



**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

# INDICE

<b>AGRADECIMIENTOS .....</b>	<b>3</b>
<b>RESUMEN.....</b>	<b>4</b>
<b>CAPÍTULO 1 INTRODUCCIÓN .....</b>	<b>5</b>
1.1. MOTIVACIÓN .....	5
1.2. OBJETIVOS.....	7
1.3. ESTADO DEL ARTE .....	8
1.4. ESTRUCTURA DE LA MEMORIA .....	8
<b>CAPÍTULO 2 MODELADO DEL SISTEMA.....</b>	<b>10</b>
2.1. ROBÓTICA MÓVIL.....	10
2.2. VEHÍCULOS CON RUEDAS .....	10
2.3. MODELO FÍSICO DIRECCIONAMIENTO DIFERENCIAL.....	13
2.4. NAVEGACIÓN AUTÓNOMA .....	17
<b>CAPÍTULO 3 ENTORNO TECNOLÓGICO .....</b>	<b>20</b>
3.1. OPENGL .....	20
3.2. ETAPA GEOMÉTRICA.....	22
3.2.1. Posicionamiento del modelo .....	22
3.2.2. Posicionamiento de la cámara .....	24
3.2.3. Modelo de proyección .....	26
3.3 QT .....	27
3.4. METODOLOGÍA ÁGIL.....	29
<b>CAPÍTULO 4 DESCRIPCIÓN DE LA APLICACIÓN.....</b>	<b>31</b>
4.1. ZONA IZQUIERDA: VIEWPORT .....	31
4.2. ZONA DERECHA: CAMPOS DE ENTRADA DE DATOS .....	33
4.3. CASOS DE USO.....	36
4.3.1. Robot de referencia .....	36
4.3.2. Separación entre ruedas.....	37
4.3.3. Radio de las ruedas .....	38
4.3.4. Separación entre sensores.....	39
4.3.5. Separación entre ruedas y sensores .....	40
4.3.6. Circuito diferente .....	41
4.3.7. Vista en perspectiva.....	41
<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.

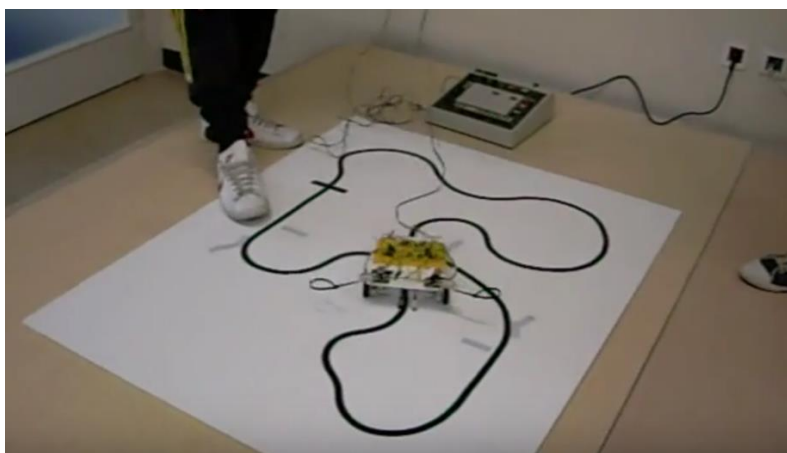
Para ello, se ha comenzado estudiando la mecánica de tal sistema a través de las ecuaciones que relacionan el giro de los motores con la posición y orientación del robot. A continuación, se han analizado las diferentes tecnologías disponibles para la implementación de la aplicación, habiéndose elegido Qt y OpenGL. Dicha aplicación, tiene implementadas tanto la simulación del comportamiento del robot como su visualización sobre el circuito. Además, el usuario puede interaccionar con la misma, no solo introduciendo los parámetros del escenario a simular y cambiando de perspectiva, sino también en tiempo de ejecución mediante la realización de zoom mediante el ratón.

# Capítulo 1 Introducción

## 1.1. Motivación

Este trabajo nace motivado por un intento de mejorar una de las herramientas que se utilizaban en la 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 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 un circuito marcado por una línea negra sobre un fondo blanco. Este tiempo depende de ciertos parámetros geométricos que los alumnos eligen a la hora de crear el robot. La figura 1 muestra una fotografía real de dicha competición.

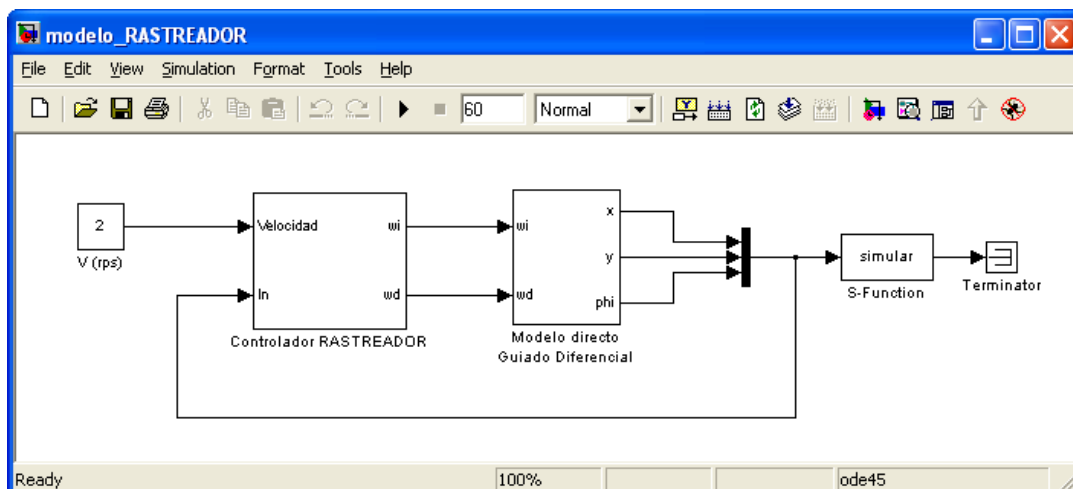
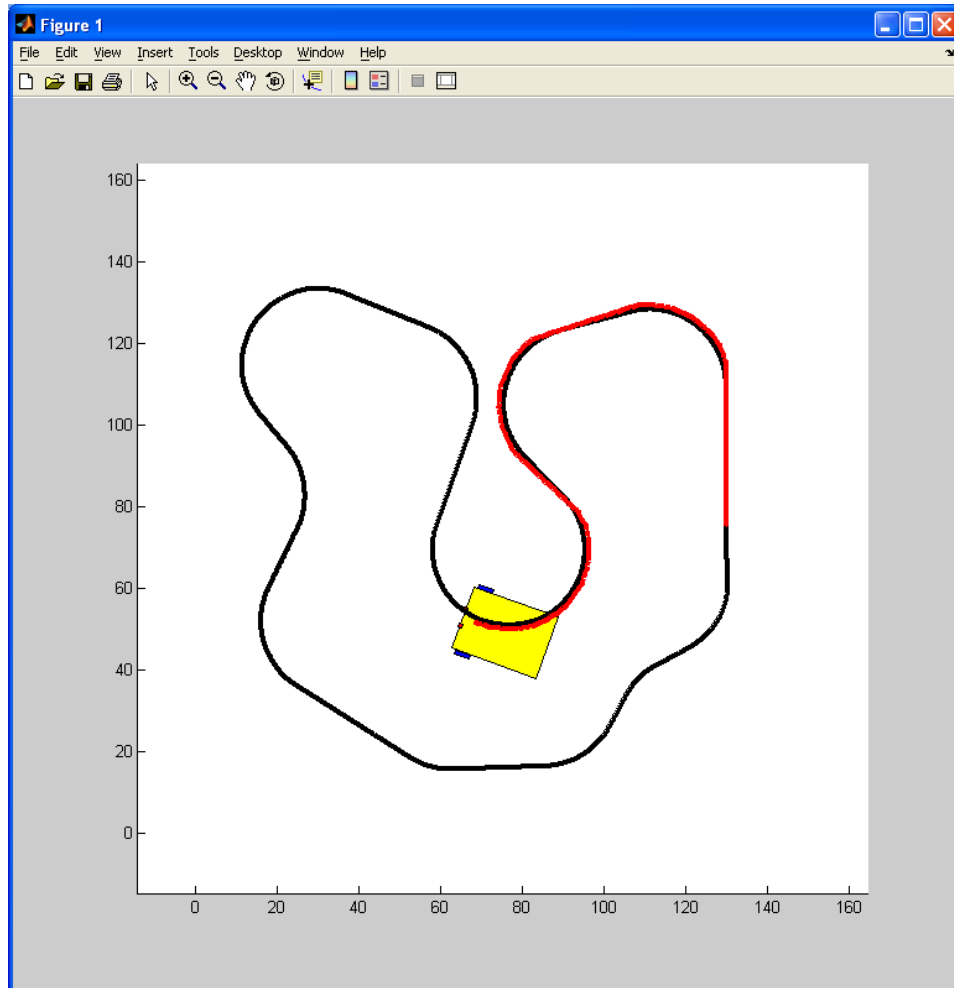


**Figura 1.** Competición de robots.

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 2 se puede observar el esquema de dicho desarrollo, en el que se pueden 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, el responsable de los cálculos de la posición y orientación del robot, y el cargado de dibujar el estado del robot en la ventana correspondiente.



