

**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

[Índice 2](#_Toc510215722)

[Agradecimientos 3](#_Toc510215723)

[Resumen 4](#_Toc510215724)

[Capítulo 1 Introducción 5](#_Toc510215725)

[1. Motivación 5](#_Toc510215726)

[2. Objetivos 8](#_Toc510215727)

[3. Estado del arte 10](#_Toc510215728)

[4. Estructura de la memoria 11](#_Toc510215729)

[Capítulo 2 Robótica móvil 12](#_Toc510215730)

[1. Vehículos con ruedas 12](#_Toc510215731)

[2. Diferentes configuraciones 15](#_Toc510215732)

[3. Direccionamiento diferencial 18](#_Toc510215733)

[4. Navegación autónoma 20](#_Toc510215734)

[Capítulo 3 Entorno tecnológico 21](#_Toc510215735)

[1. Qt 21](#_Toc510215736)

[2. OpenGL 23](#_Toc510215737)

[3. GLM 26](#_Toc510215738)

[4. Metodología ágil 27](#_Toc510215739)

[Capítulo 4 Descripción de la aplicación 29](#_Toc510215740)

[1. Zona izquierda: Viewport 29](#_Toc510215741)

[2. Zona derecha: Campos de entrada de datos 32](#_Toc510215742)

[3. Casos de uso 36](#_Toc510215743)

[Conclusiones 39](#_Toc510215744)

[Bibliografía 40](#_Toc510215745)

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 visualizar el funcionamiento de un robot móvil con direccionamiento diferencial cuando trata de seguir un 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 mismo. A continuación se ha analizado el entorno tecnológico que ha sido utilizado, habiéndose elegido Qt y OpenGL para la interacción y simulación. Posteriormente se ha realizado la implementación de la simulación y la interacción con el usuario final. Finalmente, se han implementado opciones para la mejora de la aplicación.

Capítulo 1 Introducción

1. Motivación

Este trabajo nace motivado por un intento de mejorar una de las herramientas utilizadas 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, en grupos, debían realizar robots de distintos tipos: sigue líneas, velocistas, etc. Al final del curso el profesor realizaba un circuito y se usaban los robots que se habían construido para ver cuál era el más rápido, creando así una competición de robots entre los alumnos. El tiempo en el que el robot realizaba el circuito venía dado por las medidas que los alumnos utilizasen a la hora de construir el mismo, por ejemplo un robot con una separación de sensores mayor realiza menos giros pero de mayor amplitud sin embargo un robot con una separación menor realiza más giros pero de menor magnitud y esto junto con los demás parámetros influye 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.

Como se ha mencionado anteriormente, 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 este tiene varios inconvenientes:

a) Se debe tener instalado dicho software para su uso.

b) No se 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.



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

c) No ofrece todas las funcionalidades requeridas.

Por ello, se ha desarrollado dicho software creando una aplicación (integrando las tecnologías C++, Qt y OpenGL) cuyo único requisito para su uso es 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. 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 sera necesario disponer de ningún software adicional 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 se hallaba en que una vez se decidían las medidas y se compraban o creaban las piezas en cuestión era complicado cambiar estas para utilizar distintas. Por ejemplo si se decidía que el radio de las ruedas seria de X una vez se compraba la rueda, en caso de querer cambiarla se debía comprar una nueva, otro ejemplo podría ser el tamaño del robot, si se decidía porque el robot tuviera unas dimensiones de (Y, Z) pero más tarde se daba cuenta que sería mejor tener un robot con unas dimensiones mayores, estos debían volver a crear el soporte del robot puesto que este es el indica las dimensiones.

Esto se soluciona en la aplicación con una simple interfaz user-friendly en la que se podrá cambiar tantas veces como se quiera 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 se hacía complicado poder probar el robot creado, o diferentes opciones para el mismo.

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

* 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 se tiene una forma rápida de simular el movimiento del robot, se puede experimentar haciendo cambios en las medidas para poder observar qué cambios se realizan en el movimiento del robot y por lo tanto ir aprendiendo con la visualización del mismo.

