UNIVERSIDAD POLITÉCNICA SALESIANA
SEDE GUAYAQUIL
CARRERA DE INGENIERÍA ELECTRÓNICA

DISEÑO E IMPLEMENTACIÓN DE UN CONTROL DIFUSO PARA UN ROBOT SEGUIDOR DE LINEA CON AUTOBALANCEO UTILIZANDO LEGO MINDSTORMS

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO ELECTRÓNICO

AUTORES: **CARTAGENA IZQUIERDO JULIO ANDRES**
RAMIREZ DE LA TORRE JOSUE ALBERTO

TUTOR: **ING. BYRON LIMA CEDILLO MSc.**

Guayaquil - Ecuador

2022

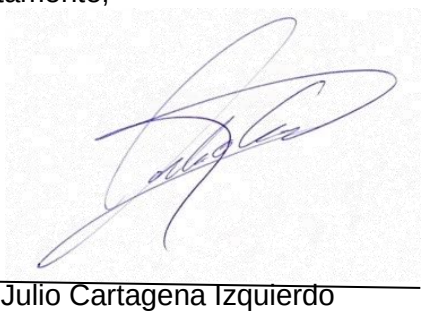
CERTIFICADO DE RESPONSABILIDAD Y AUTORÍA DEL TRABAJO DE TITULACIÓN

Nosotros, Julio Andres Cartagena Izquierdo con documento de identificación N° 0917246860 y Josué Alberto Ramírez De la Torre con documento de identificación N° 0931731707; manifestamos que:

Somos los autores y responsables del presente trabajo; y, autorizamos a que sin fines de lucro la Universidad Politécnica Salesiana pueda usar, difundir, reproducir o publicar de manera total o parcial el presente trabajo de titulación.

Guayaquil, 5 de marzo del 2022

Atentamente,



Julio Cartagena Izquierdo
CI: 0917246860

Josué Ramírez de la Torre
CI:0931731707

CERTIFICADO DE CESIÓN DE DERECHOS DE AUTOR DEL TRABAJO DE TITULACIÓN A LA UNIVERSIDAD POLITÉCNICA SALESIANA

Nosotros, Julio Andres Cartagena Izquierdo con documento de identificación N° 0917246860 y Josué Alberto Ramírez De la Torre con documento de identificación N° 0931731707, expresamos nuestra voluntad y por medio del presente documento cedemos a la Universidad Politécnica Salesiana la titularidad sobre los derechos patrimoniales en virtud de que somos autores del Trabajo de titulación: **DISEÑO E IMPLEMENTACIÓN DE UN CONTROLADOR DIFUSO PARA UN ROBOT SEGUIDOR DE LINEA CON AUTOBALANCE, UTILIZANDO LEGO MINDSTORMS**, el cual ha sido desarrollado para optar por el título de Ingeniero Electrónico, en la Universidad Politécnica Salesiana, quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente.

En concordancia con lo manifestado, suscribimos este documento en el momento que hacemos la entrega del trabajo final en formato digital a la Biblioteca de la Universidad Politécnica Salesiana.

Guayaquil, 5 de marzo del 2022

Atentamente,

Julio Cartagena Izquierdo
CI: 0917246860

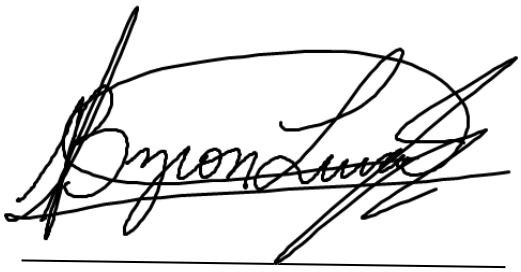
Josué Ramírez de la Torre
CI: 0931731707

CERTIFICADO DE DIRECCIÓN DEL TRABAJO DE TITULACIÓN

Yo, **ING. BYRON LIMA CEDILLO, MSc.** con documento de identificación N° 0921971768 , docente de la Universidad Politécnica Salesiana , declaró que bajo mi tutoría fue desarrollado el trabajo de titulación: **"DISEÑO E IMPLEMENTACIÓN DE UN CONTROLADOR DIFUSO PARA UN ROBOT SEGUIDOR DE LINEA CON AUTOBALANCE, UTILIZANDO LEGO MINDSTORMS"**, realizado por Julio Andres Cartagena Izquierdo con documento de identificación N° 0917246860 y por Josué Alberto Ramírez De la Torre con documento de identificación N° 0931731707, obteniendo como resultado final el trabajo de titulación bajo la opción Proyecto de titulación que cumple con todos los requisitos determinados por la Universidad Politécnica Salesiana.

Guayaquil, 5 de marzo de 2022

Atentamente,



Ing. Byron Lima Cedillo, MSc.

