



**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

Director/Tutor: Alberto Herrán González

Índice

ÍNDICE.....	2
AGRADECIMIENTOS	3
RESUMEN.....	4
CAPÍTULO 1 INTRODUCCIÓN.....	5
1. MOTIVACIÓN	5
2. OBJETIVOS.....	7
3. ESTADO DEL ARTE.....	8
4. ESTRUCTURA DE LA MEMORIA	9
CAPÍTULO 2 ROBÓTICA MÓVIL	10
1. ROBÓTICA MÓVIL.....	10
2. VEHÍCULOS CON RUEDAS	10
3. MODELO FÍSICO DIRECCIONAMIENTO DIFERENCIAL.....	13
4. NAVEGACIÓN AUTÓNOMA	17
CAPÍTULO 3 ENTORNO TECNOLÓGICO	19
1. OPENGL	19
2. ETAPA GEOMÉTRICA.....	21
3. QT	27
4. METODOLOGÍA ÁGIL.....	29
CAPÍTULO 4 DESCRIPCIÓN DE LA APLICACIÓN.....	31
1. ZONA IZQUIERDA: VIEWPORT	31
2. ZONA DERECHA: CAMPOS DE ENTRADA DE DATOS	33
3. CASOS DE USO.....	37
CONCLUSIONES.....	40
BIBLIOGRAFÍA.....	41

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, habiéndose elegido Qt y OpenGL. Posteriormente se ha realizado la implementación de la simulación del comportamiento del robot y su visualización sobre el circuito, además de la interacción con el usuario final mediante inputs en la aplicación. Finalmente, se han implementado opciones adicionales para la mejora de la experiencia del usuario y de la aplicación.

De La aplicación, habiéndose ...
Dicha aplicación, tiene implementadas tanto la
GOMO

Capítulo 1 Introducción

1. Motivación

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

En esta asignatura los alumnos, en grupos, ~~debían construir~~ robots de distintos tipos: sigue líneas, velocistas, etc. Al final del curso el profesor realizaba ~~una competición~~ en la que se ~~usaban~~ los robots que se ~~habían construido~~ para ver cuál ~~era~~ el más rápido. En esta competición se cronometraba el tiempo que tardaba cada robot en realizar el circuito, este tiempo dependía de ciertos parámetros que los alumnos elegían a la hora de crear el robot.
Construyen
reutilizar
realizando diferentes tareas. En concreto, p.ej. al robot sigue-líneas, ...

El objetivo de este proyecto es desarrollar una aplicación que permita simular y visualizar la dinámica de ~~este robot móvil con direcciónamiento diferencial. La aplicación servirá de banco de pruebas con el que analizar el comportamiento del robot sigue líneas antes de su construcción real.~~ *Ldos robot lgo diferentes*
sistema
mismo
wando?
indicadores (garrales, circuitos, etc)

Como se ha mencionado antes, actualmente se dispone de un desarrollo previo en MATLAB-Simulink, *E*. En la figura 1 se puede observar el esquema de dicho desarrollo, en el que se puede ver los diferentes componentes que se utilizan en la aplicación. La figura 2 muestra el programa en ejecución.

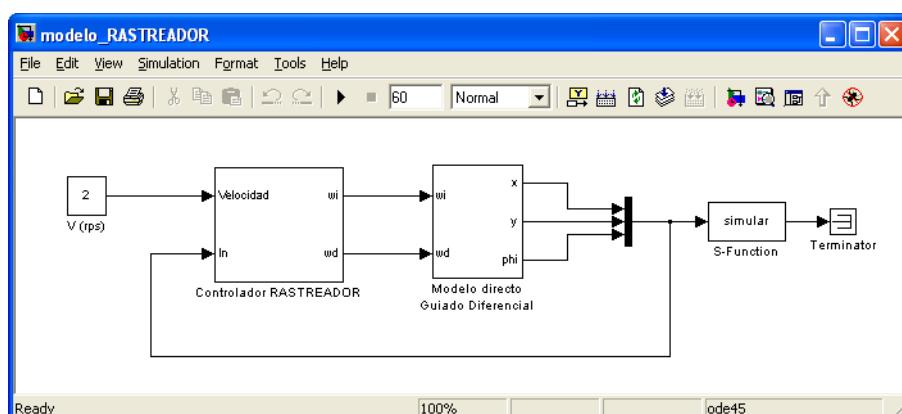


Figura 1. Modelo del desarrollo hecho en MATLAB-Simulink.

Comentar algunos parámetros que aparecen en la fig. 1. (Bloques)

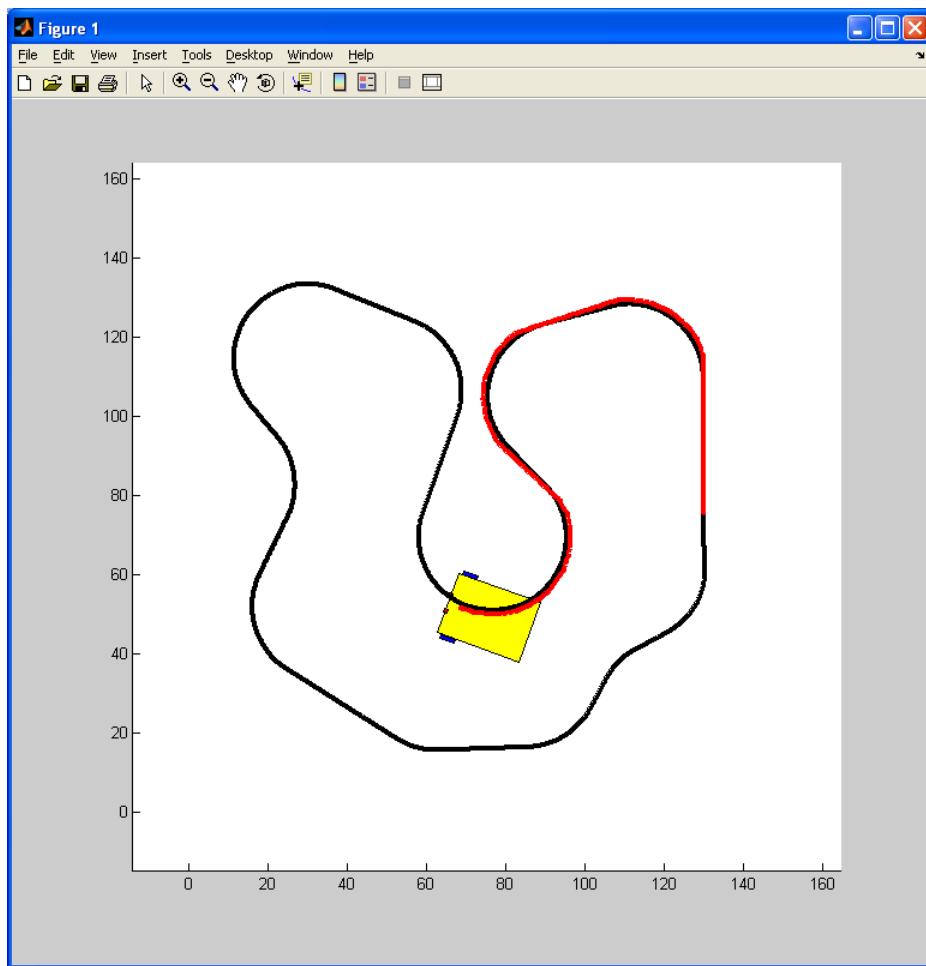


Figura 2. Vista de la simulación desarrollada en MATLAB-Simulink.

Sin embargo, este tiene varios inconvenientes:

- a) Se debe tener instalado MATLAB-Simulink para su uso.
ya que
- b) No se permite cambiar los parámetros del robot de una forma sencilla, estos
~~parámetros se deben escribir~~ en un vector cuya estructura ~~debes~~ ^{se} conocer,
~~Se encuentran almacenados~~ como se puede ver en la figura 3. (*contar qué es cada número o los primos por ejemplo....*)
- c) No ofrece todas las funcionalidades requeridas (estas se especifican más adelante en el punto 2 Objetivos.)

```
>> robot
robot =
22    16     3     2    -2     3     2     3     1
```

Figura 3. Vector en el cual se deben poner los parámetros del robot.

↙ ↘

Por ello, se ha desarrollado 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 para hacerlo funcionar.

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

2. Objetivos

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

- **Eliminar la necesidad de tener software instalado:** con esta aplicación no es necesario disponer de ningún software adicional instalado, más allá de las librerías necesarias para la aplicación, por lo que es más fácil su descarga y utilización inicial.

No me guste este párraf.

- **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 es complicado cambiar estas para utilizar distintas. Por ejemplo si se decide que el radio de las ruedas es de Xcm una vez se compra la rueda, en caso de querer cambiarla se debe comprar una nueva, otro ejemplo podría ser el tamaño del robot, si se decide porque el robot tuviera unas dimensiones de Y-Zcm pero más tarde se daba cuenta que sería mejor tener un robot con unas dimensiones mayores, estos deben volver a crear el soporte del robot puesto que este es el que indica las dimensiones.

Le dejo

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.

fácilmente los parámetros del

- **Realización de pruebas con facilidad:** debido a lo costoso de realizar un robot y la necesidad de crear un circuito por el cual el robot tenga que correr se hace complicado poder probar el robot creado, o diferentes opciones para el mismo.