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

* Facilidad de cambiar los parámetros del robot.
* Visualización de la simulación del robot de forma correcta.
* Integración de la interfaz y de la simulación en la aplicación.
* Interfaz user-friendly, fácil de utilizar para todos.
* Diseño simple de la simulación para poder visualizar mejor los movimientos de tu robot.

1. 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 de manera sencilla los parámetros del robot. Lo bueno que tenía la herramienta encontrada es que no solo servía para un tipo de robot sino que incluía en la aplicación varios tipos de robots.

1. Estructura de la memoria

Los capítulos que se abarcan a continuación son:

Robótica móvil: se comienza explicando qué es un vehículo con ruedas y diferentes configuraciones del mismo. A continuación se entra en detalle en el funcionamiento del direccionamiento diferencial. Finalmente se le aplica una navegación autónoma a un vehículo con ruedas con direccionamiento diferencial.

Entorno tecnológico: se informa mediante una descripción las diferentes tecnologías usadas: Qt, OpenGL y GLM. Además se realiza una explicación sobre su utilización en el proyecto. Además de estas también se explican otras tecnologías menos importantes para el proyecto.

Descripción de la aplicación: en este apartado se expone la aplicación al completo, comenzando por detalles sobre la simulación, pasando por la explicación de la interfaz y finalizando con casos de uso de la aplicación.

Capítulo 2 Robótica móvil

1. Vehículos con ruedas

Los vehículos con ruedas son 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 en la impulsión, además dependiendo del tipo de terreno, puede aparecer deslizamiento y vibraciones en el mismo. Como se ha dicho en la definición, estos vehículos son eficientes en terrenos duros y libres de obstáculos, por lo tanto en terrenos blandos son poco eficientes.

Otro problema que tienen este tipo de vehículos se haya 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.

A continuación se pasa a calcular las ecuaciones del modelo cinemático y discreto.

Según se ve en la figura 4, se supone 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 4.** 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:

(1)

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

(2)

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:

(3)

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

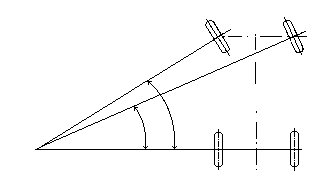
(4)

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.

1. Diferentes configuraciones

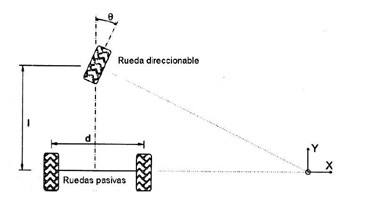
Los vehículos con ruedas emplean distintos tipos de locomoción que les da unas características y propiedades diferentes entre ellos en cuanto a eficiencia energética, dimensiones, maniobrabilidad y carga útil. A continuación se pasa a mencionar algunas de estas configuraciones:

* Ackerman: Es el habitual de los vehículos de cuatro ruedas. Las ruedas delanteras son las que se ocupan del giro, sin embargo la rueda interior gira un poco más que la exterior lo que hace que se elimine el deslizamiento. El centro de rotación del vehículo se haya en el corte de las prolongaciones de las ruedas delanteras y las ruedas traseras como se muestra en la figura 5.



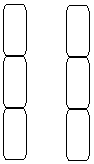
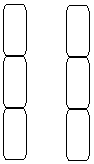
**Figura 5.** Configuración ackerman y centro de rotación.

* 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 cetro 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.



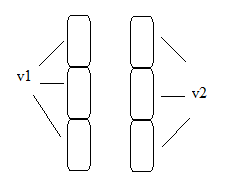
**Figura 6.** Configuración de triciclo y centro de rotación.

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



v1 > v2

v1 < v2



**Figura 7.** Configuración Skid Steer y movimiento según velocidades.

* 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.

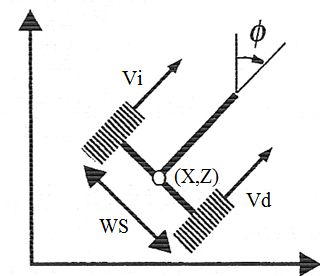


**Figura 8.** Pistas de desplazamiento.

* Síncronas: Se trata de una configuración en la que todas las ruedas actúan de forma simultánea y, por lo tanto, giran de forma síncrona.
* Direccionamiento diferencial: Esta configuración es la que utiliza el robot sigue líneas. A continuación se entra en detalle sobre su funcionamiento.