DIRECTOR DEL PROYECTO

DEDICATORIA

Dedico este Proyecto técnico a Dios por permitirme llegar hasta este momento

A mis padres por apoyarme en todo momento en este periodo universitario, por enseñarme que en la vida se puede alcanzar y lograr todo lo que nos propongamos a pesar de las dificultades que es presenten a lo largo del camino

A mis hermanos con los cuales comparto la alegría de alcanzar esta meta

A mis abuelitos Blanca Cordova y Emilio De La Torre los cuales tuve su apoyo en todo momento y sé que este momento es especial para ellos como para mi

A mis amigos y compañeros porque sin los equipos de trabajo que formamos no hubiéramos logrado este objetivo

Josue Alberto Ramirez De La Torre

DEDICATORIA

Dedico este trabajo de titulación en primer lugar a Dios por haberme guiado por el camino de la felicidad hasta ahora, en segundo lugar, a cada uno de los miembros que son parte de mi familia, a Pepita Cartagena, mi MADRE, a Julio Cartagena, mi padre, a mi segunda madre Pepa Izquierdo MI ABUELA, mis terceras madres y no menos importantes MIS TIAS Alex y Gisella, a mis hermanos y mi sobrina. Dedico este trabajo con todo mi amor y cariño a MI ESPOSA Scarley ya que con sus consejos me ha brindado ese respaldo incondicional para culminar esta etapa de mi vida.

A mi tutor Ing Byron Lima que no desistió nunca, aún sin importar que no salía como esperábamos la pruebas y no lográbamos reunirnos por cuestiones de tiempo. A mi compañero Josue con el que nos pasamos más de unas noches sufriendo, pero lo logramos. A la Universidad por todo lo aprendido y compañeros de clases por todas las experiencias.

Julio Andres Cartagena Izquierdo

AGRADECIMIENTO

En primer lugar, doy gracias a Dios por permitirme llegar a culminar esta etapa tan bonita que llamamos Universidad. Gracias a mi madre por las incontables veces que estuvo para apoyarme y guiarme. Tus esfuerzos han sido impresionantes, tu amor y dedicación es para mí es invaluable porque sé que ella me ayudó y lo sigue haciendo, además de haberme dado la vida, siempre confío en mí y nunca me abandonó. Gracias a mis abuelos, que han sido un apoyo constante para lograr todos mis objetivos y han luchado a mi lado todo este recorrido. Gracias a mis tías que siempre me han ayudado en todo lo que necesitara, por su apoyo y por demostrar la fe que tienen en mí. Gracias a mis hermanos que con cada una de sus palabras me hacían sentir orgulloso de lo que soy y de lo que les puedo enseñar. Gracias a mi esposa y amiga por estar conmigo en todos aquellos momentos en que el estudio y el esfuerzo ocupaban todo mi tiempo y esfuerzo, por ser ese apoyo incondicional en mi vida, que con su amor, respaldo y ánimos que me brinda, he logrado alcanzar nuevas metas, tanto profesionales como personales.

Y por supuesto a mi tutor, a las autoridades que conformaron la Universidad, por permitirme concluir con una etapa de mi vida, gracias por la paciencia, orientación y guiarme en el desarrollo de esta investigación.

Julio Andres Cartagena Izquierdo

RESUMEN

AÑO	ALUMNOS	DIRECTOR DE TESIS	TEMA DE PROYECTO TÉCNICO
2022	Julio Andrés Cartagena Izquierdo Josué Alberto Ramírez de la Torre	Ing. Byron Lima Cedillo, MSc.	“DISEÑO E IMPLEMENTACIÓN DE UN CONTROL DIFUSO PARA UN ROBOT SEGUIDOR DE LINEA CON AUTOBALANCEO UTILIZANDO LEGO MINDSTORMS”

El presente proyecto técnico nace de la necesidad de utilizar un equipo que hasta el momento solo se usaba para competencias y aprendizaje bajo una herramienta proporcionada por Lego. Este proyecto tiene como objetivo la implementación de un control difuso en la plataforma de National Instruments para un robot seguidor de línea auto balanceado utilizando el kit de robótica de Lego Mindstorms, se lo realiza para las instalaciones del club de robótica de la Universidad Politécnica Salesiana, mediante el cual los estudiantes de la carrera se familiarizan con la programación de algoritmos de control y el manejo del kit Lego Mindstorms para poder realizar prácticas utilizando lógica difusa para la programación del robot balancín con el sistema de péndulo invertido.

El objetivo principal de la implementación del robot balancín es que los estudiantes aprendan el uso de la lógica difusa para la programación de algoritmos de control mediante las prácticas intermedias y avanzadas que contribuyan al conocimiento y al aprendizaje para los diferentes tipos de diseños que se pueden realizar con el kit de Lego Mindstorms acorde a la necesidad del proyecto.