**Figura 2.** Antiguo desarrollo en MATLAB-Simulink.

Sin embargo, este simulador tiene varios inconvenientes:

- a) Se debe tener instalado MATLAB-Simulink para hacerlo funcionar.
- b) No se pueden 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, donde los números que aparecen en el vector robot representan en este orden, la longitud del robot en centímetros, su anchura, etc.
- c) No ofrece todas las funcionalidades requeridas por una aplicación de este tipo, tal y como se verá más adelante en la sección 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.

## 1.2. 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 que necesita la aplicación, descargadas junto a la misma.
- **Facilidad a la hora de realizar cambios:** La aplicación debe tener una simple interfaz "user-friendly" en la que se puedan cambiar rápidamente 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 los parámetros y poder observar sus consecuencias en el movimiento del robot, e ir aprendiendo sobre el mismo con su visualización

- **Realización de pruebas con facilidad:** 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 para 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.
- Integración de la interfaz gráfica de usuario y la ventana de simulación en la misma aplicación.

### 1.3. Estado del arte

Antes de diseñar nuestra propia aplicación con los requisitos que se acaban de comentar, se ha hecho una revisión del tipo de aplicaciones existentes ya disponibles para tal fin. En cuanto a las herramientas que se pueden encontrar en el mercado para simular un robot de este tipo se pueden encontrar varias:

- **Hemero**, una herramienta en Matlab-simulink para la enseñanza de la robótica. El problema que posee esta herramienta es la falta de parte gráfica [1].
- **RoboWorks**, se trata de una aplicación de modelado, simulación y animación de sistemas físicos, sin embargo, se utiliza solo para robots manipuladores [14].
- **RoboDK**, sucede lo mismo que con la anterior, se trata de una aplicación para simular robots, sin embargo, está basada en robots manipuladores [15].

Como se ve las herramientas descritas no cumplen completamente los requisitos motivando el desarrollo de la herramienta aquí descrita.

### 1.4. 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 con sus diferentes configuraciones. A continuación,



se describe la configuración seleccionada para la aplicación, el 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 en un vehículo de este tipo.

En el Capítulo 3, **Entorno tecnológico**, se habla sobre las diferentes tecnologías y librerías utilizadas: Qt, OpenGL, freeglut y GLM. De estas, se comenta su utilización en el proyecto. También se explica de manera breve el cauce gráfico, y en concreto, dada su importancia para este trabajo, 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, el sistema de Trello y el sistema de 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 gráfica con todos sus controles 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 Modelado del sistema

### 2.1. 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 [1].

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

El robot utilizado en este trabajo se enmarca dentro de los autónomos ya que es capaz de seguir la trayectoria mediante un sistema de guiado por medio de sensores de infrarrojos que no requiere de ningún tipo de control externo.

### 2.2. Vehículos con ruedas

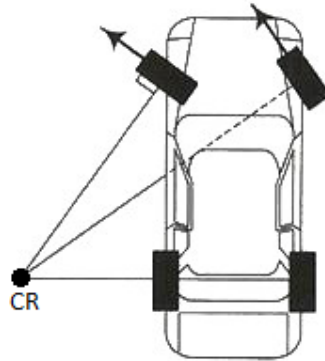
Los vehículos con ruedas son un tipo de robot móvil que proporcionan una solución simple y eficiente para conseguir movilidad sobre terrenos duros y libres de obstáculos, con los que se permite conseguir velocidades más o menos altas.

Su limitación más importante es el deslizamiento al impulsarse. Además dependiendo del tipo de terreno, pueden 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. Además, 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.



**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 del lado izquierdo con las del derecho.



**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.



**Figura 8.** Ruedas síncronas.

- **Direccionamiento diferencial:** Esta configuración es la que utiliza el robot sigue líneas utilizado en este trabajo. 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 además con esas mismas ruedas. Adicionalmente, existen una o más ruedas para el soporte en la parte posterior.



**Figura 9:** Direccionamiento diferencial.

### 2.3. Modelo físico direccionamiento diferencial

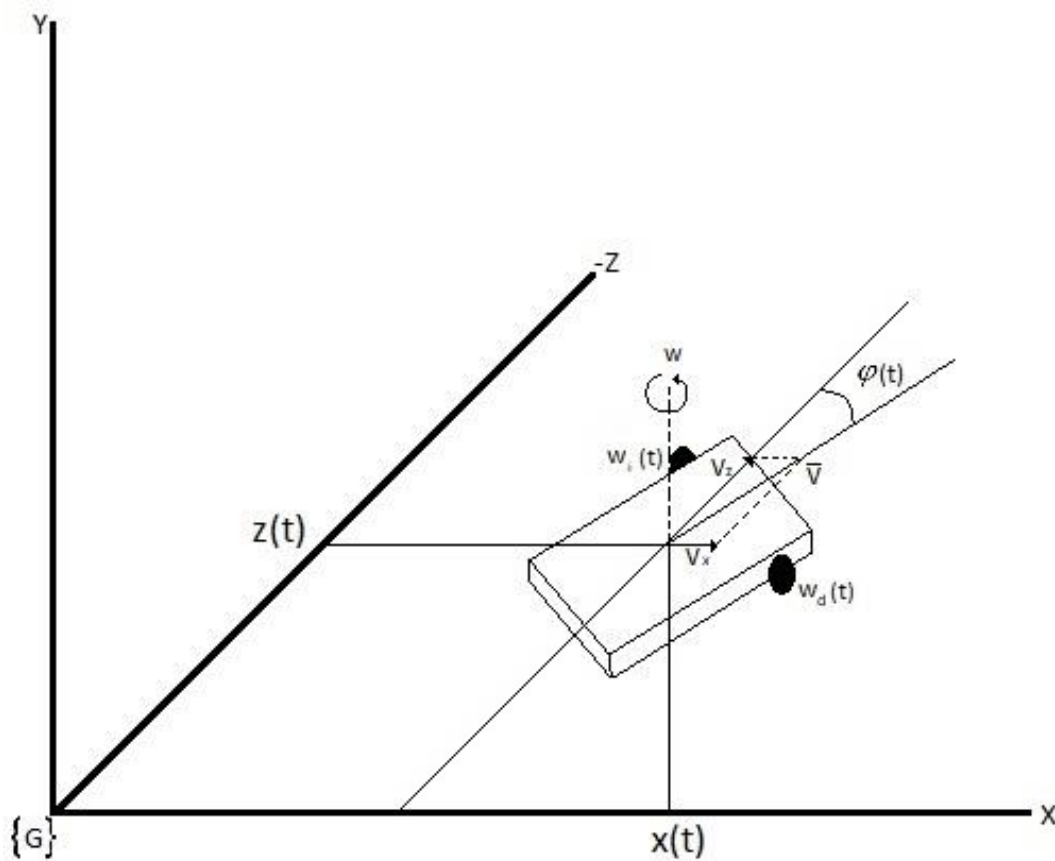
Para poder realizar la simulación del movimiento del robot se debe conocer primeramente el estado del sistema. El estado del robot viene dado por su posición y orientación. Puesto que el robot se va a mover, se necesita saber el estado del mismo en los diferentes instantes de tiempo, por lo tanto, se debe definir un modelo de cambios de estado, es decir, unas ecuaciones que a partir del estado actual y unas entradas permiten calcular el estado del sistema en el siguiente instante de tiempo. En la ecuación (1) se puede observar la ecuación que define el sistema, siendo  $\bar{s}$  el estado y  $\bar{r}$  las entradas.

$$\bar{s}(t + \Delta t) = f(\bar{s}(t), \bar{r}(t)) \quad (1)$$

Para el cálculo de un robot móvil con direccionamiento diferencial  $\bar{r}$  son las velocidades de las ruedas izquierda y derecha  $(w_i(t), w_d(t))$ , mientras que el estado  $\bar{s}$  viene dado por la posición  $(x(t), z(t))$  y orientación  $\varphi(t)$  del robot, tal y como se muestra en la ecuación (2).

$$\begin{aligned}
 \bar{s}(t) &= (x(t), z(t), \varphi(t)) \\
 \bar{r}(t) &= (w_i(t), w_d(t))
 \end{aligned}
 \tag{2}$$

En la figura 10 se pueden observar tanto el estado del sistema como sus entradas sobre el robot en cuestión.



**Figura 10.** Estado y entradas de un robot con direccionamiento diferencial.

A partir de la figura 10, si el vehículo tiene una velocidad de desplazamiento  $v$  y de rotación  $w$ , se obtiene los componentes mostrados en la ecuación (3):

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

Por otro lado, la derivada de una función puede aproximarse por el cociente incremental mostrado en la ecuación (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:

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

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

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

Aplicando el mismo resultado al resto de coordenadas del modelo cinemático directo, se obtiene la ecuación de cambio de estado (6):

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

Con el objetivo de buscar una ecuación similar a la mostrada en las ecuaciones (2), debemos relacionar las velocidades  $(v_x, v_z, v_\varphi)$  con las entradas reales del sistema  $(w_i, w_d)$ .