1. 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 9 se muestra una imagen de dicho esquema. Nótese que para especificar la configuración hay que indicar los valores de las tres variables (x, z, ), siendo las variables de control las velocidades de las ruedas laterales.



**Figura 9.** Locomoción mediante guiado diferencial.

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

(5)

Sustituyendo estas expresiones en las obtenidas a partir de la Figura 4, se obtienen las velocidades de las coordenadas del robot en el sistema {G} a partir de la velocidad de giro de cada rueda:

(6)

Finalmente, utilizando el modelo discreto, se obtiene:

(7)

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 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).

1. Navegación autónoma

El robot sigue líneas que se ha implementado realiza su movimiento de manera autónoma, se coloca el robot sobre un fondo blanco con una línea negra que representa el circuito y este deberá recorrer el circuito sin salirse del mismo. Esto se puede realizar gracias a dos sensores que son implantados en la parte posterior 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 realizara cambios en la velocidad de sus ruedas resultando en un movimiento recto, rotatorio hacia la izquierda o rotatorio hacia la derecha.

Los sensores que se implementan en este tipo de robots son sensores CNY70 los cuales 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. Cuando el sensor se haya sobre una línea negra la luz es absorbida y el fototransistor envía una señal (ya sea alta o baja dependiendo del montaje del sensor), sin embargo cuando se haya sobre fondo blanco la luz es reflejada y por lo tanto el fototransistor envía la señal contraria a la enviada al estar sobre negro.

La simulación de estos sensores se basa en el conocimiento de la posición de los sensores en todo momento y de los puntos que conforman el circuito, pudiendo así comprobar si cualquiera de los dos sensores está situado sobre el circuito.

Capítulo 3 Entorno tecnológico

1. Qt



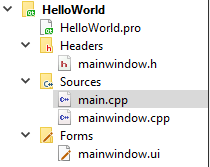
Qt es un framework de desarrollo de aplicaciones multiplataforma para ordenador, embebido y móvil que en gran parte se suele utilizar para programas que utilicen interfaz gráfica.

Internamente se utiliza C++ con alguna extensión para funciones como Signals y Slots, por lo tanto, se utiliza orientación a objetos. 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.

A continuación se muestra el “Hello World” en qt para poder observar un ejemplo sencillo. La estructura se puede ver en la figura 10. 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 10.** Estructura “Hello World”.

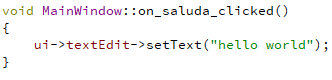
El fichero HelloWorld.pro es el que contiene la configuración del proyecto.

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 11) y se les nombra como se desee, en este caso el botón se llama “saluda” y el cuadro de texto “textEdit”.



**Figura 11.** Mainwindow.ui.

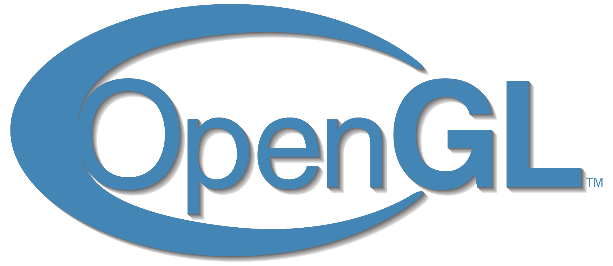
El fichero mainwindow.cpp contiene el método *on\_saluda\_clicked()* que se llamará cuando se pulse el botón “saluda”. Este contiene el código que accederá al cuadro de texto y escribirá en él “Hello World”, este método se puede observar en la figura 12.



**Figura 12.** 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.

1. 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.

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

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

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.

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.

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 estés 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 simulacion. 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.

1. GLM



A pesar del control que ofrece OpenGL sobre el cauce gráfico, en esta aplicación se ha necesitado llegar más en detalle sobre la matriz MVP, siendo exactos en la matriz Model (M), puesto que se necesita tener en todo momento las coordenadas de los sensores del robot para realizar los cálculos sobre las colisiones de los mismos con el circuito.