Adicionalmente este trabajo de titulación cuenta con un bloque de control del kit en el cual tiene cargado el programa, dos motores de fuerzas, una estructura a base de las piezas del kit que ha sido diseñada para la mejor estabilidad del robot, un giroscopio para detectar los cambios de inclinación del robot mientras realiza la función de balanceo, para su control difuso y su visualización de parámetros se usó el software

de LabVIEW para delimitar el ángulo de elevación en el cual trabaja el robot y además realizar la presentación mediante una gráfica. Este proyecto queda con una base sólida para futuras investigaciones de programación en Python y en LabVIEW con la idea que puede diversificar su uso para diferentes tipos de aplicaciones.

Palabras claves: Labview, péndulo invertido, lógica difusa, lego Mindstorms, Ethernet, uart, Arduino.

ABSTRACT

YEAR	STUDENTS	DIRECTOR OF TECHNICAL PROJECT	TECHNICAL PROJECT THEME
2022	Julio Andrés Cartagena Izquierdo Josué Alberto Ramírez de la Torre	Ing. Byron Lima MSc.	"DESIGN AND IMPLEMENTATION OF A FUZZY CONTROL FOR A SELF-BALANCED LINE FOLLOWER ROBOT, USING THE LEGO MINDSTORMS ROBOTICS KIT ".

This technical project arises from the need to use equipment that until now was only used for competitions and learning under a tool provided by Lego. This project aims to implement a fuzzy control on the National Instruments platform for a self-balancing line follower robot using the Lego Mindstorms robotics kit, it is done for the robotics club facilities of the Salesian Polytechnic University, through which students of the race are familiarized with the programming of control algorithms and handling of the Lego Mindstorms kit in order to perform practices using fuzzy logic for programming the rocker robot with the inverted pendulum system.

The main objective of the implementation of the rocker robot is that students learn the use of fuzzy logic for programming control algorithms through intermediate and advanced practices that contribute to the knowledge and learning for the different types of designs that can be done with the Lego Mindstorms kit according to the need of the project.

Additionally, this degree work has a control block of the kit in which the program is loaded, two force motors, a structure based on the pieces of the kit that has been designed for the best stability of the robot, a gyroscope to detect the changes of inclination of the robot while performing the balancing function, for its fuzzy control and visualization of parameters LabVIEW software was used to define the angle of elevation in which the robot works and also make the presentation through a graph.

This project is left with a solid foundation for future research in Python and LabVIEW programming with the idea that it can diversify its use for different types of applications.

Keywords: Labview, inverted pendulum, diffuse logic, Lego Mindstorms, Ethernet, uart, Arduino

Contenido

Índice de Figuras

INTRODUCCIÓN

El presente proyecto está enfocado a estudiantes de la carrera de Ingeniería Electrónica con mención en sistemas industriales de la universidad Politécnica Salesiana sede Guayaquil para que de esta manera obtengan un conocimiento aplicado en lógica difusa y la programación de algoritmos de control , por consiguiente, el proyecto está basado en la creación de un robot balance seguidor de línea, a partir de un starter kit de Lego Mindstorms el cual servirá para demostrar en base a prácticas como actúan los controladores PID y difusos en sus diferentes etapas de control, que fomentara el aprendizaje y al desarrollo de nuevas propuestas tecnológicas en torno a la propuesta planteada.

Actualmente el club de robótica de la Universidad Politécnica Salesiana sede Guayaquil cuenta con starter kits de robóticas de la marca LEGO MINDSTORMS, donde se ha notado que el estudio de algoritmos de control no se ha estudiado con una profundidad más compleja y didáctica para los robots que se pueden diseñar con estos elementos del kit.

El presente tema tiene como finalidad implementar un robot seguidor de línea auto balanceado con el starter kit de Lego Mindstorms, que pueda ser empleado por el software de LabVIEW, contara con un control difuso para poder auto balancearse mientras sigue una línea de color negra. En este proyecto se busca profundizar en los conocimientos de los controladores difusos para la tecnología que nos brinda los kits de Lego Mindstorms.

1. EL PROBLEMA

1.1 Planteamiento del Problema

En la actualidad en las instalaciones del club de robótica de la universidad Politécnica Salesiana de Guayaquil, existen kits de la marca Lego Mindstorms de la línea de robótica, donde se ha notado que no cuentan con un control difuso que le permita al robot auto balancearse mientras siga su movimiento mediante una línea negra y que no se ha estudiado con una profundidad más compleja y didáctica los algoritmos de control que se pueden implementar con estos kits

1.2 Antecedentes

La propuesta del proyecto está basada en la creación de un robot auto balanceado seguidor de línea utilizando el kit de lego Mindstorms, el cual mostraría en base a pruebas como actúan los controladores difusos en sus diferentes etapas de control , implementando un controlador difuso para el balanceo y también un control PID para poder comparar qué controlador es más óptimo para dichos kits ,implementándolo en el club de robótica con el propósito que se utilicen estos controladores para futuras competencias o demostraciones con los kits de lego Mindstorms

