Grado en Ingeniería Electrónica Industrial y Automática

Curso: 2017/2018

Trabajo Fin de Grado

“Desarrollo de un prototipo de robot educacional tipo Segway con control remoto”

Alumno: Vizcaíno Espejo, Juan Rodrigo
Tutor: Pizá Fernández, Ricardo

Septiembre 2018

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Resumen

Este Trabajo Fin de Grado tiene como objetivo desarrollar una aplicación que permita controlar de forma remota un robot tipo segway construido con el kit educacional de Lego Mindstorms. La aplicación diseñada constará de un “modo remoto” que permite mover el robot en la dirección deseada y un “modo trayectoria” que permite al robot realizar trayectorias previamente conocidas.

Durante el desarrollo del trabajo se definirán los objetivos que se pretenden alcanzar. Una vez fijados los objetivos se analizarán los requerimientos del trabajo. Se verán por una parte las piezas, sensores y actuadores necesarios para construir el robot tipo segway. Y por otra parte se analizarán diferentes opciones para el software de programación, necesario para implementar los algoritmos de control.

A continuación se implementará el algoritmo de control de equilibrio para mantener el robot en posición vertical. Luego se diseñará el algoritmo de seguimiento de trayectoria, en este apartado se analizarán dos posibles algoritmos y se comentarán las ventajas e inconvenientes de cada uno, este algoritmo permitirá al robot realizar trayectorias previamente conocidas. Seguidamente se diseñará el algoritmo de control de dirección que permite al robot moverse hacia determinada dirección indicada por el usuario de forma remota desde el PC. Para terminar con el diseño del algoritmo de control se definirá la arquitectura del control completo que permita el funcionamiento conjunto de los tres algoritmos implementados por separado, y se comentarán las modificaciones que se tuvieron que realizar en cada uno de ellos para lograrlo. Esta unión permite que el robot realice las trayectorias y siga determinada dirección mientras se mantiene en equilibrio.

Una vez definido el algoritmo de control completo se implementa la interfaz de usuario que permite el control del robot de forma remota. En este apartado se explicará cómo se ha implementado la interfaz de usuario, el funcionamiento de la aplicación y una pequeña guía de usuario.

Para finalizar se mostrarán los resultados de las pruebas realizadas para verificar el buen funcionamiento de la aplicación. Por último, se realizará el presupuesto del proyecto, y se analizarán las conclusiones y posibles mejoras para el proyecto realizado.

1. Introducción

En el presente Trabajo de Fin de Grado se implementará un robot tipo segway basado en Lego. Y se diseñará una aplicación para:

- Teleoperar el robot según las referencias deseadas.
- Seguimiento de trayectorias previamente conocidas por parte del robot.

1.1 Objetivos.

Para llevar a cabo el robot tipo segway se ha utilizado la plataforma Lego Mindstorms, concretamente con el ladrillo EV3 junto con un sensor giroscopio y los encoders incluidos en los motores de Lego.

Los objetivos del proyecto son:

- Construir con los elementos del kit de Lego Mindstorms EV3 un robot del tipo segway.

- Implementar en LabView el control de estabilidad que permita mantener el robot en equilibrio. Para esto se tomará como punto de partida el control realizado por Lego para el control de un robot tipo segway en la plataforma de Lego Mindstorms, con los ajustes necesarios para adaptarlo a nuestras necesidades.

- Diseñar e implementar el algoritmo que permita dirigir el robot de manera remota a través de comandos enviados desde el PC mediante bluetooth.

- Diseñar e implementar el algoritmo de control que permita al robot seguir una trayectoria previamente conocida.

1.2 Antecedentes.

En diciembre de 2001 es presentado el segway, un invento de Dean Kamen. Se trata del primer dispositivo de transporte con auto balanceo controlado por ordenador. Es un vehículo ligero de dos ruedas que funciona íntegramente con electricidad y se mantiene en equilibrio en todo momento. Para cambiar de dirección el usuario debe inclinarse hacia la dirección que quiera tomar.

Figura 1. Segway utilizado como medio de transporte.

Su principio de funcionamiento está basado en un péndulo invertido sobre dos ruedas accionadas por dos servomotores DC independientes. Se trata de un sistema inestable y no lineal que requiere un sistema de control específico para mantenerlo en equilibrio.

1.3 Lego Mindstorms

LEGO Mindstorms es un kit de robótica educativa que permite construir, programar y controlar tus propios robots haciendo uso de piezas ensamblables y sensores acoplables con un dispositivo programable llamado brick o ladrillo. Cada vez es más habitual utilizar esta plataforma con fines educativos ya que permite experimentar con la tecnología de una manera creativa y sencilla.

Hasta el momento existen 3 generaciones de Lego Mindstorms: El bloque RCX, el bloque NXT y el bloque EV3.

Primera generación: RCX

El primer ladrillo inteligente es desarrollado conjuntamente por LEGO y el Massachusetts Institute of Technology (MIT). A partir del trabajo realizado se desarrolla la primera generación que sale al mercado en 1998. El bloque RCX tiene tres versiones oficiales: 1.0, 1.5 y 2.0.

Figura 2.Lego Mindstorms RCX.

Segunda generación: NXT

El nuevo ladrillo salió al mercado en el año 2006. Lego vendió la generación NXT en dos versiones: Home Edition y Education Edition. Además Lego lanzó al mercado diferentes kits según las características de los programas que se deseara desarrollar. Existen tres versiones del bloque NXT: 1.0, 1.1 y 2.0.

Figura 3.Lego Mindstorms NXT.

Tercera generación: EV3

Esta es la protagonista de nuestro trabajo. Es la última generación y llegó al mercado en 2013. Esta nueva generación presenta una mayor capacidad de procesamiento y sensores mejorados. También tiene una memoria mayor y una conectividad mejorada. Existen dos versiones: Home Edition y Education Edition.

Figura 4.Lego Mindstorms EV3.

En la tabla 1 se muestra una comparativa entre los tres ladrillos.

	RCX	**NTX**	**EV3**
Aparición	1998	2006	2013
Procesador	Hitachi H8/300	Atmel AT91SAM7S256 (ARM7TDMI core)	TI Sitara AM1808 (ARM926EJ-S core)
Velocidad	16 MHz	48 MHz	300 MHz
Pantalla	LCD monocromo segmentado	LCD monocromo 100x64 pixel	LCD monocromo 178x128 pixel
Memoria	32 KB RAM 16 KB ROM	64 KB RAM 256 KB Flash	64 MB RAM 16 MB Flash
Memoria Ampliable	NO	NO	microSD
USB	NO	NO	SI
WIFI	NO	NO	Opcional vía USB (host USB)
Bluetooth	NO	SI	SI
Salidas	3 Puertos (A, B y C)	3 Puertos (A, B y C)	4 Puertos (A, B, C y D)
Entradas	3 Puertos (1, 2 y 3)	4 Puertos (1, 2, 3 y 4)	4 Puertos (1, 2, 3 y 4)
Altavoz	SI	SI	SI
Interfaz usuario	4 botones	4 botones	6 botones retroiluminados
Alimentación	6 pilas AA	6 pilas AA o kit de baterías recargables	6 pilas AA o kit de baterías recargables
Compatible con Apple	SI	NO	SI

Tabla 1.Comparativa entre RCX, NXT y EV3

2. Requerimientos del trabajo.

2.1 Montaje.

Para el montaje del robot tipo segway se utiliza la plataforma Lego Mindstorms, concretamente el kit educacional de Lego Mindstorms EV3.

Uno de los motivos de utilizar esta plataforma es la ventaja de tener el problema mecánico y electrónico muy simplificado. Además esta plataforma permite elegir entre diferentes opciones de programación, ya que el dispositivo programable admite programas desarrollados por diferentes aplicaciones software con solo cambiar el firmware. Esto lo dota de flexibilidad a la hora de seleccionar el software de programación pudiendo elegir el que mejor se acople a nuestras necesidades.

Dentro de la plataforma de Lego Mindstorms existen diferentes opciones para la construcción del robot tipo segway. Algunas de estas posibilidades se resumen a continuación:

- NXT Segway with Rider y NXTway:

Ambos construidos con Lego Mindstorms NXT. Para mantenerse en equilibrio usan el sensor de color (en modo sensor de luz) como un sensor de proximidad al suelo para detectar de forma aproximada la inclinación del robot.

(a) (b)

Figura 5. (a) NXT Segway with Rider y (b) NXTway.

- NXTway-G:

Se trata de una actualización del modelo NXTway. Construido con Lego Mindstorms NXT. Para mantenerse en equilibrio utiliza un sensor giroscopio.

Figura 6. NXTway.

Como se dijo anteriormente para la realización de este trabajo se utiliza el kit educacional de Lego Mindstorms EV3. Este permite la construcción del robot tipo segway llamado Gyro Boy, que utiliza un giroscopio para mantenerse en equilibrio.

Figura 7.Gyro Boy

Este kit se eligió debido a que dispone de todas las piezas, sensores, actuadores e instrucciones necesarias para el desarrollo del robot tipo segway. Además, la generación EV3 tiene mayor capacidad de procesamiento, sensores y conectividad mejorados lo que le confiere cierta ventaja sobre el ladrillo NXT.

2.1.1 Lego Mindstorms EV3

Existen dos versiones de Lego Mindstorms EV3:

- Kit Home.
- Kit Education.

Algunas de las diferencias entre ambas versiones son el número y tipo de piezas, el número y tipo de sensores y la batería. Cada versión incluye los elementos necesarios para la construcción de diferentes modelos de robot.

Como se dijo anteriormente para el desarrollo de este proyecto se utilizará la versión educacional, y más concretamente el kit básico, ya que este dispone de todo lo necesario para la construcción del robot tipo segway. Existe un set de expansión con piezas e instrucciones para otros modelos de robot.

El kit básico contiene:

- Ladrillo inteligente EV3.
- x2 Motores grandes.
- Motor mediano
- Sensor de color.
- Sensor giroscopio.
- Sensor ultrasónico.
- x2 sensores de contacto.
- x4 cables de 25 cm.
- x2 cables 35 cm.
- x1 cable 50 cm.
- x1 cable USB.
- Batería recargable.
- Los cables de conexión.
- Instrucciones de construcción.
- 541 piezas Lego de construcción.
- Software LEGO®MINDSTORMS® Education EV3.

Figura 8.Kit Básico de Lego Mindstorms Education EV3.

2.1.2 Sensores y actuadores.

Un sensor es un dispositivo capaz de detectar magnitudes físicas o químicas y transfórmalas en magnitudes eléctricas. Los sensores aportan la información necesaria para conocer el estado del robot. Mediante esta información se puede conocer el error del proceso respecto a las referencias establecidas. A partir de esta información se pueden diseñar e implementar los algoritmos de control necesarios.

Un actuador es un dispositivo capaz de transformar una magnitud eléctrica en una magnitud física. Por medio de estos la unidad de control puede manipular el comportamiento del actuador para que el sistema se ajuste a las condiciones de funcionamiento.

A partir la información de los sensores y de los actuadores se puede controlar la estabilidad del robot, conocer su localización, etc.

Los sensores y actuadores utilizados son:

- *Sensor táctil:*

El sensor táctil es un sensor analógico que puede detectar el momento en el que se presiona y se suelta el botón rojo del sensor. Este puede funcionar de tres formas: presionado, suelto, o en contacto (tanto presionado como suelto).

Figura 9. Sensor táctil.

- *Giroscopio:*

Es un sensor de giro digital que detecta el movimiento de rotación en un eje simple. Si rota en la dirección que indican las flechas, que se encuentran en la caja del sensor, este puede detectar la razón de rotación en grados por segundo. Además registra el ángulo de rotación total en grados.

Características:

- Medición de rotación y cambios en la orientación.
- El modo de ángulo mide ángulos con una precisión de +/- 3 grados.
- El modo Gyro tiene una potencia máxima de 440 grados/segundo.
- Frecuencia de muestreo de 1kHz.

Este sensor debe estar totalmente quieto en su posición inicial cuando el robot se ponga en marcha.

Figura 10. Sensor giroscopio.

- *Motor grande:*

Es un motor de gran alcance que tiene un sensor de rotación incorporado con una resolución de 1 grado para un control preciso de posición y velocidad.

Características:

- Tacómetro de realimentación de 1 grado de exactitud.
- 160-170 rpm.
- Par de funcionamiento (toque) de 20 Ncm.
- Par de arranque de 40 Ncm.

Figura 11. Motor grande.

El modelo de robot segway seleccionado incorpora, además de los explicados anteriormente, otros sensores y actuadores. Estos, aunque no serán utilizados, se incorporan para respetar el modelo. Además los algoritmos implementados ya tendrán en cuenta el peso y posición de estos. De esta forma si en el futuro se quisiera ampliar o mejorar la aplicación utilizando alguno de estos elementos se podría realizar con facilidad.

Los sensores y actuadores extra son:

- *Sensor de color:*

Es un sensor digital que puede detectar el color o la intensidad de luz que se introduce por una pequeña ranura que se encuentra en una cara del sensor. Este sensor puede funcionar de tres modos: modo color, modo intensidad de luz reflejada y modo intensidad de luz ambiental.

- Modo color: Reconoce 7 colores (negro, azul, verde, amarillo, rojo, blanco y marrón, además de sin color).

- Modo intensidad de la luz reflejada: Mide la intensidad de la luz reflejada. Dispone de una lámpara emisora de luz de color rojo. El sensor utiliza escala de 0 (muy oscuro) a 100 (muy luminoso). En este modo, para optimizar su funcionamiento, debe mantenerse el sensor en ángulo recto y próximo a la superficie que examina, pero sin tocarla.

- Modo intensidad de luz ambiental: Mide la intensidad de luz que ingresa por su ranura desde su entorno. El sensor utiliza escala de 0 (muy oscuro) a 100 (muy luminoso).

La razón de muestreo es de 1 kHz.

Figura 12. Sensor de color.

- *Sensor ultrasónico:*

Es un sensor digital, su funcionamiento consiste en emitir ondas de sonido de alta frecuencia y leer el retardo de sus ecos para determinar la distancia a determinados objetos. También puede enviar ondas de sonido individuales para trabajar como sonar.

Características:

- Mediadas de distancia entre 1 y 250 cm
- Precisión de +/- 1cm
- La iluminación frontal es continua mientras emite e intermitente mientras escucha.
- Devuelve verdadero si escucha otro sonido ultrasónico.

Figura 13. Sensor ultrasónico.

- *Motor mediano:*

Este motor también incluye un sensor de rotación con una resolución de 1 grado. A diferencia del motor grande este es más pequeño, responde más rápidamente pero es menos potente.

Características:

- Tacómetro de realimentación de 1 grado de exactitud.
- 240-250 rpm.
- Par de funcionamiento (toque) de 8 Ncm.
- Par de arranque de 12 Ncm.

Figura 14. Motor mediano.

Los diferentes sensores y actuadores se conectan de manera sencilla al ladrillo EV3. La conexión se realiza mediante cables RJ-12 modificados. El conector RJ-12 se usa en comunicaciones telefónicas, dispone de 6 posiciones y 6 contactos (6P6C). La modificación que presentan estos cables es que la pestaña no está en el centro sino desviada hacia un lateral. Esta modificación impide que los cables sean compatibles con los de teléfono, que funcionan a voltajes más elevados que los de seguridad.

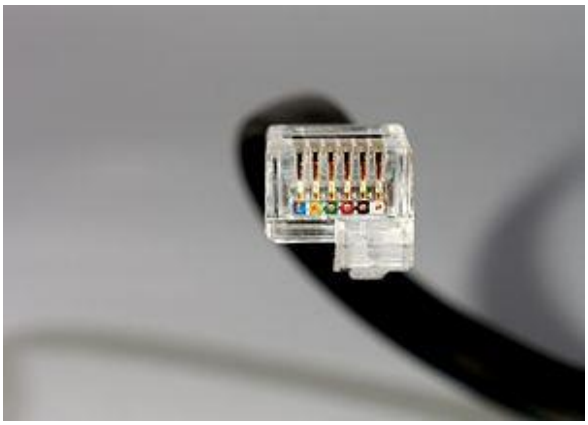

Figura 15.Cable RJ-12 modificado. Se aprecian los 6 cables y la pestaña desplazada hacia el lateral.

Los sensores se conectan a los puertos de entrada 1, 2, 3 y 4. Los motores se conectan a los puestos de salida A, B, C y D.

Motores	
Puerto de salida	Motor conectado
A	Motor grande (rueda derecha)
B	
C	Motor mediano
D	Motor grande (rueda izquierda)

Tabla 2.Conexión de los motores al ladrillo EV3.

Sensores	
Puerto de entrada	Sensor conectado
1	Sensor de color
2	Girosensor
3	Sensor táctil
4	Ultrasónico

Tabla 3.Conexión de los sensores al ladrillo EV3.

2.2 Software de programación.

Existen diferentes opciones de programación para la EV3, en este apartado se analizan algunas de estas opciones y se justifica la elección de LabView como software de programación.

Además de las opciones analizadas existen otros softwares que no se analizaron debido a que no cumplen los requisitos necesarios para el desarrollo de la aplicación o no aportan ninguna ventaja respecto al software elegido. Algunos de estos softwares son Matlab, Simulink, RobotC, etc.

2.2.1 LEGO MINDSTORMS Education EV3

Este software de programación está incluido en el kit básico de la versión educativa. El entorno de desarrollo está basado en el software de LabView de la empresa National Instruments.

El entorno de programación es visual y se basa en la conexión de diferentes bloques. Estos tienen predefinidas diferentes acciones y están clasificados en 6 grupos.

- *Bloques de acción:*

Estos tienen las acciones predefinidas para los diferentes actuadores.

1. Motor mediano.
2. Motor grande.
3. Mover la dirección.
4. Mover tanque.
5. Pantalla.
6. Altavoz.
7. Luz de estado del bloque EV3.

- *Bloques de control de flujo:*

Con estos se controla en avance del programa.

1. Iniciar.
2. Esperar.
3. Bucles.
4. Interruptor.
5. Interruptor del bucle.

- *Bloques de sensores:*

Leen datos de los sensores.

1. Botón del bloque EV3.
2. Sensor de color.
3. Girosensor.
4. Sensor infrarrojo.
5. Rotación del motor.
6. Sensor de temperatura.
7. Temporizador.
8. Sensor táctil.
9. Sensor ultrasónico.
10. Medidor de energía.
11. Sensor de sonido NXT.

- *Bloque de operaciones con datos:*

Estos te permiten guardar variables y constantes y realizar operaciones matemáticas.

1. Variable.
2. Constante.
3. Operaciones secuenciales.
4. Operaciones lógicas.
5. Matemáticas.
6. Redondear.
7. Comparar.
8. Rango.
9. Texto.
10. Aleatorio.

- *Bloques avanzados:*

Estos bloques permiten realizar acciones más avanzadas como administras archivos o establecer conexiones Bluetooth.

1. Acceso al archivo.
2. Registro de datos.
3. Mandar mensajes.
4. Conexión bluetooth.
5. Mantener activo.
6. Valor del sensor sin procesar.
7. Motor sin regular.
8. Invertir motor.
9. Detener programa.
10. Comentario.

- *Mis bloques:*

El software permite la creación de bloques propios. Estos se pueden almacenar para su utilización en otros proyectos.

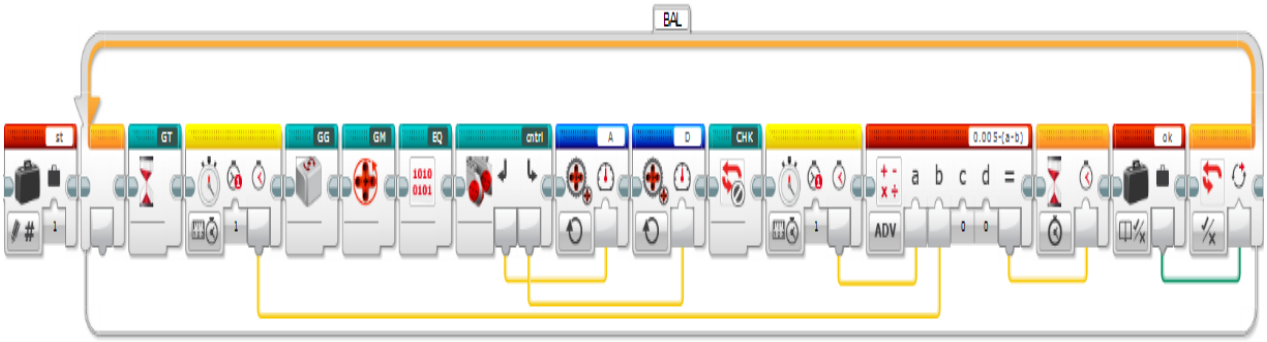

Figura 16. Fragmento del código del programa Gyro Boy desarrollado con Lego Mindstorms Education EV3.

Al analizar la posibilidad de desarrollar la aplicación con este software se analizaron las ventajas y los inconvenientes.

Las ventajas que se encontraron son las siguientes:

- Se puede conectar con el ladrillo EV3 mediante cable USB, Bluetooth o WIFI (usando un cable USB).

- Permite la ejecución de tareas en paralelo.

- Está enfocado a desarrollar robots propuestos, por lo que incluye instrucciones de montaje y ejemplos de programación para el Gyro Boy.

- Los programas se pueden ejecutar en el ladrillo de forma autónoma o con el ladrillo conectado al PC.

- Permite registro de datos.

Se trata de un software destinado al aprendizaje, por lo que está diseñado para que la programación sea fácil e intuitiva. Esto provoca que tenga ciertas limitaciones a la hora de desarrollar programas más complejos. A continuación se muestran algunos de los inconvenientes que se encontraron y por los que se decidió analizar otras posibilidades.

Los inconvenientes encontrados son:

- Para salir de un bucle solo se puede analizar una variable. Esto complica la gestión del programa. Para el correcto funcionamiento de la aplicación desarrollada es imprescindible poder evaluar el valor de diferentes variables, por lo que esto es un inconveniente insalvable.

- Dispone de pocas estructuras y funciones de programación.

- No se pueden recibir mensajes por vía bluetooth desde el PC. Solo permite recibir mensajes enviados desde otro ladrillo EV3. Este es otro inconveniente insalvable, ya que el objetivo del trabajo es controlar de forma remota el robot desde el PC.

- Es complicado gestionar las variables, estas se van incluyendo en los diferentes bloques, pero no hay forma de poder organizar las variable en listas ni de borrar las variables que ya no se utiliza. En el programa desarrollado existe un número considerable de variables y su organización hacen más sencillo el trabajo de programación, revisión y explicación del código.

- Los bloques de programación tienen un tamaño muy grande, lo que hace que un código simple ocupe mucho espacio en la pantalla. Esto también dificulta el trabajo de revisión y modificación cuando los códigos desarrollados son más complejos.

2.2.2 LabView

LabView (*Laboratory Virtual Instrumentation Engineering Workbench*) es un entorno de programación que utiliza un lenguaje visual gráfico. Este software fue desarrollado por National Instruments (NI) que lanzó al mercado la primera versión en 1986. El lenguaje de programación se llama lenguaje G y es 100%

gráfico. La programación consiste en unir diferentes bloques que representan funciones o hardware. Los programas en LabView se llaman instrumentos virtuales (VI).

Para poder interactuar con el ladrillo EV3 NI ha desarrollado un módulo de LabView para Lego Mindstorms que permite programar y controlar el Lego Mindstorms EV3 con LabView. Este módulo no está disponible para la versión de LabView 2017, por este motivo para el desarrollo de este trabajo se utiliza LabView 2016. Las características del módulo según la web de NI son las siguientes:

- *Centro de Proyecto del Robot:* Incorpore contenido del plan académico y comparta resultados en un solo lugar.

- *Editor de Control Remoto:* Configure y controle visualmente su NXT/EV3 usando un control de palanca o un teclado.

- *Reproductor de Piano:* Reproduzca sus propios sonidos y canciones en el dispositivo NXT/EV3.

- *Terminal NXT/EV3:* Administre la memoria y los programas del NXT/EV3 desde su pantalla.

- *Visualización Remota:* Vea todas las pantallas y botones de su NXT/EV3 en el monitor de su PC.

- *Editor de Esquemáticos:* Configure gráficamente y pruebe conexiones de motor y sensores

- *Visualizador de Sensores*: Vea los datos desde sus sensores desde su proyecto en tiempo real.

- *Visualizador de Datos:* Registre y analice fácilmente los datos que adquiere desde su NXT.

- *Editor de Imágenes:* Cree sus propias imágenes para visualizar en la pantalla del NXT.

Este módulo dispone de todos los bloques de funciones y hardware necesarios para el desarrollo de la aplicación.

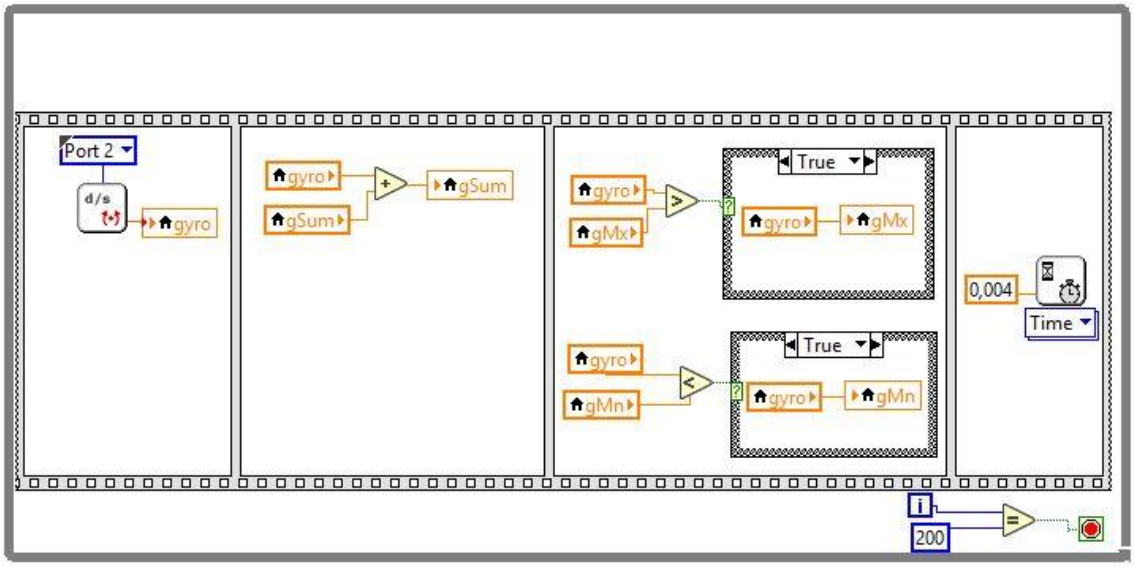

Figura 17.Fragmento del código del programa Gyro Boy desarrollado con LabView.

Al analizar la posibilidad de desarrollar la aplicación con este software se analizaron las ventajas y los inconvenientes.

Al igual que el software de LEGO MINDSTROMS Education EV3 dispone de las siguientes ventajas:

- Se puede conectar con el ladrillo EV3 mediante cable USB, Bluetooth o WIFI (usando un cable USB).
- Permite la ejecución de tareas en paralelo.
- Los programas se pueden ejecutar en el ladrillo de forma autónoma o con el ladrillo conectado al PC.
- Permite registro de datos.

Además en comparación con el software de Lego presenta las siguientes ventajas:

- Permite analizar más de una variable para salir de un bucle. Lo que permite el desarrollo de programas más sofisticados.
- Dispone de bloques que permiten comunicarse por vía bluetooth con un PC. Estos bloques sirven para mandar y para recibir mensajes por vía bluetooth. Por lo que se puede trabajar de forma remota. Esto es una condición necesaria para el desarrollo de la aplicación.
- Dispone de más funciones y estructuras, que permiten el desarrollo de programas más complejos.

- La gestión de las variables es más sencilla. Se pueden agrupar y añadir comentarios para facilitar el análisis. Además permite borrar las variables que no se van a utilizar.

Las desventajas encontradas son:

- Al ser un lenguaje de programación gráfico los programas ocupan mucho espacio en la pantalla, aunque al tener los iconos más pequeños un programa desarrollado en LabView ocupa menos que el mismo programa desarrollado con el software de Lego.

En resumen, a pesar de no haber trabajado antes con LabView se trata de un lenguaje de programación fácil de aprender. Además es un software maduro por lo que existen numerosos programas de ejemplo y tutoriales, proporcionados tanto por la comunidad como por el fabricante.

Además el propio software de Lego se basa en LabView, siendo este último un software más potente.

Por todo esto para el desarrollo de este trabajo se utiliza LabView 2016 ya que es el que más prestaciones aporta. Este software proporciona un potente entorno de desarrollo gráfico que dispone de todas las condiciones requeridas para el desarrollo del presente trabajo.

3. Diseño e implementación de los sistemas de control.

Tal y como se dijo en el apartado *"1.1 objetivos"* el trabajo consiste en diseñar una aplicación que permita controlar de forma remota, desde el PC, el robot tipo segway construido. En este aparatado se muestra el diseño de los diferentes algoritmos de control implementados para controlar el robot. En primer lugar, se presenta el control de equilibrio, que permite mantener el robot estable en todo momento. En segundo lugar, se presenta el control de trayectoria (Modo trayectoria), que permite que el robot realice trayectorias previamente conocidas. A continuación se presenta el control de dirección (Modo remoto), que permite mover el robot en la dirección deseada. Por último se define la estructura de control para el sistema completo, aquí se explica las modificaciones que se realiza en cada uno de los códigos explicados anteriormente por separado para que funcionen conjuntamente.

3.1 Control de equilibrio.

3.1.1 Breve explicación teórica.

El funcionamiento del robot tipo segway se basa en un péndulo invertido sobre dos ruedas. El péndulo invertido es un problema clásico de la teoría de control para el cual se han obtenido diferentes modelos matemáticos, por lo que existen diferentes soluciones. En este trabajo se partirá del modelo matemático utilizado por Lego para el diseño de su algoritmo de control de estabilidad.

Un péndulo simple tiene el punto de giro en la parte superior y se balancea por debajo de este punto. Por el contrario, el péndulo invertido tiene el punto de giro en la parte inferior y se balancea por encima de este punto. Se trata de un sistema inestable, debido a que su centro de masas se encuentra por encima de su punto de giro, y no lineal.

A partir de las ecuaciones diferenciales linealizadas del modelo se puede obtener la representación en variables de estado.

$$\left.\begin{aligned} \dot{x}(t) &= A \cdot x(t) + B \cdot u(t) \\ y(t) &= C \cdot x(t) + D \cdot u(t) \end{aligned}\right\} \quad \begin{aligned} &(1) \\ &(2) \end{aligned}$$

Siendo:

$x(t)$ → Variables de estado.

$y(t)$ → Salida del sistema.

$u(t)$ → Acción de control.

A → Matriz del sistema.

B → Matriz de entrada.

C → Matriz de salida.

D → Matriz de acoplamiento.

Las variables de estado resumen toda la información anterior del sistema y junto a las entradas futuras se puede saber el comportamiento futuro del sistema. Para realizar el control se utilizan las 4 variables medidas que son las siguientes:

- $\theta\ (rad)$ Ángulo de inclinación del robot.
- $\dot{\theta}\ \left(\frac{rad}{s}\right)$ Velocidad angular de inclinación del robot.
- $x\ (grados)$ Posición del motor.
- $\dot{x}\ (\frac{grados}{s})$ Velocidad del motor.

El objetivo para el segway es minimizar el balanceo y mantener la posición vertical. Para ello se realiza un control por realimentación de estados, que consiste en elegir el valor de K para que el sistema en bucle cerrado tenga la dinámica deseada.

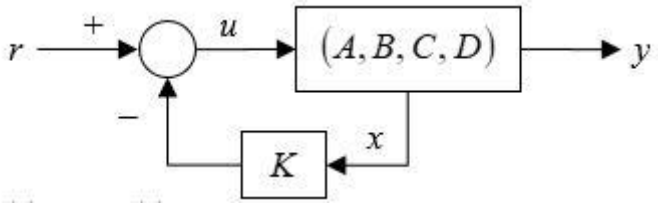

Figura 18. Realimentación de estados.

El caso ideal es que el número de sensores sea igual al número de variables de estado. En este caso no es así, ya que solo disponemos del giroscopio y los encoders de los motores. El giroscopio aporta de forma directa la medida de la velocidad angular de inclinación del robot ($\dot{\theta}$) y este valor se integra para obtener el ángulo de inclinación del robot (θ). Los encoders miden la posición del motor (x) y se deriva este valor para obtener la velocidad del motor ($\dot{x}$)

El control de estabilidad que se implementará en este trabajo se basa en el control de estabilidad diseñado por Lego para el Gyro Boy con pequeñas modificaciones para adaptarlo a las necesidades del proyecto y con alguna pequeña mejora. Por este motivo no se entra de manera más profunda en la explicación teórica.

3.1.2 Algoritmo de control.

Para comenzar a funcionar los sensores se tienen que calibrar, mientras tanto el robot está en reposo y los botones del ladrillo se iluminan de color naranja y muestra por pantalla *"Calculando gOS…" (gOS* es el nombre de la variable que representa la deriva del giroscopio). El sensor giroscopio es delicado por lo que se recomiendo calibrarlo manualmente (desconectar y conectar nuevamente) antes de iniciar el programa. Puede que el robot no deje el estado de reposo, esto es probable que sea debido a que el giroscopio esté a la deriva y se deba reiniciar manualmente. Cuando el robot comienza a funcionar los botones del ladrillo se iluminan de color verde y se muestra por pantalla *"Manteniendo equilibrio"*. Si el robot pierde el equilibrio, y se cae, los botones del ladrillo se iluminan con una luz naranja intermitente mientras se apagan los motores, y

posteriormente se ilumina con una luz roja y muestra por pantalla *"Esperando reinicio"*, para reiniciar el robot se utiliza el botón del sensor táctil. Una vez pulsado el botón de reinicio se ha añadido una espera de 2 segundos para asegurar que el robot está totalmente quieto en la posición de equilibrio para evitar que el programa comience con el robot en movimiento y así favorecer al buen funcionamiento del giroscopio, si se inicia el programa con el robot en movimiento el giroscopio irá a la deriva. Durante esta espera se muestra por pantalla *"El robot debe estar en la posición de reposo".*

Como se ha visto el programa se puede dividir en tres partes:

- Robot en reposo.
- Robot en funcionamiento.
- Robot desequilibrado.

Todo el programa está dentro del *bucle principal*, que se repite de forma infinita. Dentro de este hay un *bucle de equilibrio* que se ejecuta cuando el robot está en funcionamiento, es decir, manteniendo el equilibrio.

En la tabla 4 se muestran las variables utilizadas.

Nombre de la variable	Tipo de dato	Explicación
gMn	DBL	Almacena el máximo valor del giroscopio.
gMx	DBL	Almacena el mínimo valor del giroscopio.
gSum	DBL	Suma de los valores del giroscopio.
gyro	DBL	Valor del giroscopio.
Salirdelbucle	Boolean	Se utiliza para salir del bucle que calcula la deriva del giroscopio.
gOS	DBL	Valor medio de los valores del giroscopio mientras se calcula su deriva. Calculo de error del giroscopio.
gAng	DBL	**Variable medida**. Ángulo de inclinación del robot.
cLo	I16	Número de repeticiones del bucle de equilibrio.
tInt	DBL	Tiempo de integración.
gSpd	DBL	**Variable medida**. Velocidad angular de inclinación del robot.
mSum	DBL	Suma del valor de posición de los dos encoders.
mD	DBL	Diferencia entre la posición medida en el instante k y k-1.
mD1	DBL	Diferencia entre la posición medida en el instante k-1 y k-2
mD2	DBL	Diferencia entre la posición medida en el instante k-2 y k-3.
mD3	DBL	Diferencia entre la posición medida en el instante k-3 y k-4.
mSpd	DBL	**Variable medida**. Velocidad de los motores.
mPos	DBL	**Variable medida**. Posición de los motores.
pwr	DBL	Acción de control.

okaux	Boolean	Variable auxiliar analizada para salir del bucle de equilibrio.
ok	Boolean	Variable para salir del bucle de equilibrio.

Tabla 4. Variables utilizadas en el algoritmo de control de equilibrio.

A continuación se explica el funcionamiento de cada una de las partes del programa.

3.1.2.1 Robot en reposo.

El programa comienza inicializando todas las variables y reseteando el giroscopio, los encoders de las ruedas y el temporizador 2. Los botones del ladrillo se iluminan de color naranja.

A continuación se mide el error de polarización constante en la velocidad del sensor giroscopio. Este puede tener cierta deriva incluso después de reiniciar el sensor. Para algunas aplicaciones esta deriva no tendrá importancia, pero en este trabajo se necesita cierta precisión por este motivo se calcula este error en el valor de la tasa de giro al inicio del programa, y de esta forma se puede restar en cálculos posteriores.

