

**UNIVERSIDAD**

**REY JUAN CARLOS**

**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA**

**DOBLE** **GRADO EN INGENIERIA INFORMATICA E INGENIERIA SOFTWARE**

**Curso Académico 2017/2018**

**Trabajo Fin de Grado**

## **SIMULACIÓN DE UN ROBOT SIGUE LINEAS**

### **Autor: David Vacas Miguel**

### **Director/Tutor: Alberto Herrán González**

Índice

[Resumen 3](#_Toc508543095)

[Capítulo 1 – Introducción 4](#_Toc508543096)

[1.1 Motivación 4](#_Toc508543097)

[1.2 Objetivos 7](#_Toc508543098)

[1.3 Estado del arte 9](#_Toc508543099)

[1.4 Requisitos 10](#_Toc508543100)

[Capítulo 2 - Entorno tecnológico 11](#_Toc508543101)

[2.1 Cauce grafico 11](#_Toc508543102)

[2.2 OpenGL 17](#_Toc508543103)

[2.3 Qt 18](#_Toc508543104)

[Capítulo 3 – Implementación 21](#_Toc508543105)

[3.1 Dinámica del robot 21](#_Toc508543106)

[3.2 Interfaz grafica 25](#_Toc508543107)

[3.3 Detalles implementación 29](#_Toc508543108)

[Capítulo 4 – Implementación 34](#_Toc508543109)

[4.1 Casos de uso 34](#_Toc508543110)

[4.2 Resultados 35](#_Toc508543111)

[Capítulo 5 – Conclusiones 38](#_Toc508543112)

Resumen

La aplicación realiza la simulación de un robot sigue líneas que, dado los parámetros del robot y un circuito, completará este último sin salirse del mismo.

Mecánicamente el robot se basa en dos sensores situados en la parte delantera del robot que comprueban si están sobrepuestos al circuito, en caso afirmativo la rueda correspondiente se frenará, reducirá su velocidad o, incluso, ira marcha atrás para que el robot realice un giro que le permita continuar su trayectoria dentro del circuito. En nuestro caso la rueda frenará.

Capítulo 1 – Introducción

1.1 Motivación

En una clase de la asignatura robótica, los alumnos, en grupos, debían realizar robots de distintos tipos: sigue líneas, velocistas, etc. Al final del curso el profesor realizaba un circuito y los alumnos usaban los robots que habían construido para ver cuál era el más rápido, creando así una competición de robots entre los alumnos. Lo rápido que el robot realizaba el circuito venía dado por las medidas que los alumnos utilizaban a la hora de construir el mismo, por ejemplo un robot con una separación de sensores mayor realizará menos giros pero más grandes sin embargo un robot con una separación menor realizará más giros pero de menor magnitud y esto junto con los demás parámetros influirá en la velocidad para realizar el circuito.

El objetivo de este proyecto es desarrollar una aplicación que permita simular y visualizar la dinámica de este robot móvil con direccionamiento diferencial. La aplicación servirá de banco de pruebas con el que analizar el comportamiento del robot sigue líneas antes de su construcción real.

Actualmente se dispone de un desarrollo previo en MATLAB-Simulink, ver figura 1 y figura 2.



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



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

Sin embargo tiene varios inconvenientes como: a) para poder utilizarlo hay que tener instalado dicho software; b) no permite cambiar los parámetros del robot de una forma sencilla, estos parámetros se debían escribir en un vector cuya estructura debes conocer, como se puede ver en la figura 3; y c) no ofrece todas las funcionalidades requeridas.



**Figura 3.** Se puede observar el vector en el cual se deben poner los parámetros del robot.

Por ello, se pretende desarrollar dicho software creando una aplicación (integrando las tecnologías C++, Qt y OpenGL) cuyo único requisito para su uso sea el de tener instaladas en el sistema las librerías necesarias.

El destinatario de la aplicación son alumnos de una asignatura en la que se pretende que jueguen con el simulador antes de la construcción del robot real.

1.2 Objetivos

Una vez vistas las carencias de la aplicación que se tenía anteriormente, los objetivos que servirían para subsanarlas, además de otros objetivos adicionales serían los siguientes:

* Eliminar la necesidad de tener software instalado: con esta aplicación no tendrás la necesidad de tener software instalado, más allá de las librerías necesarias para la aplicación, por lo que será más fácil su descarga y utilización inicial.
* Facilidad a la hora de realizar cambios: el mayor problema a la hora de crear el robot era que una vez decidas las medidas y comprado o creado la pieza en cuestión era complicado cambiar esta para utilizar una distinta. Por ejemplo si se decidía que el radio de las ruedas seria de X una vez comprada la rueda deberían comprar una distinta si querían cambiarlo, otro ejemplo es el tamaño del robot, si se decidían porque el robot tuviera unas dimensiones de (Y, Z) pero más tarde se daban cuenta de que sería mejor tener un robot con unas dimensiones mayores, estos deberían volver a crear el soporte del robot que es el que indica las dimensiones.