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:

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

Sustituyendo estas expresiones en las obtenidas en la ecuación (3), se obtienen los componentes de la velocidad 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 mostrado en la ecuación (6), se obtiene:

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

Para terminar, si utilizamos las variables  $s$  para representar el estado de los sensores ( $s=0$  si esta sobre la línea y  $s=1$  en caso contrario), se obtiene:

$$\begin{aligned}
 x(t + \Delta t) &= x(t) - w \cdot (s_d + s_i) \cdot \frac{WR \cdot \sin(\varphi(t))}{2} \cdot \Delta t \\
 z(t + \Delta t) &= z(t) + w \cdot (s_d + s_i) \cdot \frac{WR \cdot \cos(\varphi(t))}{2} \cdot \Delta t \\
 \varphi(t + \Delta t) &= \varphi(t) + w \cdot (s_d - s_i) \cdot \frac{WR}{WS} \cdot \Delta t
 \end{aligned} \tag{10}$$

Y definiendo las constantes  $k_1 = \frac{w \cdot \Delta t}{2}$  y  $k_2 = w \cdot \Delta t$  resulta:

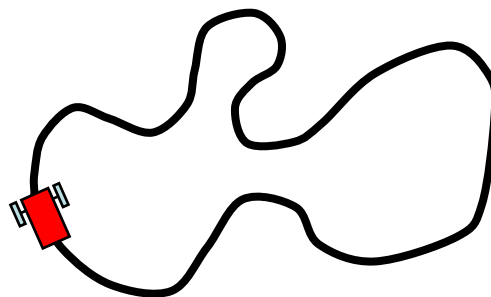
$$\begin{aligned}
 x(t + \Delta t) &= x(t) - k_1 \cdot (s_d + s_i) \cdot WR \cdot \sin(\varphi(t)) \\
 z(t + \Delta t) &= z(t) + k_1 \cdot (s_d + s_i) \cdot WR \cdot \cos(\varphi(t)) \\
 \varphi(t + \Delta t) &= \varphi(t) + k_2 \cdot (s_d - s_i) \cdot \frac{WR}{WS}
 \end{aligned} \tag{11}$$



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  $(t + \Delta t)$ . 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 de las ruedas  $WR$  y separación entre las mismas  $WS$ , así como la velocidad de giro de los motores  $w$  y el paso de la simulación  $\Delta t$ . El cálculo de las variables  $s$  se explica a continuación.

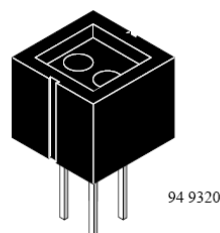
## 2.4. Navegación autónoma

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. El robot sigue líneas que se ha implementado realiza su movimiento de manera autónoma, 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.



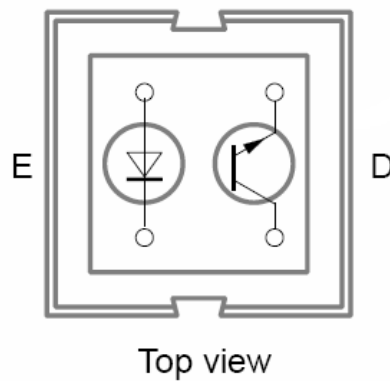
**Figura 11.** Navegación autónoma del robot sigue líneas.

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. La figura 13 muestra de forma simplificada su estructura interna. Cuando el sensor se haya sobre una línea negra la luz emitida por el fotodiodo es absorbida por la misma y el fototransistor la señal correspondiente al circuito de control. 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:

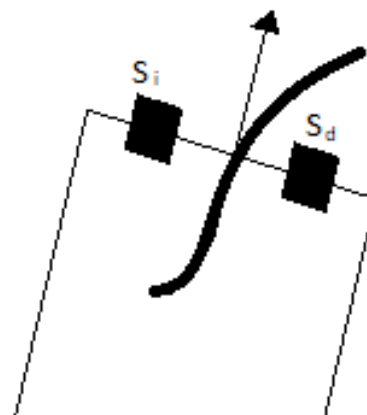
$s_i$  -> sensor izquierdo. |  $s_d$  -> sensor derecho.

---

**Algoritmo 1:** Algoritmo sensor

---

1. **if**  $IsOnCircuite(s_i)$
  2.      $s_i \leftarrow 0$
  3. **if**  $IsOnCircuite(s_d)$
  4.      $s_d \leftarrow 0$
- 



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

---

**Algoritmo 2:** IsOnCircuite(pos)

---

```
1. for (i=0; i<circuite.length; i++)
2.     distance = raiz(cuadrado(circuite(i).x - pos.x)) + cuadrado(circuite(i).z - pos.z)
3.     if distance<threshold
4.         return true
5. endfor
6. return false
```

---

Esta es una de las posibles implementaciones, otras opciones son: la rueda en vez de frenar realiza el giro hacia atrás  $s_i = -1$  y/o  $s_d = -1$  o disminuye la velocidad en vez de frenar completamente  $s_i = s_i - \Delta$  y/o  $s_d = s_d - \Delta$  con  $\Delta \in (0,1)$ .

## Capítulo 3 Entorno tecnológico

Para el desarrollo del trabajo se han utilizado diferentes tecnologías:

- Freeglut para la creación de los modelos 3D utilizados.
- Glm para el cálculo de matrices necesarias para el posicionamiento del robot y sus sensores.
- OpenGL para la visualización gráfica (3D) del robot en diferentes perspectivas.
- Qt, tanto para la creación de la interfaz de usuario, como para la creación de la ventana en la que mostrar la simulación del robot.



### 3.1. 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) u OpenGL. Se decidió utilizar OpenGL puesto que es la librería más conocida, pública y multiplataforma, además de que 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 proporciona funciones con las cuales se pueden renderizar imágenes, realizar animaciones, juegos, simulaciones, etc. Adicionalmente, OpenGL también permite:

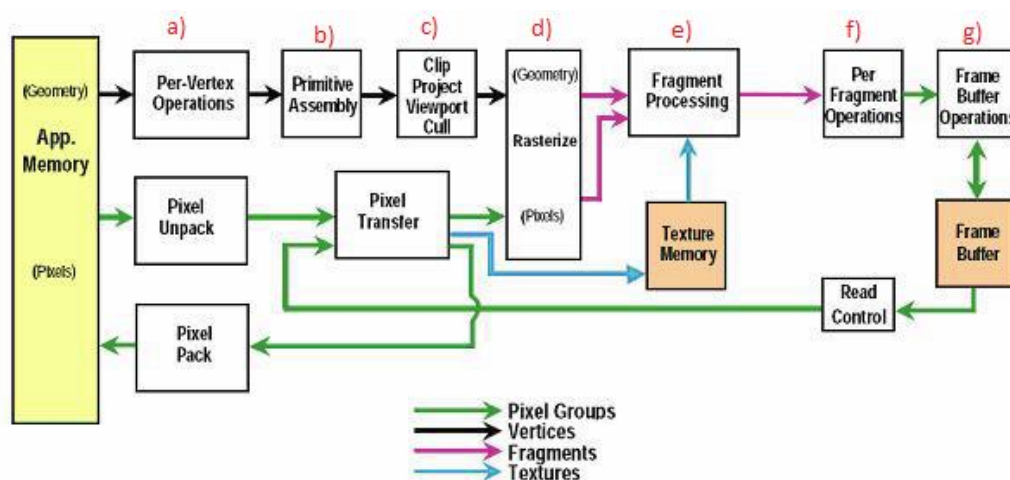
- Construir formas geométricas a partir de las primitivas que este proporciona.
- Ubicar los objetos en la escena.
- Ubicar el punto desde el que se visualiza la escena.
- Poner color o texturas.
- Crear luces.
- Realizar la rasterización.

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, como se verá más adelante, se ha utiliza Qt en este proyecto.

El cauce grafico 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. Dicho cauce 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. OpenGL define las etapas de un cauce grafico común para todos los tipos de hardware.



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

De todas estas etapas, se describen a continuación las asociadas al procesamiento de los vértices de la escena (a) por ser las más importantes para el desarrollo de este proyecto.

## 3.2. Etapa Geométrica

En esta etapa se transforma las coordenadas de los vértices de los objetos 3D en su sistema local a su proyección en 2D. Dicha etapa lleva a cabo su tarea mediante una serie de cálculos matriciales. En concreto, para obtener las coordenadas donde se encuentra cada punto del objeto dentro de la imagen final es necesario calcular la matriz model-view-projection (MVP). Como su nombre indica, esta matriz viene dada por la multiplicación de tres matrices distintas, que implementan diferentes funcionalidades. La ecuación (12) muestra el cálculo llevado a cabo por dicha etapa, donde  $\bar{v}'$  es el vértice proyectado visto desde la cámara y  $\bar{v}$  el vértice del modelo del objeto a visualizar [3].

$$\bar{v}' = M_{projection} \cdot M_{view} \cdot M_{model} \cdot \bar{v} \quad (12)$$

Esta ecuación requiere como entrada cada uno de los vértices de los modelos a visualizar. Para la creación de los modelos 3D se utiliza la librería *freeglut*.