Para calcular el error se mide el valor del giroscopio 200 veces, esperando 0,004 segundos entre cada medida. De estas medidas se obtiene el máximo valor (*gMx*), el mínimo valor (*gMn*) y se suman todos los valores medidos (*gSum*).

Si la diferencia entre el valor máximo y el mínimo es menor de 2 grados/segundo la variación de la tasa es mínima y se sale del bucle (*SalirBucle*). El valor del error (*gOS*) se obtiene al dividir las 200 medidas (*gSum*) entre 200 para obtener el error medio.

Si por el contrario la diferencia entre *gMx* y *gMn* es mayor de 2 grados/segundo quiere decir que hay una variación considerable en la velocidad del giroscopio y se repite el bucle.

Esta secuencia de inicio puede llevar algo de tiempo y el programa no continuará hasta que se completen todas las calibraciones.

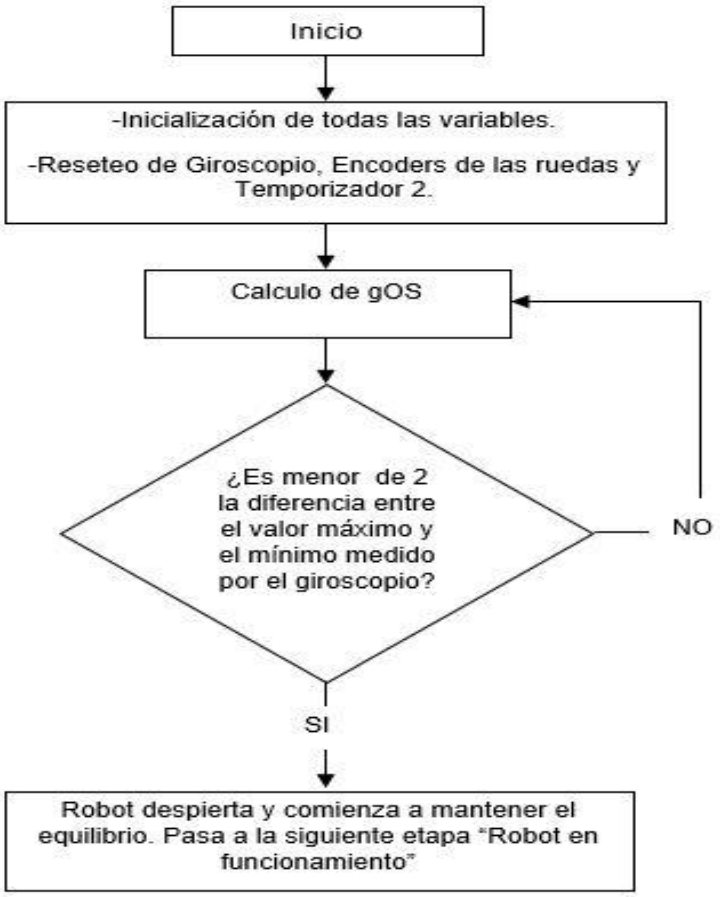

Figura 19. Diagrama de flujo de la primera parte del control de estabilidad.

3.1.2.2 Robot en funcionamiento.

Una vez inicializadas todas las variables, reseteados los sensores y calculada la tasa de error del giroscopio el robot comienza a mantener el equilibrio.

Se empieza estableciendo un pequeño valor a la variable *gAng* debido a que el robot en reposo sobre el soporte tiene una ligera inclinación hacia delante. A continuación el robot "despierta", para avisar al usuario que se encuentra completamente activo los botones del ladrillo se iluminan de color verde y el robot emite un pitido. En este momento entra en el bucle de equilibrio, este bucle se repetirá infinitamente mientras el robot no pierda el equilibrio.

Dentro del bucle de equilibrio el primer paso es obtener la información del temporizador 1, que se utiliza para determinar el valor de *tInt*, que es el periodo de tiempo promedio de una iteración del bucle de equilibrio. Sobre este tiempo se integra el valor de velocidad angular de inclinación del robot (*gSpd)* para obtener el ángulo de inclinación del robot (*gAng).* La variable *tInt* también se utiliza para calcular la derivada de la posición de los motores (*mPos)* para determinar la velocidad de los motores (*mSpd).*

El número de veces que se ha repetido el bucle se guarda en la variable *cLo* que es cero en la primera iteración y aumenta en 1 cada vez que se calcula el tiempo de integración (*tInt)*.

En la primera iteración (*cLo=0)* se establece un valor para *tInt* de 0,014 segundos y se resetea el temporizador 1. En las siguientes iteraciones el tiempo de integración (*tInt)* se calcula dividiendo el valor del temporizador 1 entre el número de vueltas que haya dado al bucle, indicado por la variable *cLo.* Por tanto *tInt* no es el tiempo de la iteración más reciente, sino el promedio de todas las iteraciones.

Una vez calculado el tiempo de integración el siguiente paso es leer el giroscopio para calcular la velocidad angular de inclinación del robot (*gSpd*) y el ángulo de inclinación del robot (*gAng*). El giroscopio mide la velocidad angular de inclinación en grados/segundo y por tanto el ángulo de inclinación está expresado en grados.

El valor de la velocidad angular de inclinación no se obtiene directamente de la lectura del sensor, ya que se debe tener en cuenta el error del giroscopio (*gOS).* El procedimiento es el siguiente:

1. Se actualiza el valor de *gOS*. Este nuevo desplazamiento se calcula a partir del 99,95% del valor anterior de *gOS* y del 0,05% de la tasa actual del giroscopio.

$$gOS\left(\frac{grados}{s}\right) = 0{,}0005 \cdot velocidad\ angular(lectura\ sensor) + 0{,}9995 \cdot gOS \quad (3)$$

2. Este nuevo valor del desplazamiento se resta al valor actual del giroscopio para obtener el valor de velocidad angular de inclinación del robot (*gSpd).*

$$gSpd\ \left(\frac{grados}{segundo}\right) = velocidad\ angular(lectura\ sensor) - gOS \quad (4)$$

A continuación se calcula el ángulo de inclinación del robot (*gAng*). Este valor se calcula integrando el valor de velocidad angular de inclinación del robot (*gSpd).* La variable *gSpd* se multiplica por el tiempo sobre el que está integrado, es decir, el cambio del ángulo es igual a la velocidad de giro (*gSpd)* multiplicado por el tiempo transcurrido (*tInt*). A este cambio de ángulo se le debe sumar el valor anterior del ángulo de inclinación. Todo esto queda resumido en la siguiente ecuación:

$$gAng\ (grados) = gAng + gSpd \cdot tInt \quad (5)$$

El siguiente paso es obtener la información de los encoders situados en eje del motor, para calcular la posición de los motores (*mPos)* y la velocidad de los motores (*mSpd*). En este caso la lectura de la posición de los motores es casi una medida directa de los sensores, con un pequeño ajuste para tener en cuenta las variaciones entre los dos motores debido al giro del robot. Para el cálculo de la velocidad de los motores se calcula la derivada de la posición, es decir, se mide cuánto ha cambiado el valor de las posiciones del motor a lo largo del tiempo de iteración del bucle de equilibrio. El sensor mide la posición del motor en grados y por tanto la velocidad del motor está expresada en grados/segundo.

Obtener la posición y velocidad de los motores es imprescindible para poder calcular las aceleraciones que hay que aplicar a los motores para mantener el robot en posición vertical. A continuación se muestra el proceso para la obtención de *mPos* y *mSpd:*

1. Se lee los sensores de las dos ruedas y se suman los valores, el resultado se guarda en la variable *mSum.* La suma se realiza porque cuando el robot gira se suma y se resta la misma cantidad al valor de la posición del motor, de esta forma se cancela el cambio de posición debido al giro. Aquí ya se tuvo en cuenta que el robot debía seguir trayectorias con giros (en el apartado "*3.4 Diseño del sistema de control completo*" se explicaran todas las modificaciones que se han añadido a los diferentes algoritmos de control para realizar una estructura de control del sistema completo).

2. Ahora se calcula la diferencia entre *mSum* actual y la *mSum* de la iteración anterior. El resultado indica cuánto han girado los motores en la última iteración, esta diferencia se almacena en la variable *mD.*

3. A continuación se obtiene la posición de los motores. Para ello se suma al valor de la posición anterior la distancia recorrida en la última iteración (*mD*).

$$mPos\ (grados) = mPos + mD \tag{6}$$

4. La velocidad de los motores (*mSpd*) se calcula mediante la derivada de la posición. Para esto se almacenan los valores de variación de la posición de los motores de la iteración actual (*mD)* y de las tres iteraciones anteriores, que se almacenan en las variables *mD1, mD2* y *mD3*. Se calcula la media de estos valores, que representan el promedio de cambio del encoders en 4 iteraciones, y se divide entre *tInt.* Esto se resume en la siguiente ecuación:

$$mSpd\left(\frac{grados}{segundo}\right) = \frac{\frac{mD + mD1 + mD2 + mD4}{4}}{tInt} \tag{7}$$

5. Para terminar se actualizan los valores de variación de posición (*mD)* y se desplazan los valores de la variable de *mD3=mD2, mD2=mD1* y *mD1=mD.*

Una vez se dispone de toda la información de los sensores el siguiente paso es calcular la acción de control para que el robot pueda mantener el equilibrio. La potencia aplicada a los motores viene dada por la siguiente ecuación:

$$pwr = K1 \cdot gSpd + K2 \cdot gAng + K3 \cdot mSpd + K4 \cdot mPos \quad (8)$$

Esta ecuación determina, a partir de la información de los sensores, si el robot necesita acelerar para mantener la posición de equilibrio. Con este cálculo se resuelve el problema del péndulo invertido.

La variable de potencia *pwr* depende de las 4 variables medidas que son ángulo de inclinación del robot (*gAng),* velocidad angular de inclinación del robot (*gSpd),* posición de los motores (*mPos)* y velocidad de los motores (*mSpd).* Estas variables se ponderan multiplicando los valores por sus respectivas ganancias y aportan la información de los sensores para el cálculo de potencia del motor.

Los valores de ganancia diseñados por Lego para ajustar y optimizar el funcionamiento del robot son:

K1 = 0,8

K2 = 15

K3 = 0,08

K4 = 0,12

Como se puede ver la variable de ángulo de inclinación del robot (*gAng)* tiene un valor de ganancia mucho más grande que los otros términos. Esto es debido a su importancia, se quiere que el robot se mantenga en equilibrio y tenga un valor muy pequeño (o nulo) para el ángulo de inclinación del robot. La posición y la velocidad de los motores son importantes, ya que para poder acelerar el robot es útil conocer la posición y velocidad actual de las ruedas. El objetivo del sistema de control es que los estados converjan a 0 en el menor tiempo posible.

Para terminar con el cálculo de la acción de control (*pwr)* que se va a aplicar a cada uno de los motores, esta se limita. Los motores de Lego aceptan una potencia máxima de 100. Si el valor de potencia calculado es mayor de 100, entonces se iguala la variable *pwr* a 100. En el sentido contrario si la potencia calculada es menor que -100 se iguala la variable *pwr* a -100. Si el valor de la variable *pwr* se encuentra entre [-100 y 100] no se realiza ningún cambio sobre el valor de la variable.

A continuación se aplica la potencia calculada a cada motor, para ello se utiliza el bloque "*move DC motors".* Este bloque tiene como entradas la potencia y el puerto donde está conectado el motor. La potencia aplicada es la misma para los dos motores.

Para terminar esta etapa se comprueba si el robot mantiene la posición de equilibrio. Para ello se utiliza el valor de la variable *pwr.* Si esta tiene un valor absoluto menor de 100, se reinicia el temporizador 2. A continuación se comprueba si el tiempo medido por el temporizador 2 es mayor de 1 segundo, como se acaba de reiniciar esta comparación será falsa, y no se realiza ninguna acción sobre la variable *okaux* (que inicialmente es falsa por lo que continuará en ese estado). Si por el contrario el valor absoluto de *pwr* no es menor de 100 (entonces debe ser 100 ya que no puede ser mayor) el temporizador 2 no se reinicia, por lo que el valor de este será mayor de 1 segundo y la variable *okaux* se declara verdadera. Esta variable es un valor auxiliar de la variable *ok* que se utiliza para salir del bucle de equilibrio, como se comentará a continuación.

Cada cierto tiempo se obtienen los valores de los sensores y se calcula la acción de control, este tiempo debe ser pequeño ya que la tarea de mantener el robot en equilibrio requiere cierta rapidez. El tiempo de muestreo elegido es de 5 ms, pero el bucle de equilibrio no siempre tarda lo mismo en ejecutarse. Para mantener una constancia en la duración de cada iteración se calcula la diferencia entre el valor del temporizador 1 tomado al comienzo del bucle de equilibrio y el valor actual del temporizador 1, para determinar el tiempo de espera (5ms – tiempo transcurrido iteración actual) antes de comenzar la siguiente iteración. Una duración constante de cada iteración permite que los valores tomados en las diferentes iteraciones tengan el mismo peso y así evitamos tener que ponderar el promedio de los valores.

Una vez esperado este tiempo es cuando se evalúa la variable *okaux,* si esta es falsa significa que el robot mantiene el equilibrio y la variable *ok* se mantiene falsa, por lo que el bucle de equilibrio continua ejecutándose. Si por el contrario el robot ha perdido el equilibrio la variable *okaux* será verdadera y la variable *ok* se declara verdadera, lo que hace que termine el bucle de equilibrio.

La variable *okaux* es un valor auxiliar de la variable *ok* que es la que gestiona la salida del bucle de equilibrio. Estas dos variables tendrán el mismo valor, pero se utiliza la auxiliar para asegurar que se termina el bucle de equilibrio correctamente. Si se hiciera verdadera la variable *ok* cuando se evalúa el estado del robot, se terminaría el bucle de equilibrio en este momento, sin llegar al final.

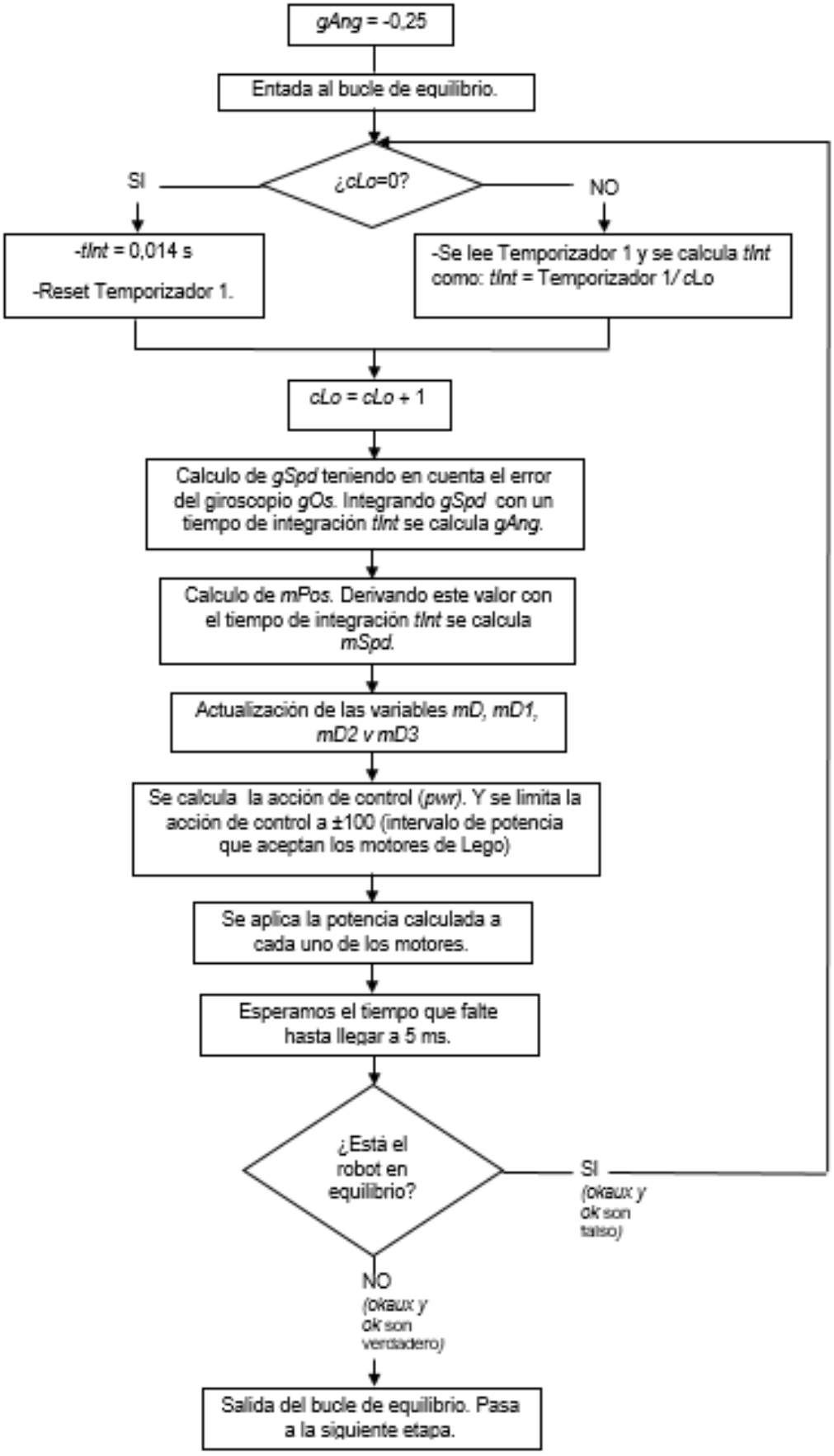

Figura 20.Diagrama de flujo de la segunda parte del control de estabilidad.

3.1.2.3 Robot desequilibrado.

Una vez el robot se desestabiliza se sale del bucle de equilibrio mediante la variable *ok.*

Cuando esto ocurre el primer paso es parar los motores, estos se paran utilizando el bloque *"stop DC motors".* Para avisar al usuario que el robot se ha desequilibrado se iluminan los botones de luz naranja intermitente y el robot emite un pitido.

A continuación el robot espera a que el sensor táctil se pulse para reiniciarse. Para avisar al usuario que el robot está listo para reiniciarse se ilumina los botones de color rojo y se muestras por pantalla el mensaje *"Esperando reinicio".* Cuando se reinicia el robot es muy importante que esté en posición vertical y quieto, para favorecer el buen funcionamiento del giroscopio. Para ello se ha añadido una espera de 2 segundos desde que se pulsa el botón hasta que vuelve a comenzar el bucle principal. Durante estos dos segundos se muestra por pantalla un mensaje para recordar al usuario que el robot debe estar quieto en posición vertical *"Robot debe estar en posición de reposo".*

Una vez pulsado el botón del sensor táctil y esperado los 2 segundos, se reinicia la pantalla y vuelve a comenzar el bucle principal.

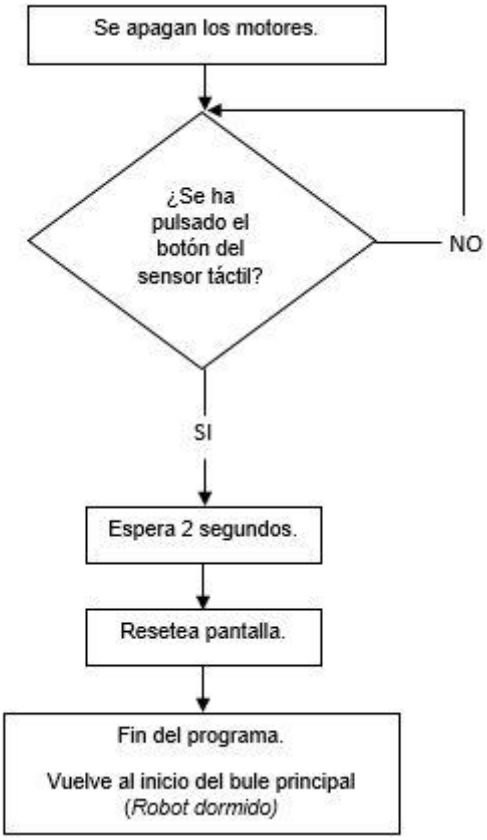

Figura 21. Diagrama de flujo de la tercera parte del control de estabilidad.

3.2 Control de trayectoria.

En este apartado se realiza el diseño del algoritmo de seguimiento de trayectoria. El objetivo es diseñar e implementar un sistema que permita al robot realizar trayectorias previamente conocidas, mediante el control de velocidad de las ruedas, y un algoritmo de seguimiento de trayectoria. Para ello se parte de que el robot desarrollado tiene una configuración diferencial. Por ello se comienza explicando cómo es un robot con configuración diferencial y cómo se realiza la localización de los robots móviles. A continuación se explica las estrategia del control de trayectoria, para este trabajo se han analizado dos métodos geométricos: punto descentralizado (o follow the carrot) y algoritmo de persecución pura.

3.2.1 Robot diferencial y localización.

Un vehículo de configuración diferencial tiene dos ruedas paralelas entre sí y separadas una distancia de *2b.* Estas están movidas por actuadores independientes, cada rueda puede moverse en dos sentidos, hacia delante o hacia atrás.

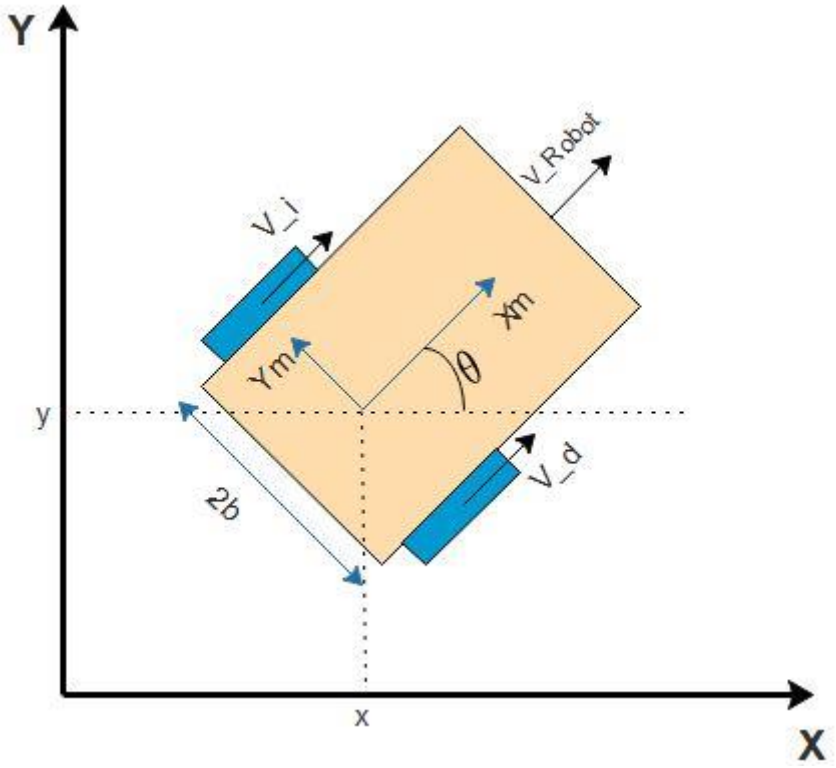

Figura 22. Robot con configuración diferencial.

Para poder seguir una determinada trayectoria es imprescindible que el robot sepa su posición exacta en el espacio. Por tanto, para la navegación autónoma se requiere robots móviles dotados de sensores que aporten información para poder determinar la posición del robot.

Para seguir una trayectoria con precisión se necesita que, tanto el control diseñado como el hardware utilizado sean apropiados. Aspectos como el control de velocidad, la correcta localización del robot, la precisión de los sensores y el algoritmo de seguimiento son aspectos importantes para asegurar el correcto funcionamiento del sistema.

La localización de un robot se puede realizar de formas diferentes. Para la aplicación diseñada se supone que la posición inicial del robot es conocida, a partir de esta se puede estimar la posición actual del robot utilizando la información local del movimiento que aportan los sensores, de forma que se calcula la distancia recorrida desde el punto inicial. Esto se llama odometría, para estimar su posición el robot obtiene información de su entorno a través de los encoders de las ruedas. Se trata de una estimación de la posición debido a que los encoders presentan errores en sus medidas, la precisión de los encoders de Lego no es muy elevada pero es suficiente para la aplicación desarrollada. A partir de las medidas de los encoders y utilizando modelos matemáticos se obtiene la ubicación del robot. La configuración diferencial permite unos cálculos odométricos sencillos.

La trayectoria seguida por el robot depende de la diferencia de velocidad entre las dos ruedas. Por ejemplo, si las ruedas tienen la misma velocidad y sentido el robot seguirá una trayectoria recta, hacia delante o hacia detrás dependiendo del sentido de giro. Si tienen la misma velocidad pero diferente sentido el robot girará sobre su propio eje (eje perpendicular a la línea imaginaria que une el centro de las dos ruedas y que pasa por el medio de esta línea). Se pueden conseguir otras combinaciones de movimiento como giros en sentido horarios y antihorarios mientras se avanza, etc. Estas combinaciones se realizan estableciendo el sentido y la velocidad de giro adecuada a cada una de las ruedas.

La navegación basada en cálculos odométricos tiene ciertas ventajas e inconvenientes. A continuación se analizan estas ventajas e inconvenientes y se compara con otros métodos para conocer la localización del robot como puede ser un sistema de posicionamiento global (GPS o cámara cenital) que permiten conocer la posición absoluta sin necesidad de utilizar la información local.

Uno de los grandes inconvenientes de la odometría es la acumulación de errores a lo largo del tiempo. Estos errores pueden ser sistemáticos, que se producen debido a errores mecánicos como asimetría en los diámetros de los ejes de las ruedas, mala alineación de estas o la resolución de los encoders. Por otro lado, los errores también pueden ser no sistemáticos, que son provocados por el entorno, por ejemplo deslizamiento de las ruedas a causa de suelo irregular (mal contacto de las ruedas con el suelo), suelo resbaladizo o interacción con objetos externos. Los errores sistemáticos son peores ya que se deben a la cinemática del robot y se acumulan constantemente. Las pruebas realizadas con el robot se

hicieron sobre una alfombra sin objetos externos alrededor, para minimizar los errores no sistemáticos.

Además este método tiene el inconveniente de que el error entre la posición real y la estimada se acumula a lo largo del tiempo. Esto provoca que tras recorrer cierta distancia el error acumulado sea grande y la estimación de la posición sea diferente a la posición real. En la aplicación diseñada las distancias que se recorren son pequeñas, y la odometría proporciona una estimación de la posición muy próxima a la posición real del robot para distancias cortas, por lo que este problema no supone ningún inconveniente para el correcto funcionamiento de nuestra aplicación.

Los métodos de estimación de posición global tienen un tiempo de respuesta elevado, además el sensor utilizado te limita a trabajar en exteriores (GPS) o en interiores (cámara). El algoritmo de navegación necesita la posición del robot por lo que este tiempo elevado de respuesta es un inconveniente importante. Si no están actualizados los datos el algoritmo de control no producirá la acción de control requerida.

Una buena solución sería unir ambos métodos. El algoritmo de seguimiento de trayectoria usa la estimación local de la posición y el sistema de posicionamiento global va dando la información, cuando esté disponible, y se utiliza para corregir la estimación local. Este es un método más complejo que no se utiliza debido a que la aplicación no requiere tanta precisión, y para las distancias recorridas por el robot la odometría es suficiente.

En definitiva, los cálculos por métodos odométricos son más baratos que los sistemas de posicionamiento global, ya que se utilizan sensores de bajo coste, en este trabajo se utilizan los sensores de las ruedas y no se necesitan sensores externos como GPS o cámara. Además este método proporciona buena precisión para distancias cortas. También permite frecuencias de muestreo altas, aunque en nuestra aplicación el tiempo de muestreo será elevado, debido a que la tarea de mantener el equilibrio tiene prioridad y por este motivo la realización de la trayectoria se hará lentamente dando prioridad a mantener el robot estable. Por último, las ecuaciones son muy fáciles de implementar en software.

Por todos los motivos vistos anteriormente se utiliza la odometría. Esta estima la posición y la orientación del robot, que se mueve por un plano bidimensional, a partir de la posición inicial y la velocidad lineal y angular del robot.

A partir de la velocidad de la rueda derecha e izquierda se puede calcular la velocidad lineal y angular del robot:

$$\begin{bmatrix} V_{Robot} \\ W_{Robot} \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2b} & -\frac{1}{2b} \end{bmatrix} \cdot \begin{bmatrix} V_d \\ V_i \end{bmatrix} \quad (9)$$

Siendo V_{Robot} la velocidad lineal del robot en mm/s, W_{Robot} la velocidad angular del robot en rad/s, V_d y V_i velocidad lineal de la rueda derecha e izquierda respectivamente en mm/s. Las velocidades de las ruedas del motor se pueden obtener directamente a partir de la información de los encoders de las ruedas.

A partir del modelo cinemático del robot diferencial se calculan las componentes de la velocidad en el eje horizontal del plano ($\dot{X}$), eje vertical del plano ($\dot{Y}$) y la velocidad angular ($\dot{\theta}$):

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} cos\ \theta & 0 \\ sen\ \theta & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} V_{Robot} \\ W_{Robot} \end{bmatrix} \quad (10)$$

Para obtener la posición (X, Y) y la orientación (θ) del robot se debe integrar los valores de velocidad obtenidos. Se busca un método que permita obtener los resultados de las integrales de forma aproximada. Una aproximación a la integral de estas funciones se muestra en la siguiente expresión:

$$\begin{bmatrix} X_{k+1} \\ Y_{k+1} \\ \theta_{k+1} \end{bmatrix} = \begin{bmatrix} X_k + V_{Robot_k} \cdot \cos\theta \cdot T_s \\ Y_k + V_{Robot_k} \cdot \text{sen}\,\theta \cdot T_s \\ \theta_k + W_{Robot_k} \cdot T_s \end{bmatrix} \quad (11)$$

Donde X_k, Y_k , θ_k , V_{Robot_k} , W_{Robot_k} son la posición, la orientación, la velocidad lineal del robot y la velocidad angular del robot respectivamente en el instante *k* y T_s es el periodo de muestreo.

3.2.2 Algoritmo de seguimiento de trayectoria.

El algoritmo de seguimiento de trayectoria se encarga de ajustar el valor de velocidad y sentido a cada una de las ruedas, en cada instante de muestreo, para que el robot realice una determinada trayectoria. Para ello se definen las trayectorias con un conjunto de coordenadas de posición.

Existen diferentes formas para representar la trayectoria, estas suelen representarse como un conjunto de puntos de control por los que el robot debe pasar. Por tanto la trayectoria debe discretizarse para transformarse en una serie de puntos bidimensionales. La trayectoria solo puede ser recorrida por el robot en un sentido, es decir, existe un orden para recorrer los puntos de la trayectoria. De esta forma se debe alcanzar el punto *i* antes que el punto *i+1.*

Existen diferentes tipos de algoritmos de seguimiento de trayectoria basados en control predictivo, linealización de modelo cinemático, métodos geométricos, etc. Para el desarrollo del presente trabajo se han estudiado dos métodos

geométricos diferentes: algoritmo de punto descentralizado (o follow the carrot) y algoritmo de persecución pura (pure pursuit). Este último será el utilizado para el desarrollo de nuestra aplicación debido a los buenos resultados que ofrece.

El algoritmo de seguimiento debe ser capaz, a partir de la diferencia entre la posición y orientación deseada (coordenada de la trayectoria) y la posición y orientación estimada del robot (localización del robot por odometría), de calcular la acción de control requerida para cada rueda, para lograr que el robot siga la ruta establecida de la manera más aproximada posible.

A continuación se desarrollara el diseño de ambos algoritmos de seguimiento de trayectoria, se analizará el comportamiento de ambos y se justificará la elección del algoritmo de persecución pura frente al algoritmo de punto descentralizado.

3.2.2.1 Algoritmo de punto descentralizado o follow the carrot.

El funcionamiento de este algoritmo es muy sencillo. Para comenzar se define la trayectoria, esta se divide en varios puntos que sirven como referencia al robot. En cada iteración se le pasa al robot valores de referencias de posición y velocidad para los ejes x e y que se almacenan en las variables *refx, refy, velxref* y *velyref*.

Por tanto, para cada iteración el robot tiene un punto objetivo, que es el punto de zanahoria, que alcanzar. El funcionamiento consiste en ir corrigiendo la orientación del robot para dirigirlo hacia dicho punto.

Antes de que el robot haya alcanzado el punto objetivo este cambiará al siguiente punto de referencia, que se convertirá en el nuevo punto objetivo. De esta forma el robot va “persiguiendo” el punto objetivo sin llegar a alcanzarlo.

Debido al funcionamiento de este algoritmo de seguimiento de trayectoria, se conoce como follow the carrot.

Figura 23. Follow the carrot.

Para poder perseguir el punto objetivo el algoritmo de punto descentralizado realiza el control a partir de la posición y la velocidad de un punto que está a una distancia *e* del eje de tracción del robot.

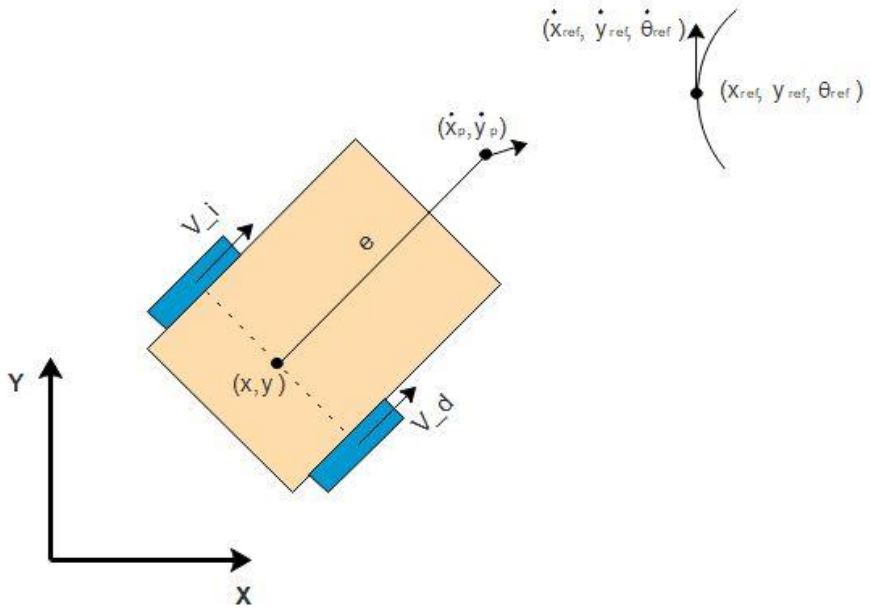

Figura 24. Punto descentralizado.

Para realizar el control de velocidad de las ruedas se necesita diseñar un regulador PID que a partir de la diferencia entre la velocidad angular de referencia y la velocidad angular del robot calcule la acción de control necesaria. Para implementar un control de velocidad el primer paso es identificar el sistema.

A continuación se explicará brevemente, no se entrará en profundidad debido a que el uso de este algoritmo fue descartado, la identificación de los motores de Lego, el diseño del regulador PID para el control de velocidad de los motores Lego, el funcionamiento del algoritmo de seguimiento de trayectoria y finalmente se comentarán los resultados y los motivos por los que el uso de este algoritmo fue descartado.

En la tabla 5 se muestran las variables utilizadas para el algoritmo de seguimiento de trayectoria por el método de punto descentralizado.

Nombre de la variable	Tipo de dato	Explicación
tfinal	DBL	Tiempo que tarda en realizar la trayectoria.
niteraciones	I16	Indica en cuantos puntos se ha dividido la trayectoria. Y las iteraciones que debe realizar el bucle de control.
Kpi	DBL	Ganancia del regulador diseñado.
Ti	DBL	Tiempo de integración del regulador diseñado.
q0	DBL	Valor del regulador PI diseñado.
q1	DBL	Valor del regulador PI diseñado.
Ts	DBL	Periodo de muestreo
U_d	DBL	Acción de control para rueda derecha. En el instante k.
u_d1	DBL	Acción de control para rueda derecha. En el instante k-1.
u_i	DBL	Acción de control para rueda izquierda. En el instante k.
u_i1	DBL	Acción de control para rueda izquierda. En el instante k-1.
refx	DBL	Referencia de posición para el eje x.
refy	DBL	Referencia de posición para el eje y.
velxref	DBL	Referencia de velocidad para el eje x.
velyref	DBL	Referencia de velocidad para el eje y.
x	DBL	Posición del robot en el eje x en mm.
y	DBL	Posición del robot en el eje y en mm.
theta	DBL	Orientación del robot en rad.
w_d	DBL	Velocidad angular de la rueda derecha (rad/s).
w_i	DBL	Velocidad angular de la rueda izquierda (rad/s).
v_d	DBL	Velocidad lineal de la rueda derecha (mm/s).
v_i	DBL	Velocidad lineal de la rueda izquierda (mm/s).
v_R	DBL	Velocidad lineal del robot (mm/s).
w_R	DBL	Velocidad angular del robot (rad/s)
xpunto	DBL	Expresión de control cinemático.
ypunto	DBL	Expresión de control cinemático.
kx	DBL	Expresión de control cinemático.
ky	DBL	Expresión de control cinemático.
radiorueda	DBL	Radio de la rueda del robot en mm.
b	DBL	La mitad de la distancia entre las ruedas del robot en mm.
e	DBL	Distancia en mm desde en eje de tracción del robot hasta un punto sobre el que realiza el control.
vref_d	DBL	Expresión del modelo cinemático inverso. Velocidad lineal de referencia rueda derecha (mm/s).
vref_i	DBL	Expresión del modelo cinemático inverso. Velocidad lineal de referencia rueda izquierda (mm/s).

wref_d	DBL	Expresión del modelo cinemático inverso. Velocidad angular de referencia rueda derecha (rad/s).
wref_i	DBL	Expresión del modelo cinemático inverso. Velocidad angular de referencia rueda izquierda (rad/s).
PosD	DBL	Posición del motor derecho (rad) en el instante k. Lectura directa del encoders.
PosD1	DBL	Posición del motor derecho (rad) en el instante k-1.
PosI	DBL	Posición del motor izquierdo (rad) en el instante k. Lectura directa del encoders.
PosI1	DBL	Posición del motor izquierdo (rad) en el instante k-1.