Esto se soluciona en la aplicación con una simple interfaz user-friendly en la que podrán cambiar tantas veces como quieran las medidas de su robot, de forma sencilla y natural.

* 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 era 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 puedes tener tu robot simulado corriendo por el circuito dado sin necesidad de crear físicamente los mismos.

* Mejora del aprendizaje: igual que en el anterior punto, puesto que los alumnos no podían realizar pruebas con el robot de forma sencilla, no podían observar detenidamente los cambios que se creaban en el movimiento del robot utilizando diferentes medidas.

Puesto que aquí tienen una forma rápida de simular el movimiento del robot, pueden experimentar haciendo cambios en las medidas para poder ver qué cambios se realizan en el movimiento del robot y con esto ir aprendiendo con la visualización del mismo.

1.3 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 las cuales 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 están 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 muy fácilmente 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.

1.4 Requisitos

Los requisitos que debe tener la aplicación serían los siguientes:

* 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 ver mejor los movimientos de tu robot.

Capítulo 2 - Entorno tecnológico

2.1 Cauce grafico

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. Actualmente la generación de estas imágenes sigue una serie de pasos ya definidos.

El cauce grafico 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 4 podemos observar las diferentes etapas y como se conectan como hemos citado anteriormente.

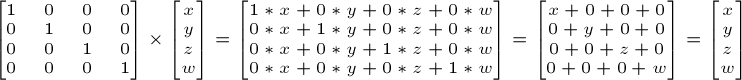


**Figura 4.** Etapas del cauce gráfico.

Las tres etapas principales del cauce grafico son: transformación, rasterizado y sombreado.

2.1.1 – Transformación

En esta etapa se transforma 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. A continuación, realizaré una breve explicación sobre las matrices más relevantes para el proyecto. Si se quiere utilizar en 3D, las matrices que se utilizan para los cálculos son matrices 4x4 siendo la matriz identidad la presentada en la figura 5.

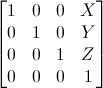


**Figura 5.** Matriz identidad.

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.

* **Model:** Esta matriz lo que realizará es 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 multiplicarán unas sobre otras, comenzando por la matriz identidad, dando así una única matriz que será la matriz model. Las matrices utilizadas para la traslación vienen dadas en la figura 6, para el escalado en la figura 7 y las diferentes matrices para la rotación, que dependen de en qué eje gires, en la figura 8.

Para realizar la traslación lo único que debes hacer es sustituir xyz por la cantidad de unidades que quieras mover en ese eje, así pues si sustituyes x por 5 realizarás una traslación de 5 unidades en el eje X.



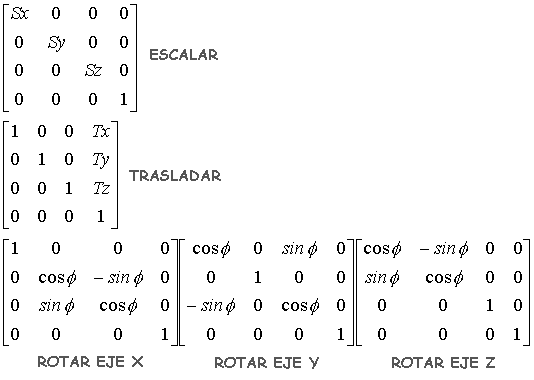
**Figura 6.** Matriz de translación.

En el caso del escalado la sustitución que realices indica el porcentaje de escalado que vas a realizar, esto resulta en que una sustitución por 1 no realiza escalado, una sustitución por 0,5 escala a la mitad de tamaño en el eje correspondiente y una sustitución por 2 escala al doble de tamaño en ese eje.



**Figura 7.** Matriz de escalado.

Para las rotaciones simplemente hay que sustituir *ϕ* por los grados que quieres hacer girar en ese eje el objeto. En caso de que quieras realizar varias rotaciones en distintos ejes, debes crear las diferentes matrices y multiplicarlas a la matriz principal.



**Figura 8.** Matrices de rotación.

* **View:** 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. Además, esta matriz no solo sitúa la cámara, sino que también realiza los cálculos para simular los movimientos que la cámara realizaría. Puesto que no se puede mover la cámara, lo que se realiza es mover el mundo en concordancia con esto, es decir se aplica al mundo la inversa del movimiento que la cámara realizaría, consiguiendo así el efecto de que la cámara se está moviendo. Por consiguiente, la posición de la cámara también es ilusoria puesto que la cámara siempre está en un punto fijo, pero como en el caso de querer mover la cámara, al decir que la posición de la cámara está en la posición X se está diciendo que el resto de objetos que no son la cámara deben reposicionarse para que al usuario le parezca que la cámara ha tenido un movimiento hacia otro punto del espacio.