1.3 Importancia y alcance

El presente trabajo técnico de titulación tiene como principal objetivo dar a conocer el uso de los controladores difusos en los kits de Lego Mindstorms, el cual de una manera sencilla y eficaz plantea una nueva estrategia para solución de diversas estructuras que se puedan implementar con dichos kits

Los principales beneficios de esta implementación de control con lógica difusa son los estudiantes de la Universidad Politécnica Salesiana Sede Guayaquil de la Carrera de Ingeniería Electrónica, especialmente los integrantes del club de robótica de la Sede de Guayaquil para potenciar el uso de estos kits de Lego Mindstorms

1.4 Delimitación del Problema

1.4.1 Delimitación Temporal

El proyecto de técnico tuvo un tiempo de duración de 2 años a partir de su aprobación por dirección de carrera

1.4.2 Delimitación Espacial

Este proyecto se realizará en las instalaciones del Club de Robótica de la Universidad Politécnica Salesiana Sede Guayaquil. [1]

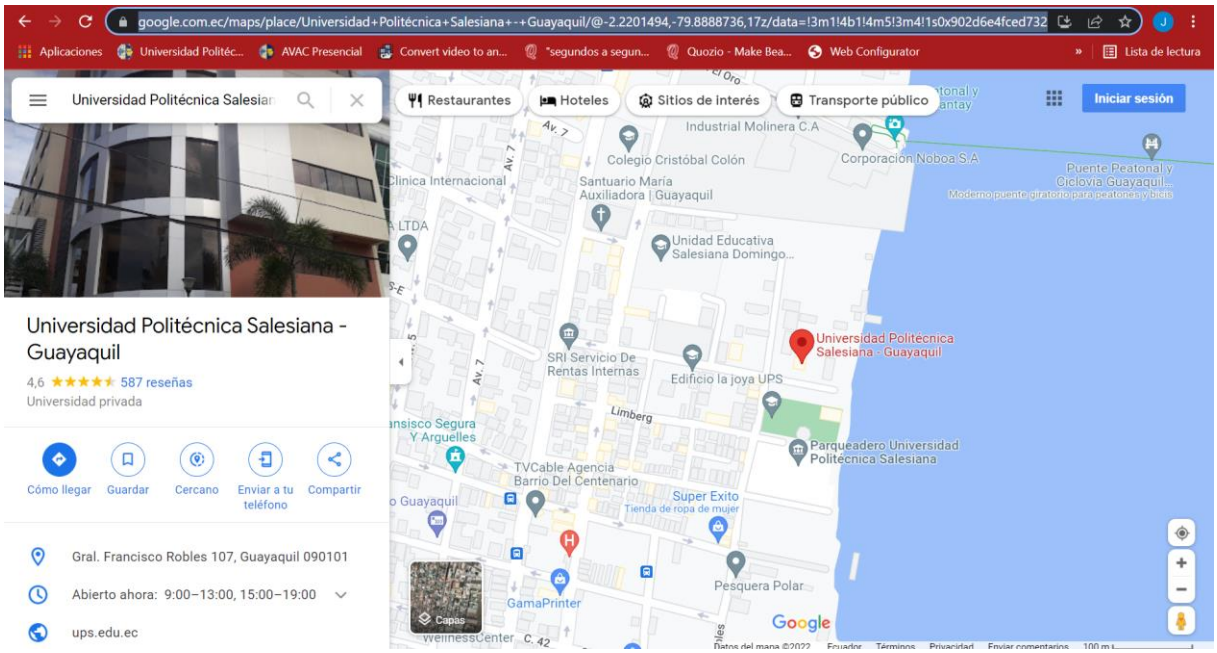


Figura 1. Locación de la Universidad Politécnica Salesiana

1.4.3 Delimitación académica

- Se realizó dos pruebas en la pista de seguidor de línea proporcionada por el club de robótica para poder observar la diferencia entre el controlador difuso y el controlador PID
- El robot tiene alcance para futuras competencias que requieran la utilización de kits de Lego Mindstorms

1.5 Objetivos

1.5.1 Objetivo General

Diseñar e implementar un controlador difuso en la plataforma de National Instruments para un robot seguidor de línea auto balanceado utilizando el starter kit de Lego Mindstorms.

1.5.2 Objetivos Específicos

- Diseñar e implementar un robot seguidor de línea auto balanceado utilizando el starter kit de Lego Mindstorms.
- Configurar los parámetros del robot seguidor de línea auto balanceado usando el software Labview.
- Plantear e implementar el algoritmo de control difuso para el sistema del robot balancín.
- Diseñar el algoritmo de control PID para el sistema del robot seguidor de línea auto balanceado.

