo así que una vez la

simulación este comenzada se vea claramente la simulación y los segundos, pudiendo ignorar el resto de la interfaz.



**Figura 30.** Tiempo de simulación.

Por último, hay dos botones, el primero cuyo texto pone "Insert Parameters", el cual cuando pulses se introducen todos los parámetros a la aplicación y se inicia la simulación. El último botón contiene el texto "Quit" que sirve para cerrar la aplicación. Estos dos últimos botones se pueden observar en la figura 31.



**Figura 31.** Botones de inicio de simulación y cerrar aplicación.

### 3. Casos de uso

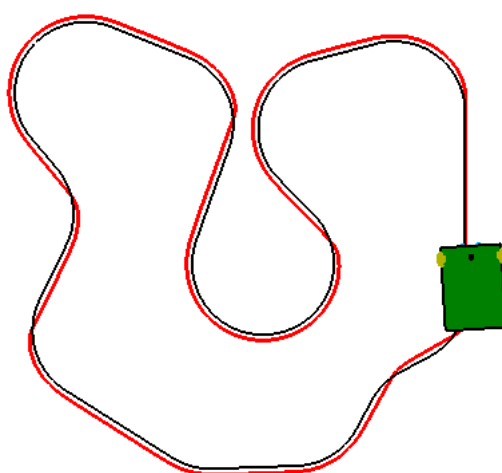
A continuación, se muestran los resultados obtenidos de diferentes configuraciones de robots y circuitos. Los casos que se van a mostrar son los siguientes:

1. Robot estándar que se va a tomar como referencia.
2. Robot estándar con cambio en la separación entre ruedas.
3. Robot estándar con cambio en el radio de las ruedas.
4. Robot estándar con cambio en la separación entre sensores.
5. Robot estándar con cambio en la distancia entre ruedas y sensores.
6. Robot estándar en un circuito distinto.

#### Caso 1. Robot de referencia.

Este robot es un robot con parámetros geométricos medios que se utiliza como referencia para el resto de los casos.

Parámetros	MS	WR	WS	WD	RL	RW	SD	SS	Tiempo
Valor	2	2	16	3	22	16	0	4	90s

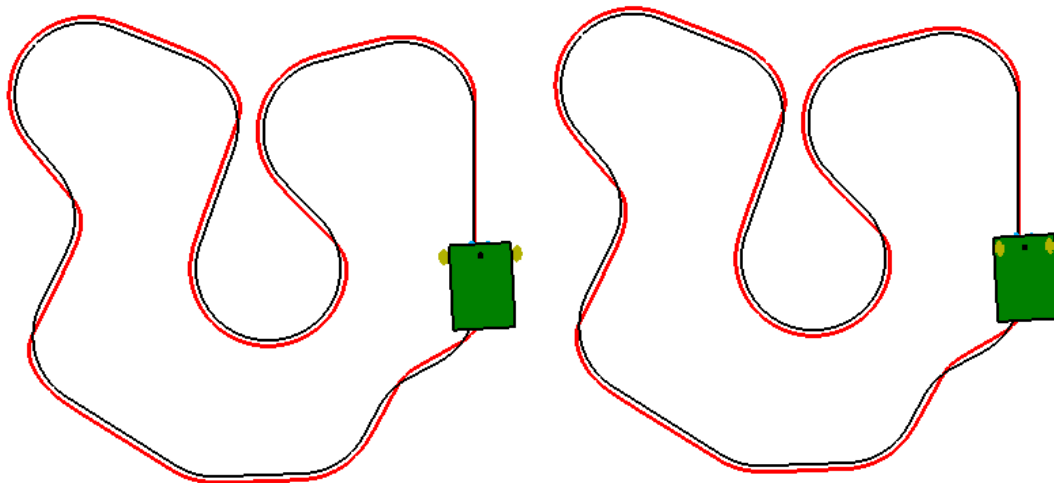


**Figura 32.** Robot estandar que se toma como referencia.

## Caso 2. Cambio en la separación entre ruedas.

En este caso se va a cambiar la distancia que tienen las ruedas entre sí, en el primer caso se muestra el aumento de esta y en la segunda su disminución.

Parámetros	MS	WR	WS	WD	RL	RW	SD	SS	Tiempo
Valor (↑)	2	2	<b>19</b>	3	22	16	0	4	<b>94s</b>
Valor (↓)	2	2	<b>13</b>	3	22	16	0	4	<b>87s</b>

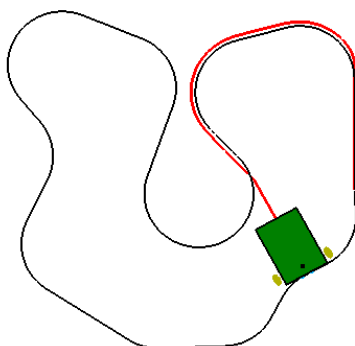


a) Mayor separación.

b) Menor separación.