* **Projection:** Esta matriz definirá como se realiza la proyección, poniendo en esta matriz si la proyección será en perspectiva u ortográfica. Además, permite realizar el clipping (desactivar la renderización) de los vértices que no son visibles.

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 la vista en perspectiva se acerca más a la vista de una persona. En la figura 9 y 10 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 9.** Vista en perspectiva. **Figura 10.** Vista ortográfica.

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.

* **MVP:** Una vez tenemos las tres matrices habrá que multiplicarlas, siempre en el mismo orden, junto con el vector de posición del objeto en cuestión (v) y esto dará las nuevas coordenadas (v’), como se muestra en la siguiente ecuación:

v’ = Mprojection \* Mview \* Mmodel \* v

2.1.2 – Rasterizado

En esta etapa se convierten los vértices en fragmentos, estos últimos pueden verse como pixeles en potencia.

2.1.3 – Sombreado

Este paso es el que calcula el color de cada pixel, por lo tanto, es la etapa en la que se leen las texturas.

2.2 OpenGL

Para poder implementar el cauce grafico 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 integrarlo 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 (explicada en el punto anterior).

OpenGL tiene diferentes versiones que siguen pudiendo ser utilizadas hoy en día, a continuación, realizaré 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 este mismo año 2017. Se añaden una gran cantidad de funcionalidades

2.3 Qt

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.

Qt es un framework de desarrollo de aplicaciones multiplataforma para ordenador, embebido y móvil que en gran parte se suele utilizar para programas que utilicen una interfaz gráfica.

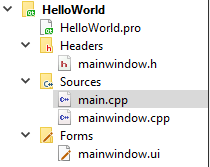
Internamente se utiliza C++ con alguna extensión para funciones como Signals y Slots, por lo tanto, es orientado a objetos. Puesto que es C++, en los proyectos se encontrarán archivos de 4 tipos:

* .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 públicas como privadas.
* .cpp: Son los archivos en los cuales está situado 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.

La integración de estas dos 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 estés utilizando que vas a utilizar esas librerías incluyendo lo siguiente “#include <QOpenGLWidget>”.

Mostrare rápidamente el “Hello World” en qt para poder observar un ejemplo sencillo. La estructura se puede observar en la figura 11. Como se puede observar, main.cpp será el que inicialice el programa, y mainwindow será el que se encargue de contener el código para mostrar en la aplicación “Hello World”.



**Figura 11.** Estructura “Hello World”.

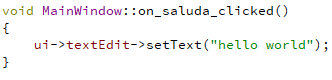
El fichero HelloWorld.pro es el que contiene la configuración del proyecto.

En primer lugar, se crean dos widgets, un botón y un espacio de texto, en el fichero mainwindow.ui (se puede utilizar la herramienta Qt Designer como se muestra en la figura 12) y se les nombra como desees, en este caso el botón se llama “saluda” y el cuadro de texto “textEdit”.



**Figura 12.** Mainwindow.ui.

El fichero mainwindow.cpp contiene el método *on\_saluda\_clicked()* que se llamará cuando se pulse el botón “saluda”. Este contiene el código que accederá al cuadro de texto y escribirá en él “Hello World”, este método se puede observar en la figura 13.



**Figura 13.** Método que escribirá “Hello World” al pulsar el botón.

Solo se necesitará añadir la cabecera del método en mainWindow.h y el programa estará terminado.

Capítulo 3 – Implementación

3.1 Dinámica del robot

3.1.1- Modelo cinemático

Según se ve en la Figura 14, suponiendo un sistema de referencia {G} y un sistema {L} con origen en el punto de guiado del vehículo y eje YL en la dirección del eje longitudinal del vehículo.



**Figura 14.** Modelo cinemático.

Por lo tanto, si el vehículo tiene una velocidad de desplazamiento y de rotación con respecto a {L}, con respecto a {G} la velocidad es:

3.1.2-Modelo discreto

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

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:

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

Por tanto, si se dispone de las coordenadas en cada instante de un determinado horizonte temporal, se puede calcular la nueva posición y orientación del robot en dicho horizonte.

3.1.3-Direccionamiento diferencial

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 15 se muestra una imagen de dicho esquema. Nótese que para especificar la configuración hay que indicar los valores de las tres variables (x, y, ), siendo las variables de control las velocidades de las ruedas laterales.