1.6 Justificación

Considerando que el starter kits de lego MINDSTORMS fue diseñado para el aprendizaje de la robótica didáctica, donde el Club de Robótica de la Universidad Politécnica Salesiana de Guayaquil cuenta en su mayoría con alguno de estos kits, y sus amplias aplicaciones que tienen a futuro.

Esto ayudará al conocimiento, aplicaciones e implementaciones de futuros robots con los kits que requieran alguno de estos tipos de control. Perfeccionando así la formación de futuros Ingenieros y competidores de la robótica.

2 FUNDAMENTACIÓN TEÓRICA

2.1. Sensor infrarrojo Mindstorms

Es un sensor digital del kit de lego diseñado para la detección de proximidad de un robot mediante luz infrarroja, su distancia máxima de medición es de 100cm, puede ser controlado con el control remoto que viene en el mismo kit, también puede recibir comandos remotos IR. [2]

Figura 2. Sensor infrarrojo Mindstorms

2.2. Servomotor Mindstorms

El servomotor Mindstorms es un motor de gran alcance que utiliza la retroalimentación del tacómetro interno que posee para un control preciso de posición y velocidad dentro de una precisión de un 1 grado de exactitud .al utilizar el sensor de rotación integrado en el motor inteligente se puede utilizar para alinearse y sincronizarse con otros servomotores del robot, para conducir en línea recta [2].

Figura 3. Servomotor Mindstorms

2.3. Giroscopio Mindstorms

El sensor digital giroscopio Mindstorms mide el movimiento y los cambios en la orientación de rotación del robot. El sensor puede medir ángulos con precisión de +/- 3 grados, el modo de giro tiene una potencia máxima de 440 grados / segundos. [2]

Figura 4. Giroscopio Mindstorms

2.4. Bloque de control Mindstorms

El brick o bloque inteligente Mindstorms, es un módulo inteligente utilizado para la construcción de robots, cuenta con 4 entradas y 4 salidas para conectar diferentes periféricos, cuenta con una interfaz iluminada de seis botones que cambia de color para indicar el estado activo del ladrillo, utiliza un procesador ARM 9 con sistema operativo basado en Linux , almacenamiento de programas a bordos que incluye 16MB de memoria flash y 64MB de RAM, tiene incorporado un altavoz dé de alta calidad , su carga de datos mediante computadora puede darse por USB o vía bluetooth. [2]

Figura 5: Bloque de control Mindstorms

2.5. Sensor de color Mindstorms

El sensor de color Mindstorms es un sensor de 3 configuraciones, con el cual se puede medir luz ambiental, diferencial entre blanco y negro ya que contiene un sensor infrarrojo de corta distancia integrado, puede medir diferencia de colores gracias a un sensor de colores integrado. [2]

Figura 6: Bloque de control Mindstorms

Características del EV3 sensor de color 45506:

- Mide luz infrarroja reflejada, y la luz ambiental.
- Es capaz de detectar entre ocho colores. No puede detectar la diferencia entre colores o blanco y negro, o entre verde, azul, rojo, blanco, marrón y amarillo.
- Tiene una frecuencia de muestreo de 1 kHz
- Auto-ID está integrado en el software EV3. [2]

2.6. Batería Lipo Mindstorms

La batería DC recargable EV3 de iones de litio está diseñada para usarse con el Bloque Inteligente EV3 y tiene una capacidad de 2050 mAh. Ofrece un tiempo de funcionamiento más prolongado que las baterías AA y se puede cargar sin desarmar el modelo. [2]

Figura 7: Batería lipo de 7.2v de 4300mA.

2.7. Bluetooth

El Bluetooth sirve para la transmisión de datos inalámbricos mediante un protocolo de comunicación, se lo utiliza para enviar fotos, videos y voz entre diferentes dispositivos con un radio alcance medio de 10 metros. [3]

2.8. Lógica Difusa

La lógica difusa es un control al cual se adapta mejor al mundo real en el que vivimos, se basa en lo relativo de lo observado como posición diferencial, donde este tipo de lógica toma dos valores aleatorio-contextualizados y referidos entre sí. [4]

2.9. Péndulo Invertido

El péndulo invertido es un sistema de balance, en el cual consiste de una estructura por lo general una representación de varilla con una masa en un extremo, y en el otro un eje que pueda pivotar bidimensionalmente, montado en un sistema capaz de trasladarse longitudinalmente, en el cual al ser inestable se utilizan técnicas de controladores para poder mantenerlo en la posición requerida [5]