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

- Es el mismo
diseño
que el Z.*
- **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 al utilizar 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 de este.

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

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

3. Estado del arte

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

- Aplicaciones creadas por una persona que muestran cómo realizarlas o dejan la aplicación en un repositorio público (pocas de estas realizan esto último). Estas aplicaciones suelen tener dos problemas: la dificultad para su descarga y utilización, que no se hallan preparadas para su uso puesto que están en las IDE correspondientes y la interfaz no es fácil de usar, en caso de que haya interfaz y no se tenga que cambiar los parámetros por código.
- Aplicaciones creadas por empresas. En este caso muchas de estas aplicaciones tienen una carencia en la UI, no se puede cambiar de manera sencilla los parámetros del robot. Lo bueno que tenía la herramienta encontrada es que no solo servía para un tipo de robot, sino que incluía en la aplicación varios tipos de robots. Ejemplos de este tipo son: Hemero, una

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

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 y en ~~específico~~ un vehículo con ruedas y diferentes configuraciones ~~del mismo~~. A continuación, se entra en detalle en el ~~funcionamiento del~~ direcciónamiento diferencial y se realiza el cálculo ~~de~~ las ecuaciones necesarias para simular un robot de este tipo. Finalmente, se le aplica una navegación autónoma a un vehículo con ~~incluyen los elementos necesarios para implementar~~ ~~de este tipo~~.

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

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

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

Capítulo 2 Robótica móvil

1. Robótica móvil

Los robots móviles son robots que, gracias a los progresos entre la inteligencia artificial y los robots físicos, son capaces de moverse a través de cualquier entorno físico. Estos normalmente son controlados por software y usan sensores y otros equipos para identificar la zona de su alrededor

Los robots móviles se pueden diferenciar en dos tipos, autónomos y no autónomos. Los robots móviles autónomos pueden explorar el entorno sin ninguna directriz externa a él, sin embargo, los robots móviles no autónomos necesitan algún tipo de sistema de guía para moverse.

2. Vehículos con ruedas

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

Su limitación más importante es el deslizamiento 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.
~~Este párrafo no aporta nada.~~

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

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

- **Ackerman:** Es el habitual de los vehículos de cuatro ruedas. Las ruedas delanteras son las que se ocupan del giro, sin embargo, la rueda interior gira un poco más que la exterior lo que hace que se elimine el deslizamiento. El

centro de rotación del vehículo se haya en el corte de las prolongaciones de las ruedas delanteras y las ruedas traseras como se muestra en la figura 4.

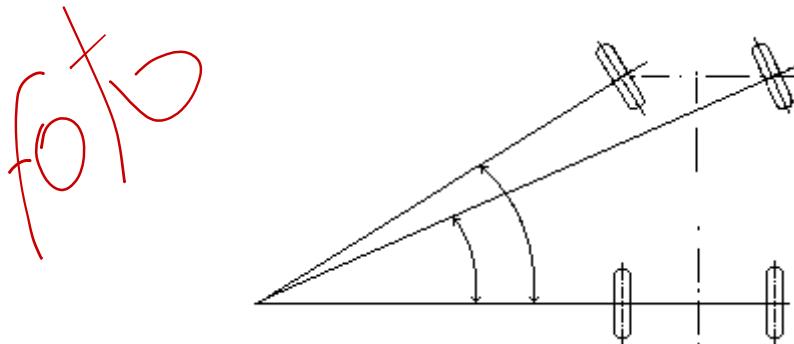


Figura 4. Configuración ackerman y centro de rotación.

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

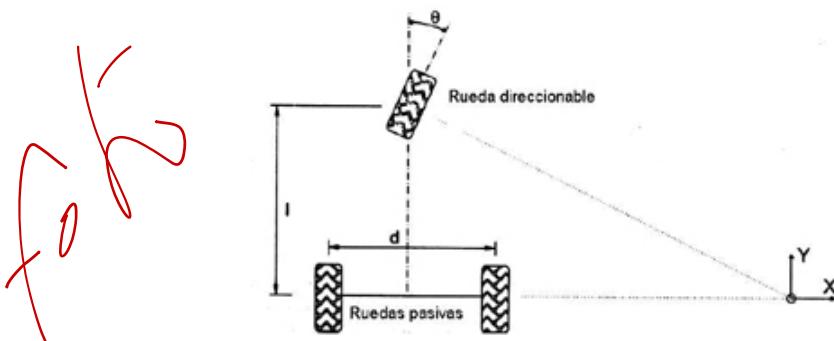


Figura 5. Configuración de triciclo y centro de rotación.

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

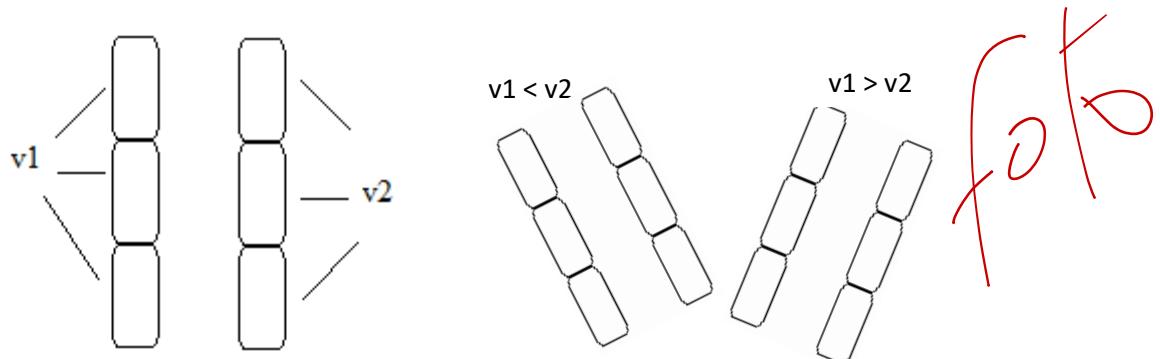


Figura 6. Configuración Skid Steer y movimiento según velocidades.

- **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 7. Pistas de desplazamiento.

- (Handwritten notes: 'Síncronas' with a circled dot, 'Diferencial' with a circled dot, and 'fot' written vertically below the text)*
- **Síncronas:** Se trata de una configuración en la que todas las ruedas actúan de forma simultánea y, por lo tanto, giran de forma síncrona.
 - **Direccionamiento diferencial:** Esta configuración es la que utiliza el robot sigue líneas. En este sistema las ruedas se sitúan de forma paralela y no realizan giros. El direccionamiento diferencial viene dado por la diferencia de velocidades de las ruedas laterales. La tracción se consigue con esas mismas ruedas. Adicionalmente, existen una o más ruedas para el soporte. En la figura 10 se muestra una imagen de dicho esquema.

3. Modelo físico direccionamiento diferencial

Para poder realizar la simulación del robot ~~y pintarlo~~ se debe conocer primeramente el estado del sistema. El ~~sistema~~ ^{gráfica} ~~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; por lo tanto, se debe definir un modelo de cambios de estado, es decir, una ecuación que a partir del estado actual (t) y unas entradas se pase al estado en el siguiente instante ($t+1$). En la ecuación X se puede observar la ecuación que define el sistema, siendo \bar{s} el estado y \bar{r} las entradas. Asimismo, en esta misma ecuación se muestra la ecuación del estado de un robot móvil, definida por su posición $(x(t), z(t))$ y su orientación $\varphi(t)$ en ese instante:

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

→ Variables que definen el estado del robot en un instante (t) ↴

Si embargo, la ecuación anterior es genérica para robots móviles, para obtener la ecuación de cambio de estado para un robot con direccionamiento diferencial se deben definir las entradas como las velocidades ~~de cada rueda~~ ^{izq y dcha.} como se explica en el punto anterior. Por lo tanto, la ecuación de las entradas resultante para un robot con direccionamiento diferencial, siendo w_i la velocidad de la rueda izquierda y w_d la velocidad de la rueda derecha, es la siguiente:

$$\bar{r} = (w_i(t), w_d(t)) \quad (X)$$

En la figura 8 se pueden observar los diferentes datos de entrada y del sistema puestos sobre un robot con direccionamiento diferencial.