Tabla 5. Variables del algoritmo de seguimiento por método de punto descentralizado.

Identificación de los motores Lego.

En este apartado se realiza la identificación del motor de corriente continua de Lego, para obtener el modelo del sistema. La obtención del modelo permite conocer el funcionamiento del sistema, y este debe ser capaz de responder de la manera más fiel posible a como lo haría el sistema físico.

La función de transferencia es un modelo matemático que se puede obtener a partir de las entradas ($V_{aplicada}$) y las salidas ($\omega_{salida}\left(\frac{rad}{s}\right)$) del proceso. Este modelo viene dado en el plano continuo mediante una ecuación diferencial, y en el plano discreto por una ecuación en diferencias.

Para realizar la identificación de los motores se implementó un programa en LabView que introduce entradas de valor conocido a los motores y guarda el valor de la salida en un fichero para su posterior análisis.

Se va a realizar la identificación en velocidad angular del motor. Se sabe por tanto, que se trata de un sistema de primer orden (la ecuación característica de la función de transferencia es de orden 1). Por tanto tendrá la siguiente ecuación diferencial:

$$\frac{\partial y(t)}{\partial t} + ay(t) = bu(t) \qquad (12)$$

Aplicando la transformada de Laplace se obtiene:

$$sY(s) + aY(s) = bU(s) \qquad (13)$$

La función de transferencia es la relación entre la salida y la entrada del sistema:

$$G_{vel}(s) = \frac{Y_{vel}(s)}{U(s)} = \frac{b}{s+a} = \frac{k_{vel}}{\tau_{vel} \cdot s + 1} \quad (14)$$

Por tanto, los parámetros a identificar son k_{vel} que es la ganancia estática (b/a) y τ_{vel} que es la constante de tiempo (1/a).

Para identificar el proceso se van a introducir una serie de escalones a la entrada y se analizará el valor de la salida. Como ya se dijo anteriormente estos valores se obtienen del programa de LabView implementado. La identificación se realizará por dos métodos. Primero de forma manual, a partir de los datos obtenidos se calcula el valor de la ganancia y la constante de tiempo. Posteriormente se utilizará la función "ident" que proporciona Matlab para realizar identificación de sistemas. Se verá que ambos métodos proporcionan resultados similares.

Para el cálculo se debe saber que el valor de la ganancia será el cociente entre el valor final de la salida $Y_{velfinal}$ y el valor del escalón introducido "*r*" tal y como muestra la siguiente ecuación.

$$k_{vel} = \frac{Y_{velfinal}}{r} \quad (15)$$

El valor de la constante de tiempo corresponde con el tiempo que transcurre desde que se aplica el escalón a la entrada hasta que la salida del proceso alcanza el 63,2% del valor final $Y_{velfinal}$.

Para que el modelo obtenido sea válido en todo el rango de trabajo es conveniente realizar varias identificaciones para diversos escalones. Por este motivo se han obtenido los valores de la salida para diferentes escalones de entrada, tanto positivos como negativos, estos escalones toman valores entre ±100, que es el rango que acepta la entrada del motor Lego. Una vez se tiene los valores de la salida del motor se calcularán los valores de k_{vel} y τ_{vel} para los diversos escalones de entrada, si los valores obtenidos son similares se calcula la media de estos valores para obtener el modelo.

En la realización de las pruebas se ha tenido en cuenta que para obtener el modelo de velocidad es más eficaz partir del motor en marcha cuando se aplica el escalón para evitar la zona muerta.

Prueba realizada:

En esta prueba se aplicaba al motor escalones de 20, 30, 50, 80, 0, -20, -30, -50 y -80. Estos valores de escalón se eligieron intentando que fueran lo más próximos posibles a los valores que se aplican al motor durante el funcionamiento del robot.

Los valores de ±20 se han aplicado para partir del motor en marcha y evitar la zona muerta. Esta referencia se aplica menor tiempo que el resto porque su función es poner el motor en marcha y cuando se estabilice aplicar el escalón.

La salida obtenida para los diferentes escalones se muestra en la siguiente figura:

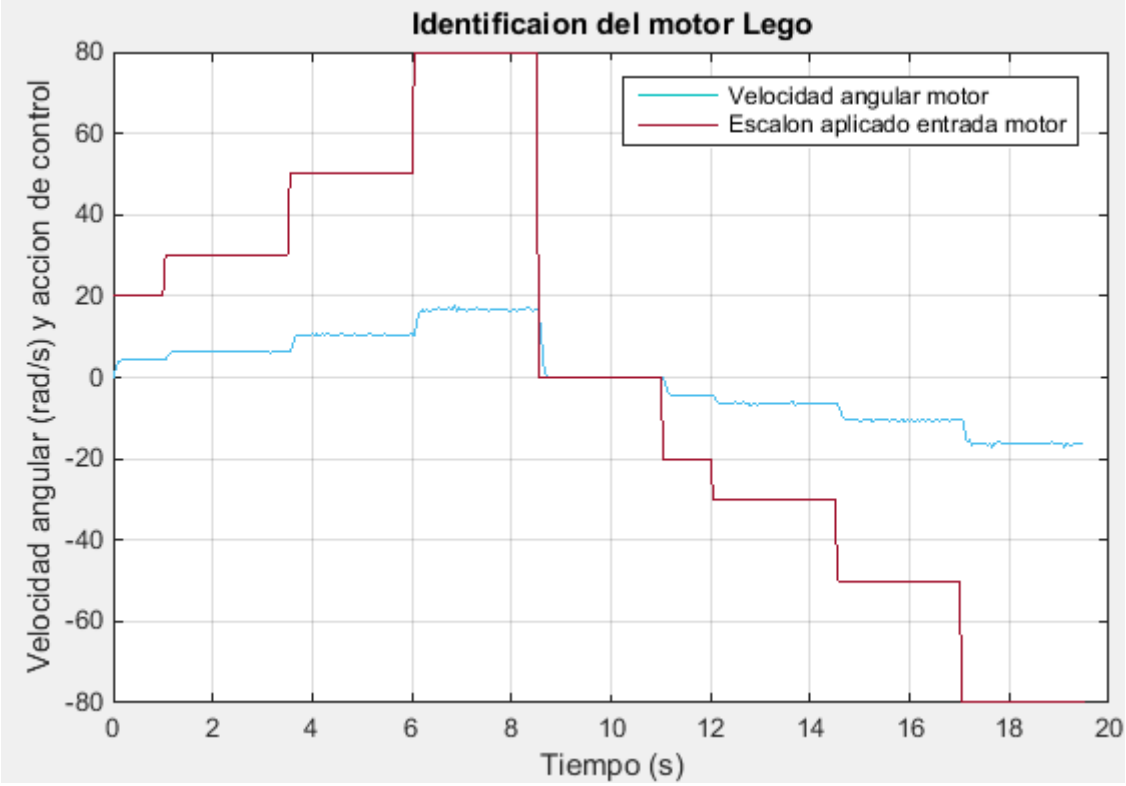

Figura 25. Respuesta del motor Lego ante diferentes escalones de entrada.

Se obtienen los valores de k_{vel} y τ_{vel} para cada referencia, esto se realiza mediante Matlab, y se calcula la media de los valores, obteniéndose la siguiente f.d.t:

$$G_{vel}(s) = \frac{Y_{vel}(s)}{U(s)} = \frac{k_{vel}}{\tau_{vel} \cdot s + 1} = \frac{0{,}2122}{0{,}0683 \cdot s + 1} \qquad (16)$$

A continuación se utiliza la herramienta de Matlab “ident”. A esta función se le pasa el valor de las entradas, el valor de las salidas obtenidas y el tiempo de muestreo utilizado para obtener dicha señal de salida (0,05 s). A partir de estos datos puede estimar un modelo, para ello pide que indiques el número de polos y de ceros que se espera que tenga el proceso, en este caso al ser el modelo de velocidad serán 1 polo y 0 ceros.

La f.d.t obtenida es:

$$G_{vel}(s) = \frac{Y_{vel}(s)}{U(s)} = \frac{k_{vel}}{\tau_{vel} \cdot s + 1} = \frac{3{,}261}{s + 15{,}68} = \frac{0{,}208}{0{,}0638 \cdot s + 1} \qquad (17)$$

Como se puede observar las f.d.t obtenidas por ambos métodos son muy similares, pero a partir de ahora se utilizará la obtenida con la herramienta “ident”.

A continuación se debe validar el modelo. En la siguiente figura se representa la salida real de la velocidad del motor y la salida del sistema identificado.

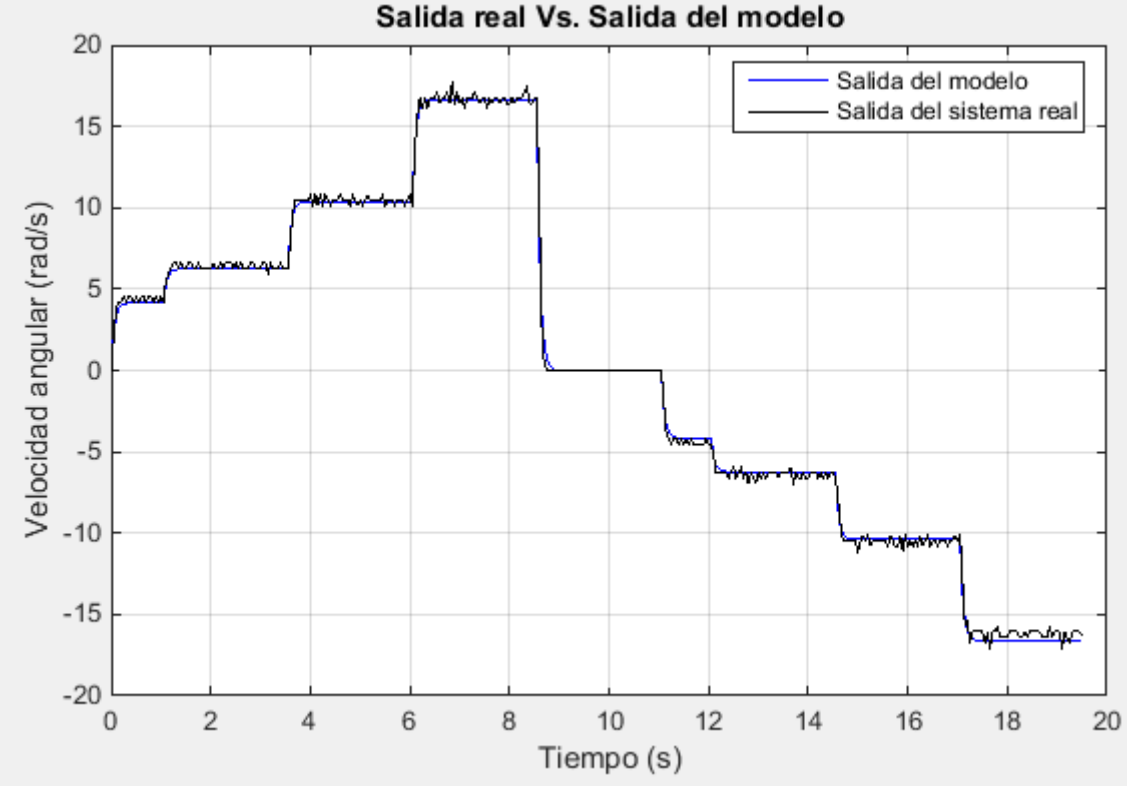

Figura 26. Comparación entre la salida del sistema real y la salida del modelo obtenido mediante la identificación.

Como se puede observar en la figura 26 el modelo obtenido se ajusta al comportamiento del sistema real. Para verlo con mayor claridad a continuación se muestra la comparación entre la salida del modelo y la salida real del sistema cuando el escalón aplicado es de 50.

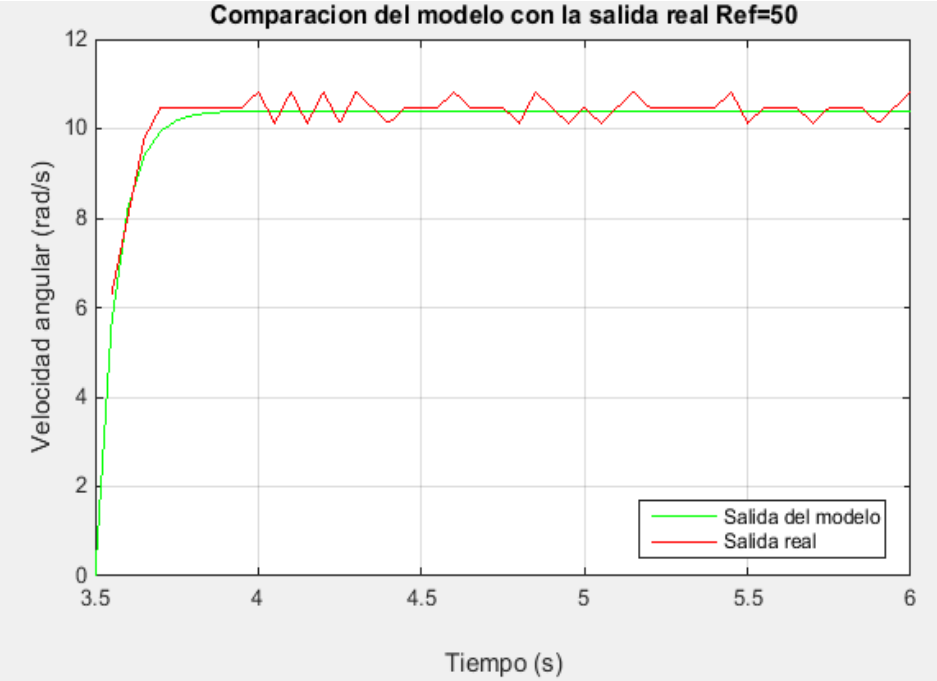

Figura 27. Comparación entre la salida del modelo y la salida real para un escalón de 50.

A continuación se discretiza el modelo obtenido. Para la discretización del sistema se utiliza el método ZOH, esto lo hace Matlab con el siguiente comando:

```
Gd=c2d(G,0.05,'zoh');
```

Del que se obtiene la siguiente f.d.t discreta:

$$G_{vel}(z) = \frac{0{,}113}{z - 0{,}4566} \qquad (18)$$

Para validar la discretización se superpone la salida del modelo continuo y la salida del modelo discreto para el mismo escalón de entrada.

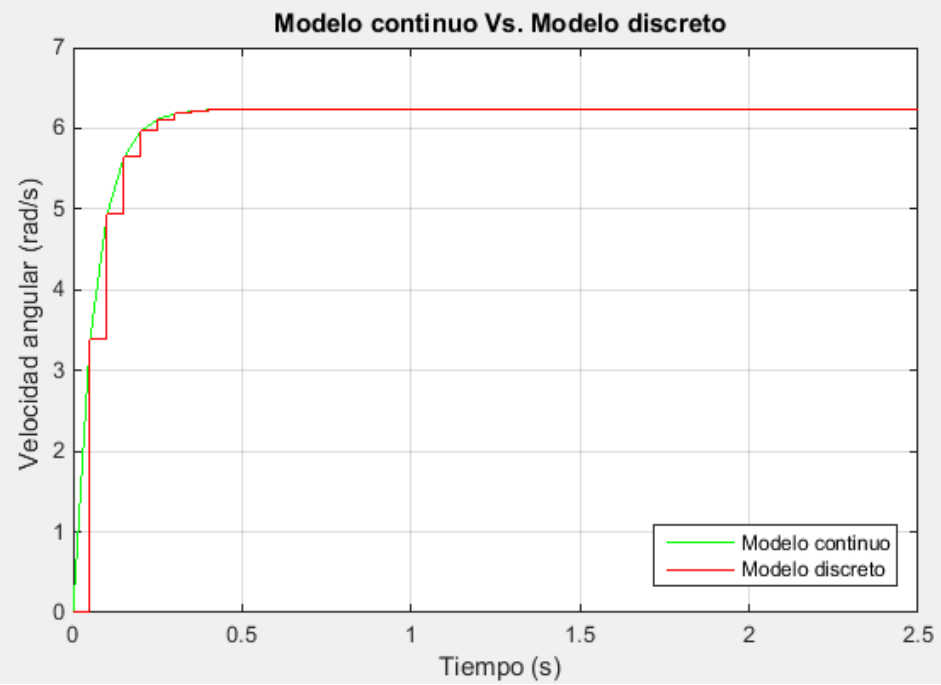

Figura 28. Comparación entre la respuesta del modelo continuo y el modelo discreto.

A continuación se superpone la salida real de velocidad angular del motor, la salida del modelo discreto y la salida del modelo continuo:

- Para un escalón de 30:

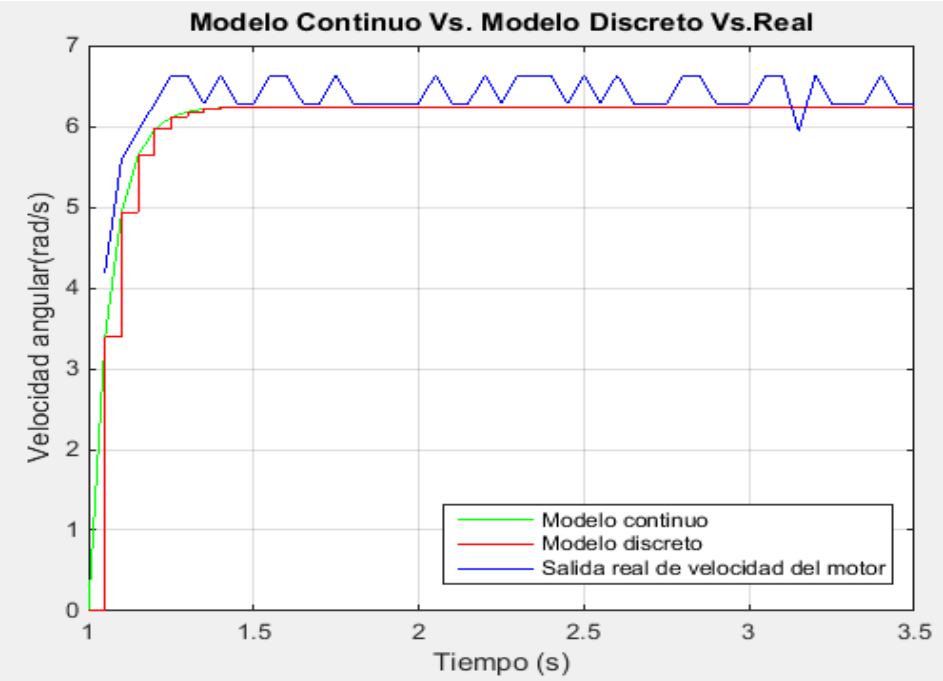

Figura 29. Comparación entre la salida del modelo continuo, el modelo discreto y el sistema real, para un escalón de 30.

- Para un escalón de 50:

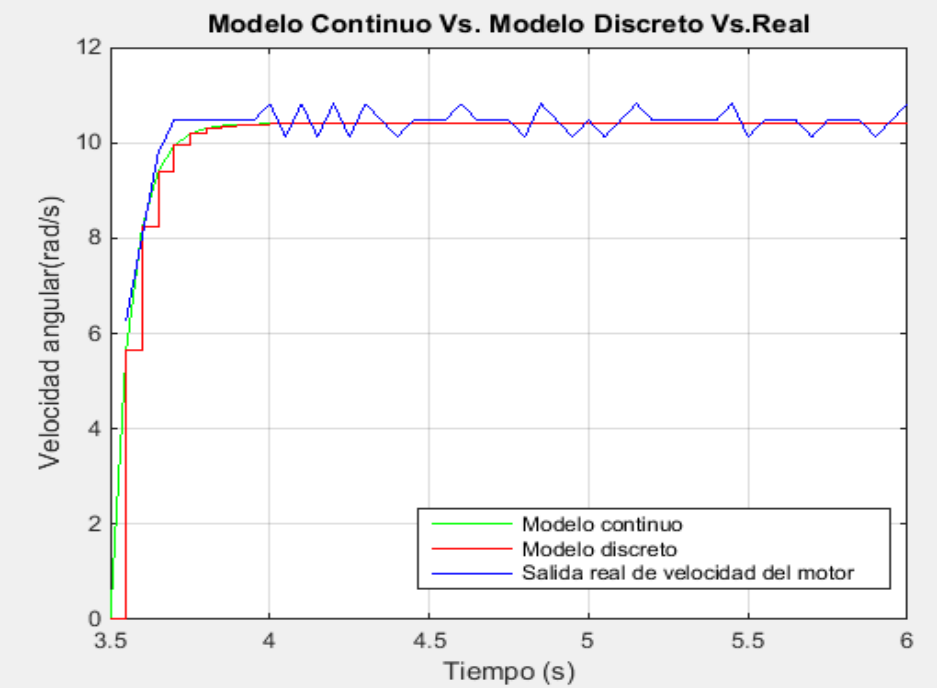

Figura 30. Comparación entre la salida del modelo continuo, el modelo discreto y el sistema real, para un escalón de 50.

Como se observa en las pruebas realizadas el modelo obtenido es correcto. Por tanto, la función de transferencia obtenida y con la que se va a trabajar es:

$$G_{vel}(s) = \frac{Y_{vel}(s)}{U(s)} = \frac{k_{vel}}{\tau_{vel} \cdot s + 1} = \frac{3{,}261}{s + 15{,}68} = \frac{0{,}208}{0{,}0638 \cdot s + 1} \qquad (19)$$

Y discretizada para un tiempo de muestreo de 0,05 s:

$$G_{vel}(z) = \frac{0{,}113}{z - 0{,}4566} \qquad (20)$$

Diseño del regulador.

Para poder seguir la trayectoria se utiliza un regulador PID. Este regulador Proporcional-Integral-Derivativo es un método muy utilizado en vehículos, robots, etc. El regulador PID funciona calculando la diferencia entre el valor medido y el valor deseado, el objetivo es corregir este error, y esto se consigue ajustando una variable manipulable.

EL regulador PID se representa con la siguiente fórmula:

$$u(t) = K_p \cdot e(t) + K_i \cdot \int_0^t e(t) + K_d \cdot \frac{de(t)}{dt} \qquad (21)$$

Siendo:

$K_p \rightarrow$ Ganancia proporcional.

$K_i \rightarrow$ Ganancia integral.

$K_d \rightarrow$ Ganancia derivativa.

$e(t) \rightarrow$ Error, diferencia entre el valor deseado y el valor medido.

Los valores K_p, K_i y K_d son los valores a sintonizar. Para poder sintonizar estos parámetros es necesario saber que función tiene cada uno, por esto se hace a continuación una breve explicación:

- Acción proporcional: La acción aplicada es proporcional a la señal de error. Es un control de presente. Cuando K_p es grande la respuesta es rápida (peligro de inestabilidad) y el error en régimen permanente es pequeño. Si por el contrario K_p es pequeña la respuesta será más lenta (mayor estabilidad) y el error en régimen permanente será grande. Por tanto una $K_p$grande mejora el régimen permanente y empeora el régimen transitorio. El principal inconveniente es que no tiene en cuenta la duración del error ni la tendencia de este. Por este motivo esta acción de control, en ocasiones, no es suficiente y debe ser complementada con la acción integral o la acción derivada.

- Acción integral: La acción aplicada es proporcional a la integral del error. Es un control de pasado, se preocupa de cuánto tiempo lleva existiendo el error. La ventaja de añadir esta acción es que tiene en cuenta la historia del error y que no descansa hasta que el error en régimen permanente sea 0. El inconveniente es que la respuesta es más lenta y hay peligro de inestabilidad. La K_i mejora el régimen permanente, pero empeora el régimen transitorio.

- Acción derivada: La acción aplicada es proporcional a la derivada del error. Es un control de futuro, tiene en cuenta la tendencia del error. La ventaja de incluir la acción derivada es que considera la tendencia del error y aporta amortiguamiento adicional. El inconveniente es que amplifica los ruidos de alta frecuencia y para cambios bruscos de referencia aplica acciones elevadas. Debido a estos inconvenientes la acción derivada solo se pondrá en caso que sea necesaria. La K_d mejora el régimen transitorio y no afecta al régimen permanente.

A continuación se plantea que tipo de regulador que se necesita en función de las especificaciones de diseño. Por una parte están las especificaciones dinámicas, en este caso se pretende que el tiempo de establecimiento sea el menor posible. Por otra parte están las especificaciones estáticas, en este caso se pretende un error de posición nulo. A parir de estas especificaciones nos planteamos que acciones son necesarias. Para ello se debe plantear las preguntas que aparecen en la figura 31.

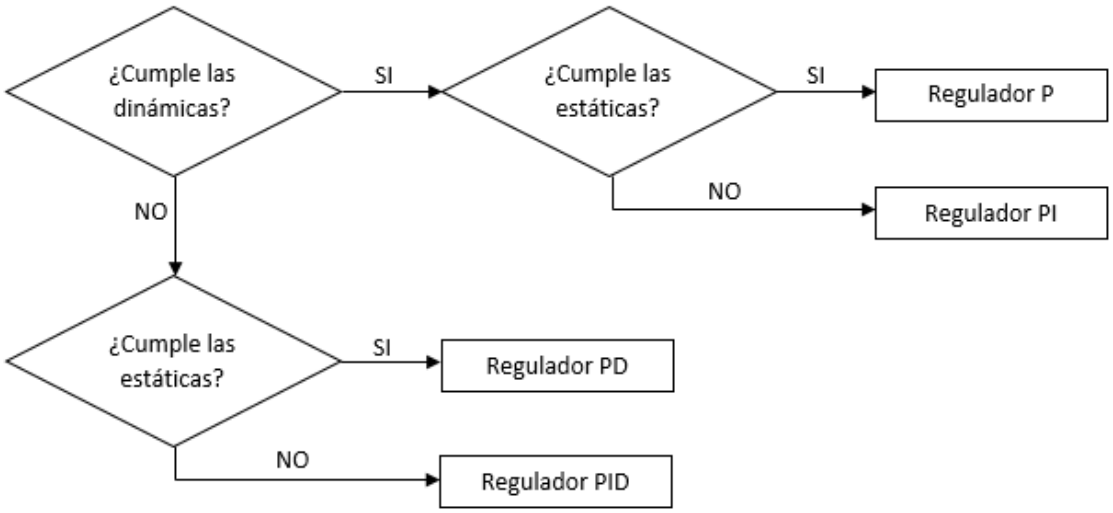

Figura 31. Diseño de reguladores PID.

Por tanto, siguiendo el flujograma anterior se sabe que para cumplir con las especificaciones del sistema se debe diseñar un regulador PI. Aunque un regulador P serviría con el regulador PI el robot seguirá mejor la trayectoria.

Una vez se ha analizado los parámetros que se necesitan para el regulador se debe conocer el valor que hay que asignar a dichos parámetros. Para esto existen diferentes opciones. En este trabajo el valor de los parámetros del regulador se van a obtener a partir del método empírico de Ziegler-Nichols. Este método permite obtener los parámetros de dos maneras: mediante la respuesta ante escalón en bucle abierto, y mediante respuesta oscilatoria mantenida en bucle cerrado. A continuación se muestran los resultados obtenidos:

- Respuesta ante escalón en BA:

Consiste en introducir un escalón en la entrada del sistema (se utiliza la f.d.t continua obtenida en el apartado de identificación de los motores Lego) y se mide el tiempo que tarda en alcanzar la salida el 28,3% (t1) y el 63,2% (t2) del valor final. A partir de estos tiempos se obtienen los parámetros T_p y T_0 , y con estos se calculan los valores del regulador PI mediante las siguientes ecuaciones:

$$K_{pi} = 0{,}9 \cdot \frac{T_p}{KT_0} \qquad (22) \qquad\qquad T_i = \frac{T_0}{0{,}3} \qquad (23)$$

Con este método se obtiene un K_{pi} = 5,81 y T_i= 0,163. En la simulación la respuesta del sistema es buena, pero en el sistema real empeoraba, por lo que sería necesario realizar un ajuste manual.

- Respuesta sostenida en BC:

Para este método se cierra el bucle de control con un regulador proporcional y se aumenta el valor de la ganancia hasta que el sistema oscile con una amplitud constante (comportamiento marginalmente estable). Este valor de ganancia es K_u y el periodo de oscilación es T_c . Para este método se ha utilizado la función de transferencia discretizada para un periodo de muestreo de 0,05 segundo. A partir de los valores de K_u y T_c se obtienen los valores del regulador PI mediante las siguientes ecuaciones:

$$K_{PI} = 0{,}45 \cdot K_u \qquad (24) \qquad\qquad T_i = \frac{T_c}{1.2} \qquad (25)$$

En la siguiente figura se muestra la gráfica de la respuesta sostenida del sistema oscilando con amplitud constante.

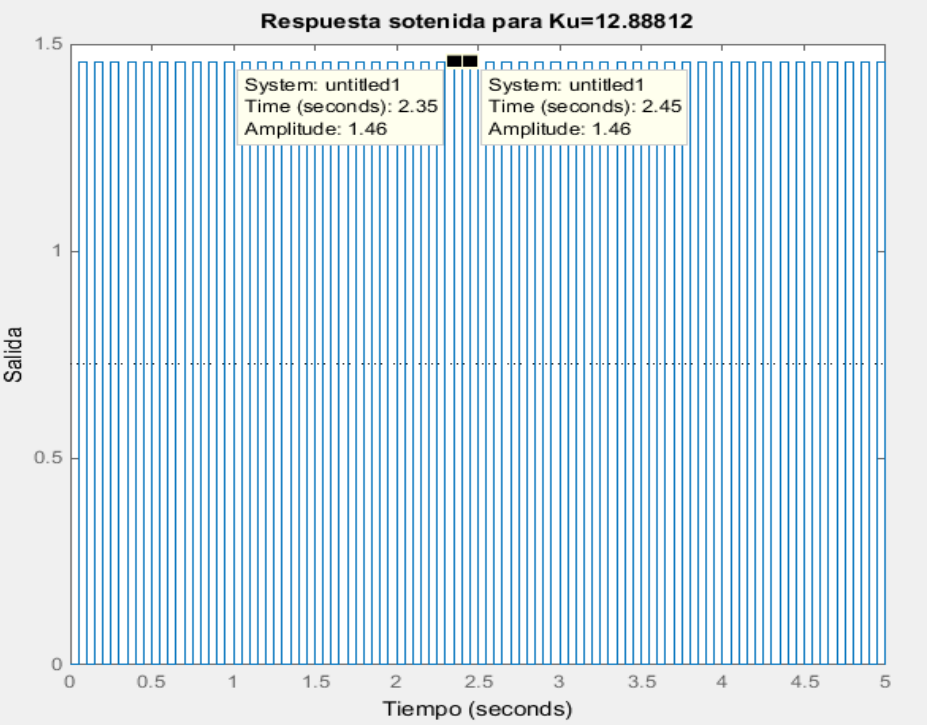

Figura 32. Respuesta marginalmente estable del sistema.

Los valores obtenidos son K_u = 12,888 y T_c =0,1 s. A partir de estos valores se obtiene K_{PI} = 5.7799 y T_i = 0.0834. Estos valores son orientativas y es necesario realizar un ajuste manual. Con este método se han obtenido resultados más precisos que con el método de respuesta ante escalón en BA, por este motivo para el diseño del regulador PI se partirá de los valores orientativos obtenidos por este método.

Una vez tenemos los parámetros de K_{PI} y T_i se obtiene el regulador PI. Este regulador la acción de control presenta la siguiente expresión:

$$u(t) = K_{PI} \cdot \left[e(t) + \frac{1}{T_i} \cdot \int_0^t e(t)\, dt \right] \tag{26}$$

Para poder implementar el algoritmo de control se debe obtener el equivalente discreto. Haciendo una aproximación de la integral como suma de áreas se obtiene la siguiente expresión del controlador:

$$G_{PI}(z) = \frac{q_0 z + q_1}{z - 1} \tag{27}$$

Donde:

$$q_0 = K_{PI} \tag{28}$$

$$q_1 = K_{PI} \cdot \frac{T - T_i}{T_i} \tag{29}$$

Siendo T el periodo de muestreo (0,05 s).

Por tanto la acción de control que se calcula en el algoritmo y que se aplica a los motores es:

$$u(k) = q_0 \cdot e(k) + q_1 \cdot e(k-1) + u(k-1) \tag{30}$$

Como puede observarse la acción de control depende del error actual, del error en el instante anterior y del valor de la acción de control en el instante anterior.

Tras realizar varias pruebas con el sistema real y partiendo de los valores orientativos obtenidos por el método de respuesta sostenida en BC, se obtiene una valor de K_{PI} = 7 y T_i = 0.12. El regulador PI discreto obtenido sustituyendo el valor de los parámetros en la ecuación 27 se muestra en la siguiente ecuación:

$$G_{PI}(z) = \frac{7 \cdot z - 4.08}{z - 1} \tag{31}$$

En la figura 33 y 34 se muestra la respuesta del sistema real.

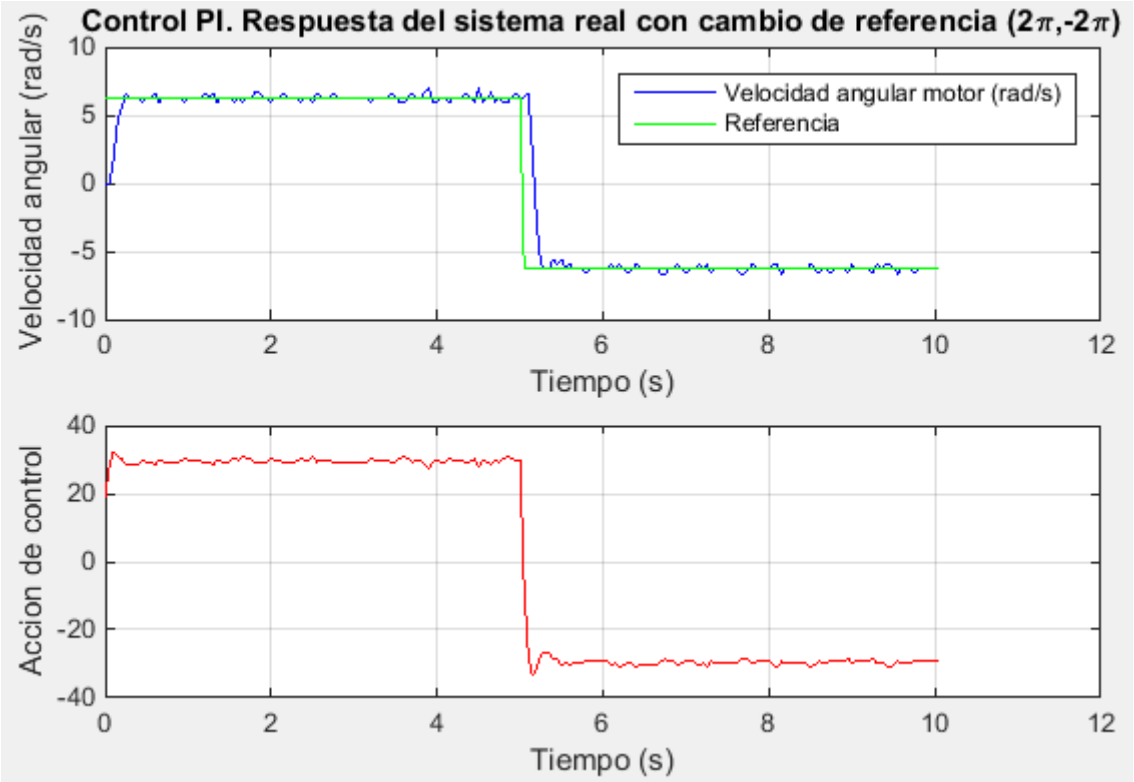

Figura 33. Control PI de velocidad angular de los motores Lego, con 2 cambios de referencia.