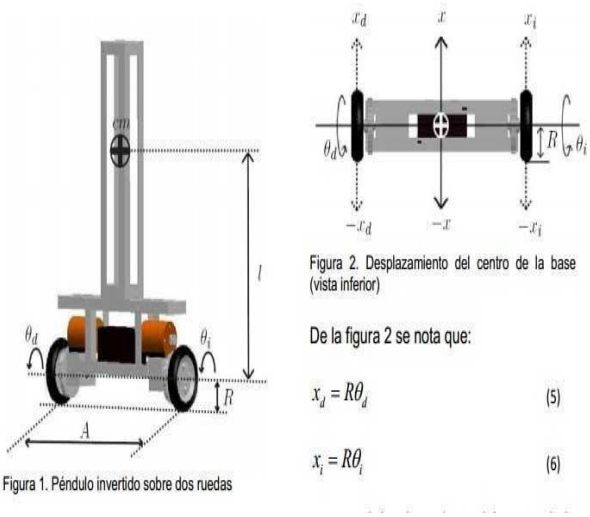


Figura 8: Péndulo invertido

2.10. LabVIEW

Es un entorno de desarrollo integrado y diseñado para ingenieros y científicos. LabVIEW es un lenguaje de programación gráfica que utiliza un modelo de flujo de datos en lugar de líneas secuenciales de código de texto, lo que permite escribir código funcional utilizando un diseño virtual que se asemeja a un proceso de pensamiento. [6]

Figura 9: Entorno de inicio del Software LabVIEW NI National Instruments

2.11. Python

Python es un lenguaje de programación de alto nivel, orientado a objetos, con una semántica dinámica integrada, principalmente para el desarrollo web y de aplicaciones informáticas.

Es un lenguaje de programación versátil multiplataforma y multiparadigma que se destaca por su código legible y limpio [7]

2.12. Arduino Uno

Arduino uno es un microcontrolador creado en el instituto de Diseño Interactivo de Ivrea en Italia

Es una placa compuesta por un microcontrolador de la marca Atmel, pines digitales, pines analógicos, la cual se le puede comunicar con otros módulos de programación

El microcontrolador de Arduino posee una interfaz de entrada, que sirve para conectar diferentes periféricos a ella. La información de estos periféricos se traslada a la placa, la cual se encargará de procesar los datos que le lleguen a través de ellos. [8]

Figura 10: Placa Arduino Uno

2.13. Arduino IDE

IDE – entorno de desarrollo integrado, llamado IDE (sigla en inglés de Integrated Development Environment), es un programa informático compuesto por un conjunto de herramientas de programación. Puede dedicarse en exclusiva a un solo lenguaje de programación o bien puede utilizarse para varios.

Es una aplicación multiplataforma basada en Java la cual nos permitirá programar las diferentes placas de Arduino del mercado [9]

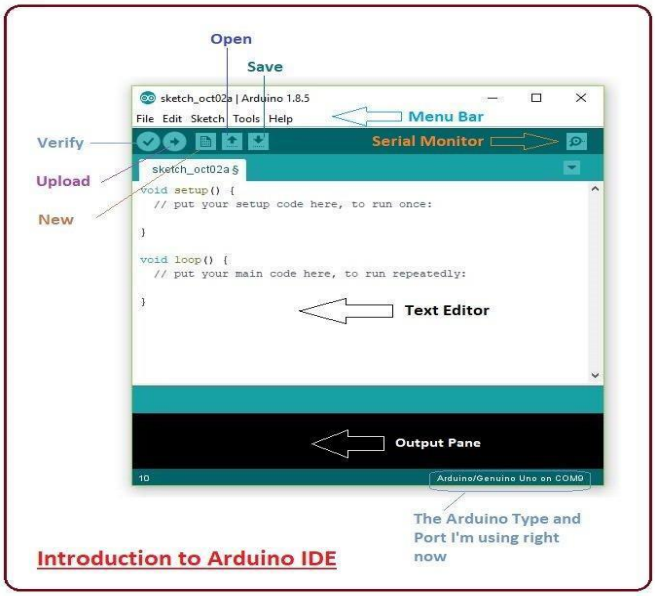


Figura 11: Arduino IDE

2.14. Visual Studio Code

Visual Studio Code es un editor de código de fuente independiente lo cual es compatible con algunos lenguajes de programación entre estos Python. [10]

2.15. Micro Python

Es un eficiente intérprete del Lenguaje de Programación Python, que esta optimizado para funcionar en microcontroladores y ambientes restringidos. [11]

Es el que se encarga de procesar el código de programación y hacer posible que el hardware (ordenador, microcontrolador…) ejecute las acciones en él descritas. [11]

2.16. Panel Frontal

Es la ventana principal que contiene el diseño de toda la instrumentación como controladores, numeradores, etc...