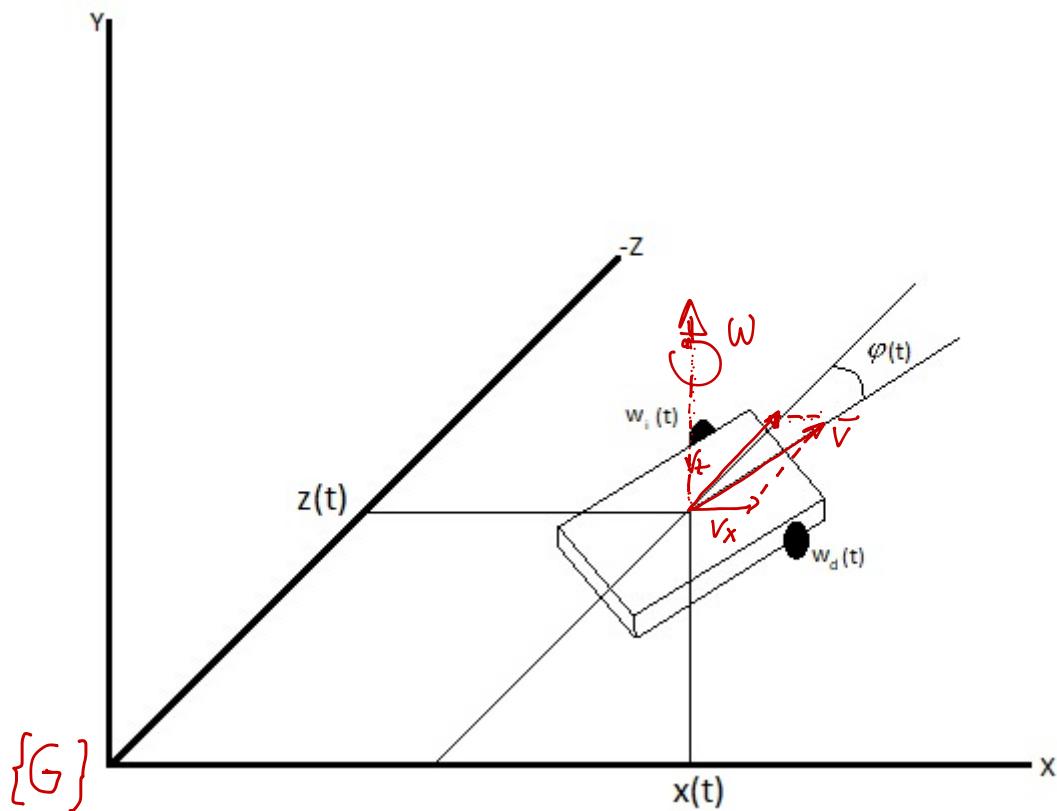


Figura 8. Sistema 3D de un robot con direccionamiento diferencial.

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

Según se ve en la figura 9, se supone un sistema de referencia $\{G\}$ y un sistema $\{L\}$ con origen en el punto de guiado del vehículo y eje \hat{Y}_L en la dirección del eje longitudinal del vehículo.

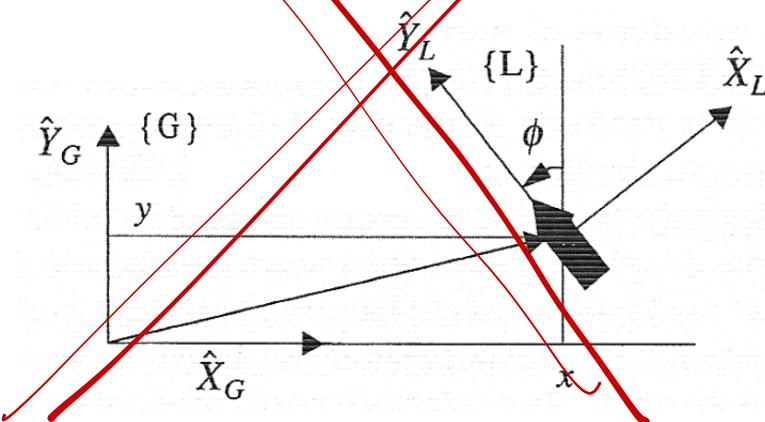


Figura 9. Modelo cinemático.

Té vale en la figura 8

Aprox. de P. C. f. 5.8

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

$$\begin{aligned} v_x &= v * \sin(\varphi) \\ v_z &= v * \cos(\varphi) \quad (1) \\ v_\varphi &= w \end{aligned}$$

Se hace así:

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

$$v_x = \frac{dx}{dt} = \frac{x(t+1) - x(t)}{\Delta t} \quad (2)$$

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:

$$x(t+1) = x(t) + v_x * \Delta t \quad (3)$$

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

$$\begin{aligned} x(t+1) &= x(t) + v_x * \Delta t \\ y(t+1) &= y(t) + v_y * \Delta t \\ \varphi(t+1) &= \varphi(t) + v_\varphi * \Delta t \end{aligned} \quad (4)$$

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

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

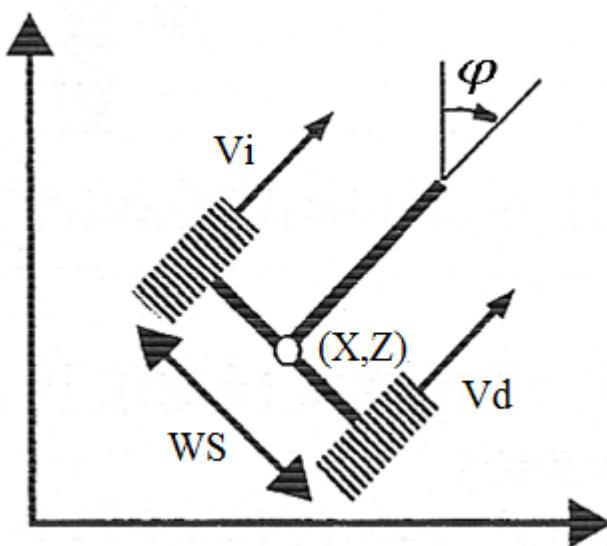


Figura 10. Locomoción mediante guiado diferencial.

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

$$\begin{aligned} v &= \frac{v_d + v_i}{2} = \frac{(v_d + v_i) * WR}{2} \\ w &= \frac{v_d - v_i}{WS} = \frac{(w_d - w_i) * WR}{WS} \end{aligned} \quad (5)$$

Sustituyendo estas expresiones en las obtenidas a partir de la Figura 9, se obtienen las velocidades de las coordenadas del robot en el sistema $\{G\}$ a partir de la velocidad de giro de cada rueda:

$$\begin{aligned} v_x &= \frac{-(w_d + w_i) * WR}{2} * \sin(\varphi) = -(w_d + w_i) * \frac{WR * \sin(\varphi)}{2} \\ v_z &= \frac{(w_d + w_i) * WR}{2} * \cos(\varphi) = (w_d + w_i) * \frac{WR * \cos(\varphi)}{2} \\ v_\varphi &= \frac{(w_d - w_i) * WR}{WS} = (w_d - w_i) * \frac{WR}{WS} \end{aligned} \quad (6)$$

Finalmente, utilizando el modelo discreto, se obtiene:

$$\begin{aligned}x(t+1) &= x(t) - (w_d + w_i) * \frac{WR * \sin(\varphi(t))}{2} * \Delta t \\z(t+1) &= z(t) + (w_d + w_i) * \frac{WR * \cos(\varphi(t))}{2} * \Delta t\end{aligned}\quad (7)$$

$$\varphi(t+1) = \varphi(t) + (w_d - w_i) * \frac{WR}{WS} * \Delta t$$

Comentar el resultado final, hacen un bucle de la ecuación y que es para variar.

4. Navegación autónoma

El robot sigue líneas que se ha implementado realiza su movimiento de manera autónoma, se coloca el robot sobre un fondo blanco con una línea negra que representa el circuito, como se muestra en la figura 11, y este deberá recorrer el circuito sin salirse del mismo. Esto se puede realizar gracias a dos sensores que son implantados en la parte delantera del robot los cuales son responsables de la detección de la línea del circuito. En función de lo que estos sensores recojan (están sobre el circuito o no) el robot realiza cambios en la velocidad de sus ruedas resultando en un movimiento recto, rotatorio hacia la izquierda o rotatorio hacia la derecha.

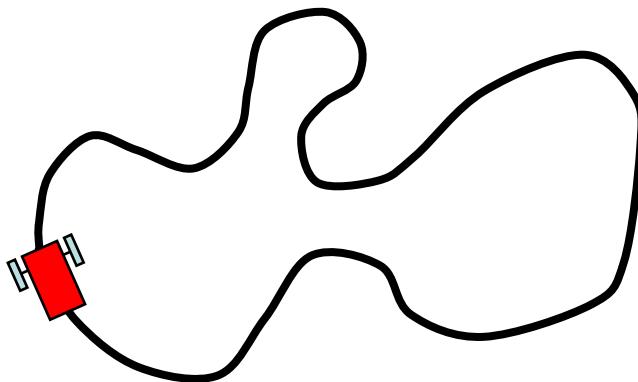


Figura 11. Colocación del robot en un circuito negro sobre fondo blanco.

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

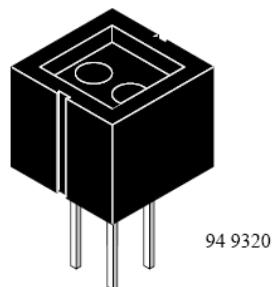


Figura 12. Sensor CNY70.

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

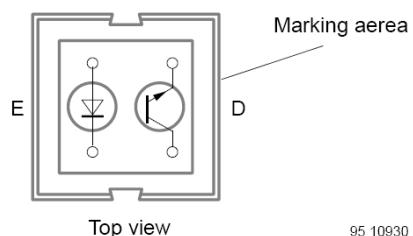


Figura 13. Estructura simplificada del sensor CNY70.

Para implementar dicho comportamiento en el simulador se hace lo siguiente.

Algoritmo

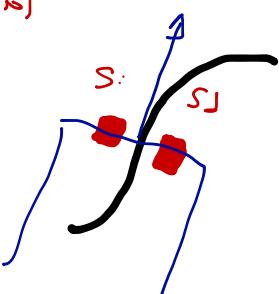
1. if ($s_1 = \text{circuito}$)
2. $w_i = 0;$
3. if ($s_2 = \text{circuito}$)
4. $w_d = 0;$
5. f.p. esto
6. $w_i = w$
7. $w_d = 0$

Inicialmente

$$w_i = w_d = w$$

Línea recta.