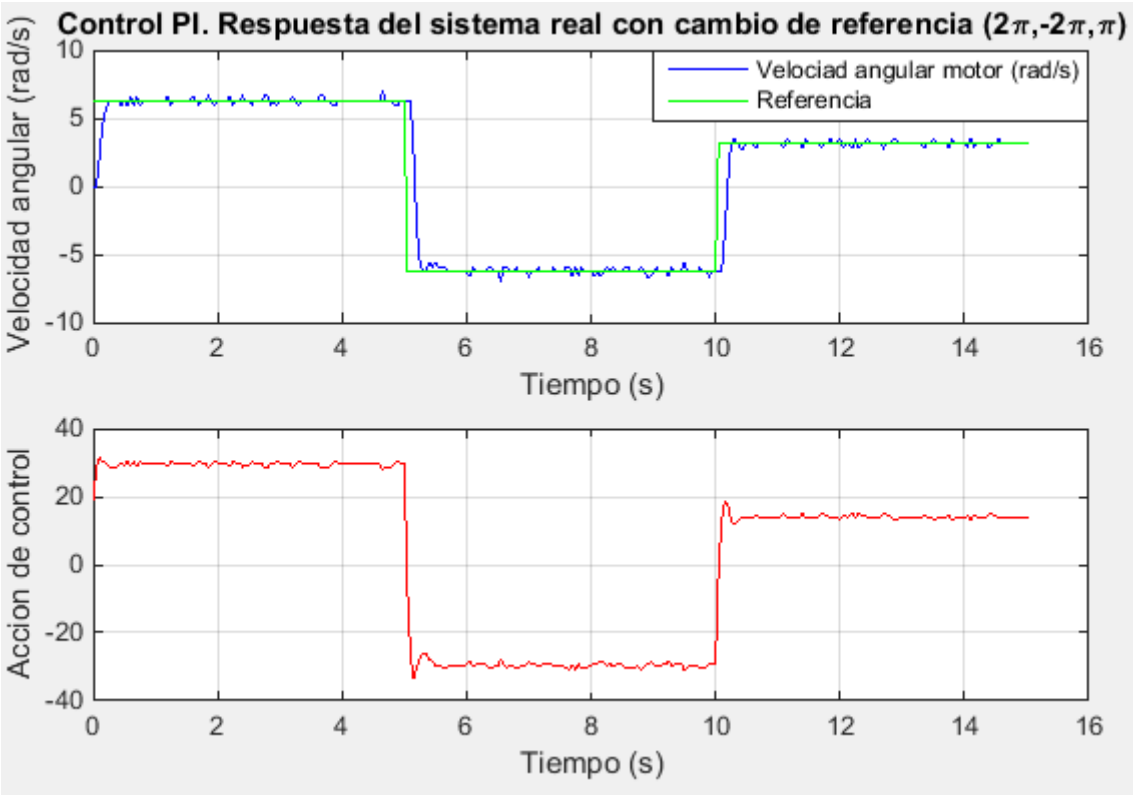

Figura 34. Control PI de velocidad angular de los motores Lego, con 3 cambios de referencia.

Con estos valores, ajustados manualmente a partir de los valores orientativos obtenidos por el método de respuesta sostenida en BC, se obtiene un regulador que sigue la referencia de manera aceptable, con un tiempo de establecimiento no muy grande y una acción de control sin picos bruscos. Con este regulador se realizó una prueba de trayectoria cuyo resultado se comentará más adelante.

Algoritmo de seguimiento de trayectoria

Una vez se ha diseñado el regulador que permite controlar la velocidad de las ruedas el siguiente paso es diseñar el algoritmo de seguimiento de trayectoria.

El primer paso es reiniciar los sensores de las ruedas y el temporizador 1, inicializar todas las variables a sus respectivos valores, calcular los valores del regulador PI a partir de los valores de K_{PI} y T_I y calcular el número de iteraciones que debe realizar el bucle de seguimiento, este número indicará en cuantas partes se ha dividido la trayectoria. Este valor se obtiene al dividir el tiempo final, que indica el tiempo que debe tardar en realizar la trayectoria, entre el periodo de muestreo, que será de 0.05 s.

Una vez se tiene todas las variables es su estado inicial comienza el bucle de control, este bucle continuará funcionando mientras el número de vueltas que ha dado el bucle (indicadas por el contador *i*), sea menor que el número de iteraciones (*niteraciones)*.

Dentro del bucle de control el primer paso es resetear en temporizador 1 y definir la trayectoria. Para definir la trayectoria se definen referencias para cada iteración, estas referencias serán el punto de zanahoria que el robot intentará alcanzar. Cada iteración tiene unas referencias de posición y de velocidad para el eje x e y. A continuación se muestran las ecuaciones utilizadas para definir las referencias para un lado horizontal de un cuadrado de lado 1000 mm.

$$refx = \frac{x_0 + (x_f - x_0) \cdot i}{\frac{niteraciones}{4}} \tag{32}$$

$$refy = 0 \tag{33}$$

$$velxref = \frac{(x_f - x_0)}{\frac{niteraciones}{4}} \tag{34}$$

$$velyref = 0 \tag{35}$$

Donde x_0 es el valor de la posición inicial en el eje x, este valor se define al comienzo del programa. x_f es el valor final de la posición en el eje x , i es un contador de bucle y $niteraciones$ es el número de iteraciones que debe realizar

el bucle de control, como puede observarse esta variable está dividida entre 4 porque el número de iteraciones se divide para realizar los cuatro lados del cuadrado.

A continuación se obtiene el valor de los encoders en radianes, los encoders de Lego expresan la rotación del motor en grados por tanto se debe multiplicar este valor por 0,0174533 (2π rad/360 grados) para obtener el valor de rotación en radianes. Estos valores indica lo que se ha movido el robot y se almacenan en la variable *PosD* y *PosI.* A partir de estos valores de posición y de los valores de posición en el instante anterior *PosD1 y PosI1* se calcula la velocidad angular de las ruedas en rad/s.

$$w_d\left(\frac{rad}{s}\right) = \frac{PosD - PosD1}{T_s} \quad (36) \qquad \omega_i\left(\frac{rad}{s}\right) = \frac{PosI - PosI1}{T_s} \quad (37)$$

Para poder calcular la velocidad angular y lineal del robot se necesita tener la velocidad lineal de cada rueda en mm/s. Para obtenerla basta con multiplicar la velocidad angular de cada rueda por el radio de la rueda (28mm).

$$V_d\left(\frac{mm}{s}\right) = \omega_d \cdot radiorueda \quad (38) \qquad V_i\left(\frac{mm}{s}\right) = \omega_i \cdot radiorueda \quad (39)$$

Ahora se puede obtener la velocidad lineal y angular del robot.

$$\begin{bmatrix} V_{Robot} \\ W_{Robot} \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2b} & -\frac{1}{2b} \end{bmatrix} \cdot \begin{bmatrix} V_d \\ V_i \end{bmatrix} \quad (9)$$

Como ya se explicó en el apartado “3.2.1 Robot diferencial y localización.” A partir de las velocidades del robot se calcula la posición (mm) y la orientación (rad) del robot.

$$\begin{bmatrix} X_{k+1} \\ Y_{k+1} \\ \theta_{k+1} \end{bmatrix} = \begin{bmatrix} X_k + V_{Robot_k} \cdot \cos\theta \cdot T_s \\ Y_k + V_{Robot_k} \cdot \operatorname{sen}\theta \cdot T_s \\ \theta_k + W_{Robot_k} \cdot T_s \end{bmatrix} \quad (11)$$

Como se dijo, el algoritmo de punto descentralizado realiza el control a partir de la posición y la velocidad de un punto que está a una distancia *e* del eje de tracción del robot. La posición de este punto viene definida por la siguiente expresión:

$$\begin{bmatrix} X_p \\ Y_p \end{bmatrix} = \begin{bmatrix} x + e \cdot \cos\theta \\ y + e \cdot sen\ \theta \end{bmatrix} \quad (40)$$

El primer paso para realizar el control de velocidad sobre las ruedas es calcularla expresión de control cinemático del robot (mm/s). Para ello se calcula la velocidad en ese punto:

$$\begin{bmatrix} \dot{X}_p \\ \dot{Y}_p \end{bmatrix} = \begin{bmatrix} \dot{x}_{ref} \\ \dot{y}_{ref} \end{bmatrix} + \begin{bmatrix} K_x & 0 \\ 0 & K_y \end{bmatrix} \cdot \begin{bmatrix} x_{ref} - (x + e \cdot \cos\theta) \\ y_{ref} - (y + e \cdot \text{sen}\,\theta \end{bmatrix} \quad (41)$$

Donde:

$\dot{x}_{ref} = velxref$, velocidad de referencia para el eje x.

$\dot{y}_{ref} = velyref$, velocidad de referencia para el eje y.

K_x y K_y son parámetros de ponderación del control cinemático con punto descentralizado. Se ha asignado un valor de 1 a ambos parámetros.

$x_{ref} = refx$, posición de referencia para el eje x.

$y_{ref} = refy$, posición de referencia para el eje y.

e (51 mm), distancia entre el eje de tracción del robot y el punto a partir del cual se realiza el control.

θ, orientación del robot.

$x\ e\ y$, posición del robot.

A continuación se calcula el modelo cinemático inverso del robot para obtener las velocidades lineales de referencia (mm/s) necesarias para cada rueda.

$$\begin{bmatrix} Vref_d \\ Vref_i \end{bmatrix} = \frac{1}{e} \cdot \begin{bmatrix} e \cdot \cos\theta - b \cdot sen\ \theta & e \cdot sen\ \theta + b \cdot \cos\theta \\ e \cdot \cos\theta + b \cdot sen\ \theta & e \cdot sen\ \theta - b \cdot \cos\theta \end{bmatrix} \cdot \begin{bmatrix} \dot{X}_p \\ \dot{Y}_p \end{bmatrix} \quad (42)$$

Donde b (51 mm) es la mitad de la distancia que hay entre las ruedas del robot.

A partir de los valores de las velocidades lineales de referencia obtenidos anteriormente se calcula la velocidad angular de referencia (rad/s) de cada rueda para el control dinámico.

$$\omega ref_d = \frac{Vref_d}{radiorueda} \quad (43) \qquad \omega ref_i = \frac{V_{refi}}{radiorueda} \quad (44)$$

El siguiente paso es calcular el error entre la velocidad angular de las ruedas y la velocidad angular de referencia de cada rueda.

$$error_{w_d} = \omega ref_d - w_d \qquad error_{w_i} = \omega ref_i - w_i \quad (45)$$

A continuación se calcula la acción de control a aplicar a cada rueda. La expresión de la acción de control para un regulador discreto se expresa en la ecuación 30. Concretando para este caso se obtiene:

$$u_{d(k)} = q_0 \cdot error_{w_d}(k) + q_1 \cdot error_{w_d}(k-1) + u_d(k-1) \quad (46)$$

$$u_i(k) = q_0 \cdot error_{w_i}(k) + q_1 \cdot error\_w_i(k-1) + u\_i(k-1) \quad (47)$$

Una vez calculada la acción de control se satura en caso de que exceda el máximo permitido (±100 en el caso de los motores de Lego). Y se aplica a los motores.

Por último se actualizan los valores *PosD1 = PosD*, *PosI1=PosI*, *error_w_d1= error_w_i* y *error_w_i1= error_w_i.* Una vez actualizado calculamos el tiempo que ha tardado la iteración, para ello se resta el valor actual del temporizador 1 y el valor que tenía al comenzar el bucle. Para que todas las iteraciones tengan la misma duración (0,05 segundos) se añade una espera. El tiempo de espera se calcula restando a 0,05 segundos, que es el periodo de muestreo, el tiempo que ha tardado en realizar el bucle la iteración (calculado mediante el temporizador1).

Si el bucle de control ha realizado menos iteraciones que el número de iteraciones necesario para la realización de la trayectoria completa, el bucle comenzará de nuevo. En caso contrario se termina el bucle de control.

Resultados

Para probar el algoritmo de seguimiento de trayectoria diseñado se realizó una prueba de seguimiento de trayectoria. En esta prueba el robot debe realizar un cuadrado de lado 1000 mm. Como el robot no mantiene el equilibrio por si solo la prueba se realizó con el robot en el aire. En la siguiente figura se muestra en azul la trayectoria que debe seguir y en rojo la ruta seguida por el robot.

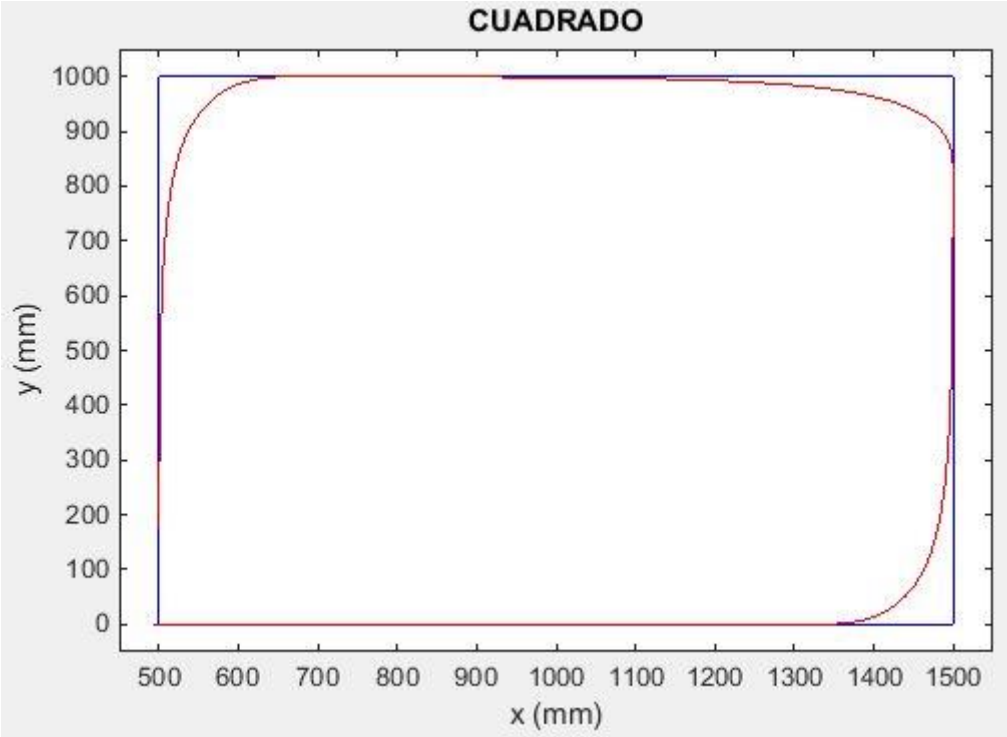

Figura 35. Posición del robot realizando trayectoria cerrada con tiempo de muestreo 50 ms.

Como puede observarse el robot realiza la trayectoria con bastante exactitud. El problema surge cuando se intenta juntar el algoritmo de seguimiento de trayectoria con el control de estabilidad del robot.

Para la aplicación diseñada el control de equilibrio debe tener prioridad ante cualquier otra función, ya que el robot debe mantener el equilibrio en todo momento. Por este motivo el control de estabilidad es más rápido que el control de trayectoria (5 ms frente a 50 ms) y la acción de control aplicada a las ruedas es la acción de control calculada para mantener el equilibrio más la acción de control calculada para seguir la trayectoria ponderada.

A pesar de ser más lento el algoritmo de seguimiento diseñado es demasiado exigente, ya que cada 50 ms cambia la referencia de posición y orientación sin importar la localización actual del robot.

Las ruedas del robot no pueden seguir la velocidad de referencia calculada por el algoritmo de seguimiento, debido a que las ruedas del robot se encargan, además de realizar la trayectoria, de mantener el equilibrio. Por ello no se mueve con velocidad constante hacia el objetivo, sino que en ocasiones se detiene parcialmente o incluso retrocede para mantenerse en posición estable. Por este motivo las velocidades de las ruedas no alcanzan la velocidad de referencia calculadas por el algoritmo de seguimiento. Además como el robot puede estar alejado de la referencia, que además continua aumentando cada 50 ms, las nuevas velocidades de referencia calculadas serán mayores para alcanzar el punto de referencia. Todo esto hace que el error entre la velocidad angular de referencia y la velocidad angular real de las ruedas sea cada vez mayor, por lo

que la acción de control aumenta hasta que es tan grande que el robot se desequilibra.

Para intentar solucionar este problema se aumentó el periodo de muestreo del control de trayectoria hasta 0.5 s. Se obtuvo un nuevo regulador PI utilizando el método de respuesta sostenida en BC, los valores de los parámetros calculados son K_{PI} = 3.5 y T_i = 0.8. Sustituyendo los parámetros en la fórmula del regulador PI discreto se obtiene el siguiente controlador:

$$G_{PI}(z) = \frac{3{,}5 \cdot z - 1{,}31}{z - 1} \qquad (48)$$

En la figura 36 muestra la respuesta del sistema real.

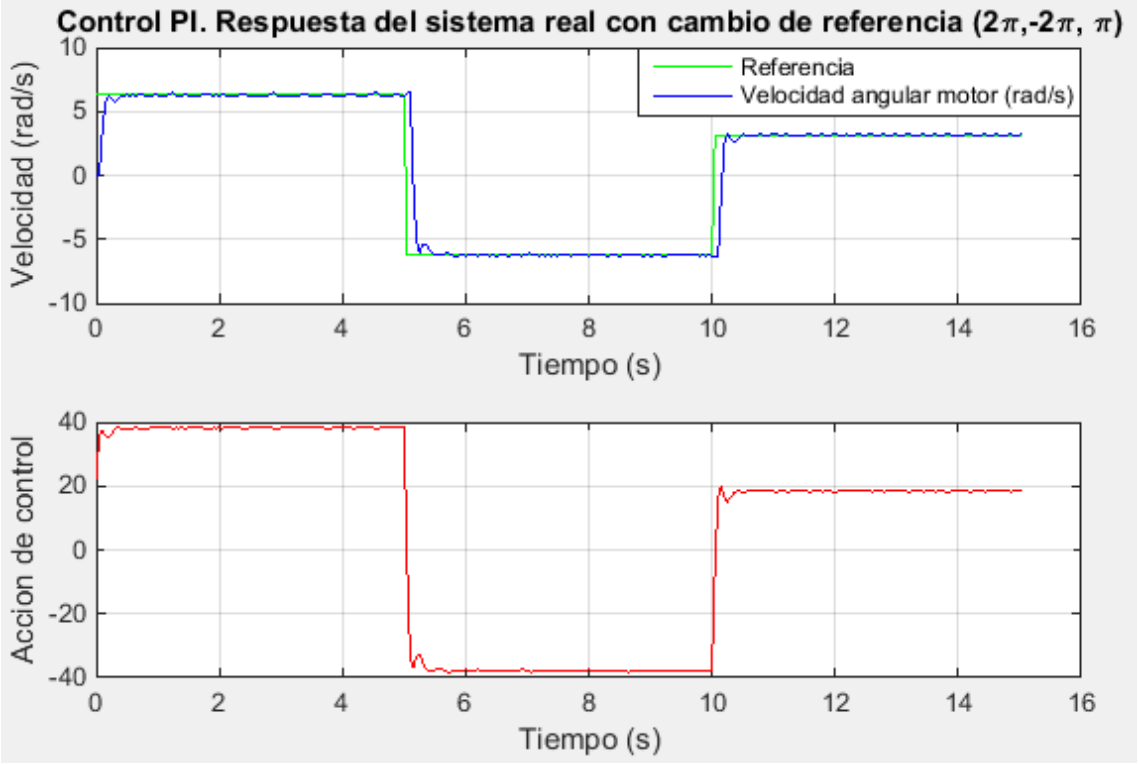

Figura 36. Control PI de velocidad angular de los motores Lego, con 3 cambios de referencia.

Con este nuevo regulador se repitió la prueba de seguimiento de una trayectoria cuadrada de 1000 mm de lado. Al igual que en la prueba anterior, esta se realizó con el robot en el aire. En la siguiente figura se muestra en azul la trayectoria que debe seguir y en rojo la ruta seguida por el robot.

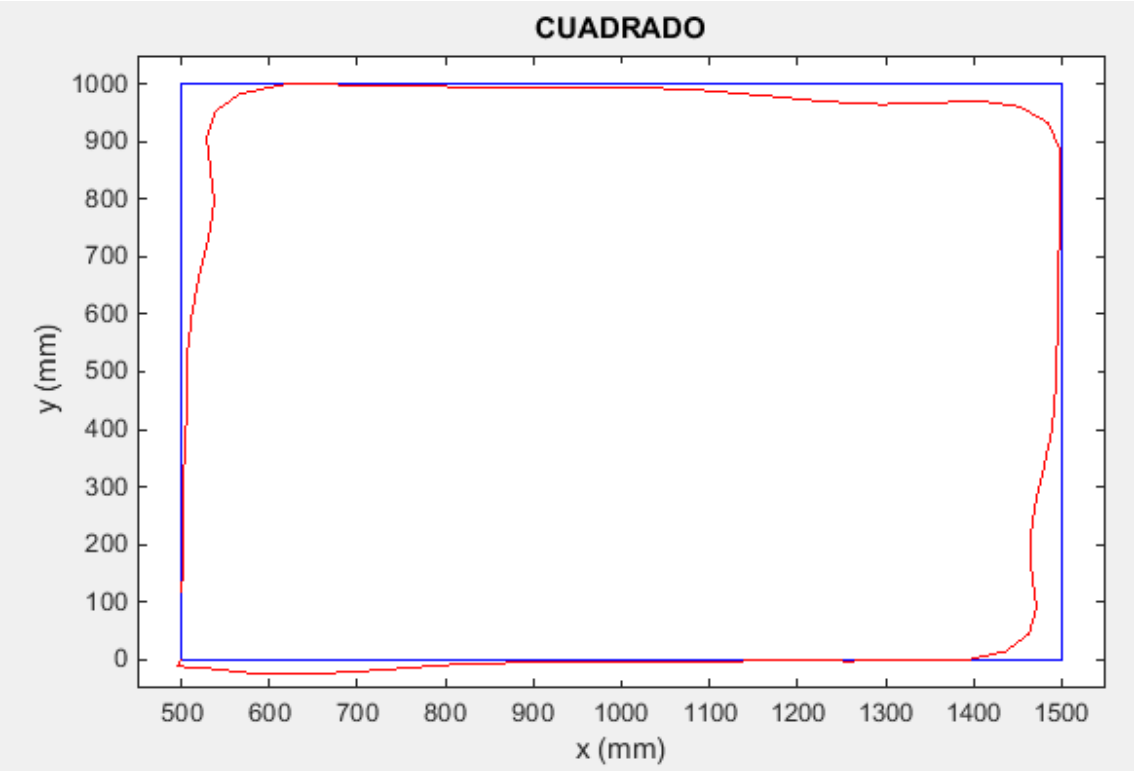

Figura 37.Posición del robot realizando trayectoria cerrada con tiempo de muestreo 500 ms.

Como se observa en la figura anterior al aumentar el periodo de muestreo empeora la exactitud con la que el robot realiza la trayectoria. Esto es lógico ya que el control es más lento.

Además de empeorar el seguimiento de trayectoria, hacer el control más lento no soluciona el problema que surge a la hora de juntar el control de trayectoria con el control de estabilidad, ya que aparecen los mismos problemas que para el caso analizado anteriormente.

En definitiva el algoritmo de punto descentralizado no es una buena opción para la aplicación que se pretende desarrollar. Este algoritmo sería una buena opción si se tratase de un robot estable, ya que los motores de las ruedas solo se encargan de seguir la trayectoria por lo que pueden seguir las velocidades de referencia calculadas por el algoritmo. En cambio, en la aplicación diseñada los motores de las ruedas se encargaran, además de seguir la trayectoria, de mantener el equilibrio. Por este motivo, como ya se dijo anteriormente, el robot no siempre puede seguir las velocidades de referencia calculadas por el algoritmo de seguimiento, sino qua a veces se para o retrocede para mantener el equilibrio. Este hecho provoca que el robot no sigua la trayectoria, se aumente la acción de control y finalmente se desequilibre. Por estos motivos se descartó utilizar este algoritmo de seguimiento.

3.2.2.2 Algoritmo de persecución pura o pure pursuit.

Para la aplicación diseñada el bucle de estabilidad manda sobre el resto de funciones. Por este motivo el algoritmo de seguimiento de trayectoria de punto descentralizado no era una buena opción. Por ello se buscó una solución que permitiera al robot realizar trayectorias previamente conocidas mientras mantenía el equilibrio. Esto se logró mediante el diseño de algoritmo de seguimiento de trayectoria por el método de persecución pura. A continuación se explican algunas diferencias entre el algoritmo de persecución pura y el algoritmo de punto descentralizado.

El algoritmo de persecución pura se encarga de alcanzar un punto objetivo mediante una trayectoria suave. Como se trata de trayectorias conocidas previamente y lineales la forma de definir las referencias en el algoritmo de punto descentralizado no es la más adecuada, esta forma es útil si no se conoce la trayectoria completa. En el nuevo algoritmo las trayectorias se definirán con pocos puntos que sirvan para definirlas. Además el cambio de referencia se hará cuando el robot este suficientemente cerca del punto objetivo y no cada cierto tiempo como se hacía en el algoritmo analizado anteriormente. También para la aplicación diseñada se necesita un algoritmo que, a diferencia del algoritmo de punto descentralizado, no dependa de un tiempo para realizar la trayectoria, en este caso dependía de *tfinal,* sino que la realice a la velocidad necesaria para poder mantener el equilibrio.

Breve explicación teórica y funcionamiento del algoritmo de persecución pura.

El funcionamiento del algoritmo de persecución pura diseñado es muy sencillo. Este consiste en pasarle un punto de referencia al robot que se convierte en el objetivo a alcanzar El algoritmo calcula la velocidad de referencia necesaria para cada rueda para que el robot alcance dicho objetivo. La velocidad calculada depende de la distancia a la que se encuentre el robot del objetivo. Una vez se encuentre lo suficientemente cerca del objetivo se cambia el valor de la referencia.

Mediante la localización del robot por odometría obtenemos su posición y su orientación (x_m, y_m y θ_m). También se conocen las coordenadas del punto objetivo (x_{ob}, y_{ob}). Con estos datos se puede calcular la distancia que tiene que recorrer el robot en el eje *x* y en el eje *y.*

$$\Delta x = x_{ob} - x_m \quad (49)$$

$$\Delta y = y_{ob} - y_m \quad (50)$$

A partir de estos valores se puede calcular la distancia del robot al punto objetivo, ρ:

$$\rho = \sqrt{\Delta x^2 + \Delta y^2} \tag{51}$$

Y también el ángulo α , que es el ángulo que debe girar el robot para orientarse hacia el punto objetivo.

$$\alpha = -\theta + atan2(\Delta x, \Delta y) \tag{52}$$

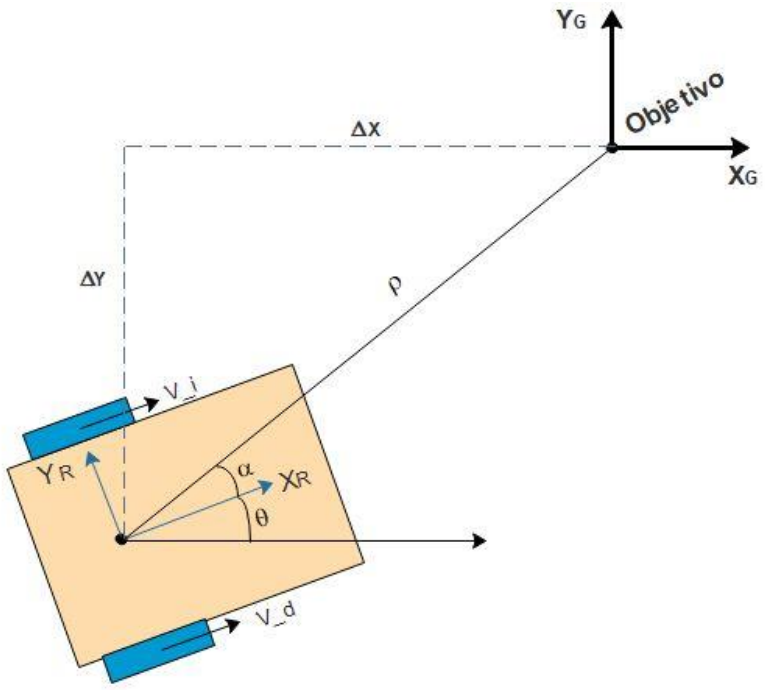

Figura 38. Persecución pura.

A partir de los valores de ρ y α se calcula la velocidad lineal y angular de control:

$$V_c = K_\rho \cdot \rho \tag{53}$$

$$W_c = K_\alpha \cdot \alpha \tag{54}$$

Y a partir de estos valores se puede calcular la velocidad lineal de referencia para cada rueda:

$$\begin{bmatrix} V_{ref\_d} \\ V_{ref\_i} \end{bmatrix} = \begin{bmatrix} 1 & \frac{b}{2} \\ 1 & -\frac{b}{2} \end{bmatrix} \cdot \begin{bmatrix} V_c \\ W_c \end{bmatrix} \tag{55}$$

Donde *b* es la distancia entre las ruedas del robot.

En la siguiente figura se muestra su diagrama de bloques.

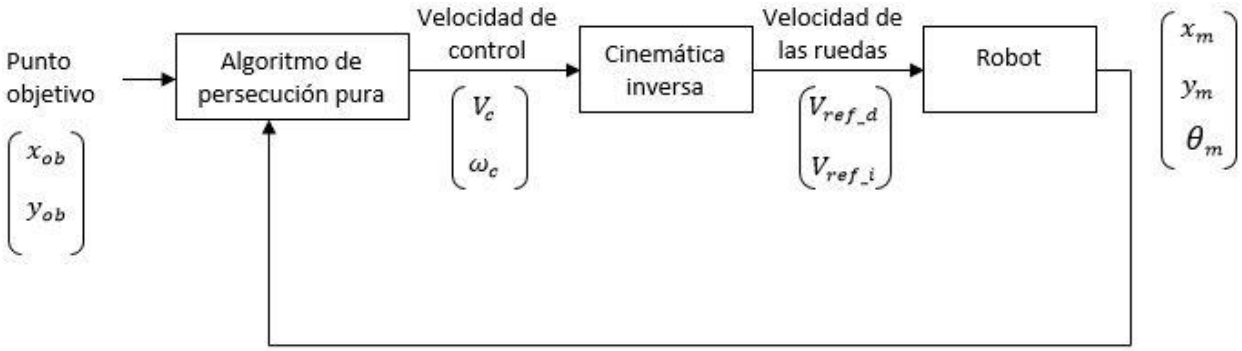

Figura 39. Diagrama bloques algoritmo persecución pura.

Como ya se dijo el control de trayectoria debe realizarse teniendo en cuenta que mantener el equilibrio es la tarea principal. Por este motivo el control diseñado es lento, tiene un periodo de muestreo de 0,5 segundo. Además para simplificar el control en el algoritmo diseñado no importa la orientación con la que el robot alcance el punto objetivo.

El algoritmo se ha diseñado para que el robot pueda seguir la trayectoria mientras mantiene el equilibrio, a continuación se explica cómo se planteó el diseño.

Al algoritmo de seguimiento se le pasan pocas referencias. El robot se dirige hacia el punto objetivo con cierta velocidad. Y cada periodo de muestreo el robot se pregunta ¿dónde estoy? y ¿a dónde debo ir? La odometría responde a la primera pregunta y el algoritmo de persecución pura calcula la distancia al punto objetivo ρ y el ángulo α que debe girar el robot para dirigirse hacia el punto objetivo y las velocidades de referencia para alcanzarlo.

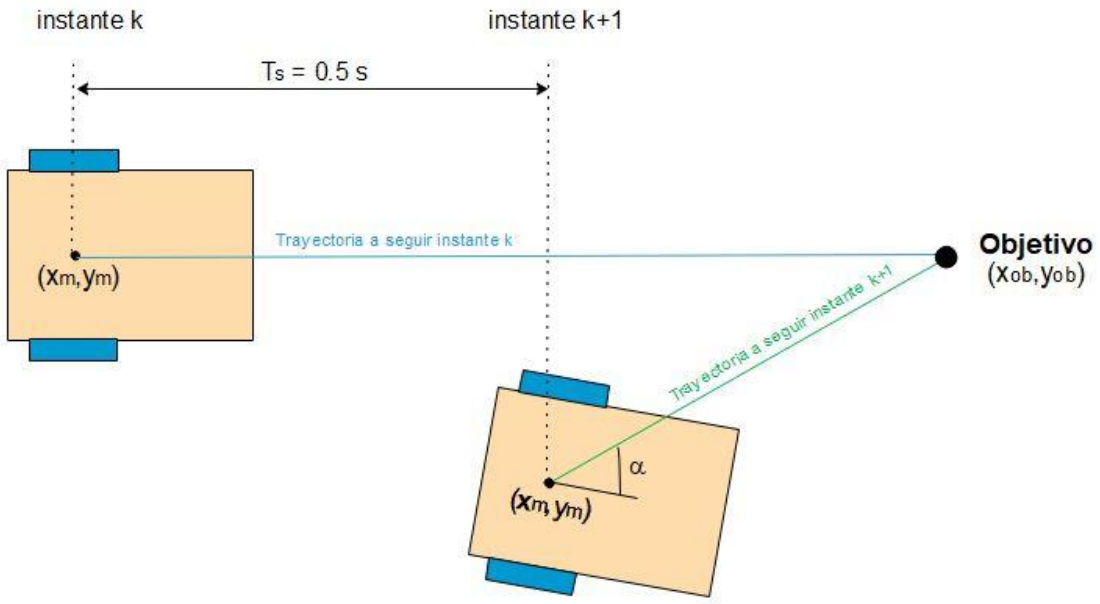

Figura 40. Explicación del funcionamiento del algoritmo diseñado.

Como se vio anteriormente las velocidades de referencia se calculan a partir de las velocidades lineales y angulares de control calculadas (ecuación 55) y estas a su vez dependen de la distancia que separa al robot del punto objetivo (ρ) y del ángulo que debe girar el robot para orientarse hacia dicho punto (α).

Para simplificar el funcionamiento se definen dos velocidades lineales de control (V_c). Una velocidad de control constante cuando el robot se encuentra alejado del punto objetivo, para ir rápido. Y cuando la distancia es lo suficientemente pequeña, la velocidad de control se calcula mediante la ecuación 53, de esta forma la velocidad irá disminuyendo conforme el robot se aproxima al punto objetivo. Este cambio en la forma de calcular V_c se produce a partir de cierto valor de ρ.

Por otro lado el robot durante la realización de la trayectoria va corrigiendo su orientación para dirigirse hacia el objetivo, siendo α el ángulo que debe girar. La velocidad de control angular (W_c) es la encargada de corregir la orientación del robot. En este caso también se definen dos velocidades angulares de control (W_c). Una velocidad para cuando el valor del ángulo que debe girar el robot es pequeño, valores de α pequeños, que se calcula con la ecuación 54 y que se usa para hacer pequeñas correcciones en la orientación mientras el robot avanza hacia su objetivo. Y una velocidad de control constante cuando el ángulo α que tiene que girar el robot es muy grande. En este caso, en el que el ángulo que debe girar el robot sea grande, la velocidad lineal de control (V_c) se hace 0 y el valor de la velocidad angular de control (W_c) tiene un valor constante. De esta forma cuando el robot debe realizar giros bruscos solo gira sin avanzar hasta que haya corregido la orientación, una vez corregida el valor de V_c y W_c dependerá del valor de ρ y α respectivamente.

Las velocidades de control no son muy grandes y además tampoco importa mucho si el robot alcanza dichas velocidades, ya que el bucle de estabilidad manda. Por este motivo para el control de velocidad bastará con un regulador con acción proporcional.

Una vez explicado el funcionamiento en el siguiente apartado se explica cómo se implementó el algoritmo el LabView.

Algoritmo de persecución pura

Para la aplicación diseñada se han definido 3 trayectorias diferentes que el robot deberá realizar. Las trayectorias definidas son un cuadrado de 1000 mm de lado, un hexágono de 500 mm de lado y un triángulo equilátero de 1000 mm de lado. En la aplicación la trayectoria que se desea realizar se le comunicará al robot mediante un mensaje bluetooth, pero para realizar el diseño del algoritmo la trayectoria a seguir se define de forma manual.

En la tabla 6 se muestran las variables utilizadas para el algoritmo de persecución pura.