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.

### 3.2.1. Posicionamiento del modelo

Para posicionar nuestro modelo del robot se utiliza la matriz *model* ( $M$ ). Esta matriz realiza una transformación de las coordenadas de los vértices del modelo para ubicarlo en la escena. Normalmente es una combinación de tres posibles transformaciones elementales: traslación, escalado y rotación. Cada una de estas transformaciones vienen dadas por matrices, las cuales se multiplican unas sobre otras dando como resultado la matriz *model*. En el caso que nos ocupa, posicionar al

robot en el plano XZ con una determinada orientación, la matriz *model* se calcula mediante la ecuación (13), es decir, se realiza primero una rotación respecto al eje y ( $R_y$ ) y después una translación en el plano XZ ( $T_{xz}$ ).

$$M = T_{xz} \cdot R_y \quad (13)$$

Las matrices específicas que se utilizan en este proyecto se pueden observar en las ecuaciones 14 y 15. En la ecuación 14 se puede ver la matriz de rotación en Y donde  $\varphi$  representa el ángulo que rota el robot.

$$R_y = \begin{bmatrix} \cos \varphi & 0 & \sin \varphi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \varphi & 0 & \cos \varphi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (14)$$

En la ecuación 15 se puede observar la matriz de translación en el plano XZ sobre el que se mueve el robot.

$$T_{xz} = \begin{bmatrix} 1 & 0 & 0 & X \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & Z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (15)$$

En ambas matrices simplemente se debe sustituir las variables X, Y y  $\varphi$  que definen el estado del robot para que OpenGL lo ubique correctamente en la escena.

Las funciones de la librería GLM utilizadas para generar dichas matrices son:

***translate(m, v)***, donde ***m*** es la matriz del modelo (inicializada a la matriz identidad) y ***v*** es el vector que contiene las cantidades a trasladar, (x,0,z) en nuestro caso. Tras ejecutar esta función ***m*** contiene la matriz mostrada en la ecuación (12), es decir, la matriz del modelo que ubica y orienta nuestro robot en la escena.

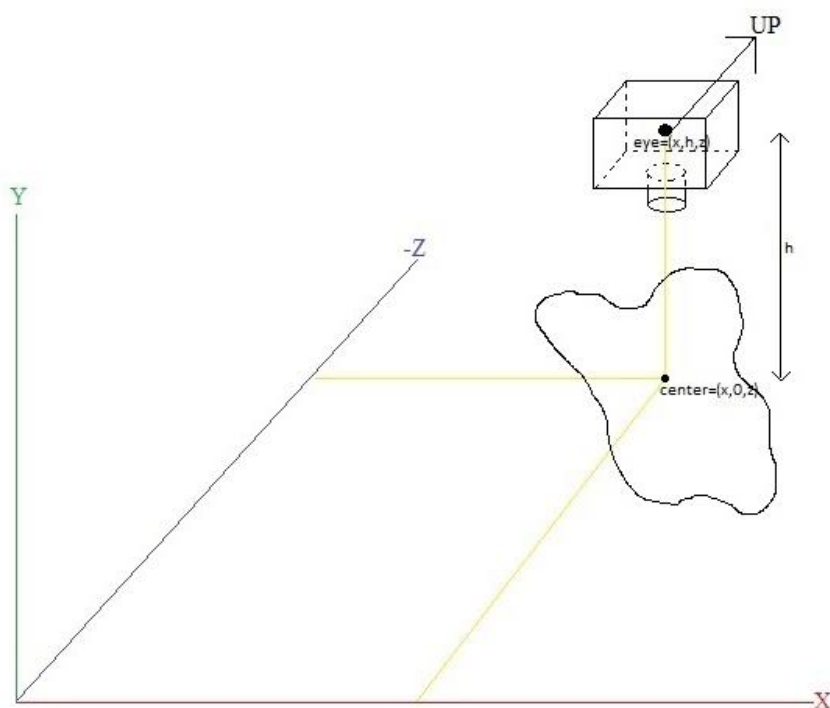
***rotate(m, angle, axis)*** donde ***m*** es la matriz anterior, ***angle*** es el ángulo que se rota y ***axis*** es un vector que indica en que eje rota, (0,1,0) en nuestro caso.

### 3.2.2. Posicionamiento de la cámara

Para posicionar la cámara se utiliza la matriz view ( $V$ ), necesitando para su creación la posición de la cámara, el punto hacia el que mira y la orientación de la cámara.

La función de la librería GLM que se utiliza para generar esta matriz es ***lookAt(eye, center, up)***, dicha función posiciona la cámara según sus parámetros de entrada que son ***center*** la posición de la cámara, ***eye*** el punto hacia donde mira, y la orientación de la misma ***up***, como se muestra en la figura 16.

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 (mayor y menor) del circuito en los dos ejes ( $x,z$ ) y con ellos se calcula el punto central del circuito.

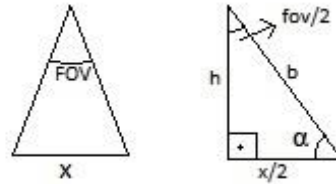


**Figura 16.** Posición y orientación de la cámara.

Una vez determinado ***center***, se calcula en qué eje ( $X,Z$ ) el circuito tiene la mayor dimensión. Con dicha información y el ***fov*** de la cámara podemos calcular mediante trigonometría la altura ( $h$ ) a la que se debe posicionar la cámara para



poder ver todo el circuito. En la ecuación (16) 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 17.** Trigonometría utilizada para el cálculo de la altura de la cámara.

$$\alpha = 180 - FOV - 90$$

$$\frac{h}{\text{sen}(\alpha)} = \frac{b}{\text{sen}(90^\circ)} = \frac{x/2}{\text{sen}(FOV/2)} \quad (16)$$

$$h = \frac{x/2 \cdot \text{sen}(\alpha)}{\text{sen}(FOV/2)}$$

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. Estas posiciones son  $center = (x_c, h, z_c)$  y  $eye = (x_c, h, z_c)$ .

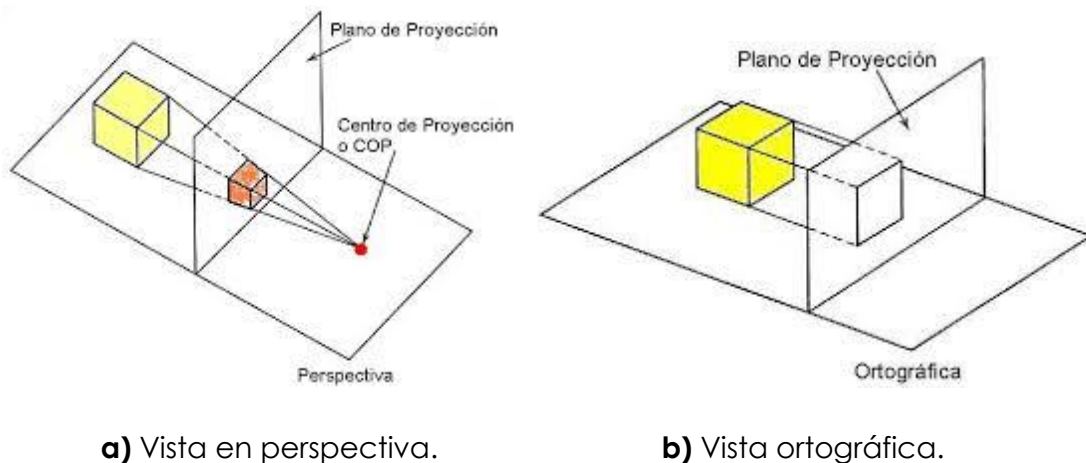
Una vez tenemos la posición de la cámara, dependiendo de que opción haya elegido el usuario se hacen los cálculos para la proyección en perspectiva u ortográfica, esto implica que se cambia también la posición de la cámara obtenida anteriormente. En caso de que se opte por una visualización en ortográfica 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 posibilidad de poder hacer zoom en ambas proyecciones con la rueda del ratón.

### 3.2.3. Modelo de proyección

El modelo de proyección se establece mediante la matriz *projection* (*V*), teniendo dos posibilidades: proyección ortográfica o en perspectiva.

La diferencia principal entre estos dos tipos de vistas 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 18 a) y 18 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 18.** 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 la librería 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, ***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 [11].

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

### 3.3 Qt

Tal y como se ha comentado en la sección 3.1, OpenGL no puede crear ventanas y no tiene una forma sencilla de interactuar con el usuario. Por todo ello, 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 de 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 [8]. 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.

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.