$s_1 \equiv$ sensor izq.
 $s_2 \equiv$ sensor der.

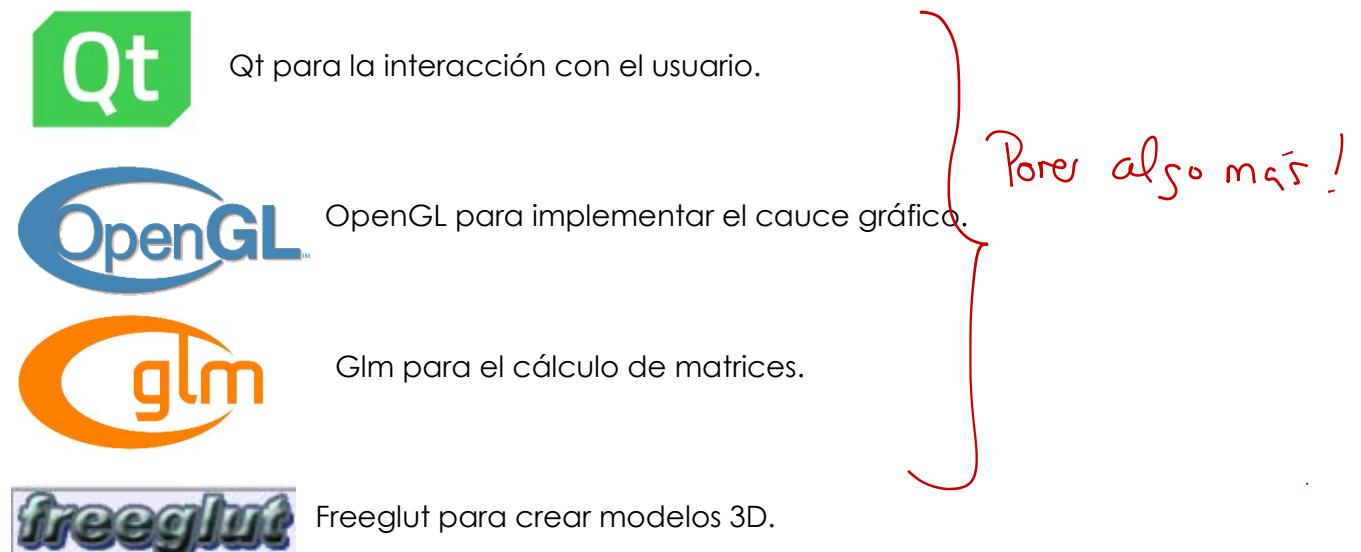


← Poner algo de este tipo por que quede claro cómo se hace implementar la integración autónoma.

Comentar otras opciones, por ejemplo los $w_i = 0$ del algoritmo podrían ser $w_i = -w$ (cambiar sentido), o $w_i = w - \Delta w$ (reducir w_i), etc....

Capítulo 3 Entorno tecnológico

Para el desarrollo del trabajo se han utilizado diferentes tecnologías:



1. OpenGL

Para la visualización de gráficos 3D

Para poder implementar el cauce gráfico, que se explica al final de este punto y el siguiente, existen diferentes APIs que se pueden utilizar como pueden ser Direct3D (para Windows), Mantle (para tarjetas AMD), Vulkan (basado en Mantle y multiplataforma) o OpenGL. Se decidió utilizar OpenGL puesto que es la librería más conocida, pública y multiplataforma, además es fácil integrarla en la mayoría de los proyectos.

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

- Construir formas geométricas a partir de las primitivas que este te proporciona.
- Ubicar los objetos en la escena.
- Ubicar el punto desde el que se visualiza la escena.

El lectura de por si
saber nada del cauce gráfico en este
punto.

- Poner color o texturas.
- Crear luces.
- Realizar la rasterización.

→ Se realiza (No las utilizó la persona antes).

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 en 2017. Se añaden una gran cantidad de funcionalidades.

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

El cauce 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 14 podemos observar las diferentes etapas y como se conectan como hemos citado anteriormente.

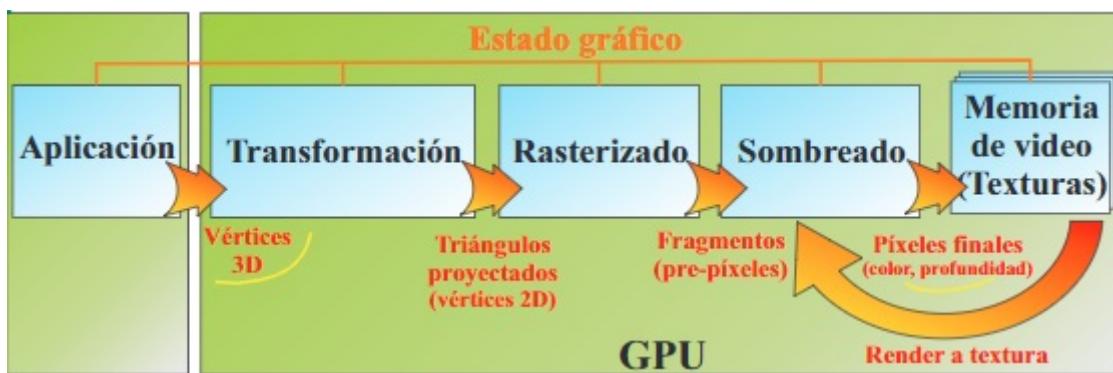


Figura 14. Etapas del cauce gráfico.

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

2. Etapa Geométrica

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. ~~Si se quiere utilizar en 3D,~~ *Para*, las matrices que se utilizan para los cálculos son matrices 4x4. Para obtener las coordenadas donde se encuentra cada punto del objeto dentro de la escena es necesario calcular la matriz model-view-projection (MVP), como su nombre indica, esta matriz viene dada por la multiplicación de tres matrices distintas las cuales realizan cálculos para la obtención de diferentes funcionalidades, es decir, la ecuación que se implementa en esta etapa es

$$v' = M_{projection} * M_{view} * M_{model} * v \quad (e f)$$

vertice proyectado visto desde la cámara

Donde v' es el vector de posición resultante del objeto en cuestión y v el vector de coordenadas en el instante anterior. *vertice del modelo del objeto considerado.*

Para la creación de los modelos 3D se utiliza la librería freeglut. Estos modelos son los que contienen las coordenadas, es decir, cada vértice de los modelos es un vector de coordenadas v . *de cada uno de los vértices del mismo.*

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

- **(P) Projection:** Esta matriz define como se realiza la proyección, ~~poniendo en esta matriz si la proyección es 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 se acerca más a la vista de una persona. En las ~~figura~~ 15 y 16 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.

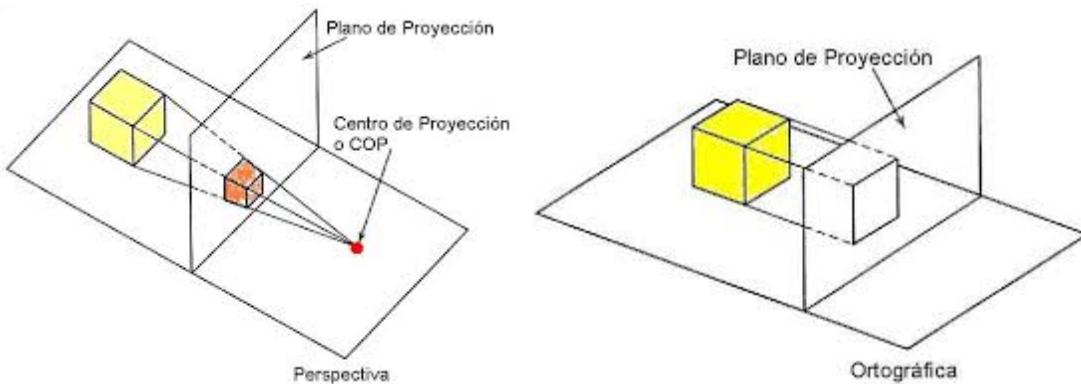


Figura 15. Vista en perspectiva.

Figura 16. 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.

Las funciones de GLM que se utilizan en el cálculo de esta matriz son `perspective(fov, aspectRatio, near, far)` para el cálculo de la matriz en perspectiva y `ortho(left, right, bottom, top, near, far)` para el cálculo de la matriz en ortográfica.

Explícate estos parámetros (ponlos en líneas separadas no entre el texto)

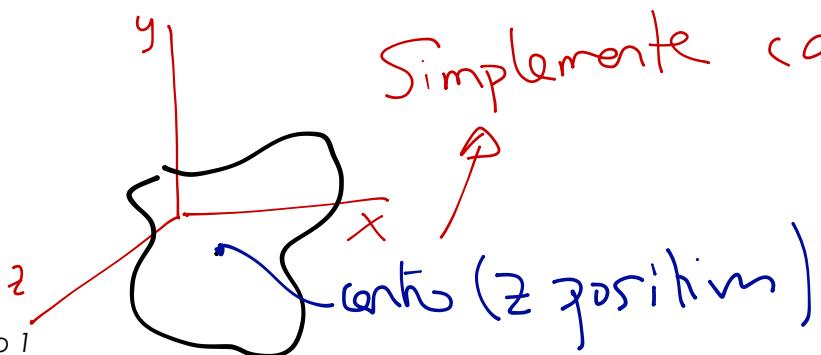
- (V) **View:** Esta matriz hace las funciones de cámara, necesitando para poder usar esta matriz la posición, el punto hacia el que mira y la orientación de la cámara. Además, esta matriz no solo sitúa la cámara, sino que también realiza los cálculos para simular los movimientos que la cámara realizaría. 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.