Nombre de la variable	Tipo de dato	Explicación
Ts	DBL	Periodo de muestreo (0.5 s).
refx	DBL	Referencia de posición en el eje x en mm.
refy	DBL	Referencia de posición en el eje y en mm.
X	DBL	Posición del robot eje x en mm.
y	DBL	Posición del robot eje y en mm.
theta	DBL	Orientación del robot en rad.
Varx	DBL	Distancia que separa al robot del objetivo en el eje x. Δx expresado en mm.
vary	DBL	Distancia que separa al robot del objetivo en el eje y. Δy expresado en mm.
rho	DBL	Distancia entre el robot y el punto objetivo. ρ expresada en mm.
alpha	DBL	Ángulo que debe girar el robot para orientarse hacia el punto objetivo. α expresado en rad.
Vc	DBL	Velocidad lineal de control en mm/s.
Wc	DBL	Velocidad angular de control en rad/s.
radiorueda	DBL	Radio de la rueda del robot 28 mm.
b	DBL	La mitad de la distancia entre las ruedas 51 mm.
Vref_d	DBL	Velocidad de referencia calculada para la rueda derecha en mm/s.
Vref_i	DBL	Velocidad de referencia calculada para la rueda izquierda en mm/s.
Wref_d	DBL	Velocidad de referencia calculada para la rueda derecha en rad/s.
Wref_i	DBL	Velocidad de referencia calculada para la rueda izquierda en rad/s.

PosD	DBL	Posición de la rueda derecha en rad en el instante k.
PosI	DBL	Posición de la rueda izquierda en rad en el instante k.
PosD1	DBL	Posición de la rueda derecha en rad en el instante k-1.
PosI1	DBL	Posición de la rueda izquierda en rad en el instante k-1.
CamRef	I16	Se encarga de cambiar la referencia cuando el robot está cerca del objetivo.

Tabla 6. Variables del algoritmo de seguimiento por método de persecución pura.

El primer paso es inicializar todas las variables y reiniciar el temporizador 3. A continuación se entra en el bucle de control, este continuará hasta que se haya completado la trayectoria. La salida del bucle está gestionada mediante la variable *CamRef* esta variable aumenta su valor en 1 cada vez que el robot alcanza un punto de referencia. Cada trayectoria tiene un número definido de referencia, por ejemplo para realizar el cuadrado hay 4 referencias, en este caso el bucle continuará mientras que *CamRef* sea menor que 5. El hexágono tiene 6 referencias, en este caso el bucle continuará mientras *CamRef* sea menor que 7. Y el triángulo tiene 3 referencias, en este caso el bucle continuará mientras *CamRef* sea menor que 4.

El primer paso dentro del bucle de control es definir el punto que debe alcanzar el robot *refx* y *refy.* Una vez se conoce la posición que se desea alcanzar se leen los encoders de las ruedas para conocer la posición de los motores, los valores de los sensores expresan la posición en grados por lo que se multiplica por 0,0174533 $(2\pi/360)$ para obtener la posición en radianes, estos valores se guardan en las variables *PosD* y *PosI.*

A continuación se calcula la posición y la orientación del robot, estos valores se calculan directamente con el valor de posición en el instante actual (*PosD, PosI)* y con los valores de posición en el instante anterior (*PosD1, PosI1)*. Esta opción es más adecuada que la de calcular la posición y la orientación a partir de las velocidades del robot, tal y como se muestra en la ecuación 11. Ya que haciendo el cálculo de esta forma tienes que derivar la posición para obtener la velocidad angular y lineal del robot y posteriormente integrarla para obtener la nueva posición, de esta forma se introducen 2 errores de ruido. Sin embargo el método utilizado usa los valores obtenidos directamente de los sensores de las ruedas para calcular la localización del robot, evitando así introducir errores innecesarios. Las ecuaciones para obtener la posición y la orientación del robot por este método se muestran a continuación:

$$\begin{bmatrix} X(mm) \\ Y(mm) \\ \theta(rad) \end{bmatrix} = \begin{bmatrix} X \\ Y \\ \theta \end{bmatrix} + \begin{bmatrix} \frac{\Delta PD + \Delta PI}{2} \cdot radiorueda \cdot \cos\theta \\ \frac{\Delta PD + \Delta PI}{2} \cdot radiorueda \cdot \text{sen}\,\theta \\ \frac{\Delta PD - \Delta PI}{2 \cdot b} \cdot radiorueda \end{bmatrix} \quad (56)$$

Donde ΔPD y es la diferencia entre la posición actual y la posición en el instante anterior de la rueda derecha. ΔPI es la diferencia entre la posición actual y la posición en el instante anterior de la rueda izquierda.

$$\Delta PD(rad) = (PD_k(grados) - PD_{k-1}(grados)) \cdot \frac{2 \cdot \pi\ (rad)}{360(grados)} \quad (57)$$

$$\Delta PDI(rad) = (PI_k(grados) - PI_{k-1}(grados)) \cdot \frac{2 \cdot \pi\ (rad)}{360(grados)} \quad (58)$$

Es decir, ΔPD y ΔPI indican el espacio que ha recorrido cada rueda en cada iteración. $2b$ es la distancia entre las dos ruedas y $radiorueda$ es el radio de la rueda.

Una vez se conoce la posición del robot se calcula la distancia a la que se encuentra del objetivo (Δx y Δy) mediante la ecuación 49 y 50. A partir de estos valores se obtiene el valor de *rho* (ρ) y *alpha* (α). El valor de *rho* se obtiene directamente mediante la ecuación 51. El valor de *alpha* se obtiene mediante la ecuación 52, pero se debe tener en cuenta que en ocasiones la función $atan2(\Delta x, \Delta y)$, dependiendo del signo de Δx y Δy, toma el ángulo negativo en vez del positivo, si este valor es negativo el valor de *alpha* se hará muy grande y el robot piensa que está muy alejado de la orientación deseada cuando en realidad esto no es así. Para solucionarlo se pone la condición de que si el ángulo calculado por la función $atan2$ es negativo se calcula el valor positivo del ángulo sumándole al valor obtenido 2π. De esta forma el robot siempre obtiene el valor de *alpha* correcto.

Una vez se han obtenido los valores se calcula las velocidades de control. Para ello se analiza el valor de la variable *alpha,* si el valor de esta es mayor de 0,5 rad el robot solo gira y se asignan los valores $V_c = 0$ y $W_c = 0{,}65\ rad/s$.

Si por el contrario *alpha* es menor de 0,5 rad el valor de la velocidad angular de control viene definido por la expresión $W_c = 1 \cdot \alpha$, y el valor de V_c depende de la trayectoria que se esté realizando y del valor de *rho.* Como se dijo existen dos velocidades, una constante cuando el robot está alejado del objetivo ($\rho > 300$, para el cuadrado y triángulo y $\rho > 100$, para el hexágono), en este caso $V_c = 150\,\frac{mm}{s}$. Y otra cuando el robot está cerca del objetivo, en este caso el valor de la velocidad de control vendrá determinado por la expresión $V_c = 0{,}5 \cdot \rho$.

Una vez obtenidas las velocidades de control se calcula la velocidad de referencia para cada rueda mediante la ecuación 55. El algoritmo fue diseñado teniendo en cuenta que se iba a juntar con el control de estabilidad, por este motivo el control de velocidad se realiza dentro del bucle de equilibrio, esto se explicará de forma detallada en el apartado "*3.4 Diseño del sistema de control completo."* donde se explica el funcionamiento conjunto.

Para finalizar se actualizan los valores de posición *PosD1=PosD* y *PosI1=PosI.* Y se analiza el valor de *rho* para ver si el robot está lo suficientemente cerca del objetivo, en cuyo caso se aumentará el valor de *CamRef* y esto provocará que se cambie el valor de la referencia o, si ya se ha completado la trayectoria, que se termine el bucle de control. Si por el contrario el robot aún se encuentra alejado del objetivo el valor de *CamRef* no varía por lo que el robot mantiene el punto objetivo. La distancia para la que se considera que el robot está cerca del punto meta es $\rho < 20$.

Todos los valores de las variables comentados en este apartado se obtuvieron tras la realización de múltiples pruebas con diferentes valores. Determinando que los valores elegidos son los más adecuados para la realización de la trayectoria por parte del robot.

A continuación se resume el funcionamiento del bucle de control del algoritmo de persecución pura mediante un flujograma.

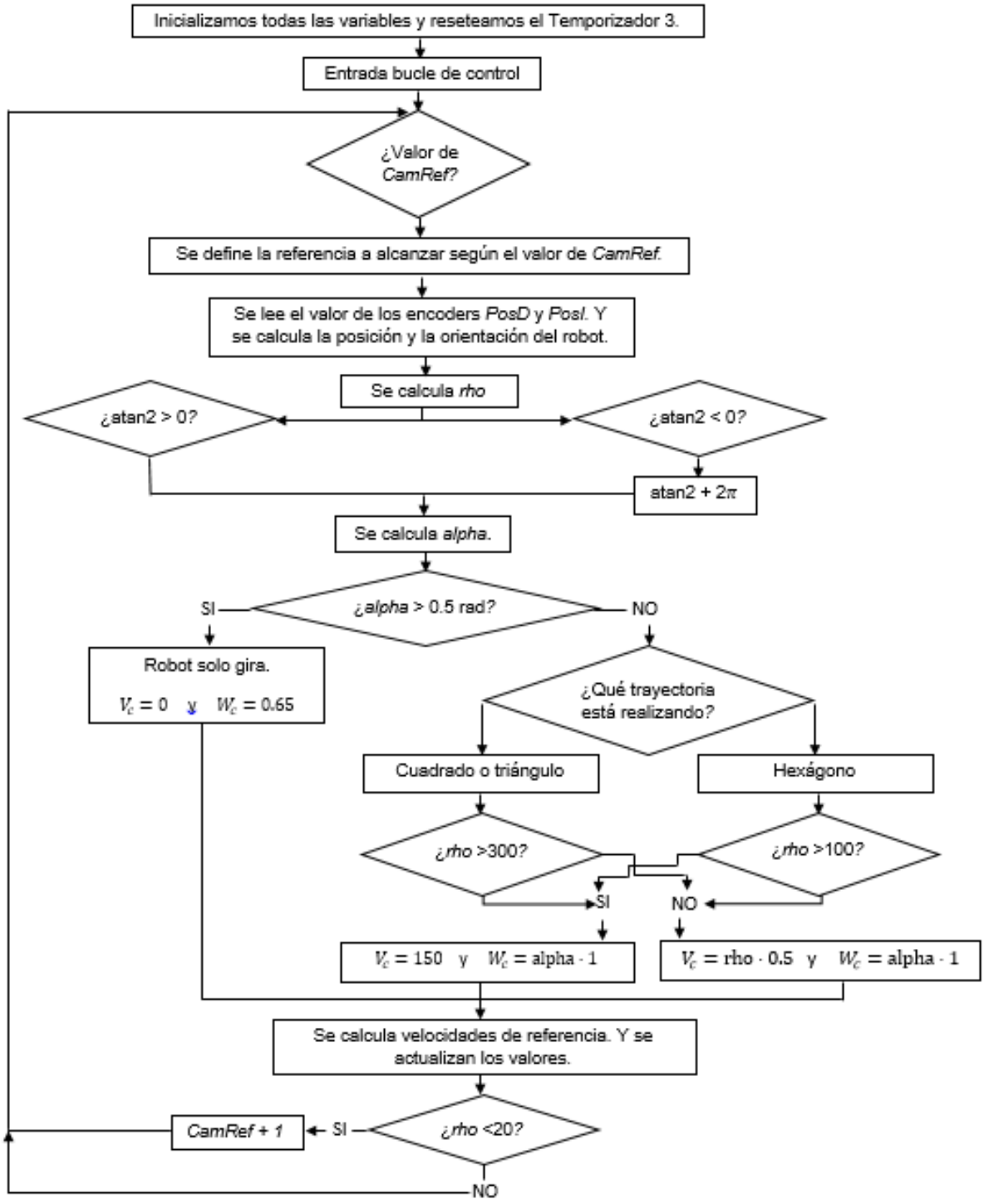

Figura 41. Explicación del funcionamiento del algoritmo persecución pura.

Como se comentó al inicio del apartado en la aplicación diseñada la trayectoria a realizar se envía al robot mediante un mensaje desde el PC al Mailbox2 con la trayectoria a realizar. Para ello al inicio del algoritmo se añadirá un una espera hasta que reciba el mensaje con la figura a realizar. Según el mensaje recibido se hace verdadera una de las nuevas variables introducidas cuya función es hacer saber al robot la trayectoria que va a seguir, estas son: "*cuadrado",*

"triangulo" o *"hexágono".* El valor de estas variables son los analizados para saber qué conjunto de referencias se va a pasar al robot y también para saber el valor de ρ para el que se hace el cambio en la forma de cálculo de V_c (recordamos que este valor era $\rho > 300$, para el cuadrado y triángulo y $\rho > 100$, para el hexágono).

3.3 Control de dirección.

A parte de mantener el equilibrio y de seguir trayectorias previamente definidas la aplicación diseñada tiene otro modo de funcionamiento que se ha llamado "Modo remoto". Este modo permite mover el robot en el espacio hacia diferentes direcciones de forma remota desde el PC.

Para este modo de funcionamiento se han definido 8 direcciones diferentes:

1. Avanzar.
2. Retroceder.
3. Girar derecha.
4. Girar izquierda.
5. Avanzar y girar a la derecha.
6. Avanzar y girar a la izquierda.
7. Retroceder y girar a la derecha.
8. Retroceder y girar a la izquierda.

El robot recibe la información de la dirección en la que se debe mover desde el PC. Para ello se establece una conexión entre el PC y el ladrillo EV3. El método elegido para establecer esta comunicación es mediante conexión bluetooth.

A continuación se explica la comunicación bluetooth, el funcionamiento de este modo y el algoritmo diseñado.

3.3.1 Comunicación bluetooth.

Para la aplicación diseñada se requiere comunicar el ladrillo EV3 con el PC de manera inalámbrica. Para ello el EV3 dispone de dos opciones: Bluetooth y WIFI.

El método elegido es la conexión bluetooth debido a que la conexión se realiza de manera sencilla, el WIFI solo permite hacer determinadas operaciones y además LabView dispone de diferentes bloques de funciones que permiten mandar y recibir mensajes por vía bluetooth, esperar hasta que se reciba un mensaje por bluetooth, etc. Para ello la EV3 dispone de 10 Mailbox donde podemos enviar (escribir) y recibir (leer) mensajes mediante bluetooth. Estos bloques serán de mucha utilidad para la implementación del algoritmo de control, y su gestión y utilización son muy sencillos.

Para establecer la comunicación entre el ladrillo EV3 y el PC se deben realizar los siguientes pasos:

1. Activar el bluetooth en el ladrillo EV3. Para ello debes desplazarte por el menú del bloque hasta la pestaña de ajustes y seleccionar la casilla “Bluetooth”.

Figura 42. Activación del bluetooth en la EV3.

Una vez seleccionado se abrirá una nueva pestaña llamada “Connections” y debes seleccionar las opciones Visibility” y “Bluetooth” y asegurarte que la casilla “iPhone/iPad/iPod” este deseleccionada.

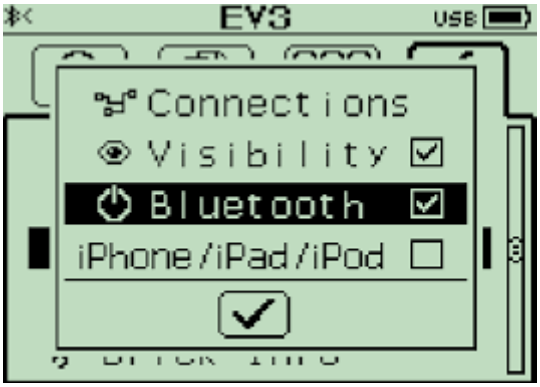

Figura 43. Ajustes para activar el bluetooth en la EV3.

2. Se busca el dispositivo desde el PC y se conecta.

3. La EV3 te preguntará si quieres conectarte a ese dispositivo y una vez aceptas debes introducir una contraseña, se utilizó “1234”.

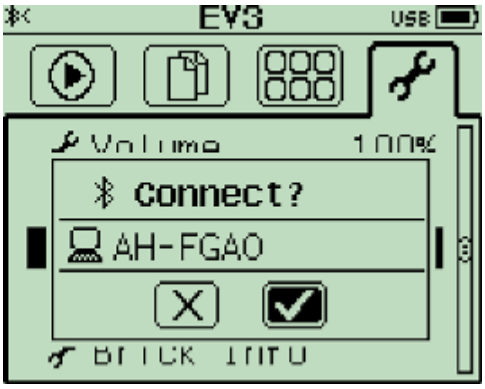

Figura 44. Emparejamiento con PC.

Figura 45. Contraseña para conectarse al PC.

4. Por último deberás introducir la misma contraseña en el PC y la conexión entre el PC y la EV3 ya está establecida.

3.3.2 Funcionamiento y algoritmo diseñado.

El funcionamiento de este modo es muy sencillo. El robot, que está manteniendo el equilibrio, está parado hasta que el usuario pulsa uno de los ocho botones de dirección, momento en el que comenzará a moverse en la dirección asociada al botón pulsado. Mientras el usuario mantenga el botón pulsado el robot continuará moviéndose en esa dirección. Si el usuario pulsa otro botón el robot cambiará su dirección. Para que el robot se mueva en la dirección deseada el botón debe permanecer pulsado, en el momento en el que se deja de pulsar el botón el robot se para.

Para poder llevar a cabo esta función se debe implementar dos programas. El primero se ejecuta en la EV3 y su función es recibir el mensaje de la dirección a seguir y definir unas velocidades de control para la dirección seleccionada. Igual que ocurría en el control de trayectoria en este modo el control de la velocidad se realiza en el bucle de equilibrio, esto se explicará en el apartado *"3.4 Diseño del sistema de control completo"*.

El segundo se ejecuta en el PC y sirve de interfaz con el usuario, presenta los 8 botones de dirección por pantalla. Su función es enviar un mensaje al programa que está ejecutándose en la EV3 con el comando de la dirección que el robot debe seguir cuando el usuario está pulsando alguno de los botones mostrados por la pantalla del PC.

En la tabla 7 se muestran las variables utilizadas en los dos programas.

Nombre de la variable	Tipo de dato	Explicación
AA	BOOLEAN	Botón avanzar. **PC**
RR	BOOLEAN	Botón retroceder. **PC**
GD	BOOLEAN	Botón girar derecha. **PC**
GI	BOOLEAN	Botón girar izquierda. **PC**
AD	BOOLEAN	Botón avanzar y girar derecha. **PC**
AI	BOOLEAN	Botón avanzar y girar izquierda. **PC**
RD	BOOLEAN	Botón retroceder y girara derecha. **PC**
RI	BOOLEAN	Botón retroceder y girar izquierda. **PC**
direccion	I16	Mensaje enviado por bluetooth con la dirección a seguir. **PC**
MensajeR	I16	Mensaje recibido por el EV3 mediante bluetooth con la dirección que debe seguir. **EV3**
Vc	DBL	Velocidad lineal de control. **EV3**
Wc	DBL	Velocidad angular de control. **EV3**
sal	BOOLEAN	Variable que gestiona salida del bucle. **EV3**

Tabla 7.Variables utilizadas control de dirección.

Programa PC.

En esta parte del programa se muestra por pantalla los 8 botones para las diferentes direcciones que el robot puede seguir. Cuando el usuario pulsa un botón el programa cambia el valor de la variable *direccion* y se encarga de enviar este mensaje por bluetooth al ladrillo EV3. En la tabla 8 se muestran los diferentes valores que toma la variable *direccion* dependiendo del botón pulsado.

Botón Pulsado	Valor de la variable *direccion*
AA	1
GI	2
GD	3
RR	4
AI	5
AD	6
RI	7
RD	8

Tabla 8. Valores asociados a los botones de dirección.

El primer paso es comprobar si hay algún botón pulsado. Si hay alguno pulsado se cambia el valor de la variable *direccion* al valor asociado a este botón (valor visto en la Tabla 8). Si no se ha pulsado ninguno la variable *direccion* estará a 0 que es el valor al que se ha inicializado. Luego se comprueba que el botón permanezca pulsado y en caso afirmativo se envía el valor de la variable

direccion al Maibox1 mediante el bloque “Send Host Mail (num)”, si no está pulsado no se envía ningún mensaje. Luego se ha añadido una pequeña espera para dar tiempo al programa a enviar el mensaje. A continuación se comprueba si el botón permanece pulsado, en cuyo caso no se realizará ninguna acción. Si por el contrario ya no hay ningún botón pulsado se enviara al Mailbox1 un cero. Esta secuencia es un bucle que se repite de forma infinita. Para ver el funcionamiento de manera más clara en la figura 46 se muestra un flujograma explicando el funcionamiento de esta parte del programa.

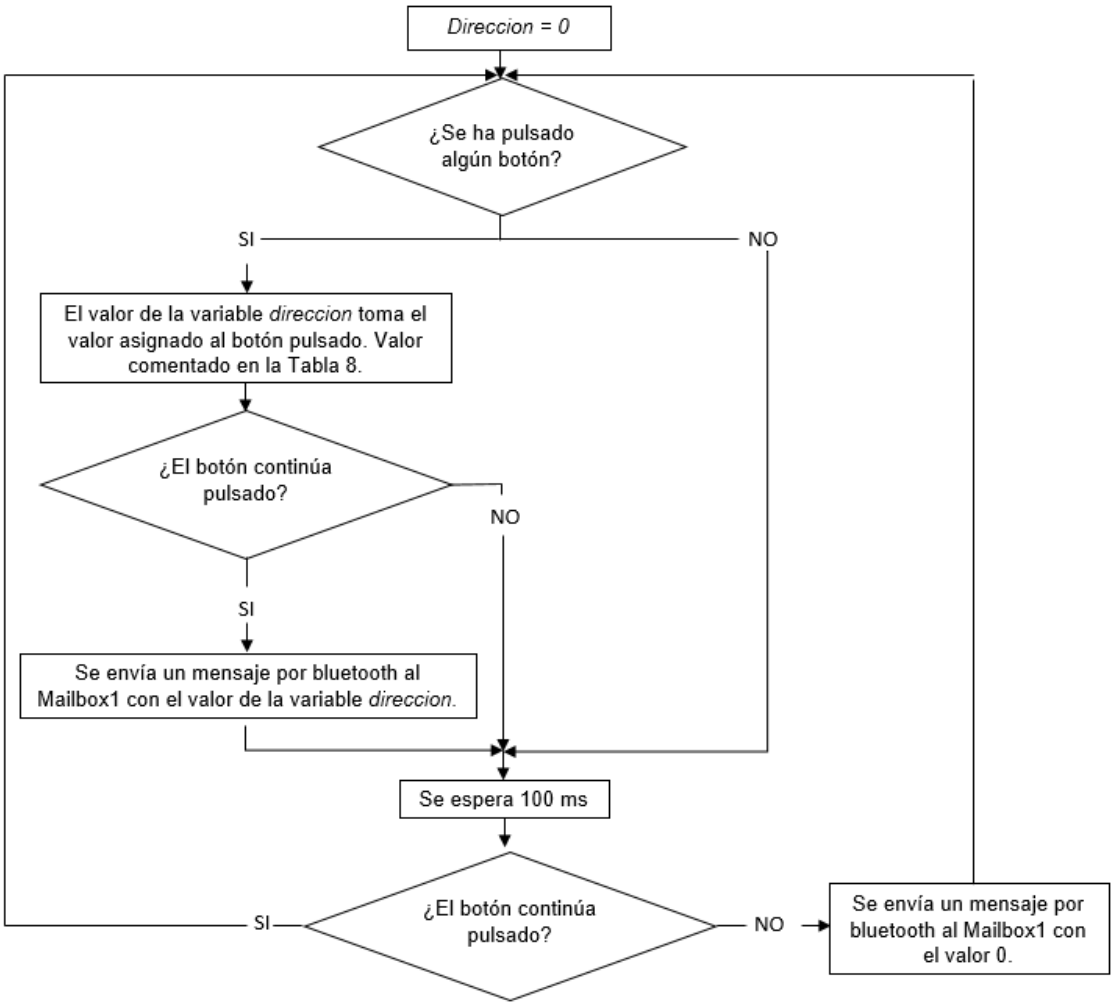

Figura 46.Funcionamiento del algoritmo ejecutado en el PC.

Programa EV3

Esta parte del programa se encarga de recibir los mensajes de dirección enviados desde el PC y definir las velocidades de control lineal y angular (V_c y W_c) en función del mensaje recibido. En la tabla 9 se muestra los valores asignados a la velocidad de control según el mensaje recibido.

NOTA: Para realizar los giros se sumará el mismo valor de signo contrario a cada rueda, de esta forma el robot girará sobre sí mismo. Por este motivo el valor de la variable W_c está expresado en mm/s.

Mensaje recibido *(MensajeR)*	Valor de las velocidades de control
0 (no hay botón pulsado)	$V_c = 0\ mm/s$ y $W_c = 0\ mm/s$
1 (AA, avanza)	$V_c = 150\ mm/s$ y $W_c = 0\ mm/s$
2 (GI, gira izquierda)	$V_c = 0\ mm/s$ y $W_c = 7\ mm/s$
3 (GD, gira derecha)	$V_c = 0\ mm/s$ y $W_c = -7\ mm/s$
4 (RR, retrocede)	$V_c = -75\ mm/s$ y $W_c = 0\ mm/s$
5 (AI, avanzar y g. izq.)	$V_c = 150\ mm/s$ y $W_c = 7\ mm/s$
6 (AD, avanzar y g. der.)	$V_c = 150\ mm/s$ y $W_c = -7\ mm/s$
7 (RI, retroceder y g. izq.)	$V_c = -75\ mm/s$ y $W_c = 7\ m/s$
8 (RD, retroceder y g. der.)	$V_c = -75\ mm/s$ y $W_c = -7mm/s$

Tabla 9. Valores de las velocidad de control según la dirección que se desea seguir.

El funcionamiento es muy sencillo y consiste en esperar a recibir un mensaje al Mailbox1. Para ello utilizamos la función “Wait for Communication” que está dentro de un bucle que se repetirá mientras no llegue ningún mensaje. Cuando se recibe un mensaje se almacena el valor recibido en la variable *MensajeR* y la variable *sal* se declara verdadera y se sale del bucle. A continuación se analiza el valor de la variable *MensajeR* y se asignan los valores correspondientes a las velocidades de control (Tabla 9). Toda esta secuencia se repite de forma infinita. Para ver el funcionamiento de manera más clara en la figura 47 se muestra un flujograma explicando el funcionamiento de esta parte del programa.

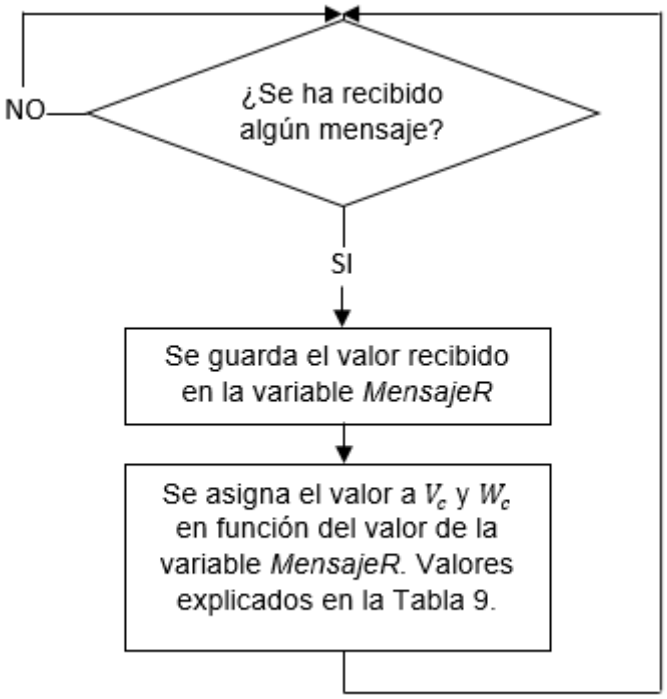

Figura 47. Funcionamiento del algoritmo ejecutado en la EV3.

3.4 Diseño del sistema de control completo.

En los apartados anteriores se ha explicado el diseño del control de estabilidad, el control de trayectoria y el control de dirección por separado. En la aplicación diseñada el robot puede realizar diferentes funciones, pero para poder realizar todas las funciones correctamente los tres controles diseñados deben funcionar de forma conjunta. Esto permitirá que el robot siga trayectorias o se mueva hacia cierta dirección mientras mantiene el equilibrio. Como se dijo anteriormente el robot no puede realizar estas funciones si el robot no está en equilibrio.

Para diseñar el control completo es necesario definir una arquitectura de control. Esta establece la relación entre los diferentes sistemas que forman el control completo y la manera que interactúan.

En el control completo hay tareas que tienen más prioridad que otras y por tanto deberán ejecutarse de manera más rápida. Las diferentes tareas se agrupan en capas, donde las capas bajas tienen mayor prioridad que las capas altas. Por esto en las capas bajas se agrupan las tareas más importantes y en las capas altas se agrupan tareas más generales.

Como se explicó anteriormente se tiene un control de equilibrio que calcula las acciones necesarias para mantener el robot equilibrado a partir de la información de los sensores. Por otra parte tenemos un control de trayectoria que a partir de la localización del robot y del punto objetivo calcula la velocidad de referencia para cada rueda para alcanzar dicho punto. Y por último un control de dirección que define la velocidad de referencia de las ruedas según la dirección que se desea seguir. Como ya se comentó el algoritmo de control de trayectoria y de dirección solo se encargan de calcular las velocidades de referencia, y estas velocidades de referencia se mandan al bucle de equilibrio donde se realiza el control de velocidad. Este bucle se encarga de calcular la acción de control necesaria para alcanzar dicha velocidad y la aplica a los motores, por tanto el bucle de equilibrio se encarga de controlar el equilibrio y el avance.

Por tanto la arquitectura para el control completo se va a dividir en 3 capas. La capa de nivel más bajo se encarga de la tarea más importante, es decir, de mantener el equilibrio en todo momento. Para ello se está ejecutando el bucle de equilibrio de forma muy rápida (tiempo de muestreo 5 ms) para poder reaccionar de manera inmediata a la información que aportan los sensores, esta capa no planifica solo reacciona ante la información recibida a través de sus sensores. La capa de nivel más alto se encargan de tareas menos prioritarias, es decir, es la encargada de la navegación del robot, en esta se define la localización del robot y se realizan los cálculos de la velocidad de referencia para cada rueda para que el robot siga la ruta deseada. Entre estas dos capas hay una capa intermedia que sirve de enlace entre la capa superior e inferior, esta capa se encarga de controlar la velocidad de giro de las ruedas. El control de trayectoria actualiza el valor de la velocidad de referencia a seguir cada 0,5 segundos y el

control de dirección actualiza el valor cada vez que se produce un cambio en la dirección deseada.

Esta arquitectura permite que las tareas más prioritarias se ejecuten de manera más rápida mientras que las tareas de la capa más altas, que se encargan de objetivos más generales, se realicen solo si las primeras se cumplen. Esto es muy importante ya que para la aplicación diseñada no importa que el robot tarde más tiempo en realizar una trayectoria, en dirigirse hacia una dirección o incluso que retroceda o se pare en determinados momentos si esto asegura que el robot va a mantener el equilibrio.

Durante el normal funcionamiento de la aplicación diseñada las tres capas están interactuando entre ellas, en la capa más baja se obtiene los datos del entono a través de los sensores y estos son utilizados por la capa alta para realizar los cálculos necesarios y a su vez esta puede enviar a la capa baja las acciones que debe ejecutar el robot.

Los tres algoritmos diseñados se ejecutan en paralelo en la EV3. En el caso del control de equilibrio el algoritmo se debe ejecutar en la EV3 completamente de manera obligatoria debido a que la comunicación bluetooth es más lenta, por lo que si el control se ejecutara en el PC el retardo introducido por el bluetooth lo hará completamente inestable. En el caso del algoritmo de seguimiento de trayectoria podría ejecutarse en el PC y enviar a la EV3 la velocidad de referencia calculada ya que no se requiere rapidez para ejecutar esta tarea. Aun así se optó por que el algoritmo se ejecutara en la EV3 ya que el ladrillo tiene suficiente capacidad de procesamiento y así prevenir fallos debidos a algún error en la comunicación. Sin embargo, hay una parte complementaria a los algoritmos de seguimiento de trayectoria y dirección que se ejecuta de forma remota. Esta parte se encarga de definir las tareas a realizar por el robot.

Por tanto las características del trabajo impiden ejecutar el programa íntegramente en la EV3 (ejecutar en local), ya que las funciones del robot se deben controlar de forma remota. Tampoco se puede tener el programa ejecutándose de forma completamente externa al robot (ejecutar en remoto), ya que como se dijo el retraso en las comunicaciones harían imposible que el robot mantuviese el equilibrio. En definitiva se tiene un sistema distribuido, una parte en local para lograr mantener el robot estable y otra parte del programa en remoto que controla la trayectoria que debe realizar el robot o la dirección a seguir.

3.4.1 Modificaciones en los algoritmos diseñados.

Para poder unir los tres controles se deben realizar ciertas modificaciones sobre los controles por separado explicados en los apartados anteriores. A continuación se comentarán las nuevas funciones y variables que aparecen en cada uno de los controles y se explica el diseño del algoritmo implementado.

En la tabla 10 se muestran las nuevas variables utilizadas para el algoritmo completo.

Nombre variable	Tipo de dato	Explicación
PosD1_aux	DBL	Valor de la posición del motor derecho que se obtiene en el bucle de estabilidad y que se utiliza en el algoritmo de seguimiento de trayectoria para inicializar la variable *PosD1,* que indica el valor de la posición del motor en el instante anterior. De esta forma el robot al iniciar una trayectoria sabe la posición en la que se encontraba en el instante anterior. El valor de *PosD* antes se inicializaba a 0, pero ahora el robot se ha podido mover antes de iniciar la trayectoria por lo que es necesario realizar este ajuste.
PosI1_aux	DBL	Valor de la posición del motor izquierdo que se obtiene en el bucle de estabilidad y que se utiliza en el algoritmo de seguimiento de trayectoria para inicializar la variable *PosI1,* que indica el valor de la posición del motor en el instante anterior. De esta forma el robot al iniciar una trayectoria sabe la posición en la que se encontraba en el instante anterior. El valor de *PosI* antes se inicializaba a 0, pero ahora el robot se ha podido mover antes de iniciar la trayectoria por lo que es necesario realizar este ajuste.
Wc_aux	DBL	Valor que se utiliza para controlar el giro del robot, su valor está relacionado con la variable Wc que se utiliza para controlar en giro del robot.

Tabla 10.Variables añadidas al algoritmo del sistema completo.

Las principales modificaciones aparecen en el bucle de equilibrio, ya que hay que añadirle la función para controlar la velocidad de referencia calculada por los algoritmos de seguimiento y dirección. Para ello se va a dividir la velocidad de referencia calculada en dos partes: una parte que se encarga del avance y retroceso del robot, este valor se almacenará en la variable V_c. Y la otra parte que se encarga del giro del robot, este valor se almacenará en la variable W_c*_aux*.

La velocidad V_c se logra mediante el control por realimentación de estados diseñado para mantener el equilibrio. Para ello se añade el valor de la velocidad deseada con su correspondiente ganancia. De esta forma la acción de control se calcula mediante la siguiente ecuación:

$$pwr = K1 \cdot gSpd + K2 \cdot gAng + K3 \cdot mSpd + K4 \cdot mPos + K5 \cdot V_c \qquad (59)$$

Por tanto ahora la potencia aplicada a los motores depende de las 4 variables medidas que son ángulo de inclinación del robot (*gAng),* velocidad angular de inclinación del robot (*gSpd),* posición de los motores (*mPos)* y velocidad de los motores (*mSpd)* y de la velocidad de referencia calculada por el algoritmo de trayectoria o por el algoritmo de control de dirección (*Vc).*

Los valores de ganancia diseñados son:

K1 = 0,8

K2 = 15

K3 = 0,08

K4 = 0,12

K5 = -0,01

El movimiento del robot debido a la velocidad de control V_c afecta a la posición del motor (*mPos*) medida por los encoders y por tanto a la velocidad del motor (*mSpd)*, estos valores se utilizan para calcular la acción de control. Por tanto hay que tener en cuenta que el avance y el retroceso del robot, debido a la realización de una trayectoria o a moverse en una determinada dirección, afecta a la variable de posición de motor (*mPos*) y a la velocidad del motor (*mSpd)*. Para no cambiar el valor de la referencia de la variable *mPos* antes de calcular la acción de control se le resta el valor del cambio de posición debido al avance o retroceso del robot.

$$mPos = mPos - V_c \cdot tInt \qquad (60)$$

Recuerda que la variable *tInt* es el tiempo de integración calculado en el bucle de equilibrio.

Y después de calcular y limitar la acción de control se vuelve a realizar el mismo ajuste mostrado en la ecuación 60 y este valor de *mPos* obtenido es el que se utilizará para calcular la posición de los motores en la siguiente iteración.