**Figura 15.** Locomoción mediante guiado diferencial.

Sean y las velocidades de giro de las ruedas izquierda y derecha, respectivamente. Si el radio de la rueda es , las velocidades lineales correspondientes son y . Es este caso, la velocidad lineal y velocidad angular correspondientes en el modelo vienen dadas por:

Sustituyendo estas expresiones en las obtenidas a partir de la Figura 14, se obtienen las velocidades de las coordenadas del robot en el sistema {G} a partir de la velocidad de giro de cada rueda:

Finalmente, utilizando el modelo discreto, se obtiene:

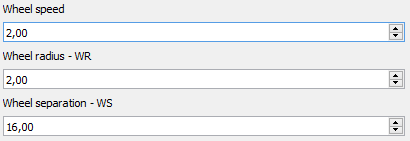
Estas últimas ecuaciones se utilizarán en el proyecto para el cálculo de la posición del robot y por lo tanto para el cálculo de la matriz model (vista en el apartado 2.1.1) mediante los métodos de la librería GLM que en este caso estos métodos realizan los mismos cálculos que los métodos de OpenGL: translate (para el cálculo de la matriz de translación, cuyo método en OpenGL seria glTranslate3f) y rotate (para el cálculo de la matriz de rotación, cuyo método en OpenGL seria glRotate3f).

3.2 Interfaz grafica

En este apartado explicaremos el porqué de las decisiones tomadas sobre la interfaz. El archivo que contiene toda la información sobre la interfaz y sus widgets es “MainWindow.ui”. 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 estará observando y donde más detalles hay que fijarse.

Una vez entrada en la parte de interacción con el usuario se decidió que estuviera a la derecha por similitud a la mayor parte de aplicaciones encontradas y por lo tanto que le resultara más natural al usuario el acceso a esta. Además, todas las labels estarán en inglés puesto que este es el idioma más hablado y por tanto dota a la aplicación de una mayor usabilidad.

A continuación, se decidió 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 decidió por utilizar tres inputs con sus respectivos labels en diferentes niveles, como se observa en la figura 16, puesto que la unión de los 3 parámetros de las ruedas en un mismo nivel no resultaba fácil de visualizar.



**Figura 16.** 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 debía 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 agruparon en 2 en un mismo nivel puesto que así se asimila a simple vista que esos parámetros están relacionados, como se ve en la figura 17.

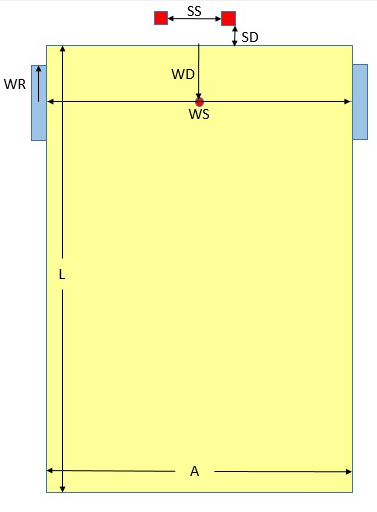


**Figura 17.** Paquete de dimensiones del robot en un nivel

Hasta aquí están los parámetros de entrada necesarios para la simulación del robot (a excepción del circuito). Por tanto, ahora deberían entrar las opciones de la aplicación que no son directamente relevantes para la simulación de la aplicación, en primer lugar, puedes encontrar dos radio buttons que deciden si el usuario desea la vista perspectiva u ortográfica. 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.

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.

A continuación, se sitúa un checkbox que activa y desactiva una imagen de referencia que tiene las indicaciones y las medidas del robot para que el usuario sepa exactamente qué es lo que esta modificando en todo momento. Además, en cada input de los anteriores se ha añadido la abreviatura al final para mayor entendimiento del usuario. Se puede observar la imagen que se da como referencia al usuario en la figura 18.



**Figura 18.** Imagen de referencia para el usuario para saber qué es lo que modifica en todo momento.

Por último, hay una zona en la que se muestran los segundos pasados desde que empezó la simulación como se muestra en la figura 19, 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.

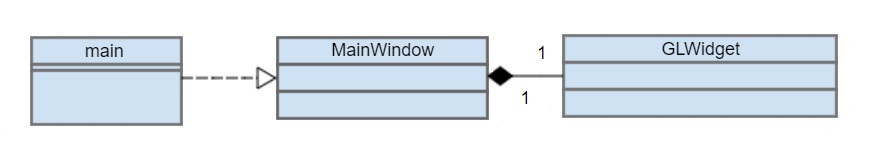
uiTiempo

**Figura 19.** Tiempo de simulación.

Por último, hay dos botones, el primero cuyo texto pone “Insert Parameters”, el cual cuando pulses se introducirán todos los parámetros a la aplicación y se iniciara la simulación. Y un último botón que contiene el texto “Quit” que sirve para cerrar la aplicación.