Simplemente decir que posiciona la cámara según (eye, center, up) explicando qué son estos parámetros (dibujar)

Las función de GLM que se utiliza para esta matriz es `lookAt(eye, center, up)` cuyos parámetros indican eye la posición de la cámara, center el punto hacia donde mira y up la orientación.

A continuación de explicar el algoritmo diseñado por posicionar la cámara de forma automática por un circuito dado. Al iniciarse esta aplicación se dibuja el circuito y se hace un cálculo para posicionar automáticamente la cámara sobre él 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 17, 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).

No tiene por qué ser así. Si por ejemplo tenemos la siguiente?



Simplemente calcular center

Escribes una parafase por decir que simplemente calculas center

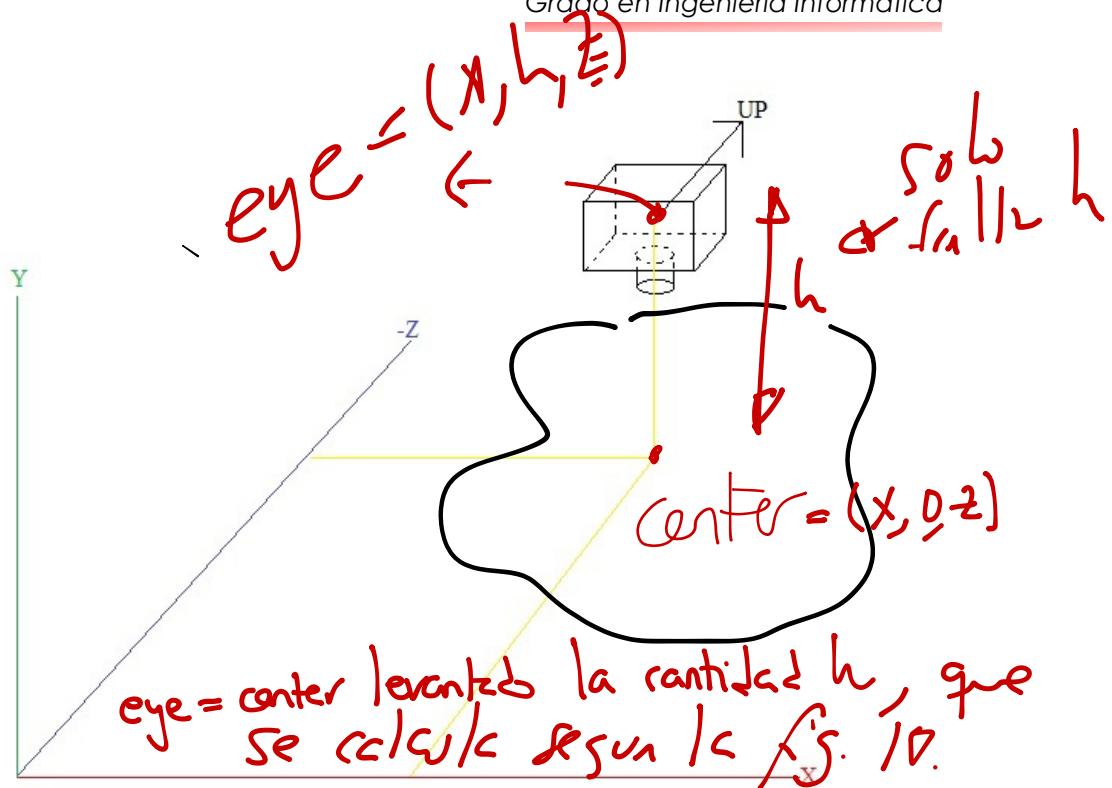
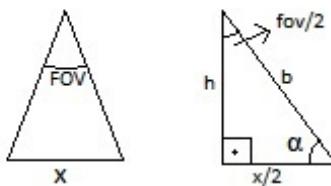


Figura 17. Posición y orientación de la cámara.

Una vez encontrados se crea un vector de 3 posiciones (zona de memoria donde caben 3 datos), 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 18.

Intenta explicar un poco mejor cómo calculas h.



$$\alpha = 180 - \text{FOV}/2 - 90$$

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

$$h = \frac{x/2 * \sin(\alpha)}{\sin(\text{FOV}/2)}$$

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

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

Una vez tenemos la posición de la cámara, dependiendo de que opción haya elegido el usuario se hacen los cálculos para la proyección en perspectiva u ortográfica, esto implica que se cambia también la posición de la cámara obtenida anteriormente. En caso de que se opte por una visualización en ortográfica la cámara se situara justo encima del circuito, sin embargo, en caso de que se opte por una visualización en perspectiva se colocara la cámara a 45° 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.

↳ ¿Cómo cambia eye en este caso?

- **(M) Model:** Esta matriz realiza una transformación de la posición en el modelo a la posición global. Normalmente es una combinación de tres posibles movimientos: trasladar, escalar y rotar. Cada uno de estos movimientos vienen dados por matrices, las cuales se multiplican unas sobre otras, comenzando

por la matriz identidad, dando así una única matriz que será la matriz modelo.

En esta aplicación se utiliza la matriz de translación y la de rotación en Y.

$$\underline{M = T \cdot R}$$

arriba?
así?

Las funciones de GLM son `translate(m,v)`, donde m es la matriz anterior y v es el vector de tres posiciones (x,y,z) que indica cuánto y hacia dónde se mueve; y la función `rotate(m,angle,axis)` donde m es la matriz anterior, $angle$ es el ángulo que se rota y $axis$ es un vector de tres posiciones que indica en qué eje rota.

Las matrices específicas que se utilizan en este proyecto se pueden observar en las figuras 19 y 20. En la figura 19 se puede observar la matriz de translación en X y Z que es donde se puede mover el robot.

$$\begin{bmatrix} 1 & 0 & 0 & X \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & Z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figura 19. Matriz de translación en X y Z.

En la figura 20 se puede ver la matriz de rotación en Y donde φ representa el ángulo que rota el robot.

$$\begin{bmatrix} \cos \varphi & 0 & \sin \varphi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \varphi & 0 & \cos \varphi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figura 20. Matriz de rotación en Y con un ángulo φ .

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

Una vez explicadas pasas a comentar su implementación con GLM.

3. Qt

OpenGL no puede crear ventanas y no tiene una forma sencilla de interactuar con el usuario por lo que es necesario cubrir estas necesidades con otro software. Algunos de los diferentes softwares que pueden hacer esto son sdl y Qt. Se ha optado por Qt puesto que tiene una interacción con el usuario más sencilla y es fácil incluir en un proyecto OpenGL.

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

Internamente se utiliza C++ con alguna extensión para funciones como Signals y Slots, por lo tanto, se utiliza orientación a objetos. 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 21 se puede observar Qt Designer con widgets colocados en una aplicación.

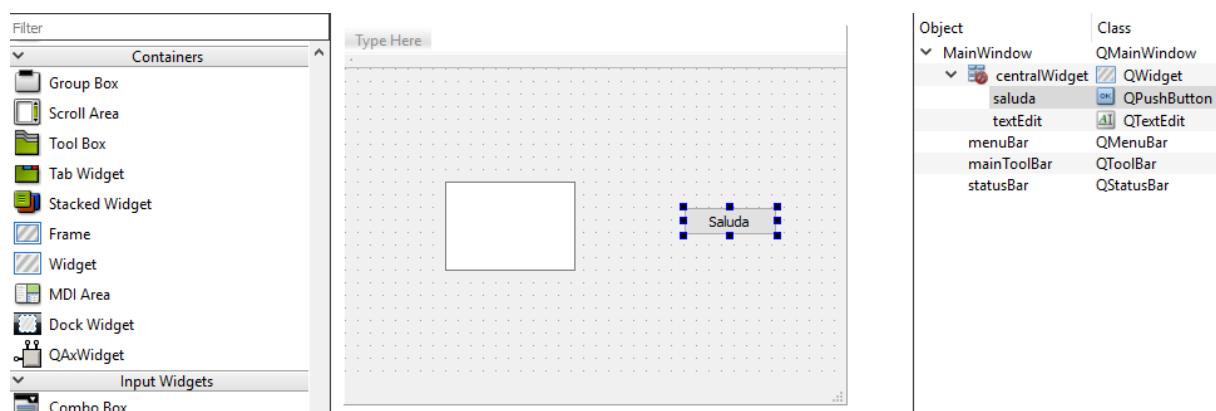


Figura 21. Qt Designer.

→ Pon siempre el código en todo el TFG.

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 simulación. En caso de utilizar librerías externas se debe incluir en el fichero .pro la palabra reservada LIBS seguido del path donde se encuentre la librería: LIBS += pathDeMiLibreria y en la clase en la que se utiliza realizar el include correspondiente.