La velocidad W_c*_aux* es la encargada de controlar el giro que debe realizar el robot para alcanzar el ángulo deseado y de esta forma controlar hacia qué dirección se dirige el robot. Para lograr el giro del robot este valor se introduce como una perturbación en la acción de control para cada motor. Este valor calculado modifica la velocidad de las ruedas sumando a una y restando a otra el valor de W_c*_aux.* Al tratarse del mismo valor de signo contrario el robot gira sobre su propio eje.

El valor de la acción de control que se aplica finalmente a los motores depende de la acción de control calculada para mantener el equilibrio y para que el robot avance y retroceda (*pwr*) y del valor de la velocidad de giro W_c*_aux.* En la siguiente ecuación se muestra la ecuación para el cálculo de la acción de control aplicada a los motores:

$$U_d = pwr + W_c\_aux \quad (61)$$

$$U_i = pwr - W_c\_aux \quad (62)$$

Para obtener el valor de V_c y W_c_aux a partir de las velocidades de referencia obtenidas por el control de trayectoria y el control de dirección se deben realizar algunas modificaciones.

En el control de trayectoria la velocidad de referencia se calcula mediante la ecuación:

$$V_{ref\_d} = V_c + \frac{b}{2} \cdot w_c \quad (63)$$

Siendo V_c y W_c las velocidades de control calculadas por el algoritmo.

Como se dijo anteriormente esta velocidad está compuesta por una parte que depende de la distancia del robot al punto objetivo (V_c) y otra parte que depende del ángulo que debe girar el robot para alcanzar la orientación deseada (W_c). Por tanto es sencillo separar esta velocidad en las dos partes necesarias para poder realizar el control:

$$V_c = V_c \quad (64)$$

$$W_{c_{aux}} = \frac{b}{2} \cdot w_c \quad (65)$$

En el control de dirección se definen las velocidades directamente como V_c y W_c_aux en función del mensaje de dirección recibido desde el PC.

Además de las modificaciones explicadas anteriormente a los algoritmos de seguimiento y dirección se les ha añadido una espera al principio del algoritmo que impide que el robot reciba órdenes de trayectorias a realizar o de direcciones a seguir hasta que se haya calibrado el giroscopio y el robot esté manteniendo el equilibrio.

4. La aplicación

Una vez se tiene los tres algoritmos diseñados funcionando conjuntamente el robot puede comenzar a realizar las funciones para las que ha sido diseñado: realizar trayectorias previamente conocidas o moverse hacia una cierta dirección mientras mantiene el equilibrio. En la aplicación diseñada el usuario puede controlar la función que realiza el robot desde el PC.

El usuario puede seleccionar entre los dos modos de funcionamiento “Modo trayectoria” y “Modo remoto”. Dentro del modo trayectoria se puede seleccionar

la trayectoria a realizar. Si el modo elegido es el remoto se puede elegir la dirección que el robot debe seguir.

Para poder enviar las órdenes desde el PC hasta el robot se ha ampliado el programa diseñado para el modo remoto "*Programa PC"* (explicado en el apartado 3.3). Ahora el primer paso es seleccionar el modo de funcionamiento y una vez elegido se enviará el mensaje con la dirección o la trayectoria. Si se ha seleccionado el modo remoto el programa solo envía el mensaje con la dirección a seguir (variable *direccion)*. Si el modo seleccionado es el modo trayectoria el programa solo envía la trayectoria a realizar (variable *trayectoria).*

Controlar el funcionamiento del robot de forma remota desde el PC requiere un constante intercambio de información entre el PC y el robot, ya que el programa que se ejecuta en el PC debe enviar mediante bluetooth la información necesaria sobre la tarea a realizar a la EV3, y esta debe mandar información al PC sobre el desarrollo de la tarea que está realizando. Esto se realiza mediante mensajes enviados por bluetooth a los diferentes Mailbox. En la tabla 11 se muestra los diferentes Mailbox utilizados y la función de cada uno y en la tabla 12 se muestran los diferentes mensajes enviados a cada Mailbox y su significado.

Mailbox	Función
Mailbox 1	Enviar desde el PC a la EV3 el valor de la dirección a seguir.
Mailbox 2	Enviar desde el PC a la EV3 el valor de la trayectoria a realizar.
Mailbox 3	Enviar desde el PC a la EV3 el modo de funcionamiento seleccionado.
Mailbox 4	Enviar desde la EV3 al PC un mensaje cuando el robot comienza a realizar la trayectoria y además indica que trayectoria va a realizar. De esta forma el usuario puede saber que el robot está realizando la trayectoria y que trayectoria.
Mailbox 5	Enviar desde la EV3 al PC un mensaje cuando el robot ha completado la trayectoria. De esta manera se informa al usuario que la trayectoria ha finalizado y que está listo para realizar otra función.
Mailbox 6	Enviar desde la EV3 al PC un mensaje cuando los sensores se hayan calibrado y el robot esté manteniendo el equilibrio. Antes de esto el robot no obedecerá las órdenes. Por tanto este mensaje hace saber al usuario que el robot está listo y va ha comenzado a obedecer las órdenes que reciba.

Tabla 11. Mailbox utilizados en la aplicación para intercambiar información entre el PC y la EV3.

Mailbox	Mensaje enviado	Función
Mailbox1 (dirección a seguir)	0	Ninguna dirección.
	1	Avanzar.
	2	Girar izquierda.
	3	Girar derecha.
	4	Retroceder.
	5	Avanzar y girar izquierda.
	6	Avanzar y girar derecha.
	7	Retroceder y girar izquierda.
	8	Retroceder y girar derecha.
Mailbox2 (trayectoria a realizar)	9	Cuadrado.
	10	Hexágono.
	11	Triángulo.
Mailbox3 (Modo funcionamiento)	12	Modo remoto.
	13	Modo trayectoria.
Mailbox4 (trayectoria que está realizando el robot)	0	Ninguna trayectoria.
	1	Realizando cuadrado.
	2	Realizando hexágono.
	3	Realizando triángulo.
Mailbox5 (trayectoria terminada)	0	No se ha terminado.
	1	Si se ha terminado.
Mailbox6 (listo para comenzar)	0	No está manteniendo equilibrio.
	1	Si está manteniendo equilibrio

Tabla 12. Mensajes recibidos en cada Mailbox y su significado.

4.1 Interfaz de usuario.

Para facilitar el uso de la aplicación por parte del usuario se ha diseñado una interfaz de usuario cuya función es facilitar la interacción entre el ser humano y la máquina La interfaz se ha diseñado de la forma que sea fácil e intuitiva. El objetivo de esta es facilitar las interacciones entre el usuario y el robot.

El usuario puede seleccionar fácilmente el modo de funcionamiento y la tarea a realizar y ver la información sobre el estado y la actividad del robot. En la figura 48 se muestra el aspecto de la interfaz diseñada.

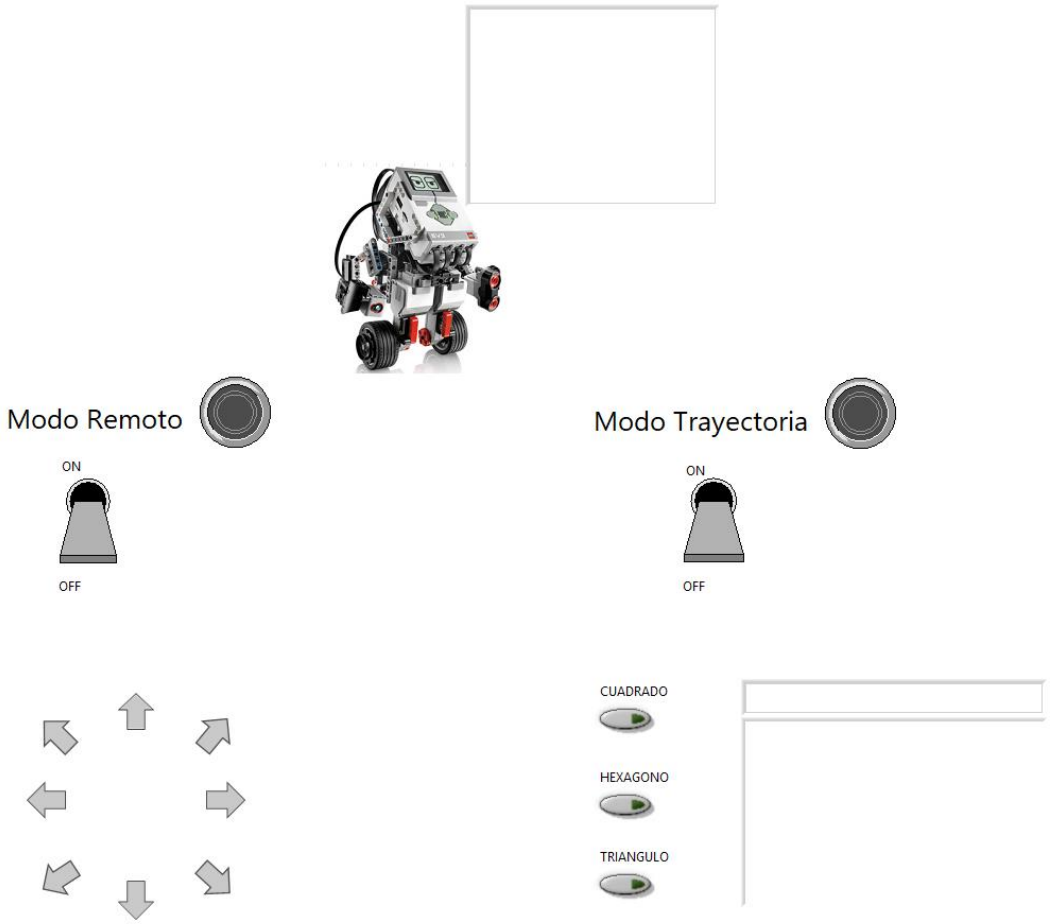

Figura 48. Interfaz de usuario.

Como puede observarse la interfaz dispone de un cuadro de dialogo donde el robot informa al usuario si está listo para obedecer órdenes o no. Para saber el estado del robot la EV3 manda un mensaje al Mailbox6 cuando el robot se enciende y está calibrando los sensores, y otro cuando está manteniendo el equilibrio y preparado para obedecer las órdenes del usuario.

Figura 49.Interfaz robot calibrando sensor. equilibrio.

Figura 50.Interfaz Robot manteniendo

Una vez el robot ha calibrado los sensores y está listo para comenzar, ya se puede elegir el modo de funcionamiento y la tarea a realizar. Para seleccionar el modo se utiliza unos interruptores de palanca, para indicar el modo elegido se enciende el led asociado a cada uno de los modos. Al elegir un modo de funcionamiento se envía a la EV3 un mensaje al Mailbox3 con el modo elegido. De esta forma solo se ejecuta el algoritmo del modo seleccionado, ya que ambos algoritmos tienen al comienzo un bucle del que no saldrán hasta que el mensaje recibido al mailbox3 les indique que se ha seleccionado su modo de funcionamiento. Además el PC solo envía los comandos asociados al modo seleccionado, es decir, si estamos en el modo remoto y pulsas el botón para realizar un cuadrado el robot no envía esa información. Por tanto cuando se ha seleccionado un modo ni se envía mensajes con órdenes del otro modo ni se ejecuta la parte del algoritmo asociada al modo no seleccionado. Si no se ha seleccionado ningún modo no se envía ningún mensaje ni de dirección ni de trayectoria desde el PC y en la EV3 los algoritmos de trayectoria y de dirección están esperando a que les llegue el mensaje al Mailbox3 con el modo elegido.

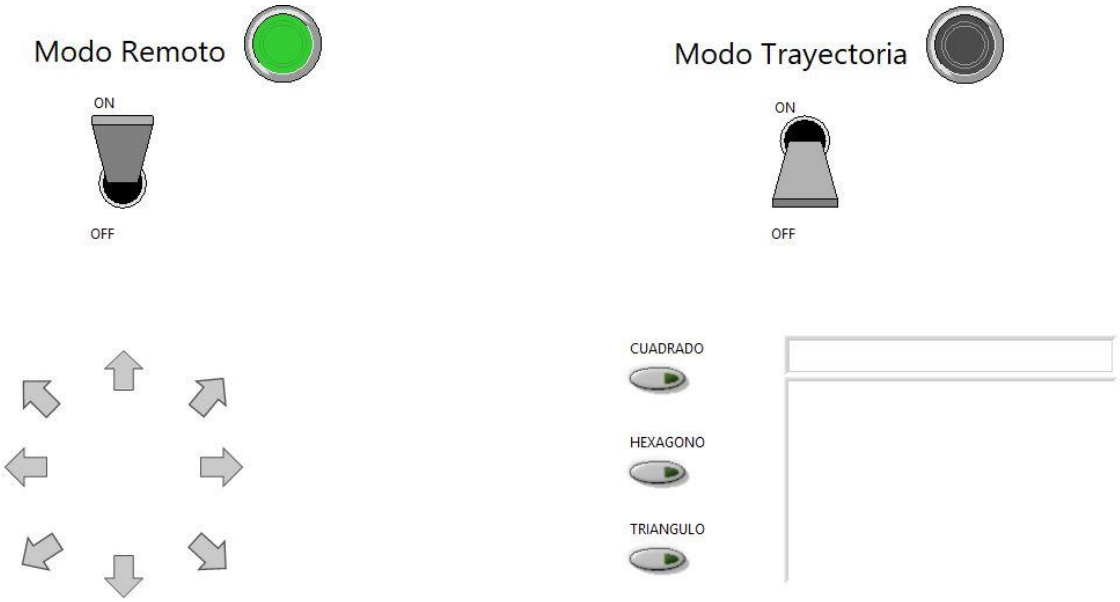

Figura 51. Interfaz, Modo remoto seleccionado.

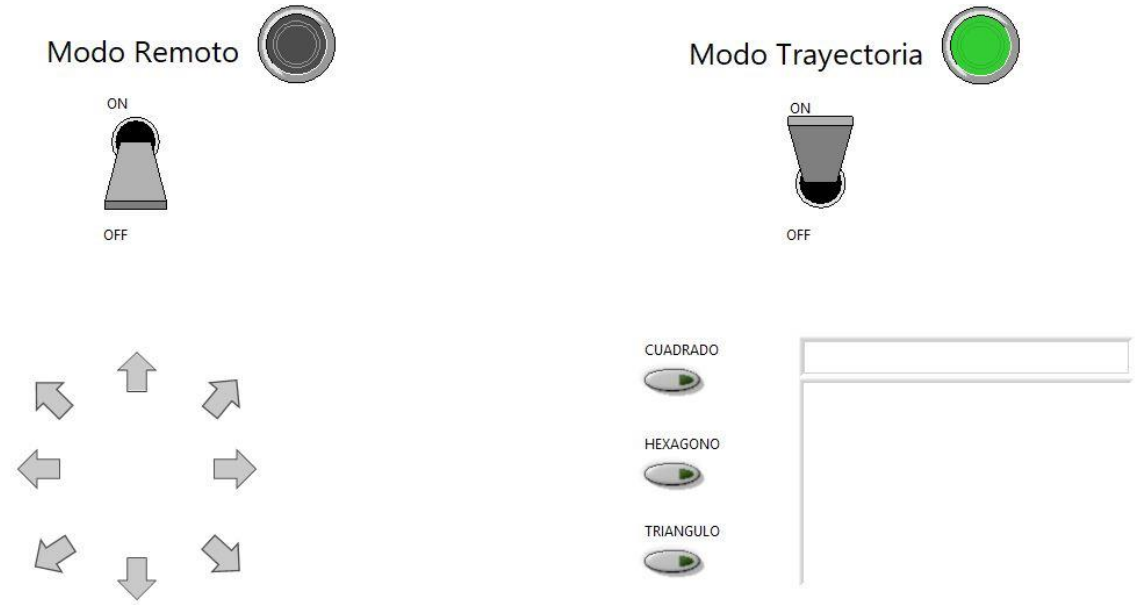

Figura 52. Interfaz, Modo trayectoria seleccionado.

Una vez se ha elegido el modo de funcionamiento se puede controlar los movimientos del robot. Si el modo elegido es el modo remoto se puede mover el robot en la dirección deseada pulsando la flecha asociada a dicha dirección. Al pulsar la flecha se envía un mensaje al Mailbox1 con la dirección deseada. Como ya se dijo anteriormente si mientras está seleccionado el modo remoto se pulsa algún botón de trayectoria el robot no obedecerá ya que solo obedece a los comandos del modo remoto. Para indicar el botón pulsado este se resalta sobre el resto.

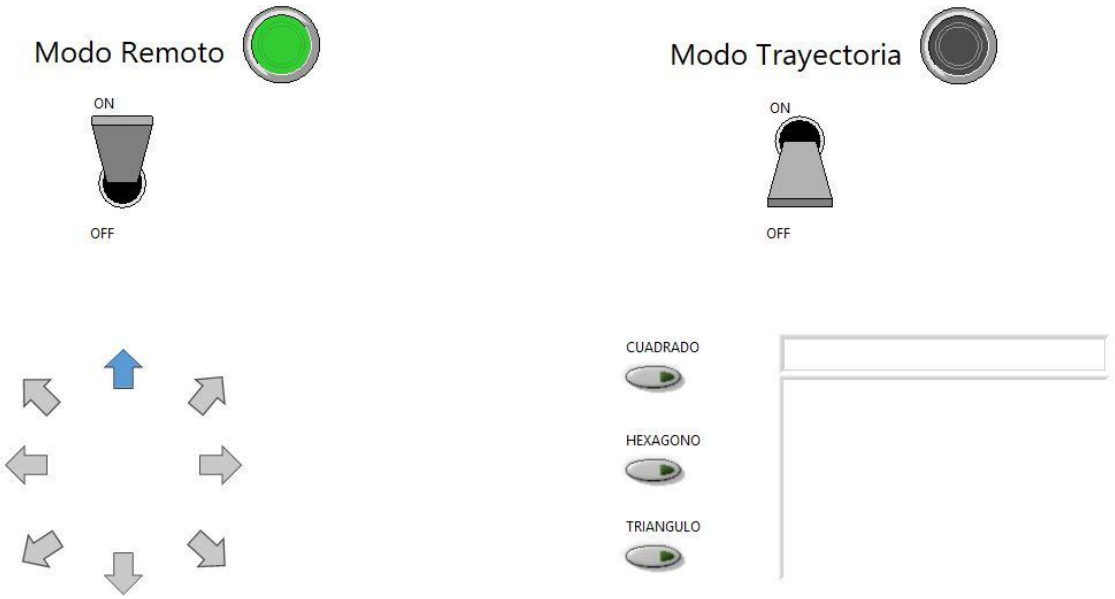

Figura 53. Modo remoto moviendo robot hacia delante.

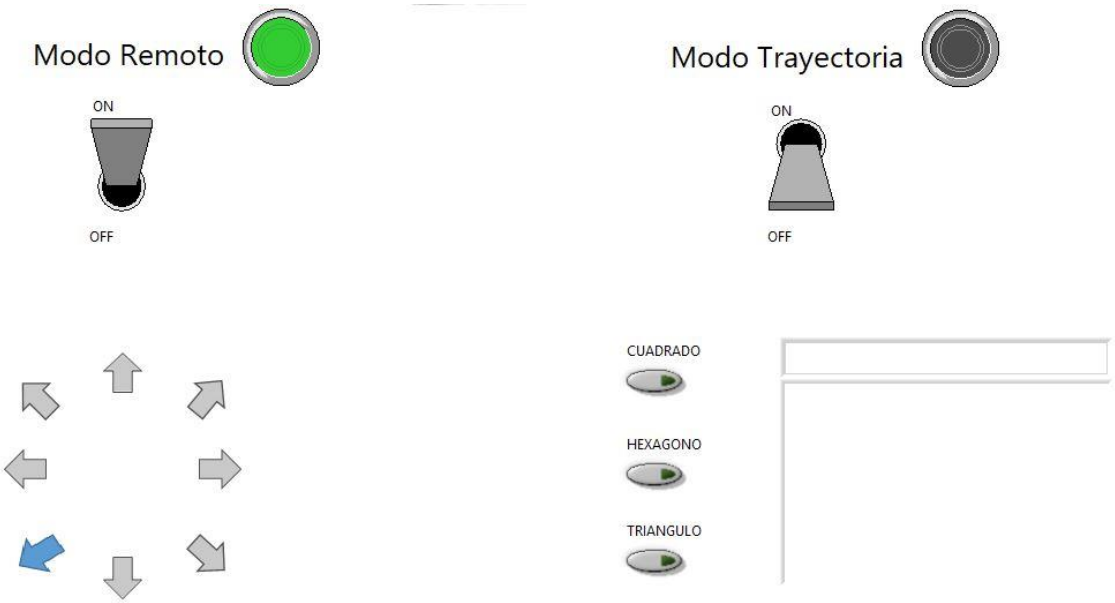

Figura 54. Modo remoto moviendo el robot hacia detrás mientras gira a la izquierda.

Si el modo elegido es el de trayectoria el robot estará listo para comenzar a realizar una trayectoria. Si se pulsa algún botón de dirección el robot no obedecerá, ya que solo obedece a los comandos del modo seleccionado. Para que el robot comience a realizar la trayectoria deseada se manda a la EV3 un mensaje al Mailbox2 con la trayectoria a realizar. Tal y como se ha diseñado el programa cuando el robot comienza a realizar una trayectoria no escucha ordenes hasta que la termina. Por tanto si está realizando un cuadrado y se pulsa el botón para realizar un triángulo el robot terminará de realizar el cuadrado. Para informar al usuario que el robot ha recibido la orden y está realizando la

trayectoria se envía desde la EV3 un mensaje con la figura que va a realizar al Mailbox4 cuando el robot comienza a realizarla. Para avisar al usuario que la trayectoria se ha completado envía un mensaje al Mailbox5 al finalizar el recorrido. Esta información se muestra por pantalla mediante un mensaje y una imagen de la figura que va a realizar el robot y cuando la trayectoria se ha completado el mensaje y la figura desaparecen quedando el espacio en blanco.

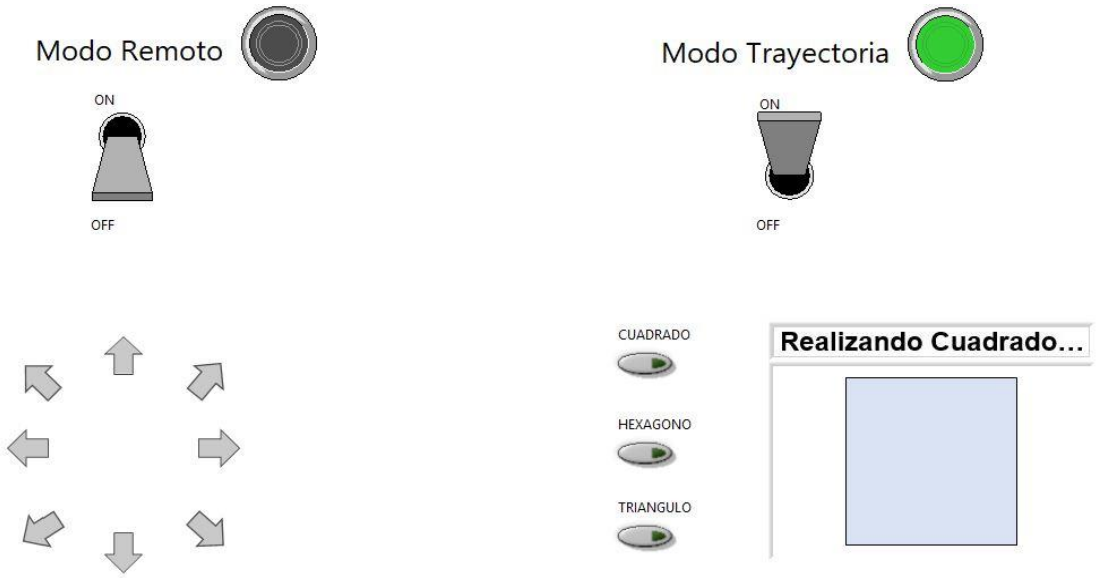

Figura 55. Modo trayectoria realizando cuadrado.

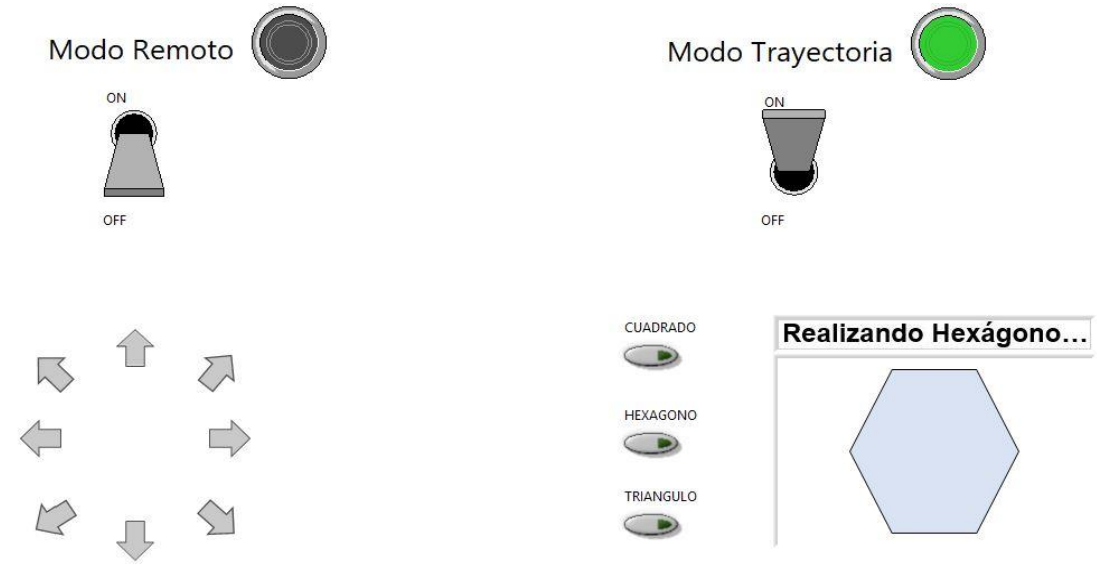

Figura 56.Modo trayectoria realizando hexágono.

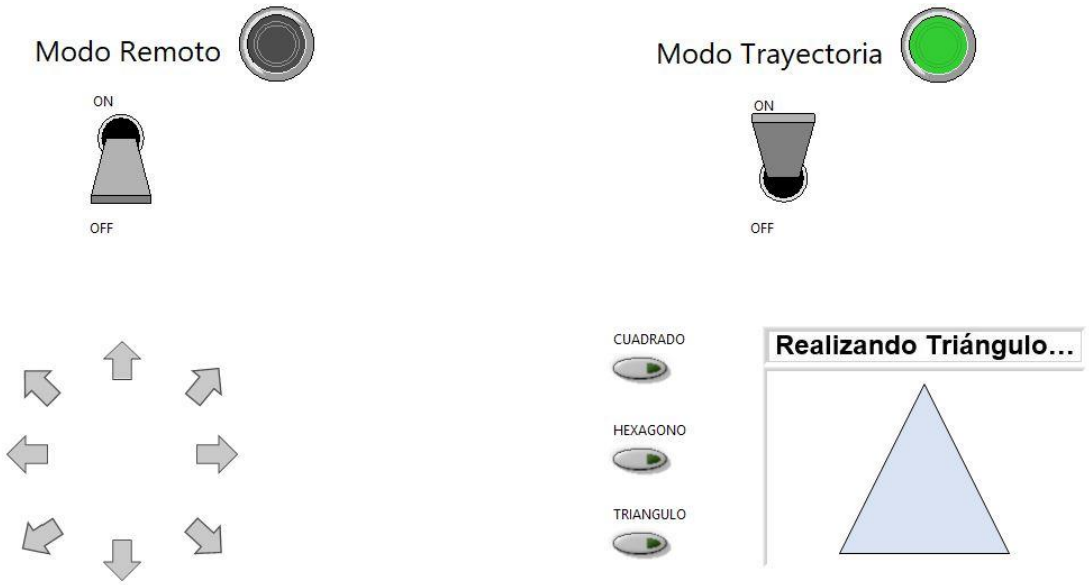

Figura 57.Modo trayectoria realizando triángulo.

Tal y como se ha visto la interfaz diseñada permite al usuario seleccionar el modo de funcionamiento y seleccionar las tareas a realizar por el robot de manera sencilla. Además muestras información al usuario sobre el estado del robot y sobre las acciones que está ejecutando.

Además de la información mostrada por la pantalla del PC, también se muestran diferentes mensajes por la pantalla de la EV3 con información sobre lo que está ocurriendo en el robot. Estos mensajes son de utilidad ya que permiten ver en el robot información sobre su estado y sobre las acciones que está realizando. El mostrar estos mensajes por la pantalla de la EV3 tiene sentido ya que el robot puede estar alejado del PC con el que se controla, y estos mensajes permiten al usuario acercarse a la zona de trabajo y saber lo que está haciendo el robot sin la necesidad de estar frente a la pantalla del PC. A continuación se explicarán los diferentes mensajes mostrados por la pantalla del robot y su utilidad.

Cuando el robot se enciende y comienza a calibrar el giroscopio se muestra "*Calculando gOS"* en la línea 1 de la pantalla. Cuando el robot termina la calibración y comienza a mantener el equilibrio se escribe en la línea 1 "*Manteniendo el equilibrio".* En este momento el algoritmo de seguimiento de trayectoria y de dirección pueden comenzar a funcionar cuando se elija el modo por ello se escribe "*Puede comenzar"* en la línea 2 y "*Esperando Modo trayectoria"* (línea 12) y "*Esperando Modo remoto"* (línea 10) de esta forma informa de que todavía no se ha elegido ningún modo de funcionamiento. Cuando se elige un modo estos mensajes desaparecen. Si se elige el modo remoto se muestra "*MODO REMOTO"* (línea 5). Si el modo elegido es el de trayectoria se muestra "*Esperando la trayectoria a realizar"* (línea 8) este mensaje desaparece cuando se recibe el mensaje con la trayectoria a seguir y

aparecen los mensajes "*MODO TRAYECTORIA"* (línea 5) "*Siguiendo trayectoria"* (línea 2) , el nombre de la trayectoria que se está realizando "*CUADRADO"*, *"TRIANGULO"* o *"HEXAGONO"* (línea 4) y en la línea 3 se muestra el lado de la figura que se está haciendo en cada momento, es decir, si aparece un "2" el robot está realizando el segundo lado de la figura. Cuando la trayectoria ha terminado se muestra "*FIN"* (línea 2) y se borran los mensajes de la línea 4 y 5. Si el robot se desequilibra se muestra "*Esperando a que pulses el botón"* (línea 1) y una vez pulsado el botón de reinicio se muestra "*Robot debe estar en la posicion de reposo".*

4.2 Funcionamiento de la aplicación y guía de usuario.

El primer paso es iniciar el robot, una vez encendido se ejecuta el programa y el robot comienza a calibrar el giroscopio. Para realizar la calibración de manera correcta el robot debe estar en reposo en posición vertical, esto es un paso delicado ya que si el robot no está totalmente quieto el giroscopio puede ir a la deriva y el robot nunca comenzará a mantener el equilibrio. Para asegurar que el giroscopio no vaya a la deriva es recomendable reiniciar de manera manual el sensor giroscopio antes de comenzar (desconectando y conectándolo). Posteriormente el robot comenzará a mantener el equilibrio en la misma posición donde se ha iniciado, hasta que recibe una señal por bluetooth con el modo de funcionamiento seleccionado. Para ello el usuario, a través de la interfaz diseñada, debe seleccionar un modo de funcionamiento. Una vez elegido un modo se puede comenzar a realizar las tareas asociadas al modo seleccionado.

Si se enciende el modo remoto el robot espera a que el usuario pulse un botón de dirección y cuando esto ocurra se moverá hacia la dirección seleccionada. Cada cierto tiempo comprueba si el botón sigue pulsado en cuyo caso continúa moviéndose. Si por el contrario el botón se ha dejado de pulsar el robot comprueba si el modo remoto sigue activado, en este caso se esperará a recibir otro mensaje con la dirección a seguir. Si el modo ya no está seleccionado, volverá a mantener el equilibrio en la misma posición donde se haya parado hasta que reciba otro mensaje desde el PC con el nuevo modo de funcionamiento seleccionado.

Si se enciende el modo trayectoria el robot espera hasta recibir un mensaje con la trayectoria a realizar. Cuando este llega comienza a realizarla, mientras el robot está realizando una trayectoria no escucha nuevas órdenes del usuario hasta que no haya completado la figura. Cada cierto tiempo el robot se pregunta si ha llegado al objetivo, cuando lo alcanza el robot se pregunta si el modo continua activado, en este caso espera el mensaje con la nueva trayectoria a realizar. Si el modo ya no está encendido vuelve a mantener el equilibrio en la misma posición donde se haya parado al alcanzar el objetivo a la espera de

recibir otro mensaje desde el PC con el nuevo modo de funcionamiento seleccionado.

En la figura 58 se muestra el diagrama de flujo del algoritmo de control.

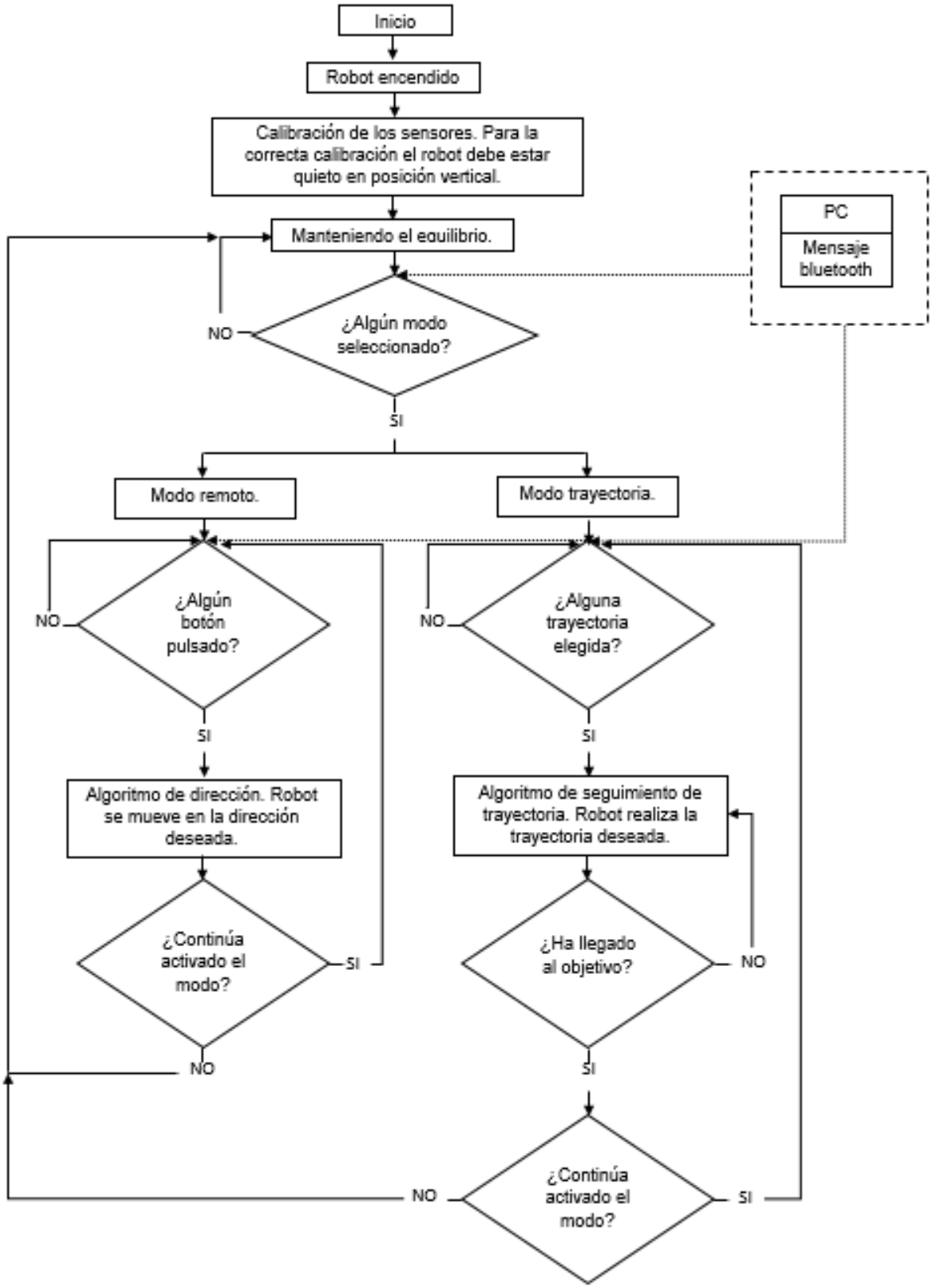

Figura 58. Diagrama de flujo del algoritmo de control.