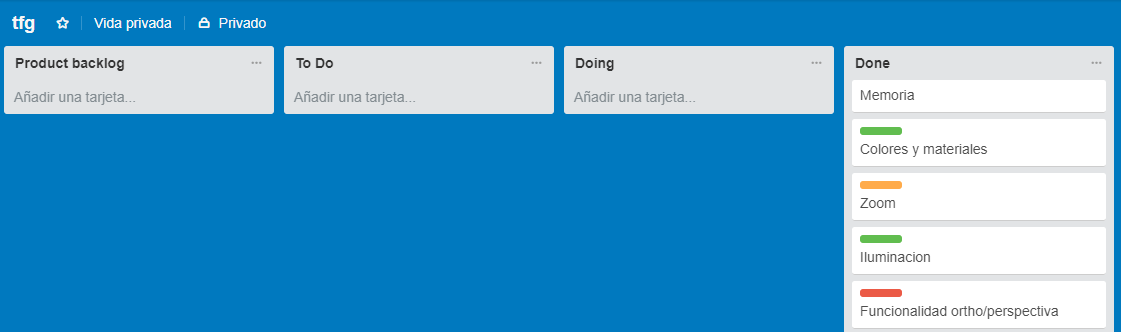
Se ha optado por la utilización de GLM como librería para el cálculo de la matriz MVP puesto que es la librería recomendada por OpenGL y por lo tanto es la más apta para este funcionamiento.

En este proyecto GLM se utiliza, como se ha dicho anteriormente, para el cálculo de la matriz MVP. Esta matriz se incluirá en la pila proporcionada por OpenGL y este realiza los cálculos pertinentes. Gracias a esto, y puesto que tenemos control total sobre la matriz Model, se puede realizar los cálculos para obtener las coordenadas de los sensores.

1. Metodología ágil

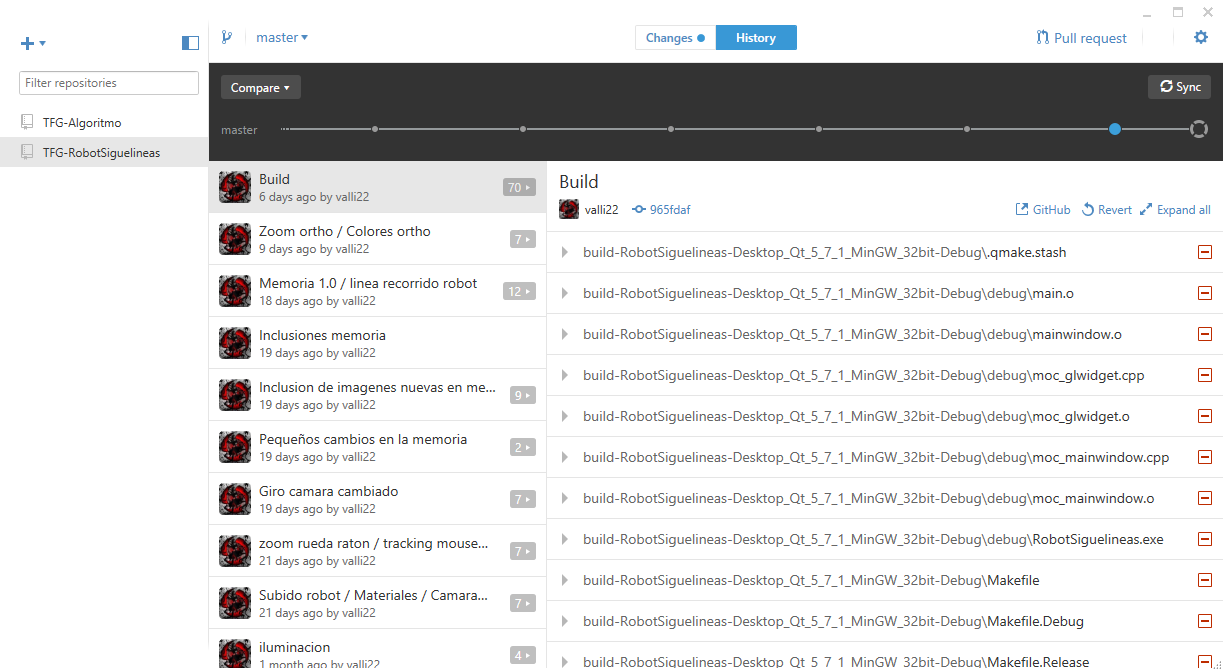
Durante el desarrollo de la aplicación se utilizado herramientas utilizadas habitualmente en proyectos en los que se aplica metodología ágil. Estas herramientas han sido:

* Trello como tablero de tareas, este se divide en las columnas habituales (Product backlog, To Do, Doing, Done). A su vez cada tarea tiene asignada una dificultad representada mediante colores. Las tareas no tienen asignadas personas puesto que hay una sola persona encargada de este tablero. A pesar de no ser relevante para la organización de un equipo y la división de tareas, puesto que solo se trata de una persona, ha resultado muy útil para no perder la visión de proyecto. Todo esto se puede ver en la figura 13.



**Figura 13:** Trello del proyecto Simulacion de un robot siguelineas.

* Git como repositorio y control de versiones (utilizado concretamente Github). Sobre este repositorio se ha ido subiendo los diferentes incrementos de funcionalidad de la aplicación de forma periódica y que gracias a él se ha podido realizar un control de versiones. Se puede observar el repositorio desde la aplicación de windows en la figura 14.



**Figura 14:** Repositorio en Github desde la aplicación

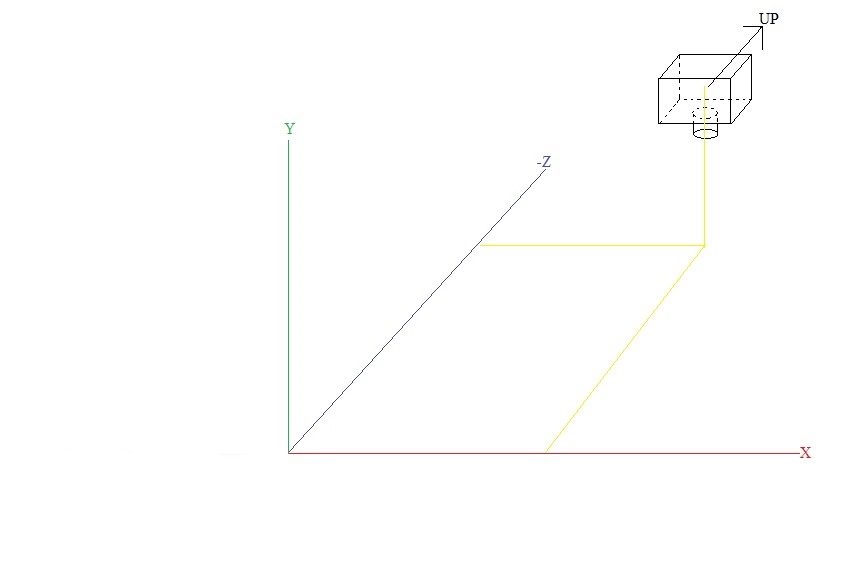
de escritorio de Windows.

Capítulo 4 Descripción de la aplicación

1. Zona izquierda: Viewport

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

Al iniciarse se dibuja el circuito y se hace un cálculo para posicionar automáticamente la cámara sobre el mismo. 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 15, 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 15.** 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 16.



**Figura 16.** Trigonometría utilizada para el cálculo de la altura de la cámara.

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

Una vez tenemos la posición de la cámara, dependiendo de que opción haya elegido el usuario se hacen los cálculos para la proyección en perspectiva u ortográfica, esto implica que se cambia también la posición de la cámara obtenida anteriormente. En caso de que se opte por una visualización en ortográfica la cámara se situara justo encima del circuito, sin embargo, en caso de que se opte por una visualización en perspectiva se colocara la cámara a 45º respecto a la posición inicial (justo encima del circuito) para que sea posible observar los modelos 3D y la perspectiva de la imagen. Como añadido, se ha implementado la funcionalidad de zoom en ambas proyecciones el cual se puede utilizar con la rueda del ratón.

Una vez realizado esto se pasa a la fase en bucle de los cálculos de la posición y el dibujado del robot. Dependiendo de los parámetros introducidos en la parte derecha de la aplicación el robot dibujado será acorde a las mismas. Los cálculos necesarios son, el cálculo de la posición del robot, el cálculo de las coordenadas de los sensores y el algoritmo que determina si los sensores del robot están tocando alguna parte del circuito.