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

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

En cuanto a las funciones relacionadas con la GUI se trata de las funciones típicas que podrías encontrar en otros software que incluyen interacción con el usuario, por ejemplo la función que se ejecuta al pulsar un botón es on_nombredel boton_clicked(). Además cada tipo de widget tiene métodos específicos, por ejemplo un campo de texto tiene el método value() que devuelve el

→ un er

texto introducido en ese campo, otro ejemplo se trata del widget checkbox el cual tiene el método isChecked() que devuelve si el checkbox está con un tick.

4. Metodología ágil

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

- **Trello** como tablero de tareas, este se divide en las columnas habituales (Product backlog, To Do, Doing, Done). A su vez cada tarea tiene asignada una dificultad representada mediante colores. Las tareas no tienen asignadas personas puesto que hay una sola persona encargada de este tablero. A pesar de no ser relevante para la organización de un equipo y la división de tareas, puesto que solo se trata de una persona, ha resultado muy útil para no perder la visión de proyecto. Todo esto se puede ver en la figura 22.

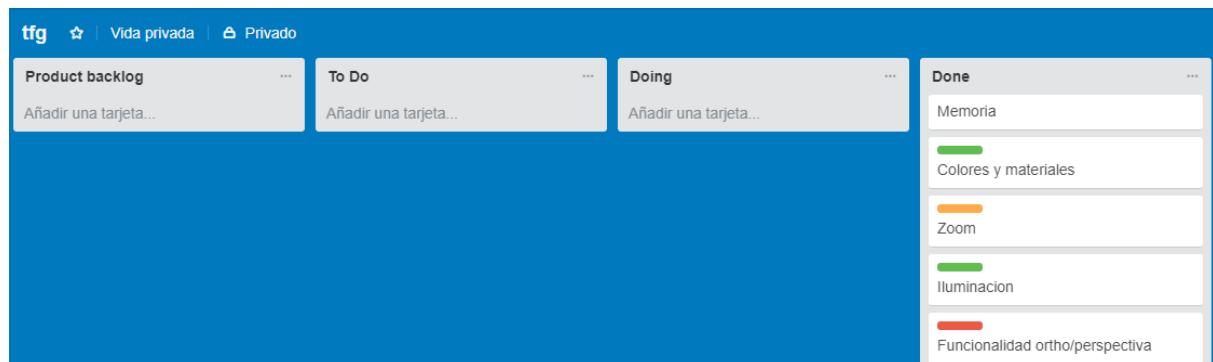


Figura 22: Trello del proyecto Simulacion de un robot siguelineas.

- **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 23.

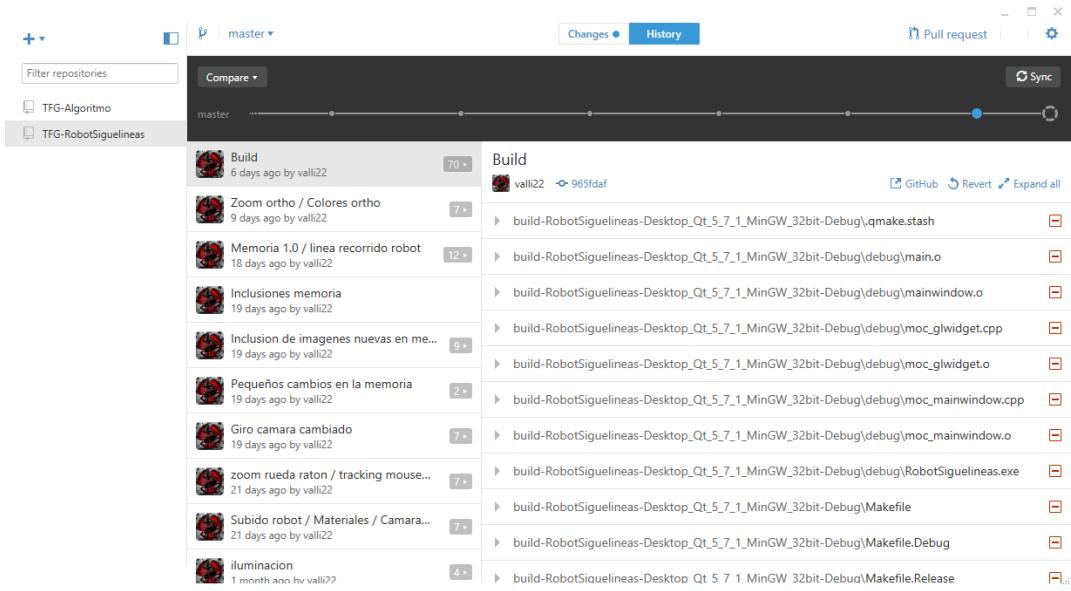


Figura 23: Repositorio en Github desde la aplicación

de escritorio de Windows.

Capítulo 4 Descripción de la aplicación

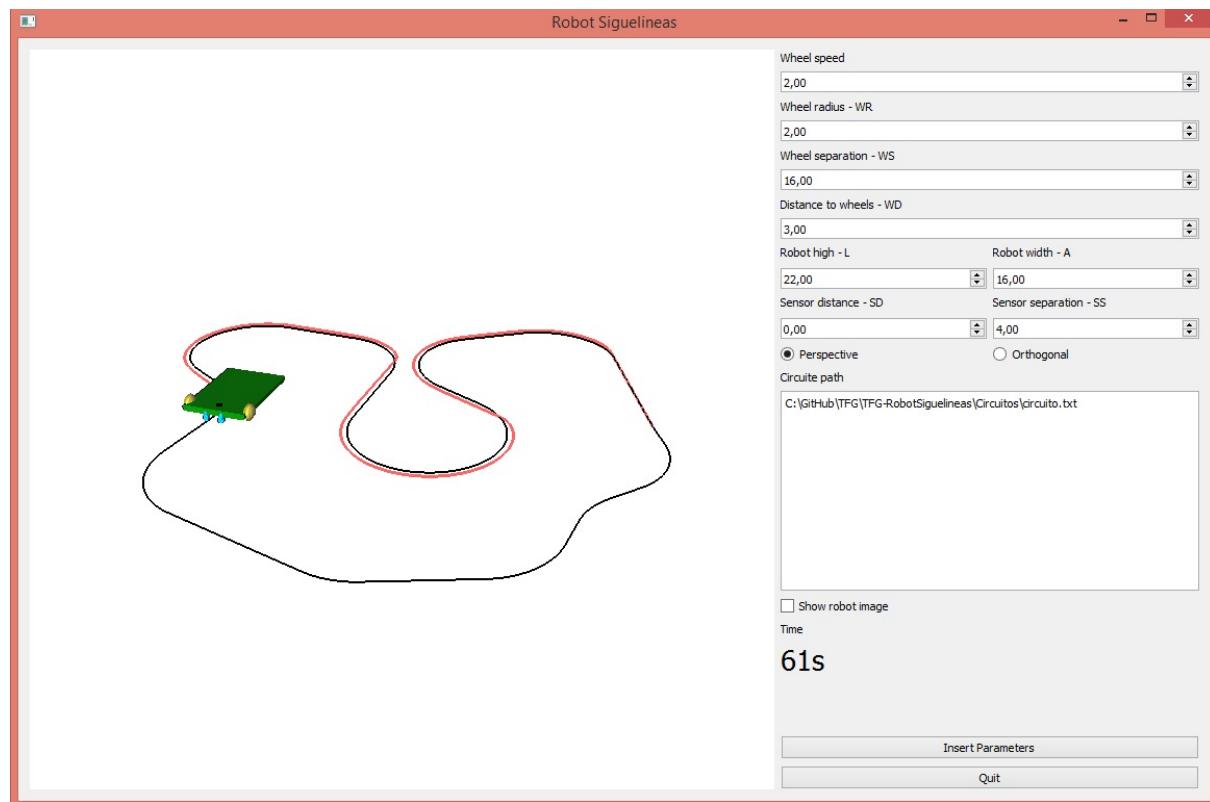


Figura 24. Vista completa de la aplicación.

La figura 24 muestra el aspecto final de la aplicación desarrollada.

Como se puede observar en la figura 24, la aplicación se puede dividir en dos zonas: izquierda y derecha. En la zona de la izquierda se muestra la simulación del robot en 3D, mientras que en la zona de la derecha se sitúan los controles de usuario.

→ 1º el texto y luego la imagen.

1. Zona izquierda: Viewport

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

En esta zona se puede observar la simulación de la aplicación, en función de lo escogido en la zona derecha, tanto en perspectiva, ver figura 25, como en ortográfica, ver figura 26.

Fig 25 muestra el resultado en 2 casos con diferentes modelos de proyección:
 a) perspectiva, y b) ortográfica.