3.3 Detalles implementación

Esta aplicación consta de un diseño simple de 3 clases. La clase ‘main’ utiliza la clase ‘MainWindow’ la cual contiene a la clase ‘GLWidget’ como se muestra en la figura 20.



**Figura 20.** UML sin variables ni métodos.

A continuación, se explica de forma simplificada el funcionamiento de las diferentes clases desarrolladas, comenzando por la más simple hasta llegar a la más compleja:

3.3.1-main

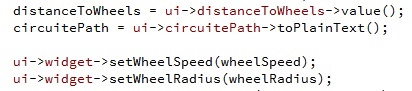
A pesar de ser esta la clase más sencilla y con menos código de todas, es la clase principal de la aplicación sobre la que se ejecutan el resto puesto que es la primera en ejecutarse al iniciar la aplicación.

Esta clase lo que realiza es la inicialización de freeglut y la creación de una instancia de la clase MainWindow. A continuación, utiliza el método show de esta última que mostrará el widget principal (“MainWindow”) y sus widgets hijos.

3.3.2-MainWindow

Esta clase se divide en tres archivos: ‘MainWindow.h’ que contiene las variables y las cabeceras de las funciones de esta clase (como se dijo anteriormente), ‘MainWindow.ui’ la cual contiene toda la información sobre la estructura de la interfaz, y ‘MainWindow.cpp’ que pasaremos a explicar su funcionamiento a continuación, que es el que contiene el código fuente de la clase.

El funcionamiento de esta clase es básico, funciona como conector entre la clase ‘GLWidget’ y la interfaz gráfica. Esta conexión se realiza a partir del método on\_insertParameters\_clicked la cual se llamará en caso de que el usuario pulse el botón con el texto “Insert Parameters”. Este método accederá a los datos escritos en los widgets de tipo input situados en la interfaz y se los enviará por medio de métodos de la clase ‘GLWidget’ a esta misma clase, como se muestra en la figura 21.



**Figura 21.** Recogida de datos y su paso a la clase ‘GLWidget’.

3.3.3-GLWidget

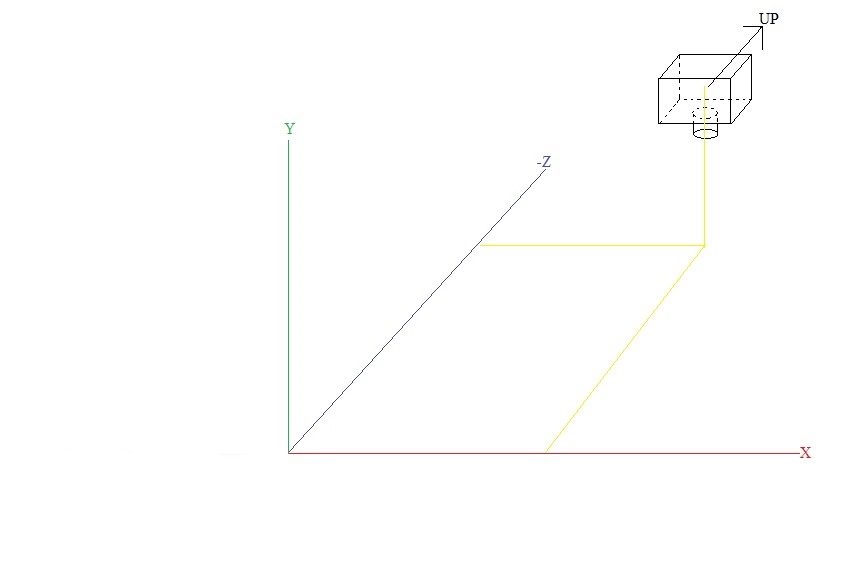
En esta clase se desarrollan todos los cálculos relacionados con la simulación del movimiento del robot y la visualización por medio de OpenGL.

La clase ‘GLWidget’ hereda de la clase ‘QOpenGLWidget’, la cual es propia de Qt y es un widget que se utiliza para renderizar gráficos en OpenGL. Esta te proporciona funciones básicas para la realización de proyectos con OpenGL. Las tres funciones más importantes que se heredan son:

* 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.

Esta clase es la que contiene todo sobre la aplicación (cálculos de posición del robot, animación, dibujado de robot y circuito, cálculo de límites en sensores, etc.), pero pasare a contar detalles que se realizan para mejorar el funcionamiento básico, contado anteriormente:

* Auto colocación de la cámara: Sabiendo que la cámara se sitúa mirando hacia y=0 y la parte de arriba de la cámara está situada hacia –z, como se muestra en la figura 22, se buscan los puntos del circuito más externos, es decir, el punto situado más a la izquierda (x menor), más a la derecha (x mayor), más arriba (z mayor) y más abajo (z menor).