**Figura 33.** Cambio en la separación entre ruedas.

Debido a tener una mayor distancia entre las ruedas este robot debe hacer un mayor recorrido para hacer el mismo giro. Al ser un circuito con muchos giros, tarda más en realizarlo. Por lo tanto, en el segundo caso, al tener una separación de ruedas menor, realiza giros más rápido.



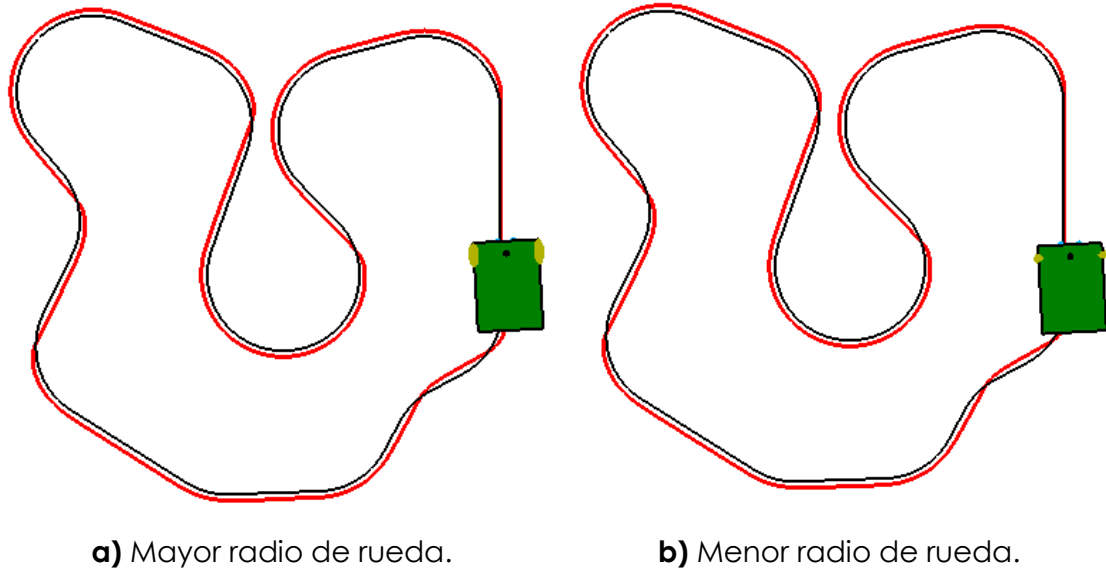
En esta imagen se puede observar el caso en el que se realiza una separación de ruedas de 20cm, el robot simplemente no puede realizar un giro y acaba saliéndose del recorrido.

**Figura 34.** Excesiva separación entre ruedas.

### Caso 3. Cambio en el radio de las ruedas.

Para este caso se aumenta y disminuye el radio de las ruedas respecto al robot estándar.

Parámetros	MS	WR	WS	WD	RL	RW	SD	SS	Tiempo
Valor (↑)	2	<b>3</b>	16	3	22	16	0	4	<b>60s</b>
Valor (↓)	2	<b>1</b>	16	3	22	16	0	4	<b>181s</b>



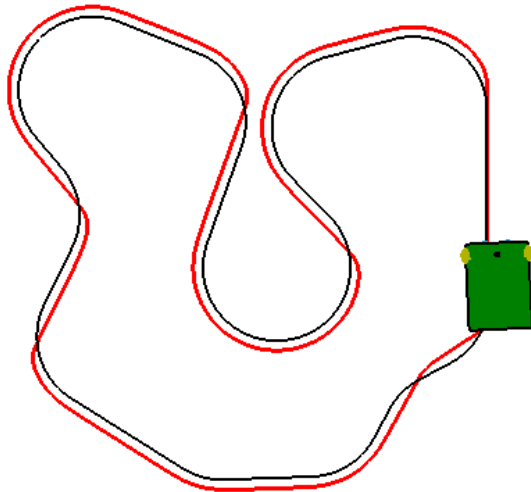
**Figura 35.** Cambio en el radio de las ruedas.

La velocidad del motor esta medida en revoluciones por segundo, por consiguiente, cuanto más radio tengan las ruedas, más veloz es el robot y cuanto menor sea el radio más lento es.