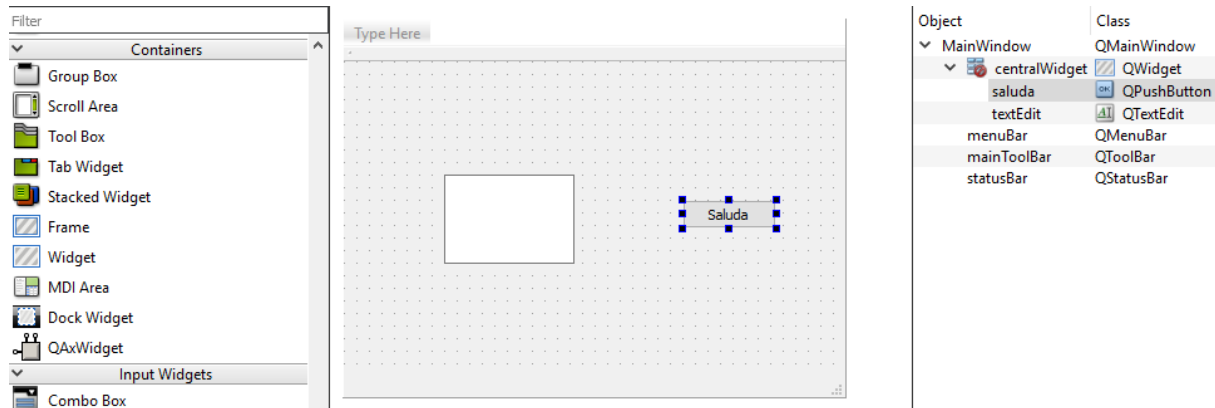


Figura 19. Qt Designer.

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

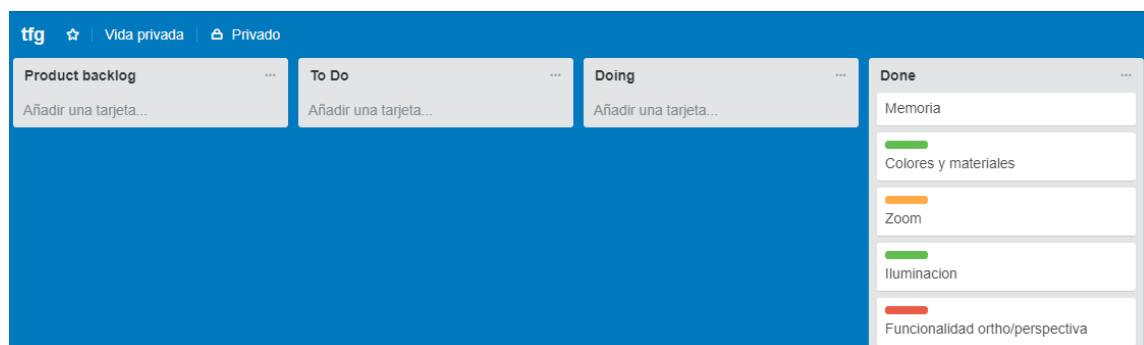
- **initializeGL():** Esta función se ejecuta una sola vez y antes que las otras dos funciones. Por lo tanto, se utiliza para inicializar y configurar todo lo necesario para la utilización de OpenGL u otros.
- **paintGL():** Esta función es la que renderiza la escena de OpenGL y se llama siempre que 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 para ajustarlo a las dimensiones de la ventana en la que mostrar la simulación, siendo *w* y *h* el ancho y alto de la misma en píxeles. 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 además de recalcular la matriz *projection* cada vez que se modifiquen las dimensiones de la ventana.

En cuanto a las funciones relacionadas con la GUI se trata de las funciones típicas que podrías encontrarte en cualquier otro software que incluya interacción con el usuario. Por ejemplo para la gestión de eventos de entrada, la función que se ejecuta al pulsar un botón es `on_nombreboton_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.

### 3.4. Metodología ágil

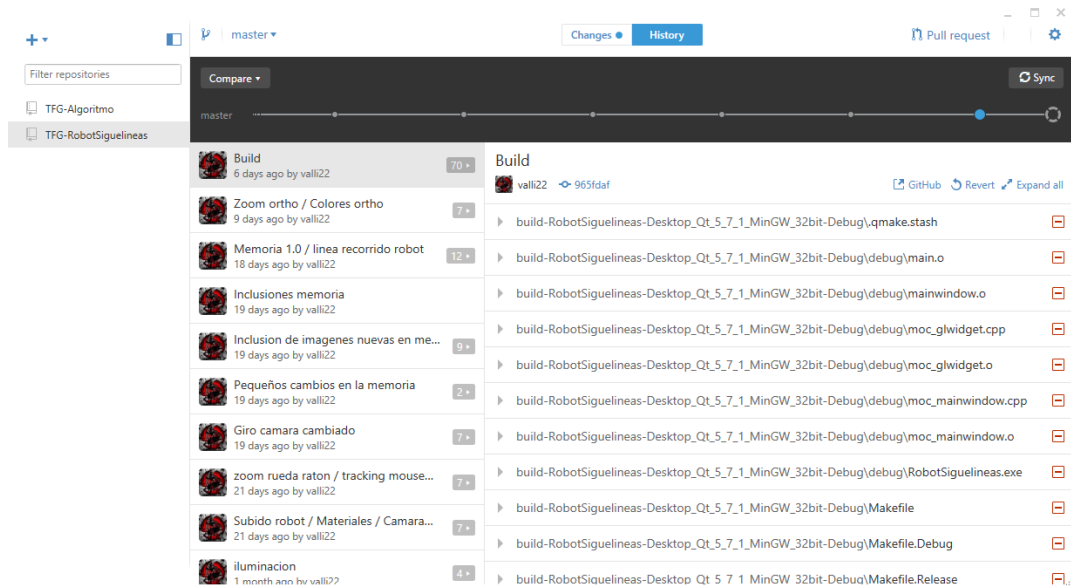
Durante el desarrollo de la aplicación se ha hecho uso de las herramientas habitualmente utilizadas en proyectos en los que se aplica metodología ágil. Estas herramientas han sido:

- **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.

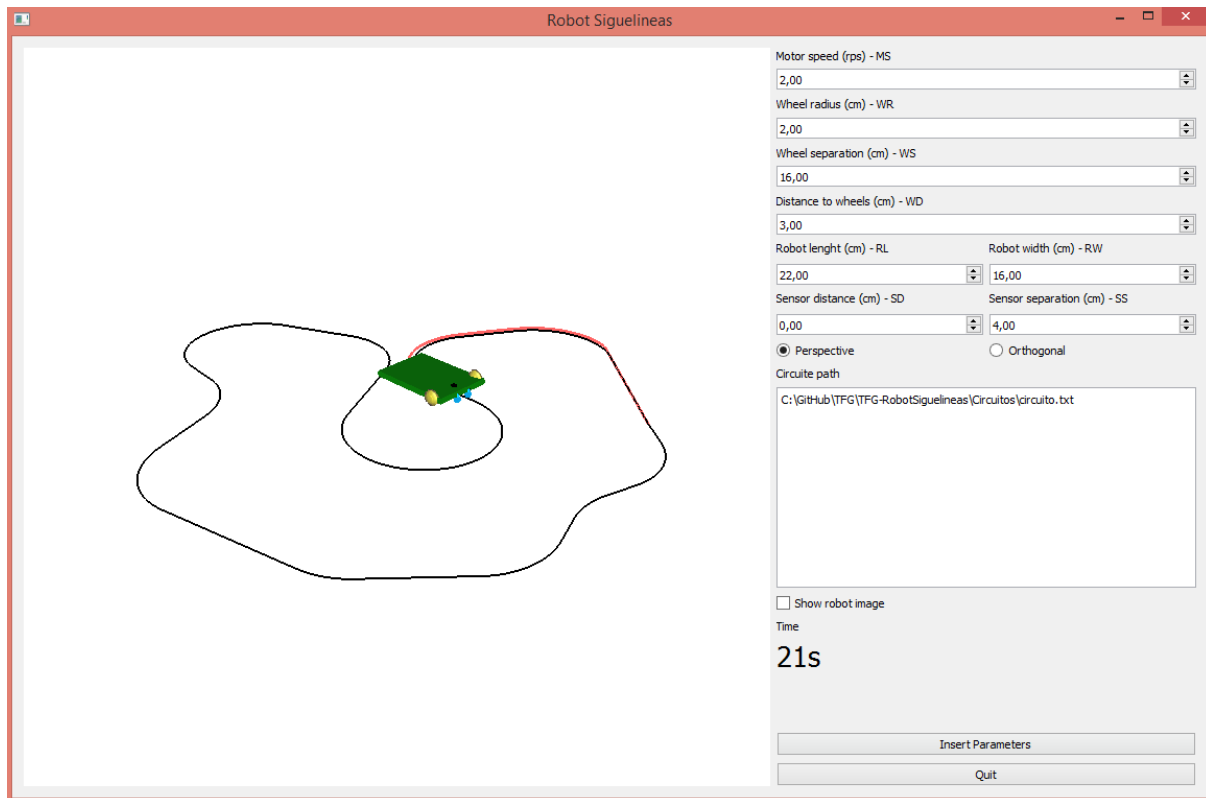
- **Git** como repositorio y control de versiones (concretamente Github). Sobre este repositorio se han ido subiendo los diferentes incrementos de funcionalidad de la aplicación de forma periódica, llevándose a cabo el control de las versiones correspondientes. Se puede observar el repositorio desde la aplicación de Windows en la figura 21.