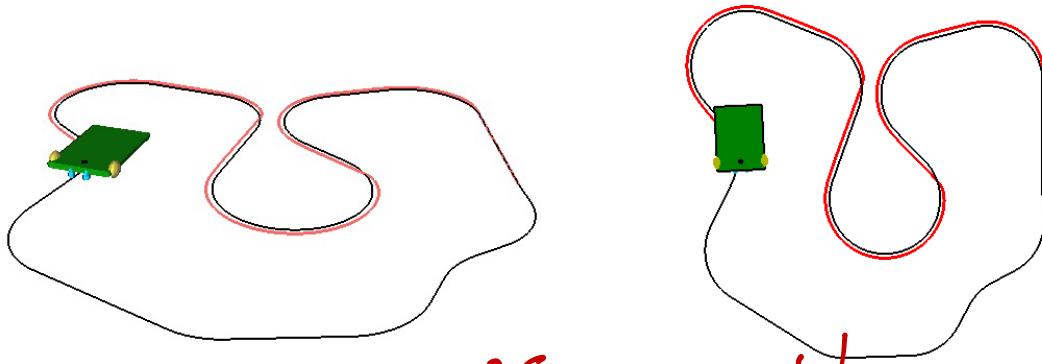


Figura 25. Vista en perspectiva.

Figura 26. Vista en ortográfica.

En el bucle de la aplicación se realizan 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.

→ Esto sobre, el navegador muestra la simulación y ya!

Además, se muestra en todo momento el recorrido que el robot ha llevado durante esa ejecución mediante una línea roja, con lo que se da un mejor feedback al usuario.

Esta zona también se aprovecha para mostrar todos los parámetros geométricos del robot sobre una imagen de este como se muestra en la figura 27. Esta imagen se puede activar y desactivar en todo momento desde la zona derecha de la aplicación. Se explica con mayor detallamiento en el siguiente apartado.

OK

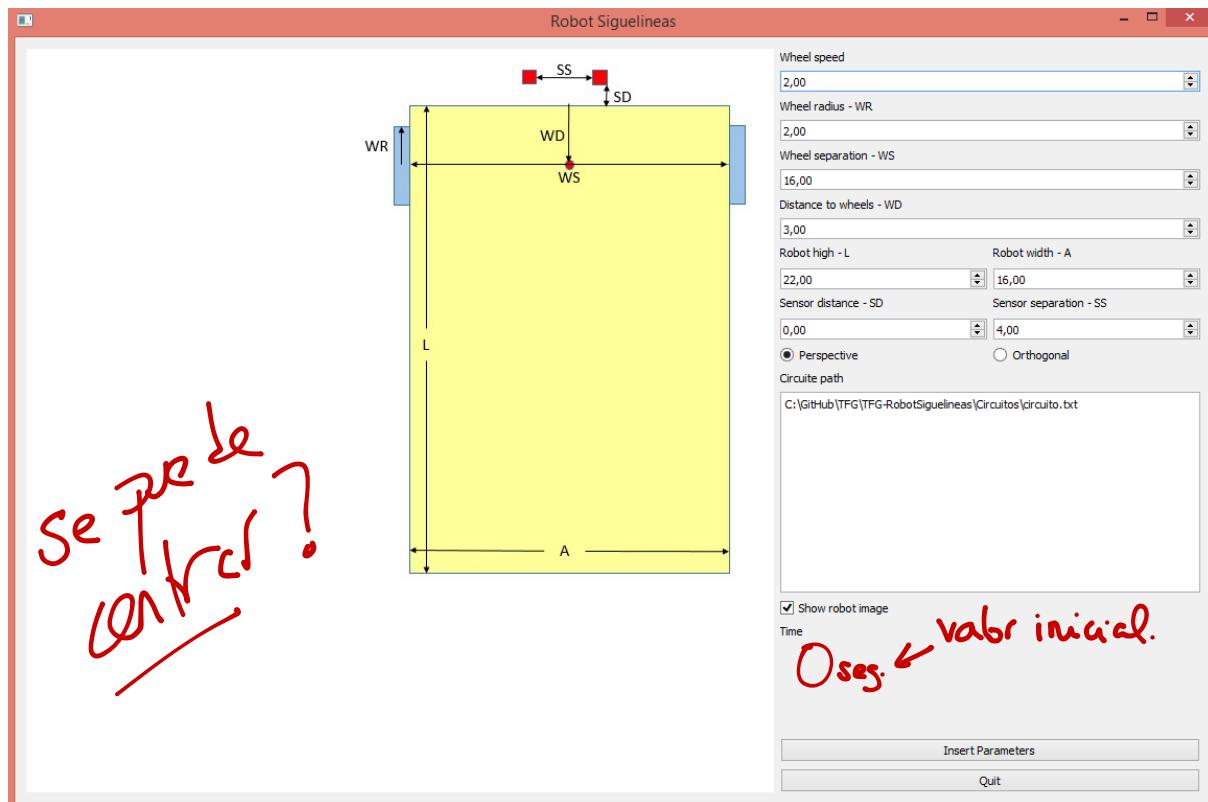


Figura 27. Aplicación con imagen de datos geométricos activa.

2. Zona derecha: Campos de entrada de datos

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

Para un mejor entendimiento de los widgets de interacción con el usuario, se ha añadido abreviaturas a cada uno de ellos y la posibilidad de superponer una imagen con las referencias visuales sobre el robot, más tarde se explica en más detalle. En la figura 28, se muestra esta imagen y una tabla de abreviaturas que explican el significado de cada una de ellas.

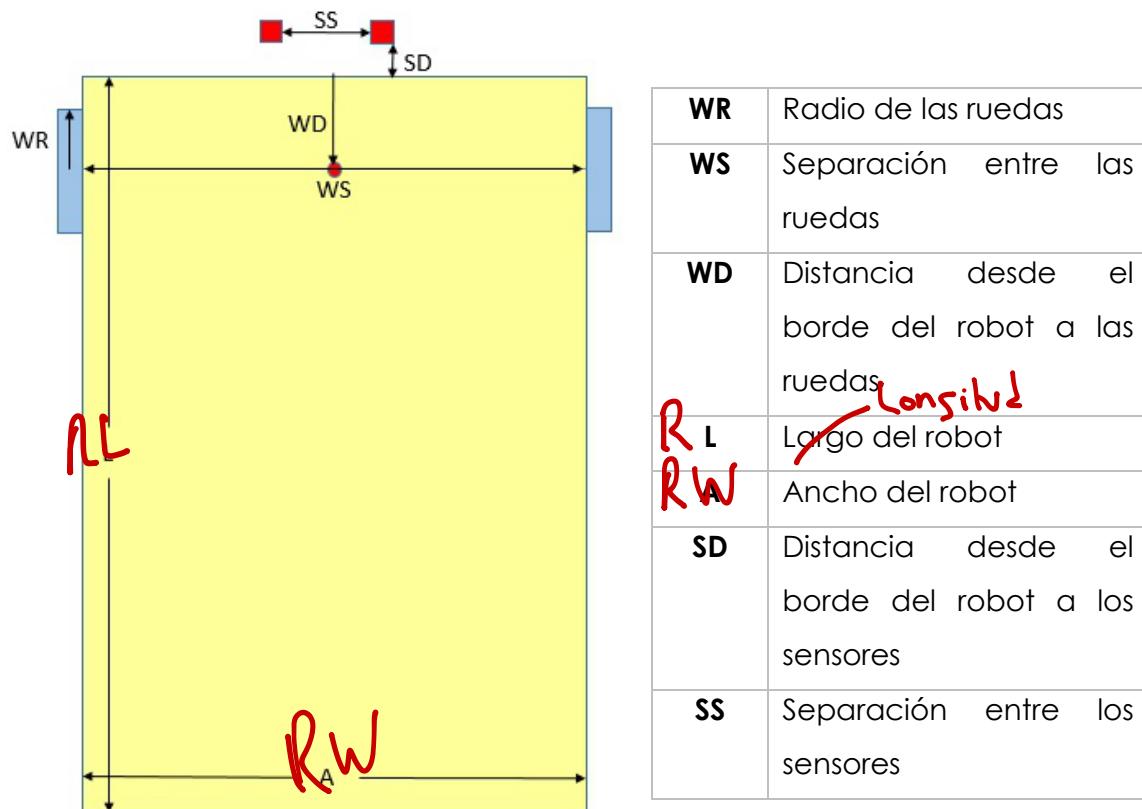


Figura 28. Imagen de referencia sobre datos geométricos del robot y tabla de abreviaturas.

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

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

A continuación, se opta por incluir los inputs en grupos de conceptos similares, como el grupo de las ruedas, el de los sensores y el del tamaño del robot. En el caso de las ruedas se utilizan tres inputs con sus respectivos labels en diferentes niveles, como se

observa en la figura 29, puesto que la unión de los 3 parámetros de las ruedas en un mismo nivel no resultaba fácil de visualizar.

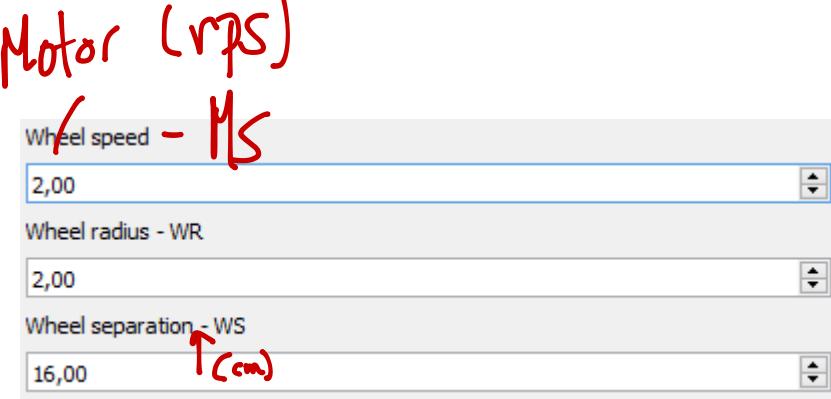


Figura 29. Paquete de ruedas en tres niveles.

incluir unidades (rev.pr.Rad), (m, etc...)