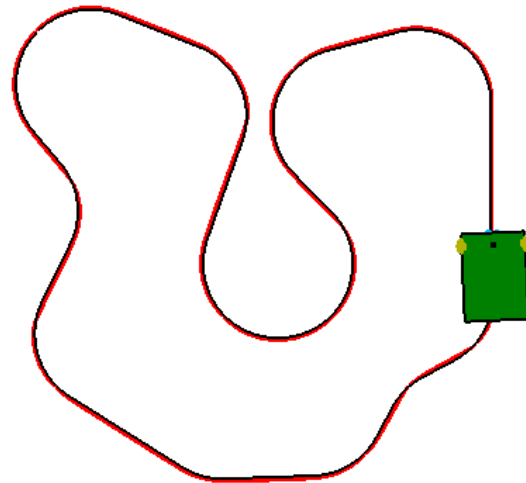
## Caso 4. Cambio en la separación entre sensores.

Se realizan modificaciones en la separación entre los sensores, aumentando o disminuyendo esta.

Parámetros	MS	WR	WS	WD	RL	RW	SD	SS	Tiempo
Valor (↑)	2	2	16	3	22	16	0	<b>6</b>	<b>92s</b>
Valor (↓)	2	2	16	3	22	16	0	<b>2</b>	<b>88s</b>



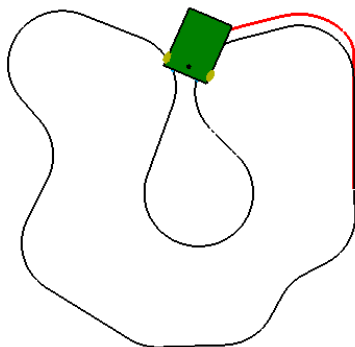
**a)** Mayor separación.



**b)** Menor separación.

**Figura 36.** Cambio en la separación entre sensores.

En el primer caso, puesto que los sensores tienen una separación mayor, hay mayor margen en la colisión con el circuito, por lo que el robot se separa un poco del circuito haciendo que realice un mayor recorrido. Sin embargo, en el segundo, al tener los sensores más juntos, el robot se ajusta más al recorrido del circuito debido a que se detectan muchas colisiones y por lo tanto el recorrido que hace es más ajustado al recorrido del circuito.



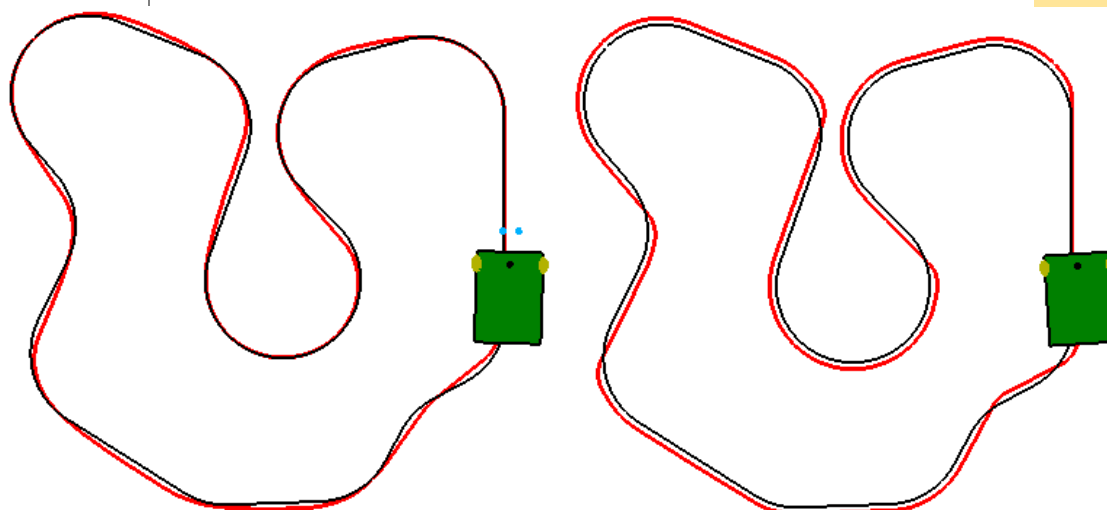
Al ampliar la separación de los sensores a 9cm estos colisionan a la vez con varias partes del circuito y este se queda parado.

**Figura 37.** Excesiva separación entre sensores.

## Caso 5. Cambio en la separación entre ruedas y sensores.

Para este caso se va a cambiar la distancia entre los ejes del robot, es decir, la distancia entre el punto central de las ruedas y el punto central de los sensores.