**Figura 21:** Repositorio en Github del proyecto.

## Capítulo 4 Descripción de la aplicación

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

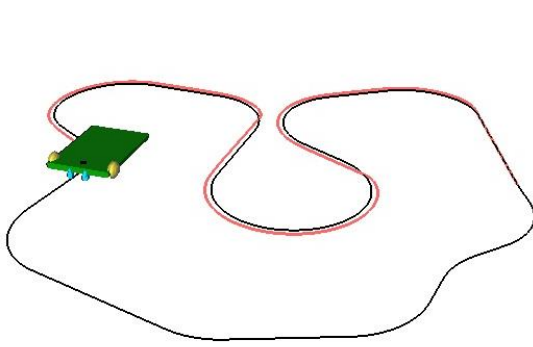


**Figura 22.** Vista completa de la aplicación.

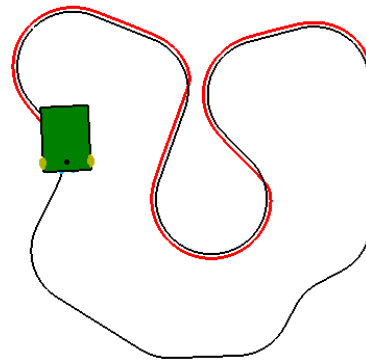
### 4.1. Zona izquierda: Viewport

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 del mismo.

En esta zona se puede observar la simulación de la aplicación, en función de los parámetros ingresados en la parte derecha de la aplicación. La figura 23 muestra el aspecto en 2 casos de uso con diferentes modelos de proyección: a) perspectiva, y b) ortográfica.



a) Vista en perspectiva.

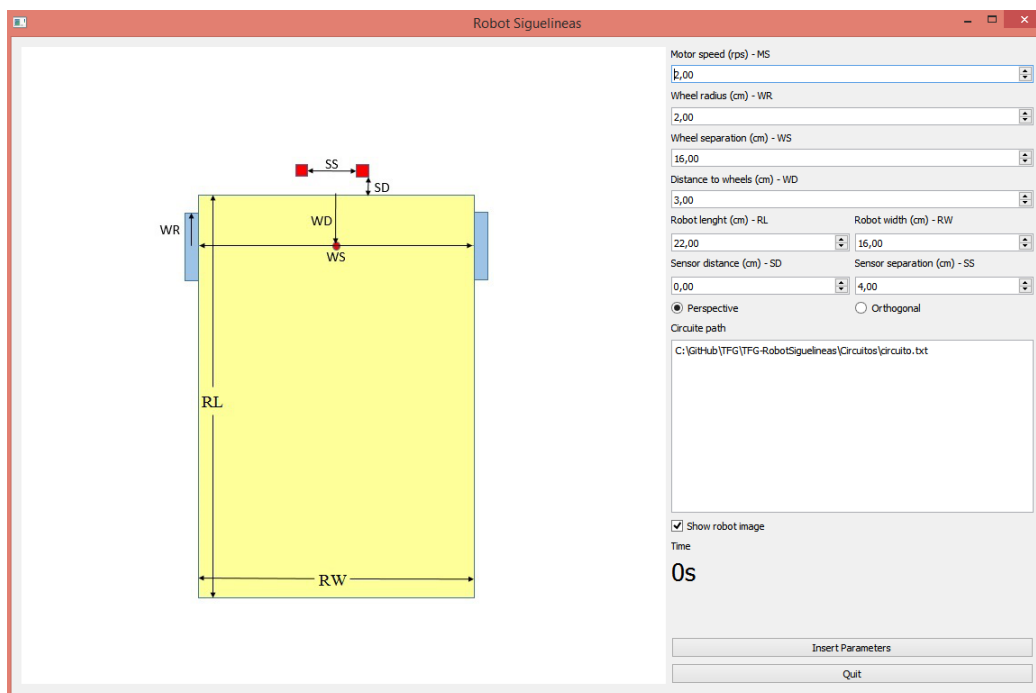


b) Vista en ortográfica.

**Figura 23.** Viewport.

El viewport muestra en todo momento, además de la simulación del robot, 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 del mismo como se muestra en la figura 24. Esta imagen se puede activar y desactivar en todo momento desde la zona derecha de la aplicación.



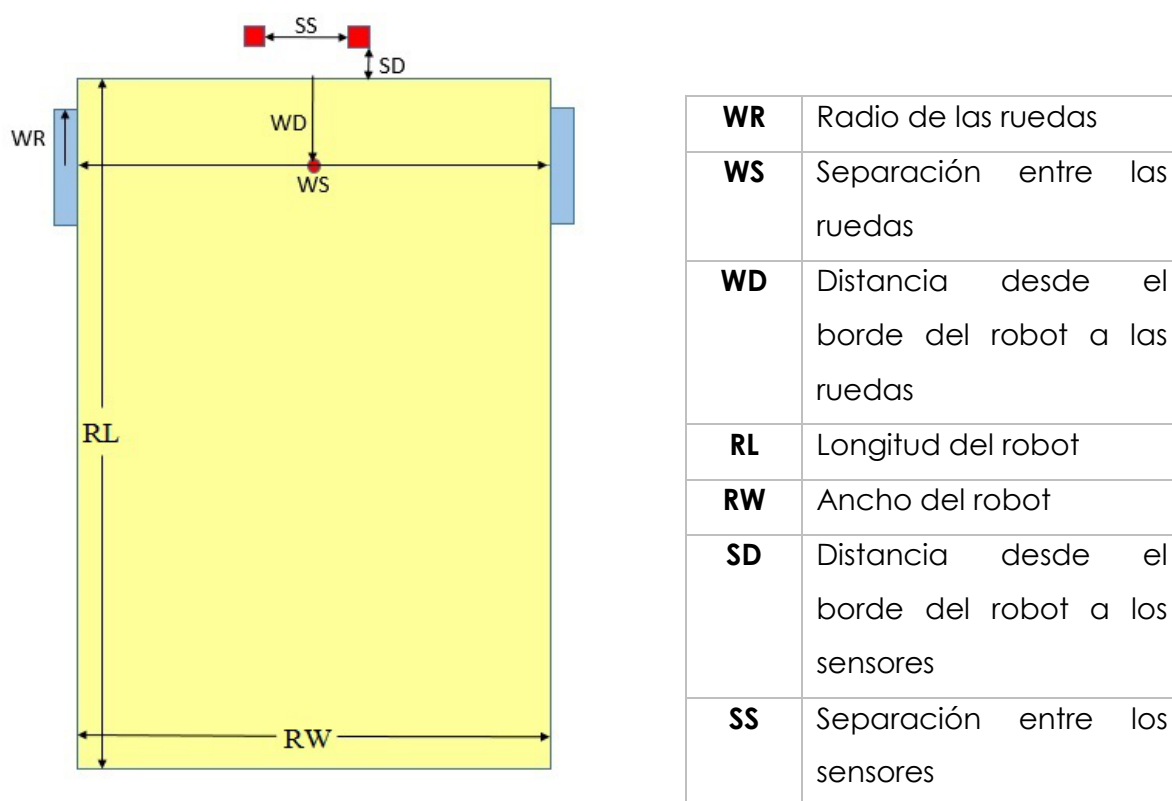
**Figura 24.** Aplicación con imagen de datos geométricos activa.



## 4.2. Zona derecha: campos de entrada de datos

La zona derecha de la aplicación se centra en la recogida de datos por parte 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. En la figura 25, se muestra esta imagen y una tabla de abreviaturas que explican el significado de cada una de ellas.

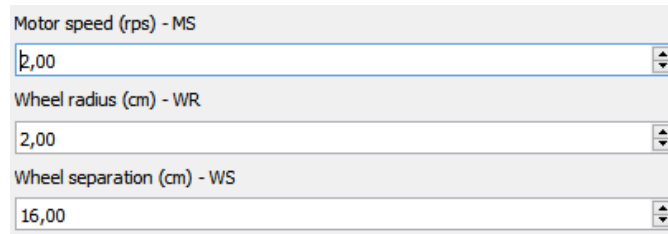


**Figura 25.** Imagen de referencia sobre datos geométricos del robot y tabla de abreviaturas.

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, donde el usuario está más tiempo observando y donde más detalles hay.

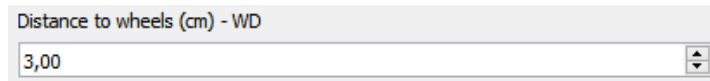
La parte de interacción con el usuario se coloca a la derecha por similitud a la mayor parte de aplicaciones encontradas, resultando así más natural el acceso a la misma por parte del usuario. Además, todas las etiquetas están en inglés puesto que este es el idioma más hablado, dotando así a la aplicación de una mayor usabilidad.

A continuación, se opta por agrupar las entradas de acuerdo a 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 entradas con sus respectivas etiquetas 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.



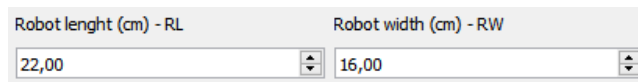
**Figura 26.** Paquete de ruedas en tres niveles.