**Figura 22.** Posición y orientación de la cámara.

Una vez encontrados se crea un vector de 3 posiciones en el cual guardaremos la posición donde más adelante situaremos la cámara. En este vector se guarda en la primera posición (x) el punto intermedio entre el punto situado más a la izquierda y el punto situado más a la derecha, a la hora de guardar la posición 3 (z) se realiza exactamente lo mismo, pero con la parte superior e inferior, es decir se calcula y se guarda el punto medio entre el punto más superior y el punto más inferior. Para poder saber la altura a la que se debe situar la cámara para que pueda ver todo el circuito, primero se comprueba cual contiene una mayor distancia si el eje x (es decir el punto más a la derecha menos el punto más a la izquierda) o el eje z (es decir el punto más arriba menos el punto más abajo). Una vez obtenida la mayor distancia se calcula mediante trigonometría la altura necesaria, como se puede ver en la figura 23.



**Figura 23.** 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.

* Posición cámara en ejecución: 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 podrá mover la cámara de forma libre con el ratón y se podrá hacer zoom con la rueda del mismo.
* Recorrido del robot: Se muestra en todo momento el recorrido que el robot ha llevado durante esa ejecución, con lo que se da un mejor feedback al usuario.

Capítulo 4 – Implementación

4.1 Casos de uso

No estoy seguro sobre que añadir aquí puesto que las acciones de la interfaz as explico en el apartado interfaz.

4.2 Resultados

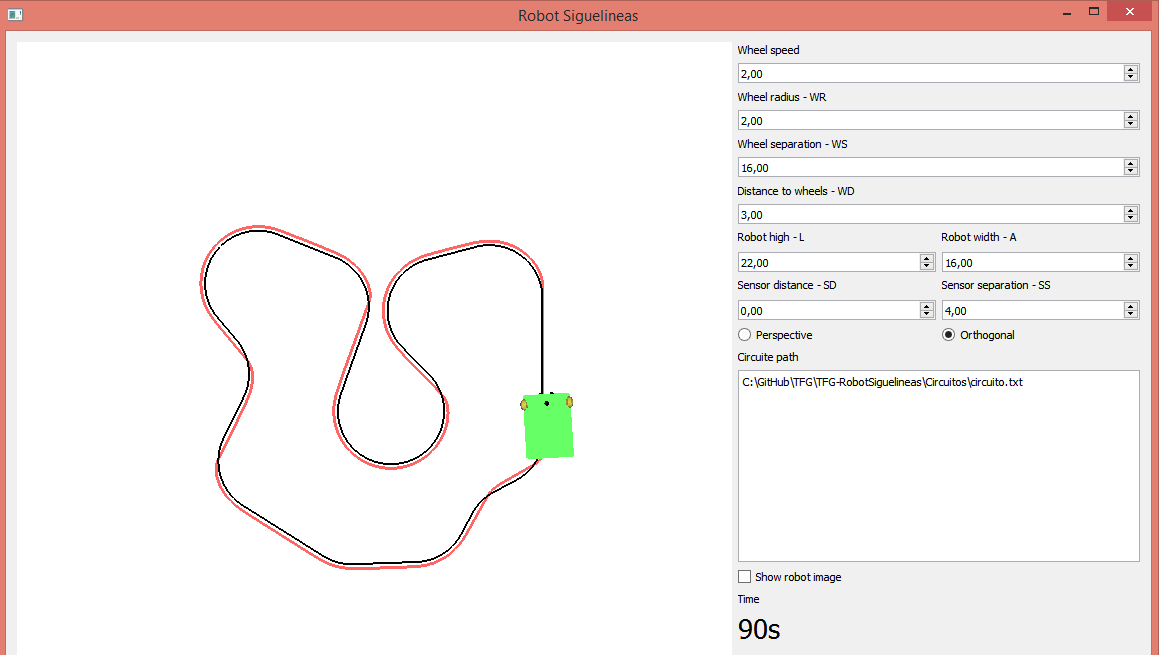
A continuación mostrare diferentes resultados de diferentes robots sobre un mismo circuito:

|  |  |
| --- | --- |
| **WR** | Radio de las ruedas |
| **WS** | Separación entre las ruedas |
| **WD** | Distancia desde el borde del robot a las ruedas |
| **L** | Largo del robot |
| **A** | Ancho del robot |
| **SD** | Distancia desde el borde del robot a los sensores |
| **SS** | Separación entre los sensores |