La secuencia explicada anteriormente se repetirá de forma permanente mientras no se produzca ningún suceso que provoque que el robot se desequilibre. En el caso que el robot pierda el equilibrio el usuario deberá poner el robot en la posición vertical de reposo y pulsar el botón de reinicio, asegurándose que el robot está totalmente quieto en dicho momento. Una vez pulsado el botón, la secuencia explicada anteriormente volverá a comenzar.

5. Pruebas y resultados.

En este apartado se presentan los resultados obtenidos para las diferentes pruebas realizadas a lo largo del desarrollo de la aplicación. Estas pruebas sirvieron para ajustar los valores de los parámetros tanto del control de equilibrio, como los parámetros del algoritmo de seguimiento de trayectoria. Con estas pruebas se pretende obtener el valor óptimo de los parámetros para conseguir que el robot realice las tareas de la mejor forma posible.

En primer lugar se va a mostrar los resultados obtenidos al realizar una prueba de equilibrio con el valor de las ganancias utilizadas en el control de equilibrio: K1 = 0,8, K2 = 15, K3 = 0,08 y K4 = 0,12. En las siguientes figuras se muestra cómo evoluciona en el tiempo el valor del ángulo de inclinación del robot (gAng) y la velocidad angular de inclinación del robot (gSpd). En los resultados obtenidos se observa como el robot se balancea entorno a su posicion de equilibrio sin perder el equilibrio y por tanto se comporta satisfactoriamente.

El robot al iniciarse está apoyado una plataforma para mantenerse estable en la posición de trabajo. Al iniciar el programa el robot se separa de la plataforma y comienza a funcionar, en las gráficas obtenidas se puede observar unos valores pequeños al principio que se corresponden al momento en el que el robot aún no se ha separado de la plataforma.

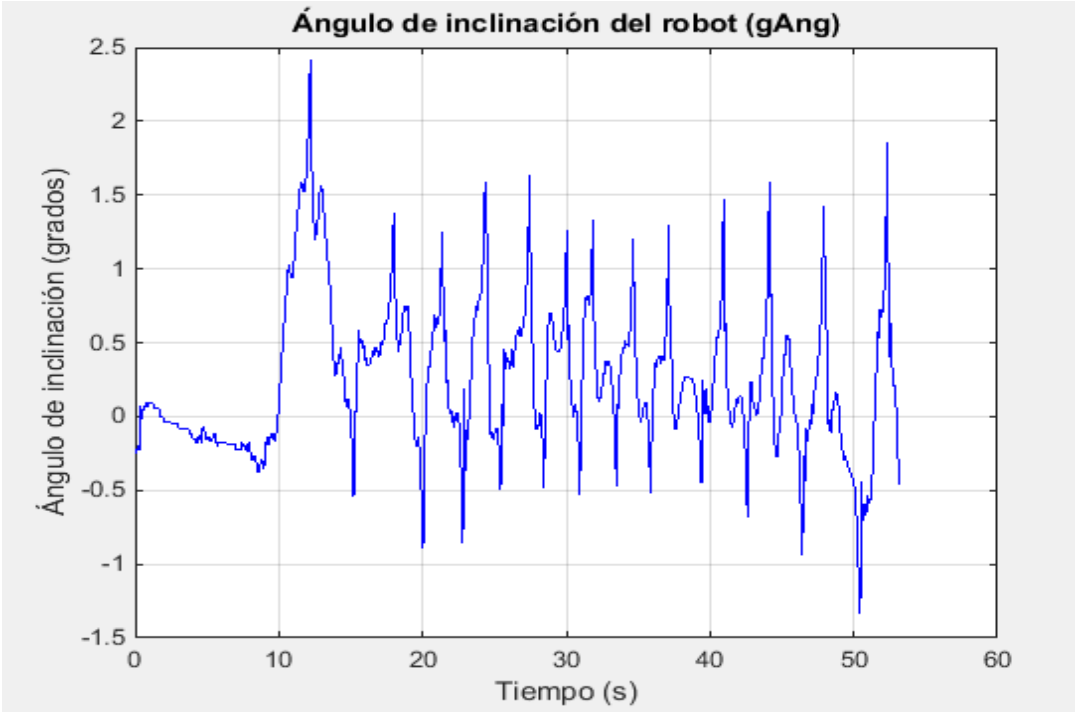

Figura 59. Valor del ángulo de inclinación del robot.

En la figura 59 se muestra el valor del ángulo de inclinación del robot. Para que el robot mantenga el equilibrio este valor debe ser muy pequeño o nulo. Se puede observar como el robot se va balanceando entonto a la posición de equilibrio, que como se dijo anteriormente debido a la estructura del robot en la posicion de reposo tiene un pequeño ángulo de 0,25 grados.

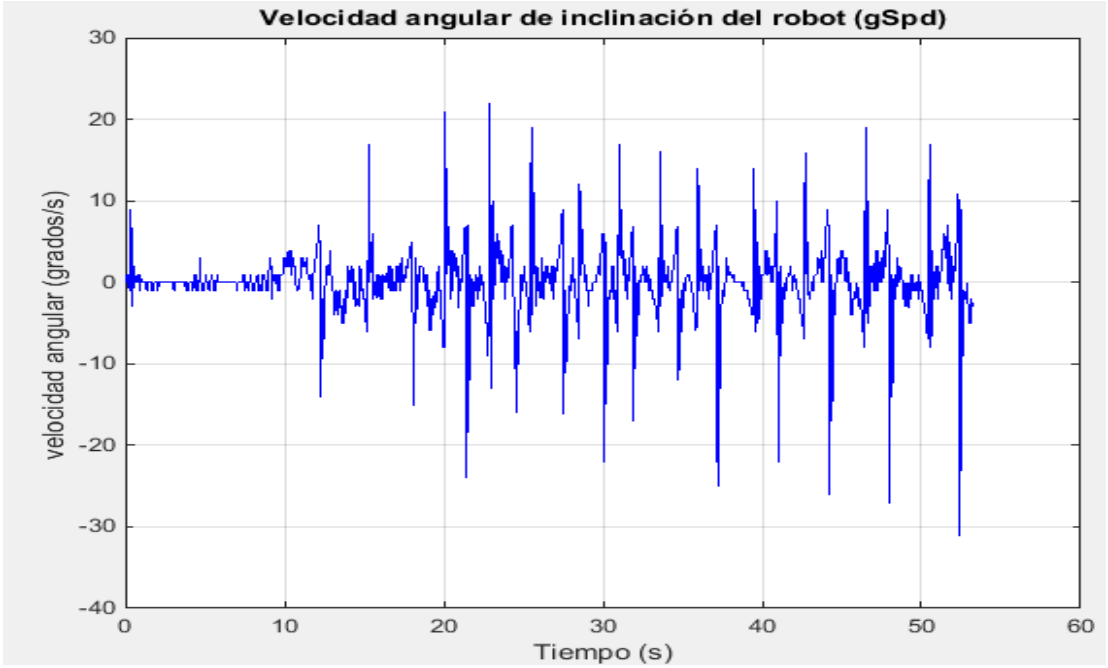

Figura 60. Velocidad del ángulo de inclinación del robot.

En la figura 60 se muestra el valor de la velocidad angular de inclinación del robot. Este valor indica la velocidad con la que el robot se va inclinando hacia un lado y hacia el otro. Se observa como el robot va moviéndose en una y otra dirección para mantener el equilibrio. Este valor está oscilando en torno a 0 grados/s.

A continuación se presentan los resultados de las diferentes pruebas para ajustar los parámetros del algoritmo de seguimiento de trayectoria. Los parámetros a ajustar en este algoritmo son los siguientes:

- Valor de α a partir del cual el robot solo gira sin avanzar para corregir la orientación. Este valor se ajustó analizando los valores de este ángulo durante la realización de diferentes trayectorias y se consideró que un valor adecuado sería $\alpha = 0{,}5\ rad$.

- Valor de W_c, valor de velocidad de giro cuando el robot solo está girando para corregir la orientación. Se realizaron pruebas con diferentes valores, debido a que cada trayectoria tiene unos giros más o menos exigentes según la trayectoria los valores que eran adecuados para unas no lo eran para otras. Para solucionar esto se decidió asignar un valor medio que permitiera una realización de los giros aceptable en todas la trayectorias. Este valor se ajustó a $W_c = 0{,}65\ \frac{rad}{s}$. Con valores de W_c menores cuando el ángulo que debe girar el robot es grande (como en el triángulo) no se lograba corregir de manera adecuada la orientación. Con valores mayores de W_c cuando el ángulo que debe girar es pequeño (como en el hexágono) el giro era muy brusco y giraba más de lo necesario.

- Valor de K_α que se utiliza para calcular el valor de W_c , valor de velocidad de giro, cuando el robot está avanzando y solo tiene que hacer pequeñas correcciones de ángulo. Debido a que el valor del ángulo α que debe girar el robot es pequeño este valor se ajustó a $K_\alpha = 1$.

- Valor de la variable ρ a partir de la cual se considera que el robot está cerca del objetivo y se reduce la velocidad con lo que se acerca a este. El valor óptimo para esta variable también depende de la trayectoria que se está realizando. En este caso la diferencia entre los valores adecuados para cada trayectoria es grande, por eso no se calculó la media como se hizo para obtener el valor de W_c, sino que se utilizó un valor para el cuadrado y el triángulo ($\rho < 300$) y otro para el hexágono ($\rho < 100$).

- Valor de V_c para ir rápido, cuando el robot está lejos del objetivo, a una distancia mayor que la ρ indicada en el párrafo anterior. El valor utilizado es de $V_c = 150\ mm/s$.

- Valor de K_ρ que se utiliza para calcular el valor de V_c cuando la distancia al objetivo es pequeña y esta velocidad se va reduciendo. Este valor depende del valor de ρ a partir del cual se empieza a ir lento el valor se ha ajustado a $K_\rho = 0{,}5$.

- Valor de la variable ρ para considerar que el robot está cerca del objetivo y cambiar la referencia. Este valor se ha ajustado a $\rho < 20$. Si este valor es más pequeño la velocidad con la que intenta llegar al objetivo es pequeña, ya que depende de ρ. Esto provoca que el robot se tambalee de forma brusca cuando está próximo al objetivo. Además como se está tambaleando puede que alcance el objetivo y se mueva a otra posición desde donde se realiza el giro. Esto, por tanto, provoca que realice el giro más lejos de la distancia deseada al objetivo y empeora la trayectoria. Para valores mayores de 20 realiza la trayectoria con menos precisión.

Una vez ajustados los valores de los diferentes parámetros se van a mostrar los resultados de las pruebas realizadas para las diferentes trayectorias. Estas pruebas se realizaron sobre una alfombra lisa y sin obstáculos.

Trayectoria cuadrada

Se trata de una trayectoria cuadrada de 1000 mm de lado.

.

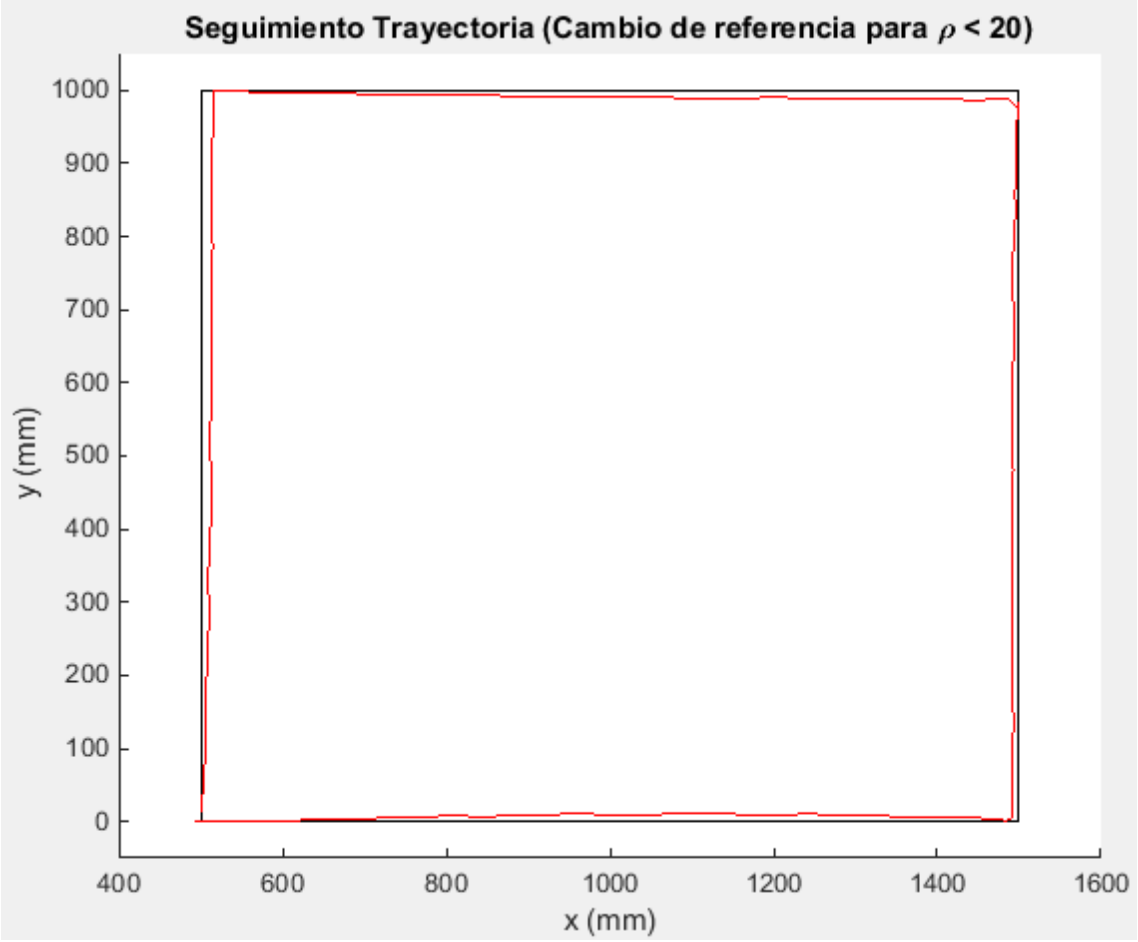

Figura 61.Posición del robot realizando cuadrado.

En la figura 61 se observa que el robot no comienza la trayectoria justo en la posición inicial, esto se debe a que mientras se recibe la instrucción de comenzar a realizar la trayectoria, el robot se está moviendo para mantener el equilibrio. Además se ve como el robot se va acercando hacia el punto de referencia corrigiendo su orientación, para realizar la trayectoria de la manera más precisa posible. También se observa como al llegar a cierta distancia de cada referencia el robot gira, debido a que se ha actualizado el valor de la referencia a seguir.

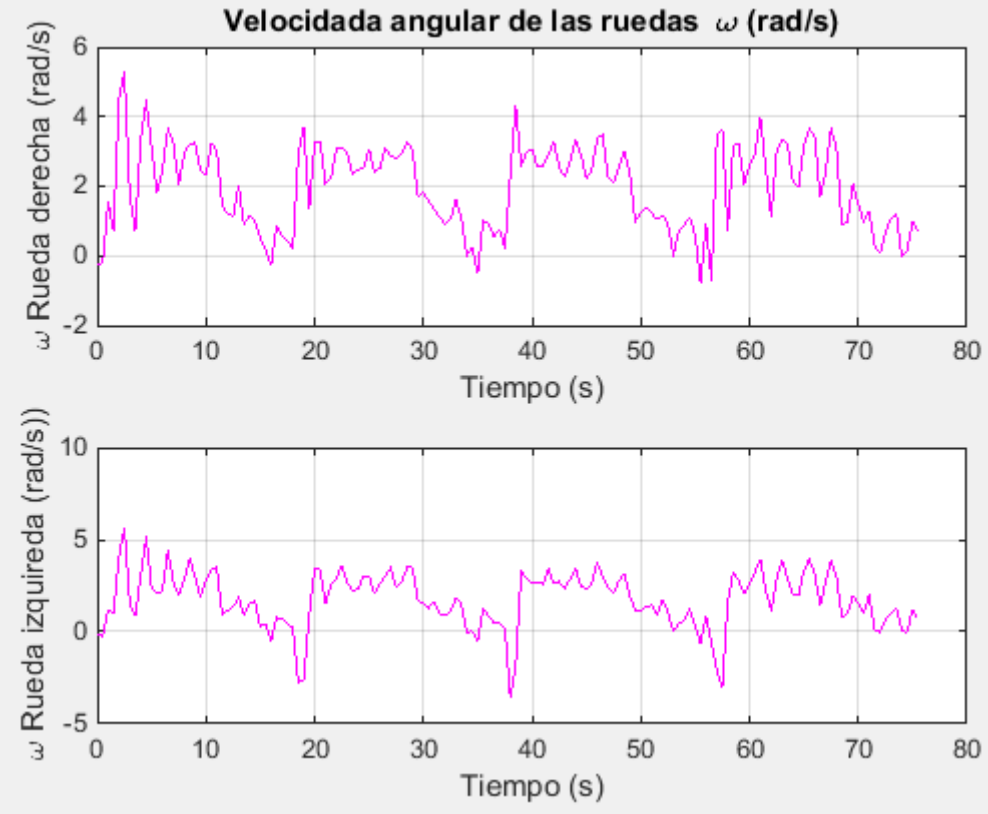

Figura 62. Velocidad angular de las ruedas mientras realiza un cuadrado.

En la figura 62 se muestra la velocidad angular de cada rueda durante la realización de la trayectoria por parte del robot. En estas se diferencian claramente 4 zonas, que se corresponden con los cuatro lados de la figura, donde la velocidad de las ruedas es más alta debido a que el robot se está moviendo hacia el objetivo. Aparecen unas zonas intermedias donde la velocidad es más pequeña que se corresponde a la realización de los giros. El valor de la velocidad de las ruedas oscila debido a que estas se encargan, además de realizar la trayectoria, de mantener el equilibrio.

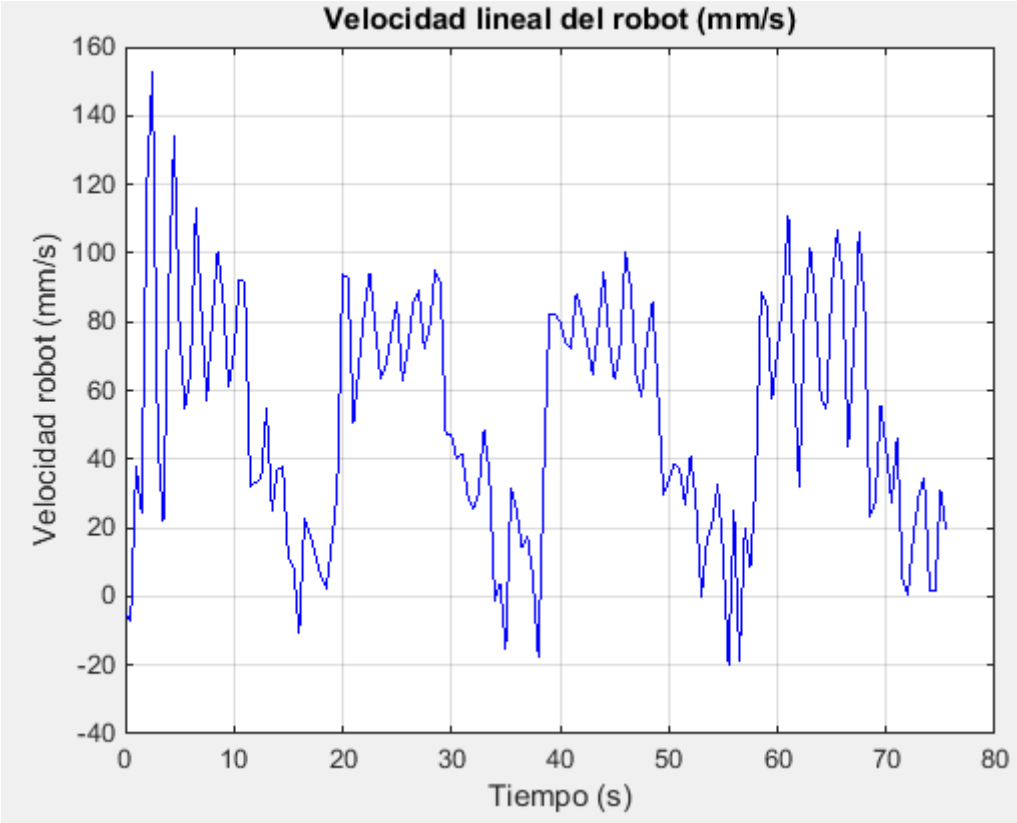

Figura 63. Velocidad lineal del robot durante la realización del cuadrado.

En la figura 63 se muestra la velocidad lineal del robot durante la realización de la trayectoria cuadrada. En esa gráfica se distinguen claramente como el robot comienza a moverse rápido hacia el objetivo y conforme se va acercando a este va disminuyendo su velocidad hasta que deja de avanzar, momento que se corresponde con la realización del giro o, en el último caso, cuando completa la trayectoria. Esto se repite 4 veces debido a que la figura que está realizando tiene 4 lados. También se puede ver que el robot solo alcanza una vez la velocidad de 150 mm/s que debería alcanzar cuando está lejos del objetivo, este valor fue el definido en el algoritmo de seguimiento de trayectoria. Sin embargo el valor de la velocidad oscila alrededor de los 80 mm/s, esto es debido a que el robot debe mantener el equilibrio. Como ya se dijo esta tarea tiene prioridad sobre el resto y por tanto no importa si se alcanza la velocidad de referencia o no mientras el robot siga manteniendo el equilibrio. Debido a que el robot debe realizar las dos tareas a la vez la velocidad no es constante sino que oscila.

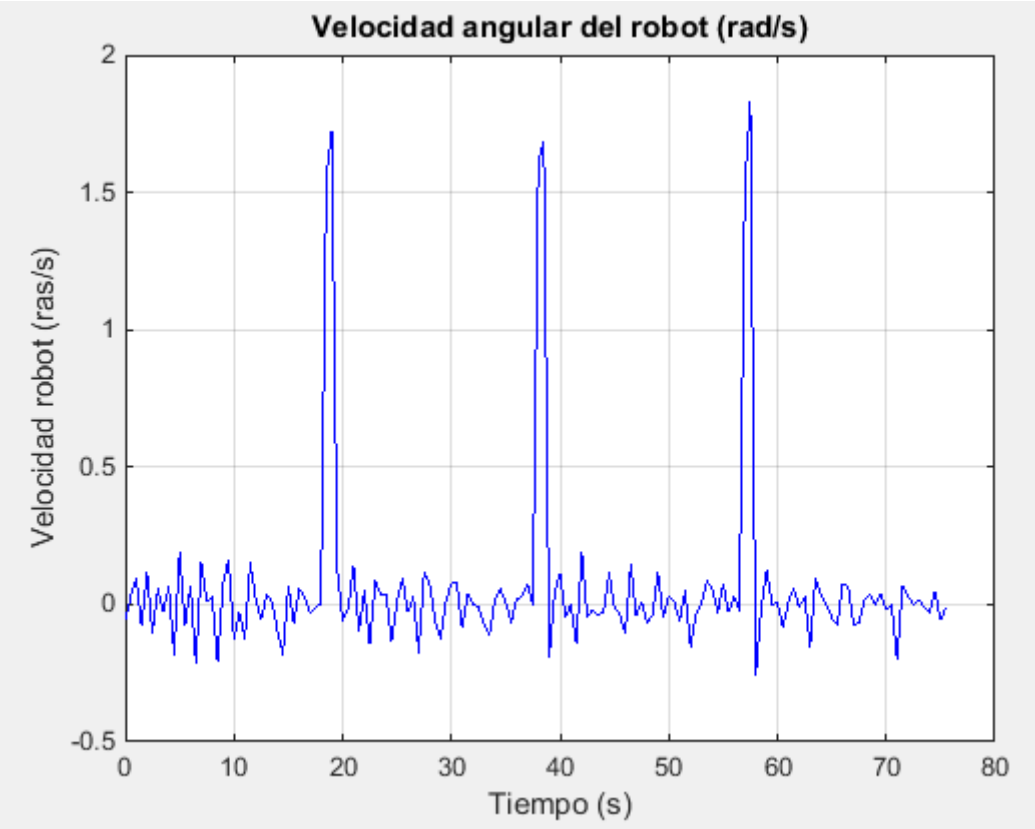

Figura 64.Velocidad angular del robot durante la realización del cuadrado.

En la figura 64 se muestra los valores de la velocidad angular del robot durante la realización del cuadrado. Se puede observar las zonas en las que el robot está avanzando y realizando pequeñas correcciones en su orientación, en estas la velocidad angular es pequeña, el valor oscila alrededor de 0. Se observan también tres picos de velocidad angular que se corresponden con la realización del robot de los giros de 1,57 radianes (90 grados) que debe realizar para completar la figura, el valor máximo de la velocidad angular es de 1,83 rad/s.

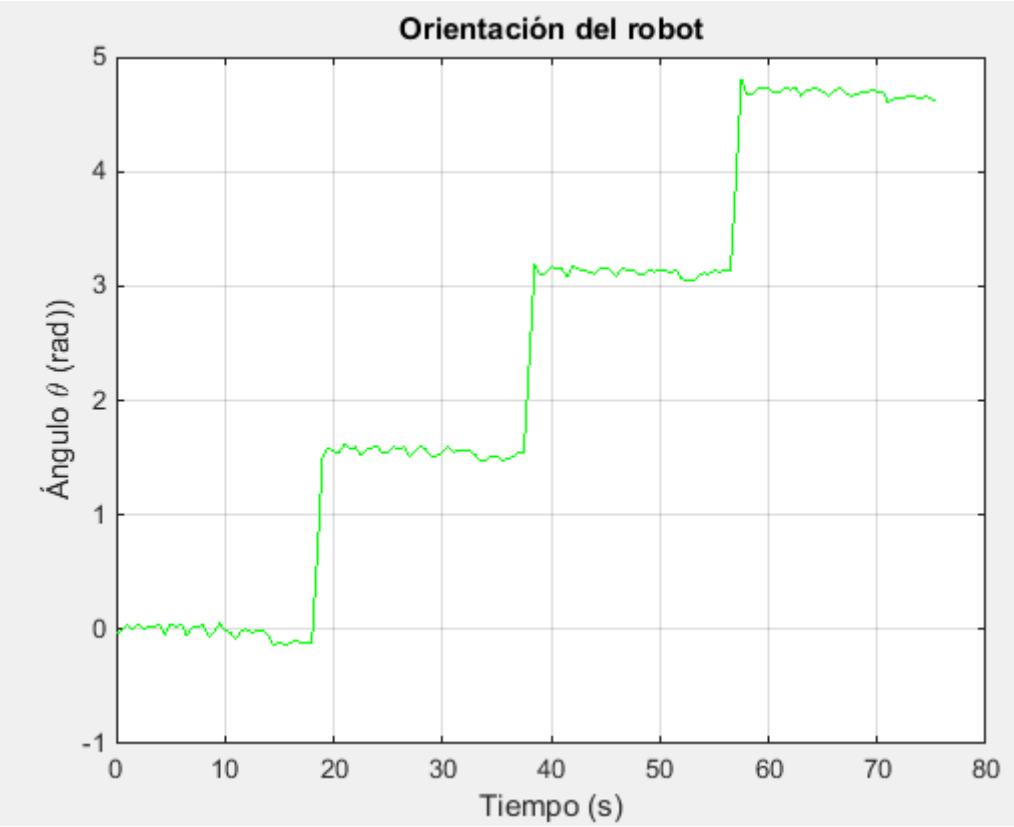

Figura 65. Orientación del robot durante la realización de la trayectoria cuadrada.

En la figura 65 se muestra el ángulo de orientación del robot. Se ven los 3 escalones correspondientes a los giros de 1,57 radianes (90 grados) que el robot realiza a lo largo de la trayectoria.

Trayectoria triangular

Se trata de una trayectoria triangular de 1000 mm de lado.

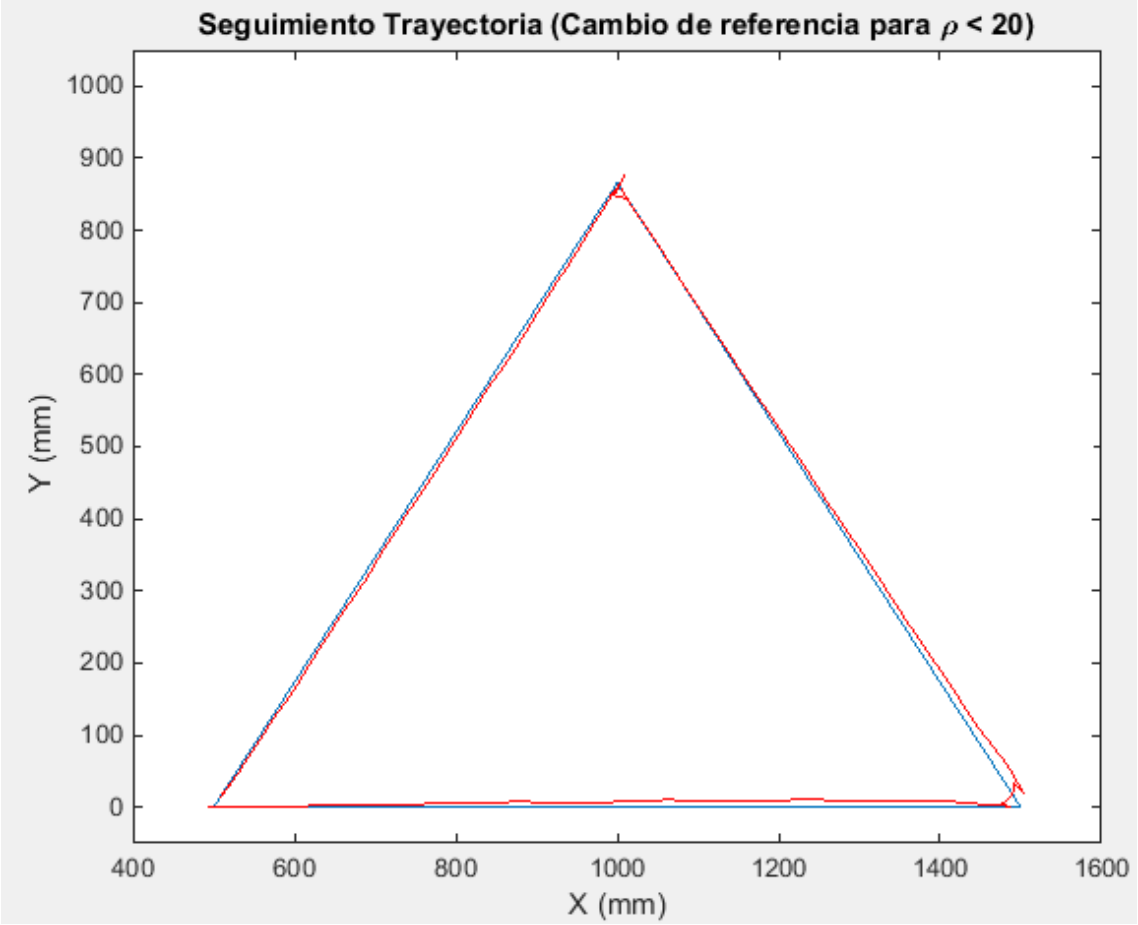

Figura 66.Posición del robot realizando triángulo.

En la figura 66 se ve como el robot tampoco comienza en la posición inicial, debido a que el robot está manteniendo el equilibrio antes de recibir la instrucción con la trayectoria a realizar. También se observa como el robot en ocasiones retrocede, esto es debido a que está manteniendo el equilibrio y esto tiene prioridad sobre realizar la trayectoria. Además se ve como el robot se va acercando hacia el punto de referencia corrigiendo su orientación. Se puede ver también como al llegar a cierta distancia de cada referencia el robot gira, debido a que se ha actualizado el valor de la referencia a seguir.

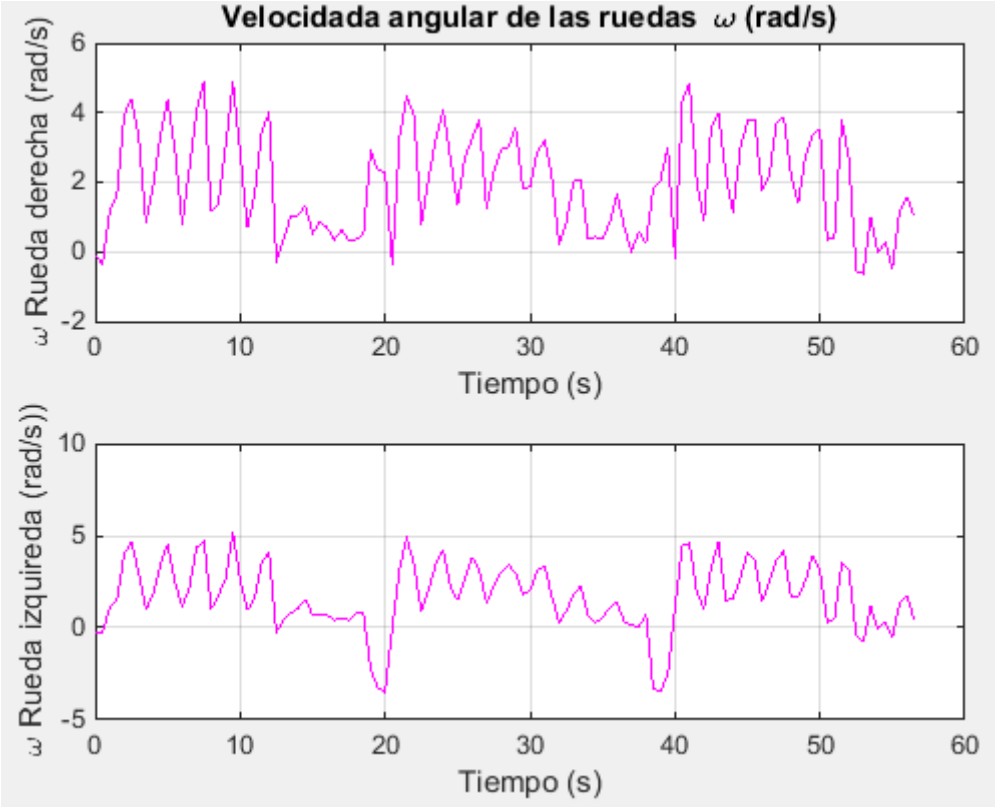

Figura 67. Velocidad angular de las ruedas mientras realiza un triángulo.

En la figura 67 se muestra la velocidad angular de cada rueda durante la realización de la trayectoria por parte del robot. En estas se diferencian claramente las 3 zonas correspondientes a los 3 lados de la figura. Mientras está realizando cada uno de los lados la velocidad de las ruedas es más alta debido a que el robot está avanzando hacia el objetivo. Aparecen unas zonas intermedias donde la velocidad es más pequeña que se corresponde a la realización de los giros, o en el último caso a que el robot ha finalizado la trayectoria. El valor de la velocidad de las ruedas oscila debido a que estas se encargan, además de realizar la trayectoria, de mantener el equilibrio.

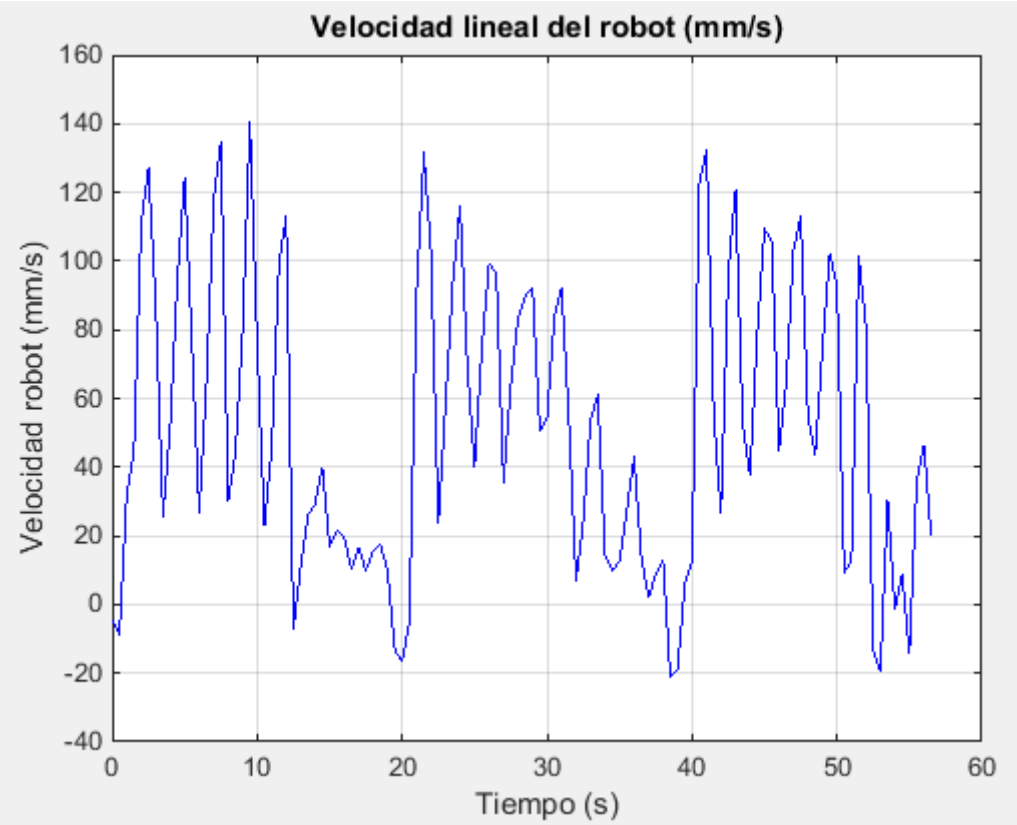

Figura 68. Velocidad lineal del robot durante la realización del triángulo.