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 WD. Por lo tanto, este parámetro debe estar situado entre los parámetros de las ruedas y los del robot. Esta opción se puede observar en la figura 27.



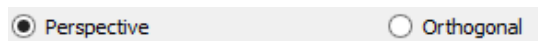
**Figura 27.** Opción para cambiar la distancia de las ruedas.

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. El resultado se muestra en la figura 28.



**Figura 28.** Paquete de dimensiones del robot en un nivel

La figura 29 muestra los botones de radio que deciden si el usuario desea la vista perspectiva u ortográfica.

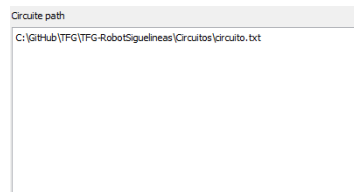


**Figura 29.** Radio buttons para elegir tipo de proyección.

Se ha decidido colocar esta entrada tras la introducción de los parámetros del robot ya que si se pusiera detrás de la entrada del circuito esta opción sería poco visible y

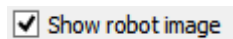
podría ignorarse, mientras que viniendo de opciones de tamaño reducido se ve más claramente.

La última entrada es un cuadro de texto editable en el que se debe introducir el path del circuito que se desea utilizar, ver figura 30. Se ha optado por la utilización de un cuadro de texto en lugar de un explorador de archivos debido a que Qt no tiene implementado un widget para la exploración de archivos.



**Figura 30.** 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 31.



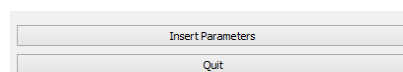
**Figura 31.** Checkbox que muestra o esconde la imagen de referencia del robot.

Después, hay una zona en la que se muestran los segundos pasados desde que empezó la simulación como se muestra en la figura 32, con un tamaño de letra que hace que resalte sobre el resto de la interfaz.



**Figura 32.** Tiempo de simulación.

Por último, hay dos botones. El primero cuyo texto pone "Insert Parameters", el cual al ser pulsado se introducen todos los parámetros a la aplicación e inicia la simulación. El último botón contiene el texto "Quit", y sirve para cerrar la aplicación. Estos dos últimos botones se pueden observar en la figura 33.



**Figura 33.** Botones de inicio de simulación y cerrar aplicación.

## 4.3. Casos de uso

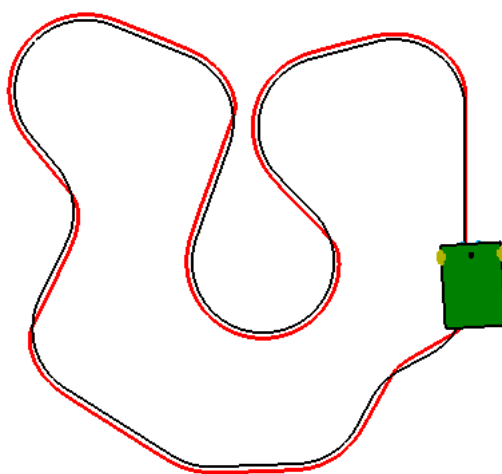
A continuación, se muestran los resultados obtenidos utilizando diferentes configuraciones de robots y circuitos. Los casos que se van a mostrar son los siguientes:

1. Robot estándar tomado como referencia.
2. Robot estándar modificando la separación entre ruedas.
3. Robot estándar modificando el radio de las ruedas.
4. Robot estándar modificando la separación entre sensores.
5. Robot estándar modificando la distancia entre ruedas y sensores.
6. Robot estándar en un circuito distinto.
7. Robot estándar con vista en perspectiva.

### 4.3.1. Robot de referencia

Este robot es un robot con parámetros geométricos medios que se utiliza como referencia para el resto de los casos. Reproduce el robot real que, a modo de ejemplo, el profesor proporcionaba a sus alumnos en la asignatura Robótica.

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

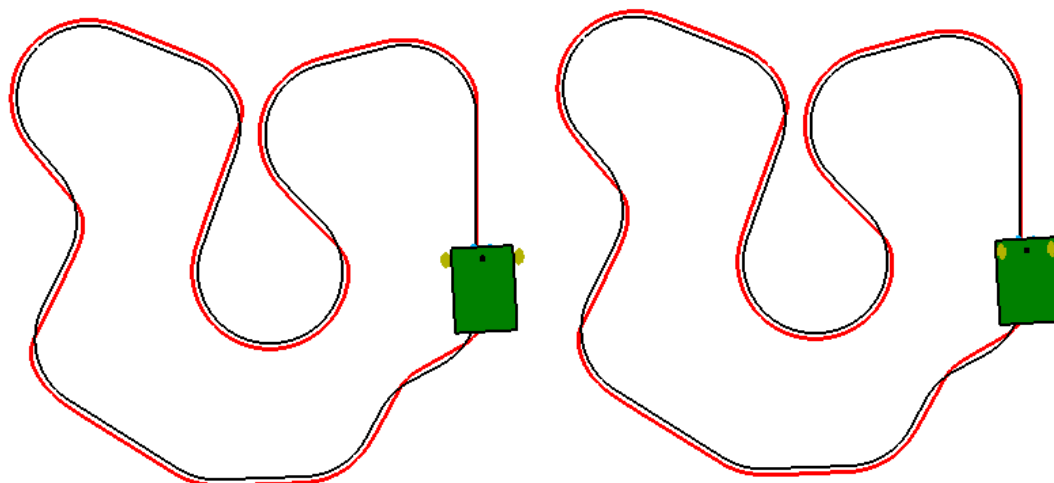


**Figura 34.** Robot estandar que se toma como referencia.

### 4.3.2. Separación entre ruedas

En este caso se va a modificar la distancia que tienen las ruedas entre sí. Respecto al robot estándar, en el primer caso se ha aumentado y en el segundo se ha reducido.

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>

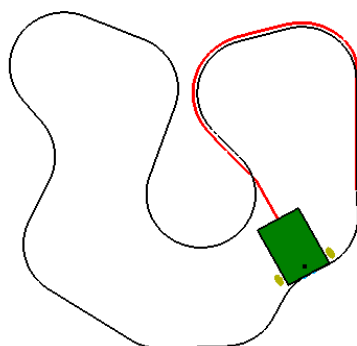


a) Mayor separación.

b) Menor separación.

**Figura 35.** 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.



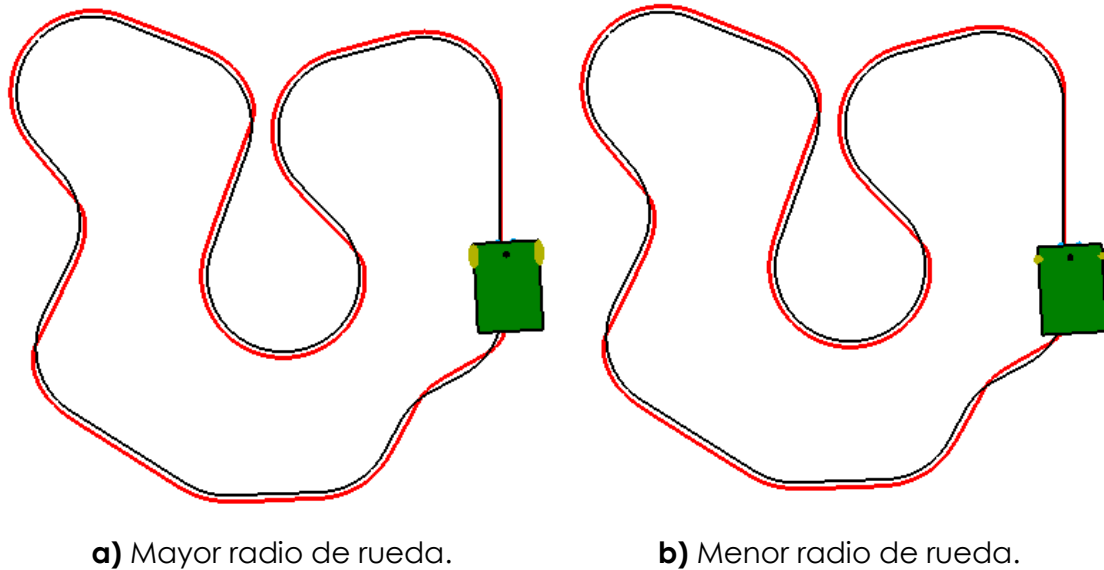
En esta imagen se puede observar el caso en el que se ha aumentado la separación de ruedas hasta 20cm. Con esta separación el robot no es capaz de girar lo suficientemente rápido como para no salirse del circuito en determinadas curvas.

**Figura 36.** Excesiva separación entre ruedas.

### 4.3.3. Radio de las ruedas

En este caso se ha aumentado y disminuido el radio de las ruedas respecto a la del 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 37.** 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.

Como se puede apreciar la modificación de este parámetro no genera ningún cambio en la trayectoria.