**Figura 24.** Leyenda de las abreviaturas usadas

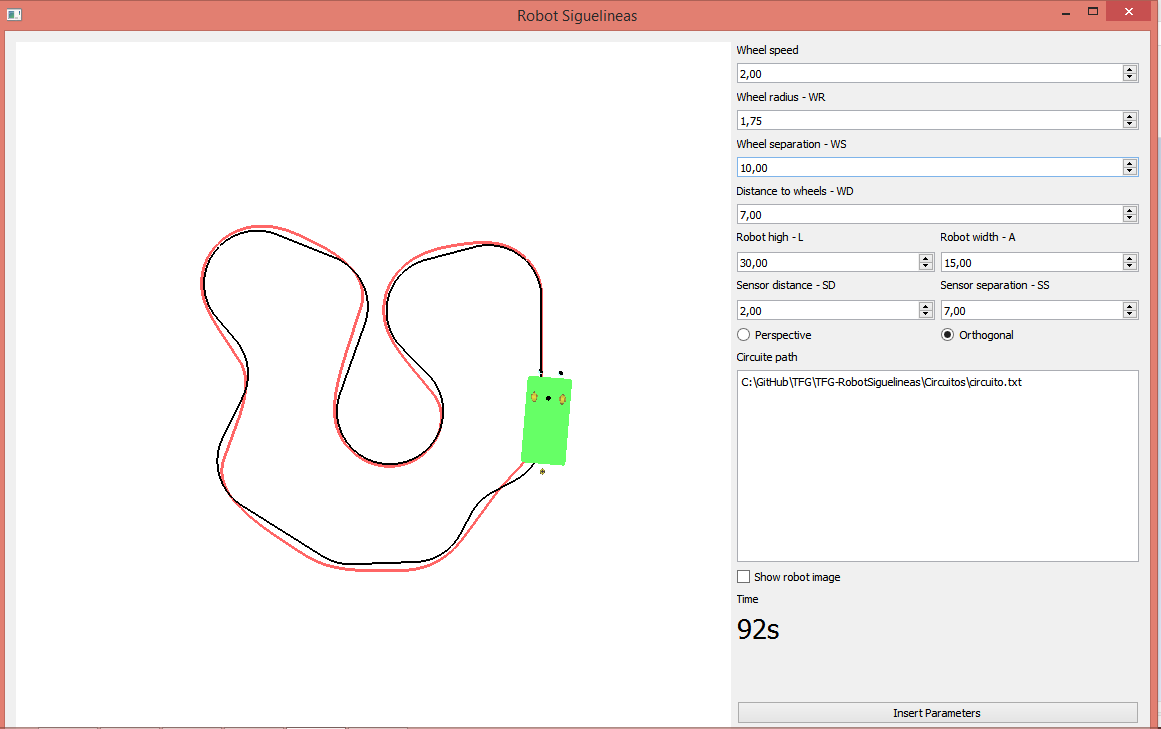
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Parámetros | Speed | WR | WS | WD | L | A | SD | SS |
| Valor | 2 | 2 | 16 | 3 | 22 | 16 | 0 | 4 |

Tiempo: 90s.



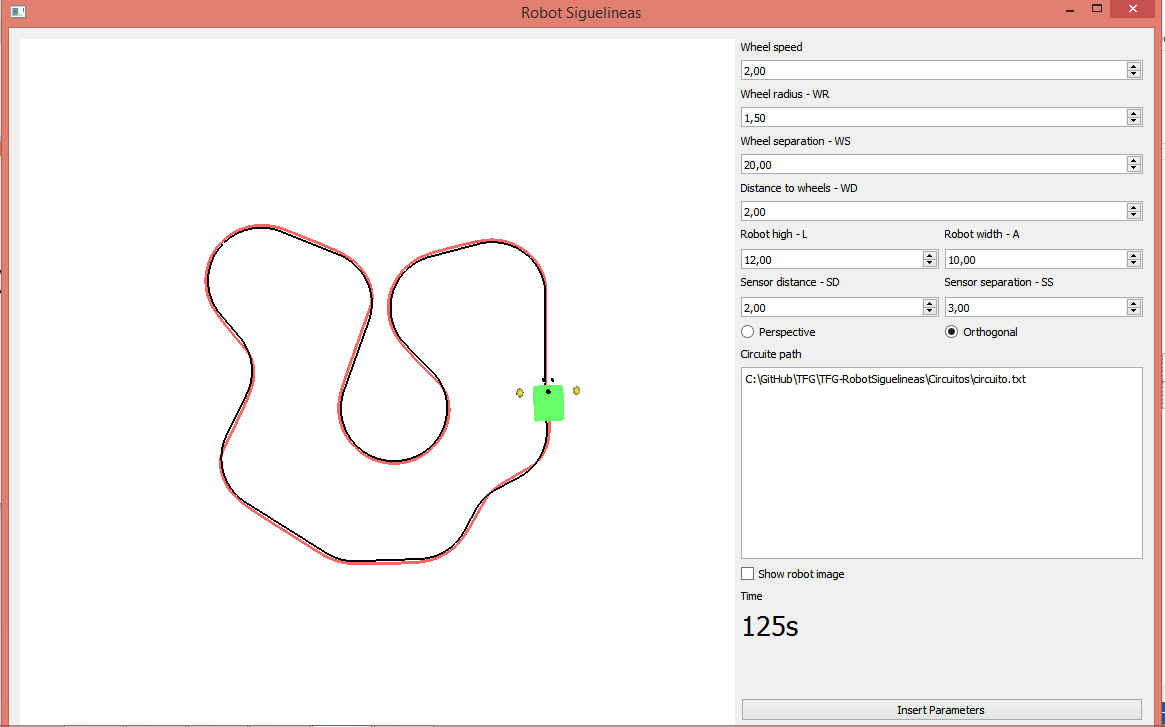
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Parámetros | Speed | WR | WS | WD | L | A | SD | SS |
| Valor | 2 | 1.75 | 10 | 7 | 30 | 15 | 2 | 7 |