Con el cual va a interactuar el usuario. En la figura 12 podemos apreciar el panel frontal que se utilizara en el proyecto técnico. [5]

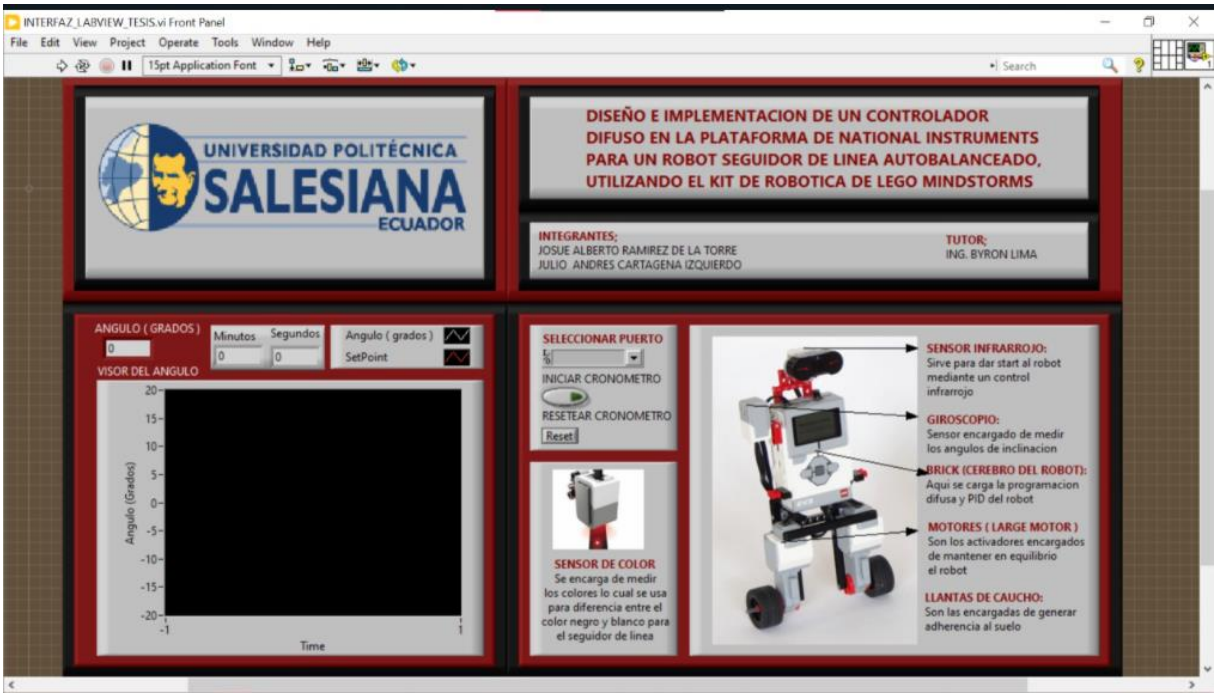


Figura 12: Panel Frontal

2.17. Sistema de control de Lazo abierto

Son técnicas de control que no cuentan con algún efecto en la acción de salida de un control, dicho en otras palabras, son sistemas que no evalúan la salida ni se retroalimentan para relacionar con la entrada

Este sistema se caracteriza por que la salida no se compara con la entrada, es sencillo y de mantenimiento fácil. [5]

2.18. Sistema de control de Lazo cerrado

Sistema de lazo cerrado (o sistema de control con realimentación): Es aquel en los que la acción está en función de la señal de salida; es decir, en los sistemas de control con realimentación, la salida que se desea controlar se realimenta para compararla con la entrada y así generar una falla que recibe el controlador para decidir qué acción tomar sobre el proceso, con el fin de disminuir dicha falla y por lo tanto, llevar la salida del sistema al valor deseado. [5]

3 MARCO METODOLÓGICO

3.1. Diseño del Robot Balance.

El robot balance es un sistema mecánico y electrónico que tiene 2 ruedas cuyos ejes son colineales, las ruedas cuentan con un motor independiente, el péndulo se encuentra fijo en el eje de las ruedas y gira alrededor del mismo, su punto de gravedad está sobre el eje de giro y se lo considera como un sistema inestable por efecto no lineal. El robot balance debe desplazarse de manera autónoma por una trayectoria mientras mantiene el equilibrio.

El robot fue diseñado basándose en modelos bases propuesto por la misma empresa de Lego Mindstorms y en el transcurso se fueron haciendo cambios y modificaciones al diseño para encontrar el más óptimo que se acomode a nuestras exigencias.