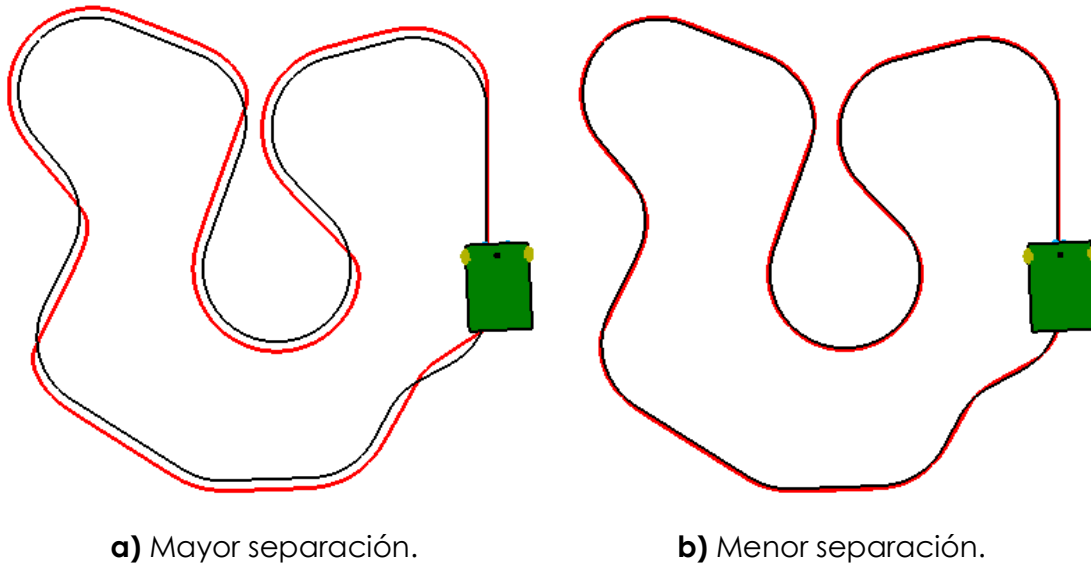
A la vista de este resultado, el alumno podría pensar en aumentar el radio de las ruedas al máximo, sin embargo, los motores podrían no ser capaces de mover un robot con un radio de rueda demasiado grande en un robot muy pesado.

Este último aspecto no se ha incluido en el simulador, por lo que para seleccionar el radio óptimo de las ruedas se debe disponer del robot real.

### 4.3.4. Separación entre sensores

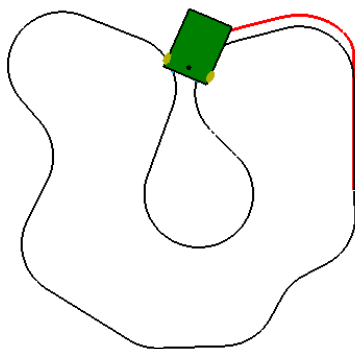
Aquí se realizan modificaciones en la separación entre los sensores, aumentando o disminuyendo esta respecto a la del caso base.

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



**Figura 38.** 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 haciéndose, por tanto, el recorrido más ajustado al camino a seguir.



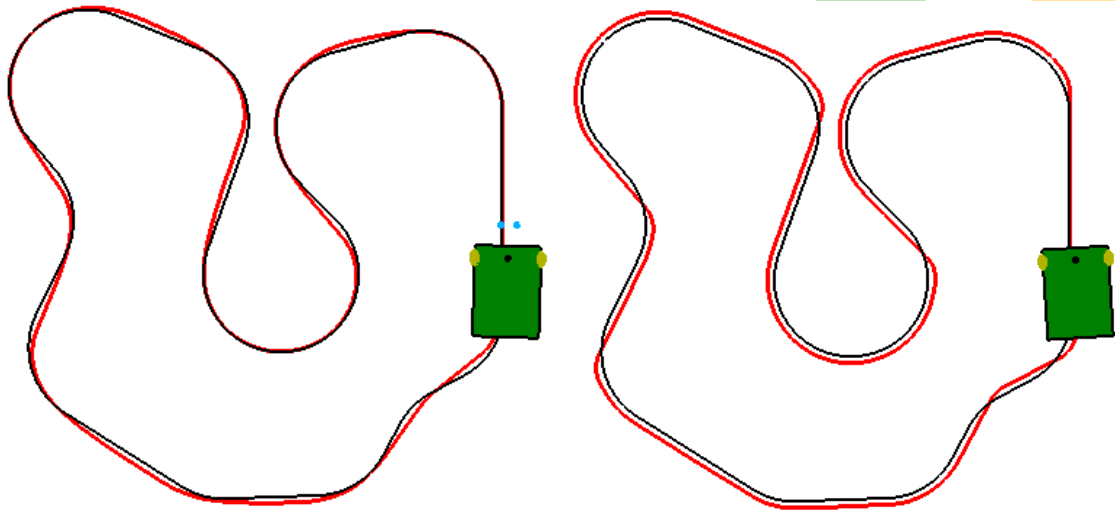
Otro caso interesante que permite probar el simulador es el de una separación excesiva de los sensores. Como ejemplo, ampliar la separación de los sensores a 9cm estos colisionan a la vez con varias partes del circuito y este se queda parado.

**Figura 39.** Excesiva separación entre sensores.

### 4.3.5. Separación entre ruedas y sensores

En este caso se ha modificado la distancia entre los ejes del robot, es decir, la distancia entre el eje de las ruedas y el eje de los sensores respecto a la del caso base.

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

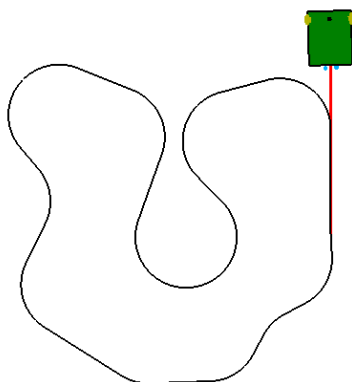


a) Mayor separación entre ejes.

b) Menor separación entre ejes.

**Figura 40.** 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.



Como curiosidad se ha probado a colocar los sensores en la parte trasera del robot, con un resultado nefasto, haciendo que el robot en la primera curva no pueda realizar el giro y por lo tanto saliéndose del circuito.

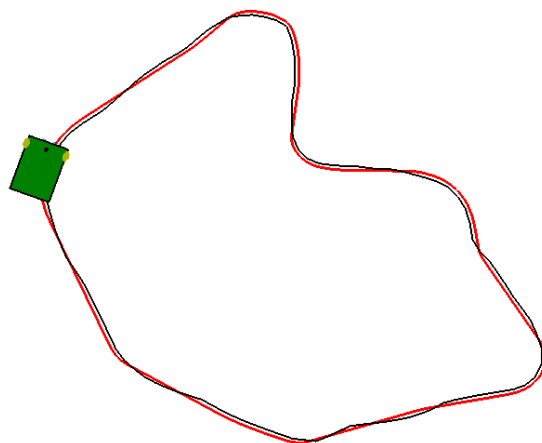
**Figura 41.** Sensores en la parte trasera.



### 4.3.6. Circuito diferente

Finalmente, se ha utilizado el robot estándar en un circuito diferente al utilizado en el resto de las pruebas.

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

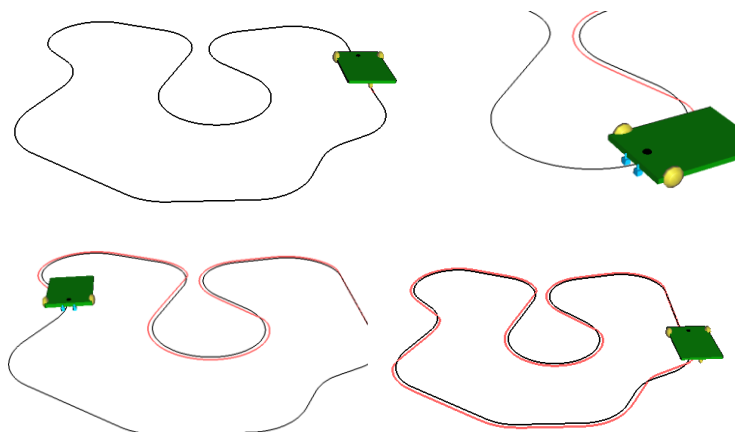


**Figura 42.** Circuito alternativo.

### 4.3.7. Vista en perspectiva

Como caso adicional se muestran varias imágenes del robot estándar realizando el circuito en vista perspectiva.

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



**Figura 43.** Vista en perspectiva

## Conclusiones

Gracias a este proyecto he aumentado 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.

En lo respectivo a las dificultades encontradas, la mayor dificultad residió en la correcta simulación del robot, la inclusión de la interacción del usuario con la aplicación ha resultado ser bastante sencilla gracias a Qt. Además, puesto que los métodos de OpenGL que utiliza Qt realizan las mismas funciones, a pesar de tener diferentes nombres, me ha resultado sencillo implementar la mayor parte de OpenGL.

De los objetivos principales propuestos a la hora de realizar la aplicación, han sido todos logrados. Aun así, hay algunas mejoras que pueden plantearse como trabajo futuro.

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. 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 este realice el recorrido en el menor tiempo posible.

Finalmente, respecto a la interfaz gráfica podría incluirse 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>