Parámetros	MS	WR	WS	WD	RL	RW	SD	SS	Tiempo
Valor (↑)	2	2	16	3	22	16	<b>5</b>	4	<b>86s</b>
Valor (↓)	2	2	16	3	22	16	<b>-1</b>	4	<b>92s</b>



**a)** Mayor separación entre ejes.

**b)** Menor separación entre ejes.

**Figura 38.** Cambio en la separación entre ruedas y sensores.

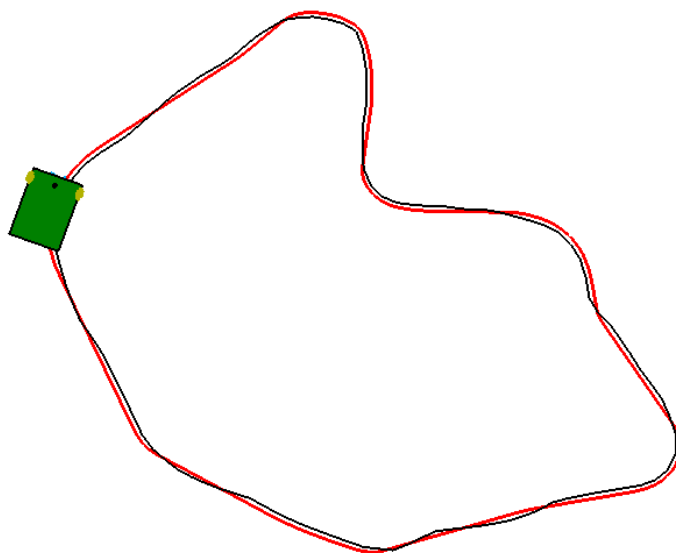
En el caso a) al estar más separado el centro de rotación de los sensores, el giro que se realiza hace que los sensores giren más rápido y por lo tanto se detecten colisiones antes. Al contrario que en el caso anterior, para el caso b), al estar más cerca el centro de rotación de los sensores, el giro del robot se hace de manera más gradual y se detectan las colisiones un poco más tarde.



## Caso 6. Circuito diferente.

Se utiliza el robot estándar en un circuito diferente al del resto de pruebas.

Parámetros	MS	WR	WS	WD	RL	RW	SD	SS	Tiempo
Valor	2	2	16	3	22	16	0	4	83s



**Figura 39.** Distinto circuito.

## Conclusiones

Gracias a este proyecto he obtenido bastante conocimiento sobre C++, además de sobre una tecnología de la cual no sabía nada como es Qt. Para poder realizarlo además ha sido necesario indagar en el funcionamiento más profundo de OpenGL, dotándome así de un mayor conocimiento sobre cómo funciona a bajo nivel.

De los objetivos principales propuestos a la hora de realizar la aplicación, han sido todos logrados.

Una de las mejoras a realizar sobre este proyecto podría ser la posibilidad de realizar carreras varios robots a la vez, o realizar estas carreras de manera online desarrollando una aplicación en red que permita a una persona ser el host y al resto conectarse a ese host para poner sus robots y poder ver la carrera. Además, se podría mejorar la forma y figuras del robot y el circuito.

Otra posible mejora sería el de desarrollar un algoritmo que, dado un circuito, encontrara los mejores parámetros posibles del robot, con los que éste realice el recorrido en el menor tiempo posible.

Otra podría ser la posibilidad de dibujar los circuitos desde la propia aplicación o, incluso, modificar los parámetros del robot desde el dibujo mediante el ratón

# Bibliografía

## Robótica

1. Fernando Reyes Cortés (2018) “Robótica. Control de robots manipuladores”.
2. Alonzo Kelly (2013) “Mobile Robotics: Mathematics, Models, and Methods”.

## Informática gráfica y OpenGL

3. Steven Marschner y Peter Shirley (2015) “Fundamentals of Computer Graphics, Fourth Edition, 4th Edition”.
4. Samuel R. Buss (2003) “3D Computer Graphics: A Mathematical Introduction with OpenGL”.
5. Graham Sellers, Richard S. Wright Jr. Y Nicholas Haemel (2013) “OpenGL SuperBible: Comprehensive Tutorial and Reference (6th Edition)”.
6. Página oficial OpenGL: <https://www.opengl.org/>

## Qt y C++

7. Bjarne Stroustrup (2013) “The C++ Programming Language, 4th Edition”.
8. Lee Zhi Eng (2018) “Hands-On GUI Programming with C++ and Qt5”.
9. Documentación online: <http://doc.qt.io/>

## Freeglut y GLM

10. Documentación online Freeglut: <http://freeglut.sourceforge.net/docs/api.php>
11. Documentación online GLM: <https://glm.g-truc.net/0.9.9/api/index.html>

## Metodología ágil

12. Andrew Stellman y Jenifer Greene (2014) “Learning Agile: Understanding Scrum, XP, Lean, and Kanban”.
13. Tipos de metodologías resumidas: <https://www.versionone.com/agile-101/agile-methodologies/>

## Otras aplicaciones parecidas

14. Roboworks: <http://www.newtonium.com/>

15. RoboDK: <https://robodk.com/index>