Además 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.

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

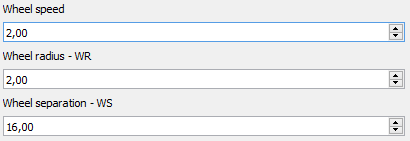
1. Zona derecha: Campos de entrada de datos

La zona derecha de la aplicación se centra en la recogida de datos del usuario que a continuación se pasa a explicar y el porqué de las decisiones tomadas.

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, la comentada en el punto anterior.

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 17, puesto que la unión de los 3 parámetros de las ruedas en un mismo nivel no resultaba fácil de visualizar.



**Figura 17.** 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 18.

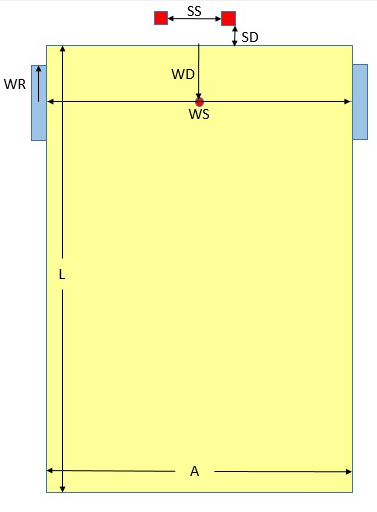
uiRobot

**Figura 18.** 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 está 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 19.

****

**Figura 19.** 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 20, 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 20.** 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.

1. Casos de uso

A continuación se muestran 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 21.** Leyenda de las abreviaturas usadas

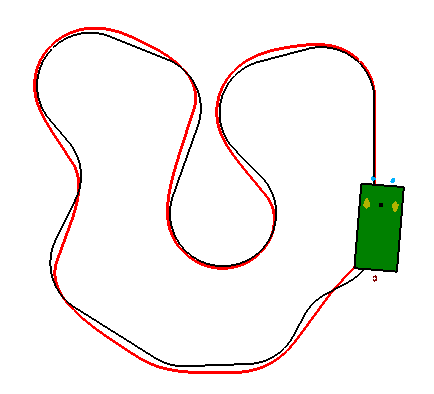
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Parámetros | Speed | WR | WS | WD | L | A | SD | SS |
| Valor | 2 | 2 | 16 | 3 | 22 | 16 | 0 | 4 |

Tiempo: 90s.



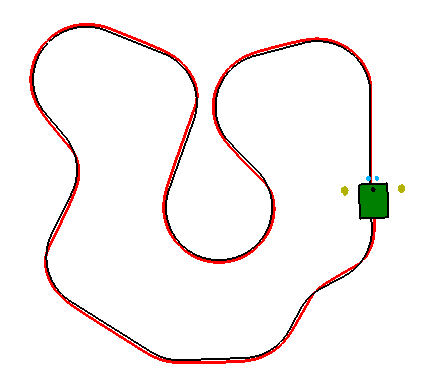
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Parámetros | Speed | WR | WS | WD | L | A | SD | SS |
| Valor | 2 | 1.75 | 10 | 7 | 30 | 15 | 2 | 7 |

Tiempo: 92s.



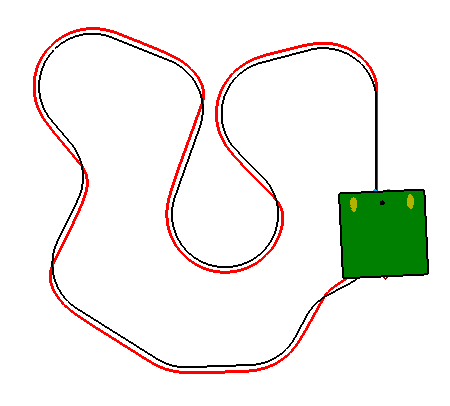
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Parámetros | Speed | WR | WS | WD | L | A | SD | SS |
| Valor | 2 | 1.5 | 20 | 2 | 12 | 10 | 2 | 3 |

Tiempo: 125s.



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

Tiempo: 76s.



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.

Bibliografía