Tiempo: 92s.



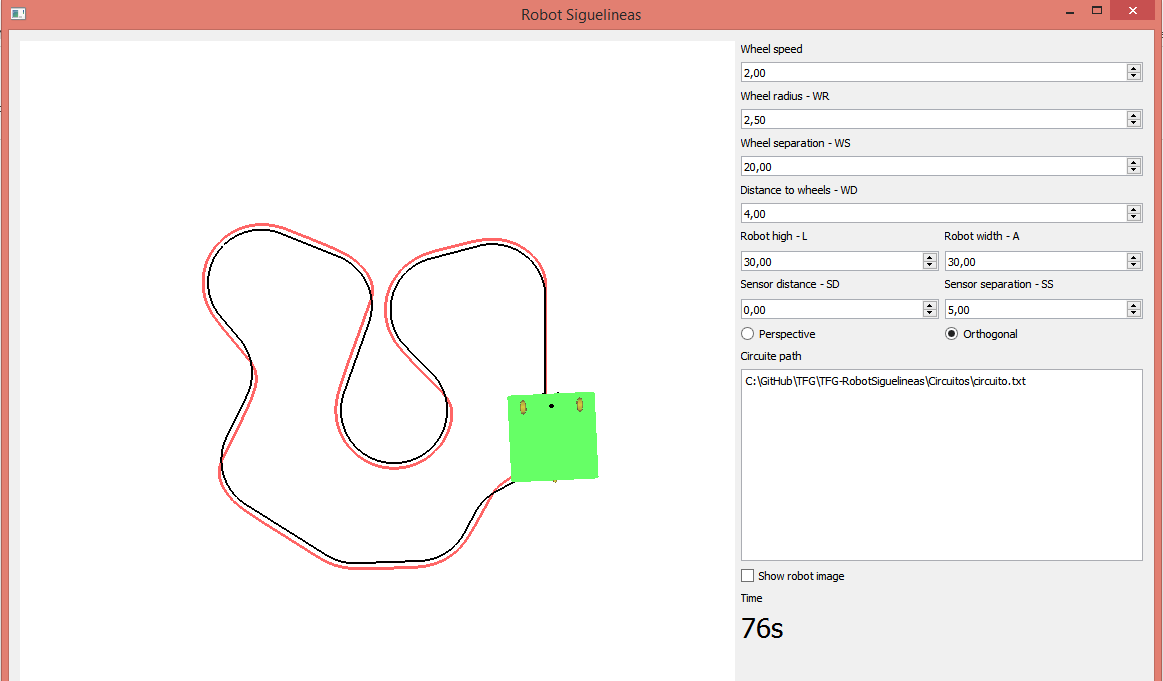
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Parámetros | Speed | WR | WS | WD | L | A | SD | SS |
| Valor | 2 | 1.5 | 20 | 2 | 12 | 10 | 2 | 3 |

Tiempo: 125s.



|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Parámetros | Speed | WR | WS | WD | L | A | SD | SS |
| Valor | 2 | 2.5 | 20 | 4 | 30 | 30 | 0 | 5 |

Tiempo: 76s.



Capítulo 5 – Conclusiones

Gracias a este proyecto he obtenido bastante conocimiento sobre C++, además de sobre una tecnología relativamente reciente 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 posibles mejoras a realizar sobre este proyecto podría ser el de incluir una forma de realizar carreras varios robots a la vez, o realizar estas carreras de manera online, una persona realizando de host y el resto poniendo sus robots y viendo 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 para realizar el robot en el menor tiempo posible.