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

Al contrario que para las ruedas, para el tamaño del robot y los parámetros de los sensores, se agrupan 2 en un mismo nivel puesto que así se asimila a simple vista que esos parámetros están relacionados, como se ve en la figura 30.

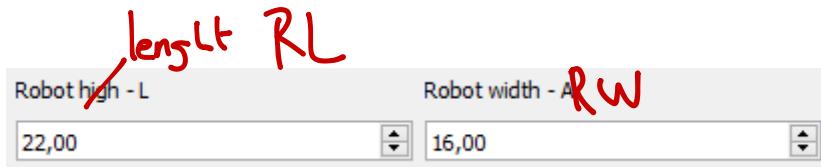


Figura 30. Paquete de dimensiones del robot en un nivel

Hasta aquí están los parámetros de entrada necesarios para la simulación del robot (a excepción del circuito). Por tanto, ahora deben entrar las opciones de la aplicación que no son directamente relevantes para la simulación de la aplicación, en primer lugar, se puede encontrar dos radio buttons que deciden si el usuario desea la vista perspectiva u ortográfica, como se puede observar en la figura 31.

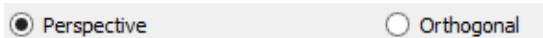


Figura 31. Radio buttons para elegir tipo de proyección.

Esta opción está colocada aquí puesto que a pesar de que todavía queda introducir el circuito, si se pusiera detrás del input para el circuito esta opción sería poco visible y podría ignorarse, mientras que viendo de opciones de tamaño reducido se ve claramente. En la figura 25 se puede observar cómo se visualiza con vista en perspectiva, la vista ortográfica se puede observar en la figura 26.

El último input se trata de un cuadro de texto editable en el que se debe introducir el path del circuito que se desea utilizar, ver figura 32.

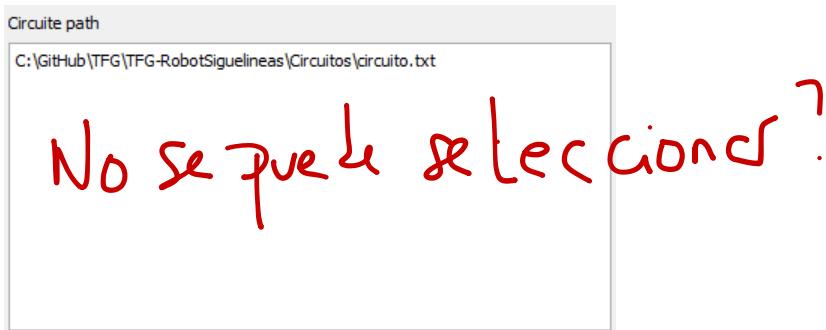


Figura 32. Campo de texto donde se debe introducir el path del circuito.

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 28.

Figura de cldxx.

Después, hay una zona en la que se muestran los segundos pasados desde que empezó la simulación como se muestra en la figura 33, 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.

Time
13s

Figura 33. Tiempo de simulación.

Por último, hay dos botones, el primero cuyo texto pone "Insert Parameters", el cual cuando pulses se introducen todos los parámetros a la aplicación y se inicia la simulación. El último botón contiene el texto "Quit" que sirve para cerrar la aplicación. Estos dos últimos botones se pueden observar en la figura 34.

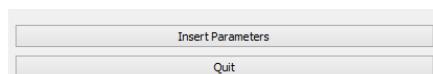


Figura 34. Botones de inicio de simulación y cerrar aplicación.

Habréis visto bien un botón de pause también.

3. Casos de uso

A continuación, se muestran diferentes resultados de diferentes robots sobre un mismo circuito:

- Caso 1: Descripción ..
- Caso 2 : ---
- Caso 3 : ...
- Caso 4 : ... ;

Como ves, así hay un criterio por escojer los casos X No se toca. SALDRÍAN QÓLO R

Una lista describiendo los casos X por qué los has escogido. Habrá que pensar mejor los

Casos : 1. Estándar (referencia)

2 (2.1 Item pero separadas ruedas.

2.2 " " alejadas "

3 (3.1 " " separadas separadas

3.2 " " alejadas " "

4. Jugar con separación ruedas-sensor

Básicamente ve el ① y luego en ②, ③ y ④ se avanza/retrae solo la primera. ⑤ ¿Otro giro?

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

Tiempo: 90s.

En todos los casos que se fijó el texto y Wegs la figura

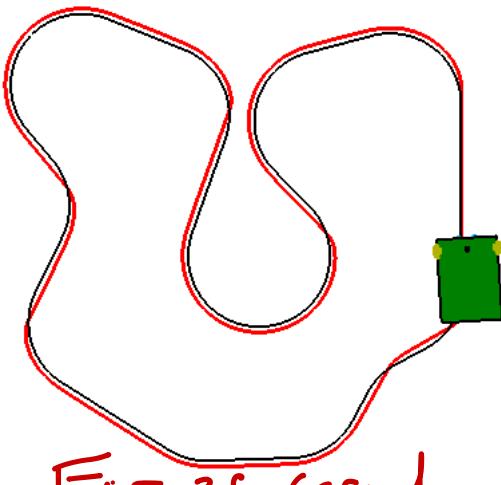


Fig 35: Caso 1

Este robot es un robot con parámetros geométricos medios lo que hace que se realice el circuito en un tiempo normal.

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

Tiempo: 92s.

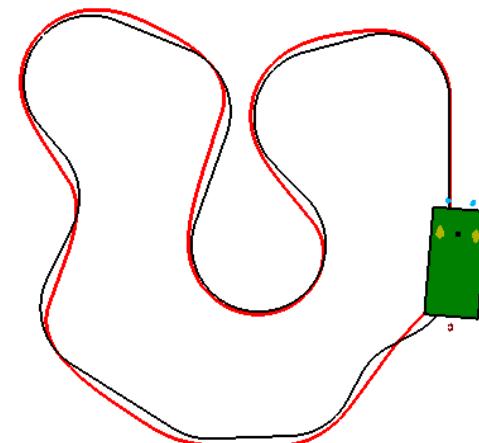
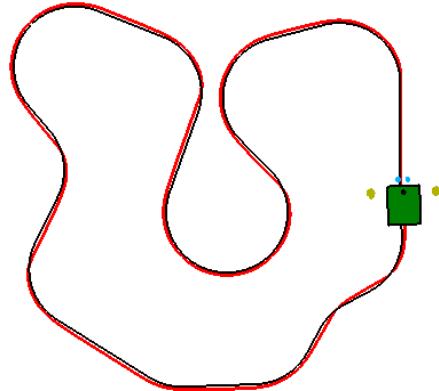


Fig 36...

Este robot tiene un menor radio de ruedas, una mayor distancia entre sensores y ruedas y una mayor distancia entre sensores, ademas tiene las ruedas mas juntas, todo esto hace que se realicen menos giros pero de mayor magnitud.

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

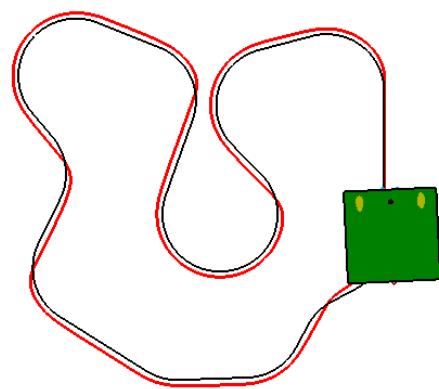
Tiempo: 125s.



Este robot tiene una separación entre ruedas muy grande pero una separación entre sensores reducida. Esto hace que se detecten muy rápido las colisiones, sin embargo, realiza giros de mayor amplitud, lo que hace que en un circuito como este con una gran cantidad de giros sea más lento.

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

Tiempo: 76s.



Este robot posee la misma distancia entre sensores y ruedas que el anterior, misma distancia entre ruedas sin embargo los sensores entre si tienen más distancia y tiene un mayor radio de rueda lo que hace que en este circuito sea más rápido.

Conclusiones

seguro?

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.

Otro podría ser la posibilidad de dibujar los circuitos desde la propia app. O modificar los parámetros del robot desde el dibujo mediante el ratón.

→ Esta está muy bien por hacer que explicarla mejor. Apliación en Red!

Bibliografía

Buscar e incluir referencias [ii] a lo largo de todo el TFG.

1. Libros de robótica.
2. Libros de informática gráfica.
3. Libros de OpenGL.
4. Libros de Qt, C++
5. Ref web de OpenGL
6. Zen para Qt, freenst, GLH.

Metodología AGILE

Trello

Github.

Etc. *frente*

- Refs. de toda aquella gente nombrada/usos en el TFG donde encontrar información adicional.

Del proyecto anterior, web de los profesores,
videos YouTube de los alumnos, etc... (el proyecto)

MATLAB, Simulink, Nemer, Robot Works

Ect ... -