En la figura 68 se muestra la velocidad lineal del robot durante la realización de la trayectoria triangular. En este caso se observa 3 zonas diferenciadas, una para cada lado de la figura. En estas el robot se mueve rápido hacia el objetivo y conforme se acerca a este se va disminuyendo la velocidad hasta que se deja de avanzar para realizar el giro o porque se ha completado la trayectoria. También se puede ver que el robot no alcanza una vez la velocidad de 150 mm/s que debería alcanzar cuando está lejos del objetivo. Sin embargo el valor de la velocidad oscila alrededor de los 80 mm/s, pero en este caso la amplitud de las oscilaciones son mayores que en la realización del cuadrado. La oscilación en la velocidad lineal del robot es debido a que el robot debe mantener el equilibrio. Como ya se dijo esto tiene prioridad sobre el resto de tareas.

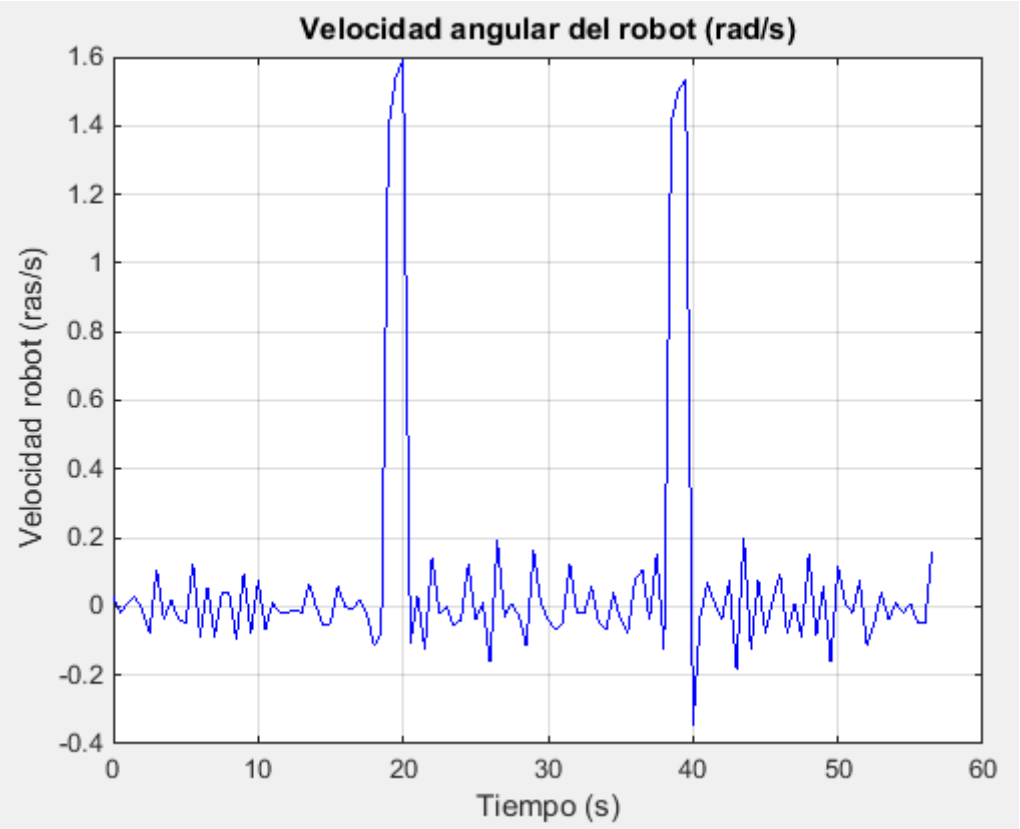

Figura 69. Velocidad angular del robot durante la realización del triángulo.

En la figura 69 se muestra los valores de la velocidad angular del robot durante la realización del triángulo. Se puede observar las zonas en las que el robot está avanzando y realizando pequeñas correcciones en su orientación, en estas la velocidad angular es pequeña, el valor oscila alrededor de 0. Se observan también dos picos de velocidad angular que se corresponden con la realización del robot de los giros de 2,0944 radianes (120 grados) que debe realizar para completar la figura, el valor máximo de la velocidad angular es de 1,591 rad/s.

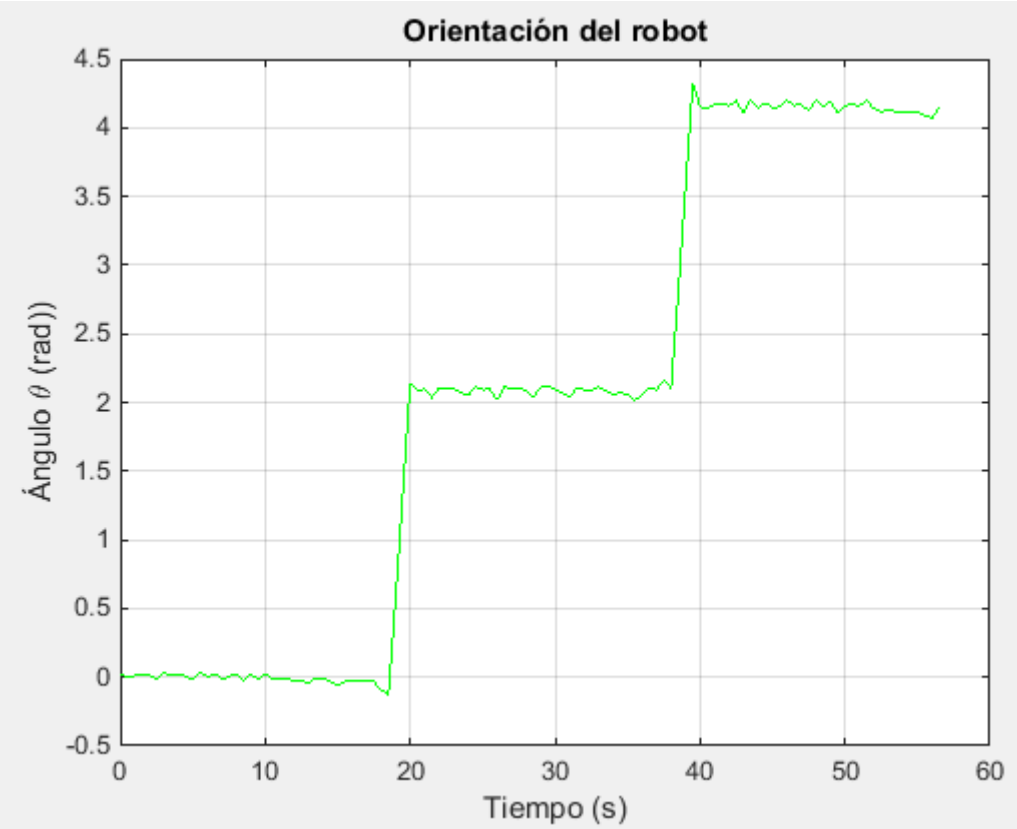

Figura 70. Orientación del robot durante la realización de la trayectoria triangular.

En la figura 70 se muestra el ángulo de orientación del robot. Se ven los 2 escalones correspondientes a los giros de 2,0944 radianes (120 grados) que el robot realiza a lo largo de la trayectoria.

Trayectoria hexagonal

Se trata de una trayectoria hexagonal de 500 mm de lado.

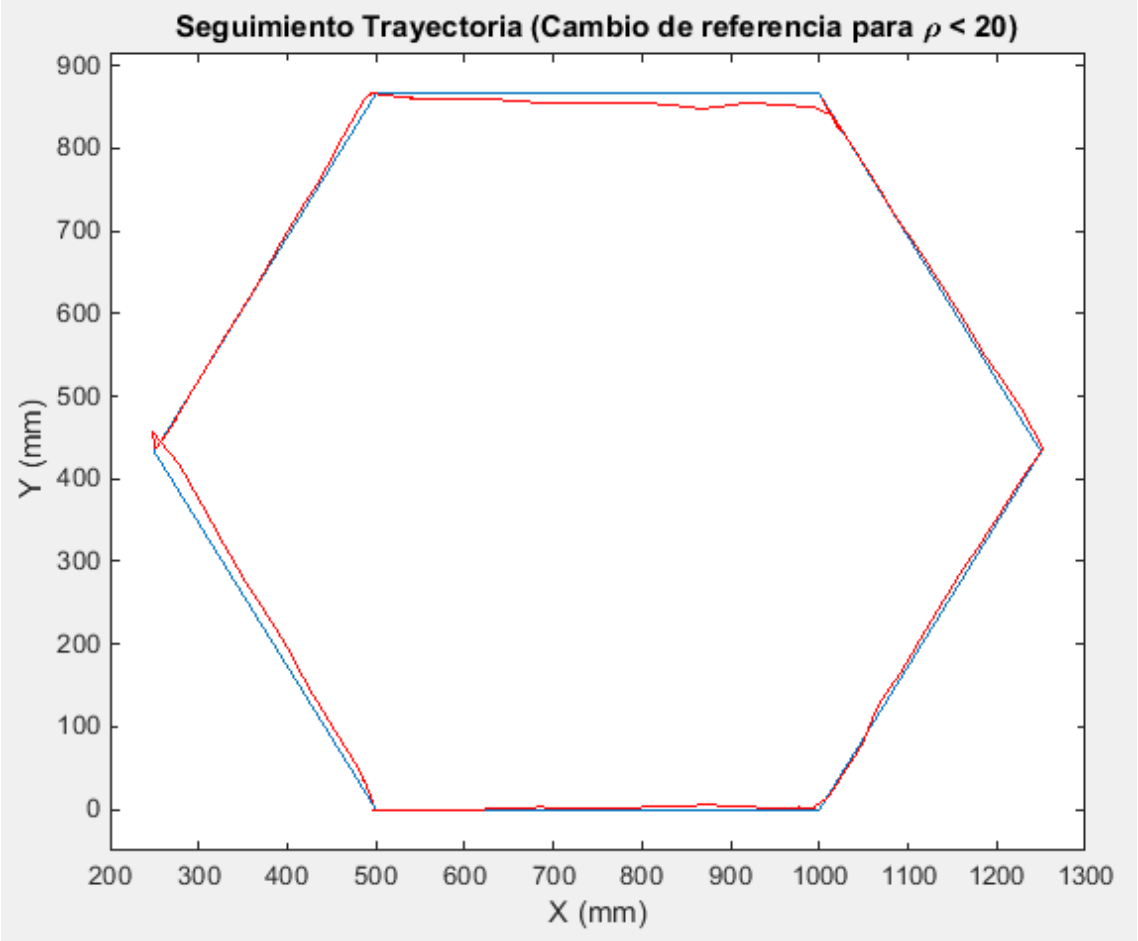

Figura 71. Posición del robot realizando el hexágono.

En la figura 71 se ve como el robot en ocasiones retrocede, sobre todo cuando realiza los giros, esto es debido a que está manteniendo el equilibrio y esto tiene prioridad sobre realizar la trayectoria. Además se ve como el robot se va acercando hacia el punto de referencia corrigiendo su orientación. Se puede ver también como al llegar a cierta distancia de cada referencia el robot gira, debido a que se ha actualizado el valor de la referencia a seguir.

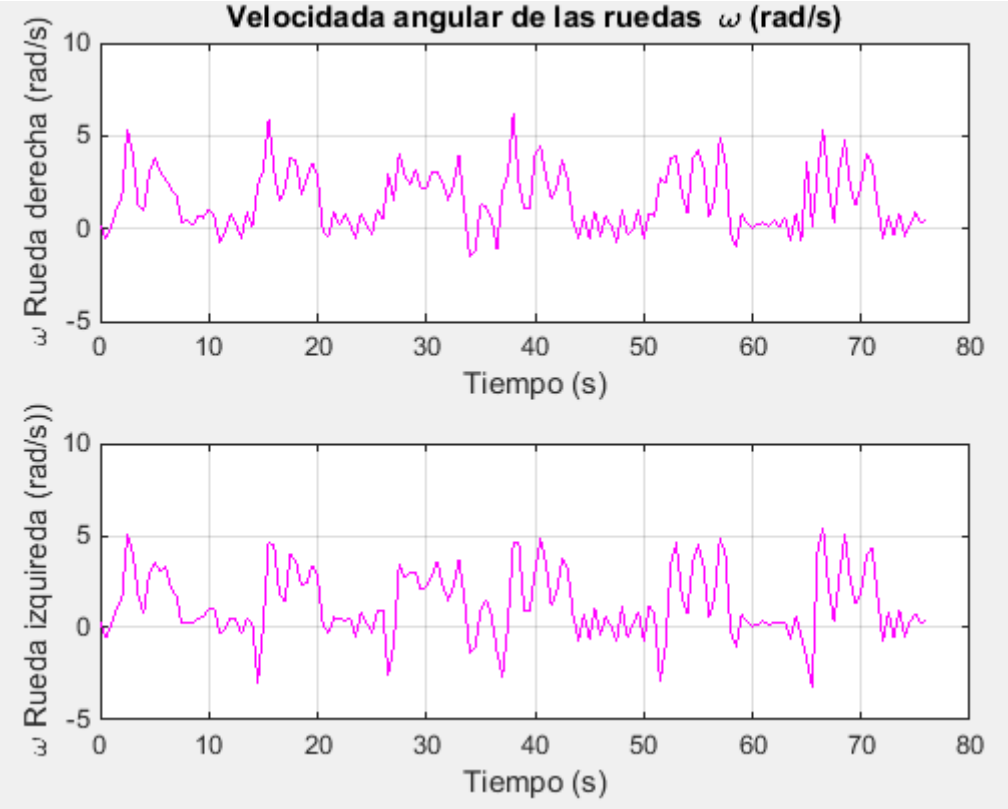

Figura 72. Velocidad angular de las ruedas mientras realiza un hexágono.

En la figura 72 se muestra la velocidad angular de cada rueda durante la realización de la trayectoria por parte del robot. En estas se diferencian claramente las 6 zonas correspondientes a los 6 lados de la figura. Mientras está realizando cada uno de los lados la velocidad de las ruedas es más alta debido a que el robot está avanzando hacia el objetivo. Aparecen unas zonas intermedias donde la velocidad es más pequeña que se corresponde a la realización de los giros, o en el último caso a que el robot ha finalizado la trayectoria. El valor de la velocidad de las ruedas oscila debido a que estas se encargan, además de realizar la trayectoria, de mantener el equilibrio.

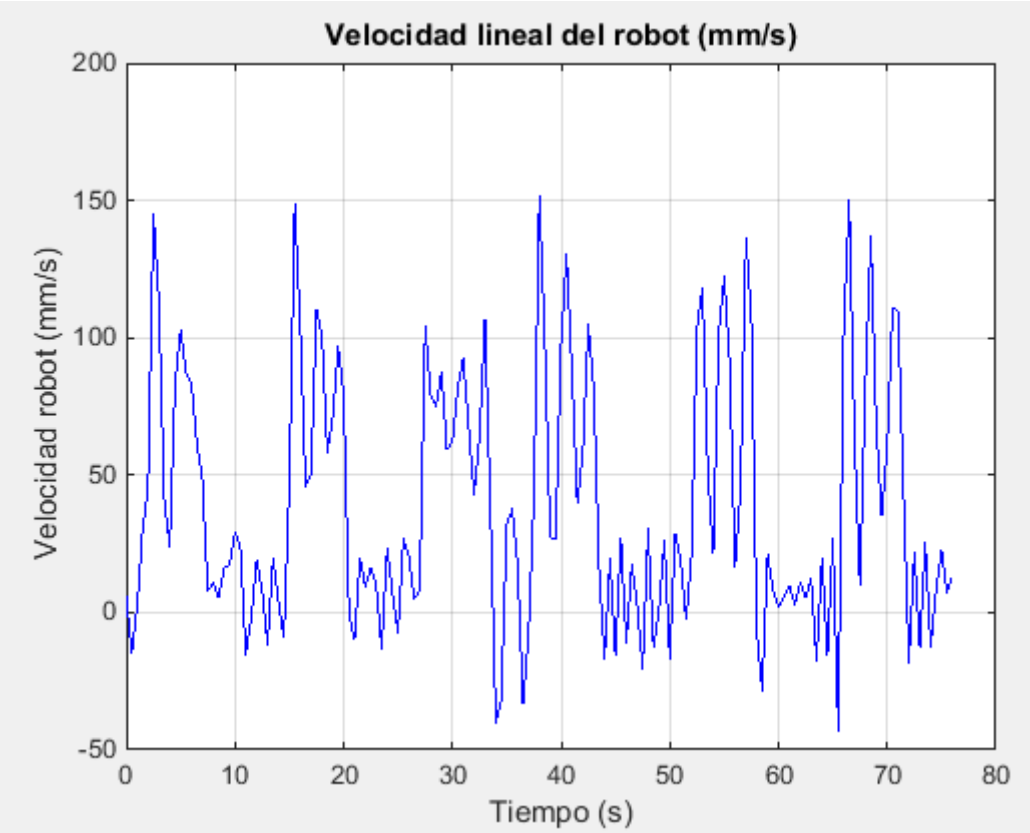

Figura 73. Velocidad lineal del robot durante la realización del hexágono.

En la figura 73 se muestra la velocidad lineal del robot durante la realización de la trayectoria triangular. En este caso se observa 6 zonas diferenciadas, una para cada lado de la figura. En estas el robot se mueve rápido hacia el objetivo y conforme se acerca a este se va disminuyendo la velocidad hasta que se deja de avanzar para realizar el giro o porque se ha completado la trayectoria. También se puede ver que el robot alcanza los 150 mm/s en tres momentos puntuales y no durante todo el trayecto mientras el robot está alejado del objetivo. Sin embargo el valor de la velocidad oscila alrededor de los 80 mm/s. El número de oscilaciones es menor que en las dos trayectorias analizadas anteriormente, esto es debido a que el los lados de las figuras son más pequeños y por tanto tarda menos en acercarse al objetivo. La oscilación en la velocidad lineal del robot es debido a que el robot debe mantener el equilibrio. Como ya se dijo esto tiene prioridad sobre el resto de tareas.

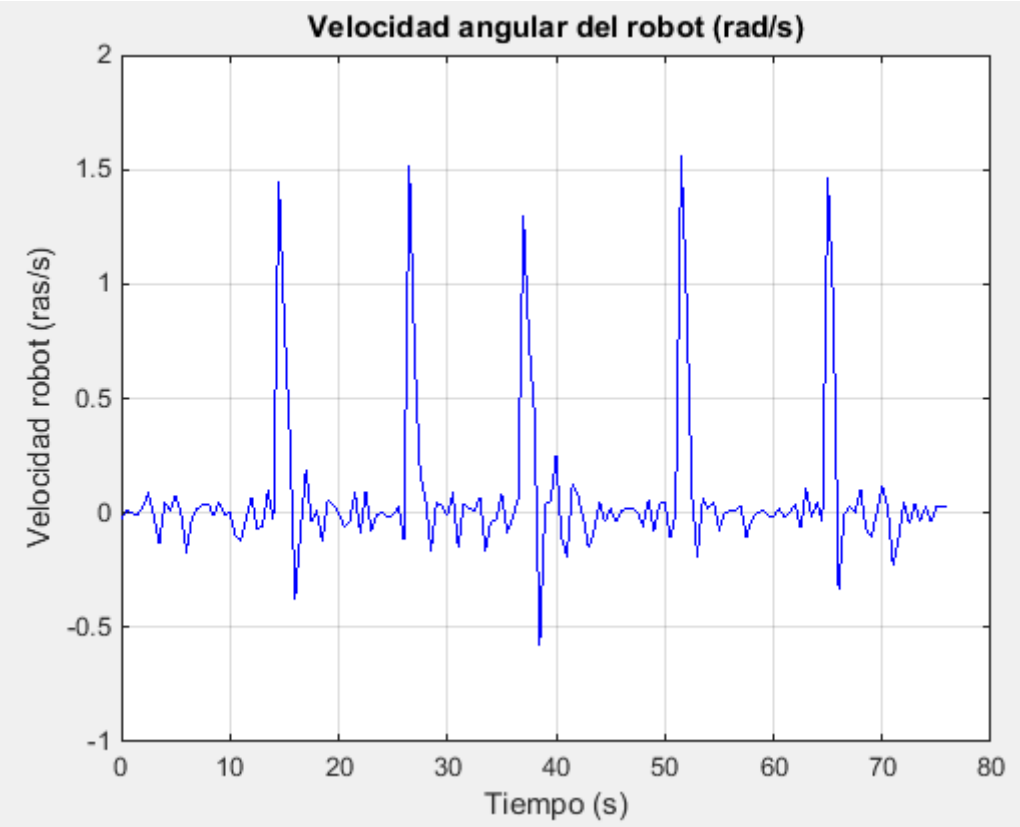

Figura 74. Velocidad angular del robot durante la realización del hexágono.

En la figura 74 se muestra los valores de la velocidad angular del robot durante la realización del hexágono. Se puede observar las zonas en las que el robot está avanzando y realizando pequeñas correcciones en su orientación, en estas la velocidad angular es pequeña, el valor oscila alrededor de 0. Se observan también 5 picos de velocidad angular que se corresponden con la realización del robot de los giros de 1,0427 radianes (60 grados) que debe realizar para completar la figura, el valor máximo de la velocidad angular es de 1,562 rad/s.

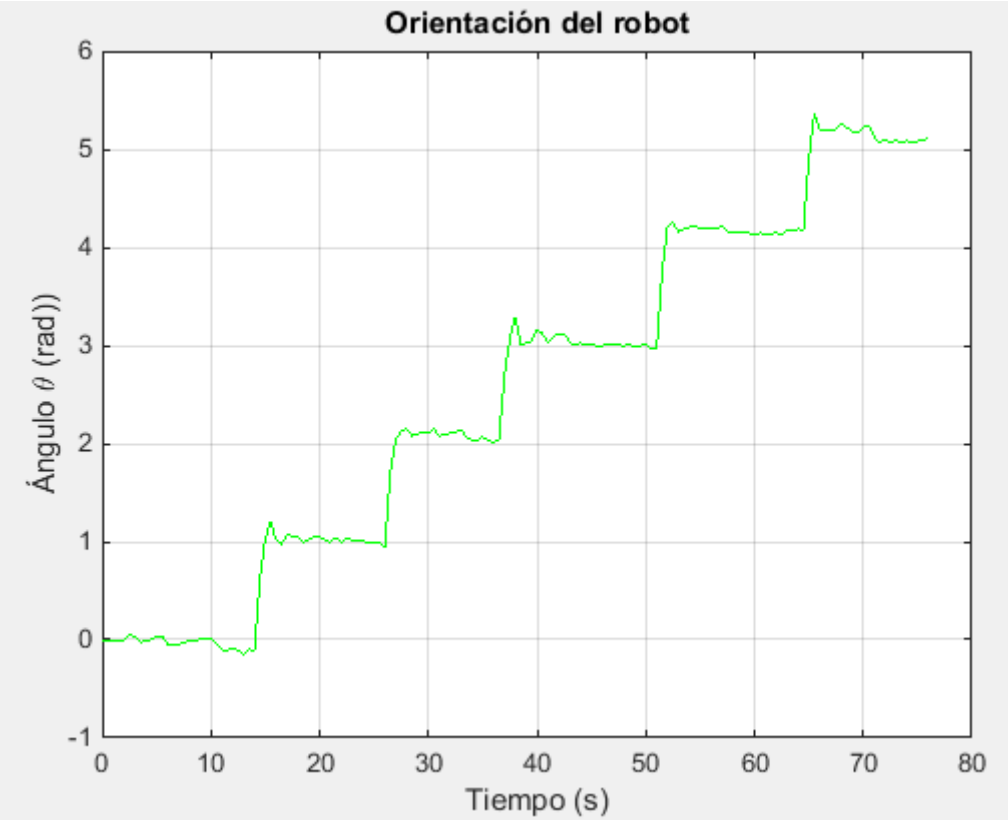

Figura 75. Orientación del robot durante la realización de la trayectoria hexagonal.

En la figura 75 se muestra el ángulo de orientación del robot. Se ven los 6 escalones correspondientes a los giros de 1,0427 radianes (60 grados) que el robot realiza a lo largo de la trayectoria.

A pesar de que existen algunos errores en la realización de la trayectoria, el robot realiza la trayectoria de manera satisfactoria. Los resultados obtenidos son satisfactorios teniendo en cuenta la complejidad de la tarea realizada, mantener el equilibrio del robot mientras se realiza una trayectoria cerrada, y que para la localización del robot solo se utilizaron los encoders de las ruedas.

6. Presupuesto.

En el presupuesto se muestran los gastos de los materiales, licencias de programas y mano de obra, necesaria para llevar a cabo el proyecto. Como el proyecto no tiene un fin empresarial no se tendrá en cuenta los impuestos (en el precio de los materiales adquiridos se incluye el IVA) ni los beneficios.

En las horas de mano de obra se incluyen todas las horas dedicadas a la realización del trabajo. Estas se dividen en varias tareas:

- Familiarizarse con el entorno de programación elegido. Nunca se había trabajado con el software de LabView por lo que se requirió de un esfuerza adicional para entender el funcionamiento, las diferentes posibilidades de programación que presenta el programa, las funciones que presenta para trabajar con Lego, etc. En definitiva tiempo necesario para aprender a trabajar con LabView.

- Horas de investigación. Destinadas a estudiar las diferentes posibilidades para desarrollar el proyecto, a obtener información sobre el control del péndulo invertido, algoritmos de seguimientos de trayectoria, configuración diferencial de un robot móvil, etc.

- Realización del montaje del robot educacional tipo segway haciendo uso del kit educacional de Lego Mindstorms EV3.

- Programación y pruebas de los diferentes algoritmos implementados para lograr el correcto funcionamiento de la aplicación.

En la tabla 13 se muestran los diferentes trabajos realizados para llevar a cabo el proyecto. En estos se incluyen los tiempos destinados a cada uno y el precio.

Concepto	Cantidad (horas)	Precio unitario (€/horas)	Precio (€)	Total (€)
Pruebas con LabView	60	20	1200	1200
Investigación	150	20	3000	4200
Montaje	10	20	200	4400
Programación y pruebas	280	20	5600	10000
TOTAL	500 horas			10000 €

Tabla 13. Diferentes tareas realizadas y con el tiempo dedicado a cada una.

En la tabla 14 se muestra el presupuesto total del proyecto. Este es el presupuesto que constaría realizar un primer ejemplar. Si se quisieran realizar más ejemplares el precio se reduciría considerablemente, debido a que las horas de mano de obra se reducirían. Esta reducción seria debido a que ya que ya se sabe trabajar con LabView, no harían falta tantas horas de investigación ni de programación y el montaje se realizaría más rápido.

SEGWAY BASADO EN LEGO MINDSTORMS EV3			
Concepto	**Cantidad**	**Precio**	**TOTAL**
Kit Lego Mindstorms EV3 Educacional	1	477,69 €	477,69 €
Licencia LabView 2016 completo	1	3451,00 €	3928,69 €
Mano de obra	500	10000 €	4928,69 €
TOTAL			13928,69 €

Tabla 14. Presupuesto para la realización del proyecto.

7. Conclusiones y posibles mejoras.

En el presente proyecto se ha construido un robot tipo segway utilizando los elementos del kit Lego Mindstorms EV3, este objetivo se realizó satisfactoriamente, ya que se logró construir el robot.

Para lograr la realización de las tareas propuestas se diseñaron por separado el control de estabilidad, que permite al robot mantener la estabilidad. Para ello se implementó un control por realimentación de estados. Mediante este control el robot es capaz de mantener el equilibrio, incluso soporta perturbaciones externas moderadas sin perder el equilibrio.

El control de trayectoria, que permite al robot realizar trayectorias previamente conocidas, y el control de dirección que permite que el robot se dirija hacia una determinada dirección, se diseñó e implementó de manera satisfactoria, ya que se logró una realización correcta de ambas tareas por parte del robot.

Posteriormente se diseñó la arquitectura de control para establecer las relaciones entre los controles diseñados por separado, para obtener el control completo. Con esto se logró el objetivo de que el robot realizara trayectorias o siguiera una determinada dirección mientras mantiene el equilibrio.

Para poder controlar todas las funciones planteadas de forma remota se realizó la interfaz de usuario. Para esto se tuvo que implementar un programa en el PC para controlar las funciones del robot de manera remota. Para poder gestionar el envío de información entre el PC y la EV3, que se realiza mediante diferentes mensajes enviados por bluetooth, se hizo uso de los diferentes Mailbox. Con esto se logra poder mandar las órdenes con la tarea a realizar al robot de manera remota, y además ver información sobre la realización de estas por parte del robot en la pantalla del PC.

En definitiva, se ha logrado que el robot realice trayectorias previamente conocidas y que se mueva en la dirección deseada por el usuario mientras el robot mantiene el equilibrio, y todo esto se controla de forma remota desde el PC. Por tanto se han alcanzado los objetivos del trabajo fin de grado de manera satisfactoria, ya que se han logrados todos los objetivos planteados al inicio de este.

A lo largo del desarrollo del proyecto aparecieron diferentes problemas, que se fueron subsanando de la mejor manera posible para lograr el objetivo del proyecto. En el control de equilibrio el sensor giroscopio presenta un error de deriva. Este se calcula para tenerlo en cuenta en los cálculos posteriores, pero en ocasiones el error es demasiado grande y el robot no comienza a mantener el equilibrio. Para reducir la probabilidad de que este error se produzca se añadió una espera antes de realizar el cálculo del error del giroscopio, para asegurar que el robot está totalmente quieto y en la posición de equilibrio cuando se realiza dicho calculo.

En la navegación del robot hay que tener en cuenta que el robot debe mantener el equilibrio en todo momento y por tanto la navegación del robot es una tarea segundaria y debe realizarse siempre que se esté manteniendo el equilibrio. Por este motivo algoritmo de punto descentralizado diseñado para la realización de trayectorias era demasiado exigente. Los cambios de referencian se hacían demasiado rápido, sin importar la localización real de robot, y las ruedas no podían seguir las velocidades de referencias ya que estas se encargan, además de seguir la trayectoria, de mantener el equilibrio. Por esto el robot no conseguía realizar las trayectorias. Además la acción de control aumentaba para lograr alcanzar las velocidades de referencia y esto desestabilizaba el robot, que comenzaba a balancearse bruscamente hasta que perdía el equilibrio. Para solucionar esto se diseñó el algoritmo de persecución pura. Este es menos exigente para el robot, ya que este se va moviendo con cierta velocidad hacia el punto objetivo, sin importar demasiado si el robot alcanza o no la velocidad de referencia. Cada cierto tiempo el robot se pregunta dónde está y hacia dónde debe ir y calcula la nueva velocidad de referencia. Esta depende de la distancia del robot al punto objetivo. Cuando el robot está lo suficientemente cerca del objetivo se cambia el valor de la referencia a alcanzar. Por tanto el nuevo algoritmo soluciona el problema al no ser tan exigente como el de punto descentralizado. Este algoritmo no exige al robot realizar la trayectoria en cierto tiempo, ni tampoco alcanzar las velocidades de referencia, por tanto permite que el robot se pare o incluso que retroceda para mantener el equilibrio, y que vaya realizando la trayectoria a la velocidad que le sea posible.

En la interfaz de usuario se intentó poder iniciar el programa que se ejecuta en la EV3 desde el PC. Para ello LabView dispone del bloque “compile and run NXT code” que permite compilar y ejecutar un programa en el ladrillo de Lego. También tiene la función “Run Program” que se utiliza para ejecutar un programa que se ha cargado previamente en el ladrillo. Se probaron ambas funciones sin lograr el objetivo. Para intentar solucionarlo se añadió una espera suficientemente larga para dar tiempo al programa a comenzar a ejecutarse, sin embargo en todas las pruebas realizadas, con ambos bloques de función, el ladrillo dejaba de responder y se tuvo que reiniciar.

Para un trabajo futuro se podría realizar ciertas mejoras como mejorar la localización del robot, para lograr que esta sea más precisa. Para esto se podría unir a la estimación local de la posición, que se utiliza en este trabajo, un sistema

de posicionamiento global (GPS o cámara cenital) que vaya dando la información sobre la posición global, y se utilice para corregir la estimación local.

También se podría aprovechar el sensor de ultrasónico y el sensor de color que forman parte del robot y, que a pesar de no utilizarse se mantuvieron para respetar el modelo del robot. Mediante el sensor ultrasónico se podría implementar una función para evitar obstáculos que el robot encontrase en el camino durante la realización de alguna trayectoria o siguiendo alguna dirección. Con el sensor de color se podría implementar una función de parada de emergencia al detectar un color determinado el robot dejara de realizar las tareas y solo se mantuviera en equilibrio en el sitio donde se detectó el color. Ambos sensores están en los brazos del robot y pueden moverse utilizando el motor mediano.

También se podría añadir botón de parada de emergencia en la interfaz de usuario, que provocara el cese de la tarea que estuviera realizando el robot y que se mantuviera en equilibrio en el sitio. En este caso solo serviría para interrumpir la realización de las trayectorias, ya que el robot para de seguir una dirección cuando se deja de pulsar el botón.

Otra opción sería construir un mando físico que permitiera el control remoto del robot de manera inalámbrica, esto se podría implementar mediante arduino. También se podría definir en la pantalla del PC los diferentes puntos de referencia a los que el robot debe dirigirse para realizar otras trayectorias.

Todas las mejoras citadas anteriormente son una buena opción para ampliar y mejorar la aplicación diseñada. Estas mejoras no se realizaron por falta de tiempo, pero son un punto de partida para posibles trabajos futuros.

8. Bibliografía.

- Antecedentes y datos sobre el segway:

1. https://es.wikipedia.org/wiki/Segway

Características e historia de Lego:

2. https://juegosrobotica.es/lego-mindstorm/

3. https://droidecomunidad.com/producto/kit-basico-lego-mindstorms-education-ev3/

4. http://sine.ni.com/nips/cds/view/p/lang/es/nid/212785

Instrucciones de montaje del robot tipo segway y guía de usuario Lego:

5. https://education.lego.com/en-us/support/mindstorms-ev3/building-instructions

6. https://www.lego.com/r/www/r/mindstorms/-/media/franchises/mindstorms%202014/downloads/user%20guides/user%20guide%20lego%20mindstorms%20ev3%2011%20all%20es.pdf?l.r2=1604863065.

Información sobre sensores y actuadores:

7. https://automotrizenvideo.com/que-son-sensores-y-actuadores/

8. http://www.esmindstorms.com/2015/07/sensor-gyro-ev3.html

9. http://ro-botica.com/tienda/LEGO-Education

Opciones de programación del Lego y opciones de segway:

10. https://juegosrobotica.es/instrucciones-lego-mindstorms-ev3/

11. http://www.nxtprograms.com/NXT2/segway/

12. https://es.mathworks.com/matlabcentral/fileexchange/19147-nxtway-gs-self-balancing-two-wheeled-robot-controller-design seg2

13. https://www.roboticastreet.com/construye-tu-propio-segway-en-30-minutos

14. https://www.lego.com/es-es/mindstorms/support

15. http://blog.electricbricks.com/2016/03/programacion-del-ev3/

16. https://es.wikipedia.org/wiki/LabVIEW

17. https://es.mathworks.com/hardware-support/lego-mindstorms-matlab.htm

Cable de conexiones

18. https://es.wikipedia.org/wiki/RJ-12

19. https://es.wikipedia.org/wiki/Lego_Mindstorms

Información sobre segway Lego y localización del robot

20. Learning Lego Mindstorms EV3 : build and create interactive, sensor-based robots using your Lego Mindstorms EV3 kit

21. https://es.wikipedia.org/wiki/Odometr%C3%ADa

22. https://www.tamps.cinvestav.mx/~mgomez/Control_Lineal/node6.html robot diferencial

23. https://e-archivo.uc3m.es/handle/10016/11629

24. http://www.redalyc.org/html/849/84920503023/index.html

Identificación de sistemas y diseño de reguladores:

25. Control automático. Tiempo continuo y tiempo discreto.