En la presente figura se muestra el robot con los componentes que lo conforman de diversas vistas del propio. [12]

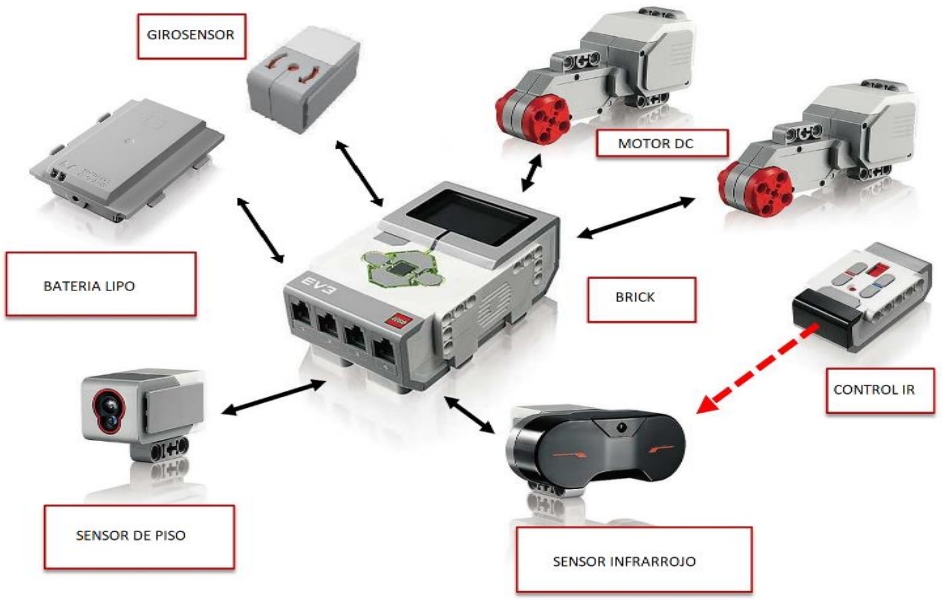


Figura 13: Esquema Físico del Robot Balance.

3.2. Cuerpo del robot Balance.

En la figura 14 podemos observar al robot propuesto por Lego Mindstorms el cual es llamado por la misma empresa como BALANCER el cual consiste en una estructura alargada dos motores denominado por Lego Mindstorms “Large Motor” el giroscopio colocado a un costado del robot, el sensor infrarrojo de distancia colocado en la parte superior del robot para hacerlo lo más parecido a una estructura de un péndulo invertido. El Brick o bloque inteligente el cual contiene las pilas es la parte más pesada del robot en esta estructura está centrada para que el centro de gravedad se encuentre lo más equilibrado posible esta estructura fue en la que más nos basamos debido que había que adaptarle un sensor de color para poder seguir la línea y pueda la estructura seguir manteniendo el equilibrio sin tanta dificultad. [13]

Figura 14: Foto del robot denominado BALANCER EV3.

En la figura 15 podemos observar al robot propuesto por Lego Mindstorms el cual es llamado por la misma empresa como GYRO BOY el cual consiste en una estructura más compacta con dos motores large motor más cerrados y el giroscopio colocado en

la mitad del robot y la parte trasera del robot , el sensor infrarrojo de distancia colocado en un costado del robot para simular como si el robot tuviera brazos agregando atractivo y decoración al robot balance y el sensor de color colado en el otro costado del robot . El Brick o bloque inteligente el cual contiene las pilas es la parte más pesada del robot en esta estructura está la parte superior del robot haciendo que el centro de gravedad se encuentre en la parte superior de los motores. de esta estructura tomamos en cuenta la importancia colocar sensores al costado para un mayor equilibrio y poder distribuir correctamente el peso del robot. nos fijamos que esta estructura es más propensa a perder el equilibrio con pequeñas perturbaciones debido que el mayor peso del robot se encuentra en la parte superior de este. [13]

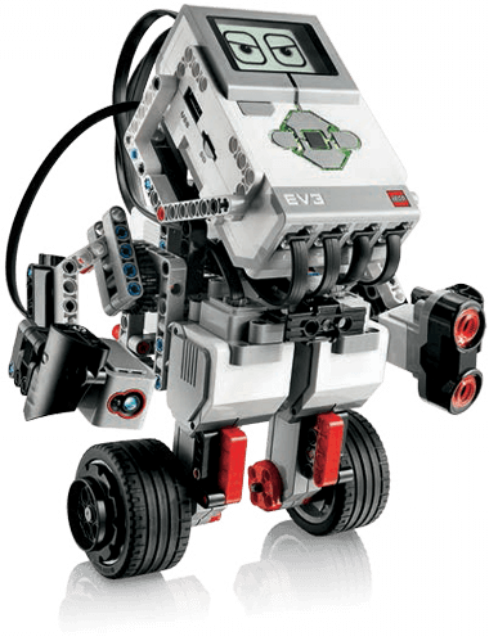


Figura 15: Foto del robot denominado GYRO BOY EV3.

En la figura 16 podemos observar el prototipo final del robot el cual optamos por combinar las dos estructuras y escoger lo mejor de cada una para tener el diseño óptimo posible para lo que nuestro proyecto técnico requería. Colocamos los sensores al costado y dejamos el Brick en la parte superior logrando que el robot balance esté proporcionado para que el centro de masa se encuentre lo más cerca el centro mismo entre el brick y los motores. [5]

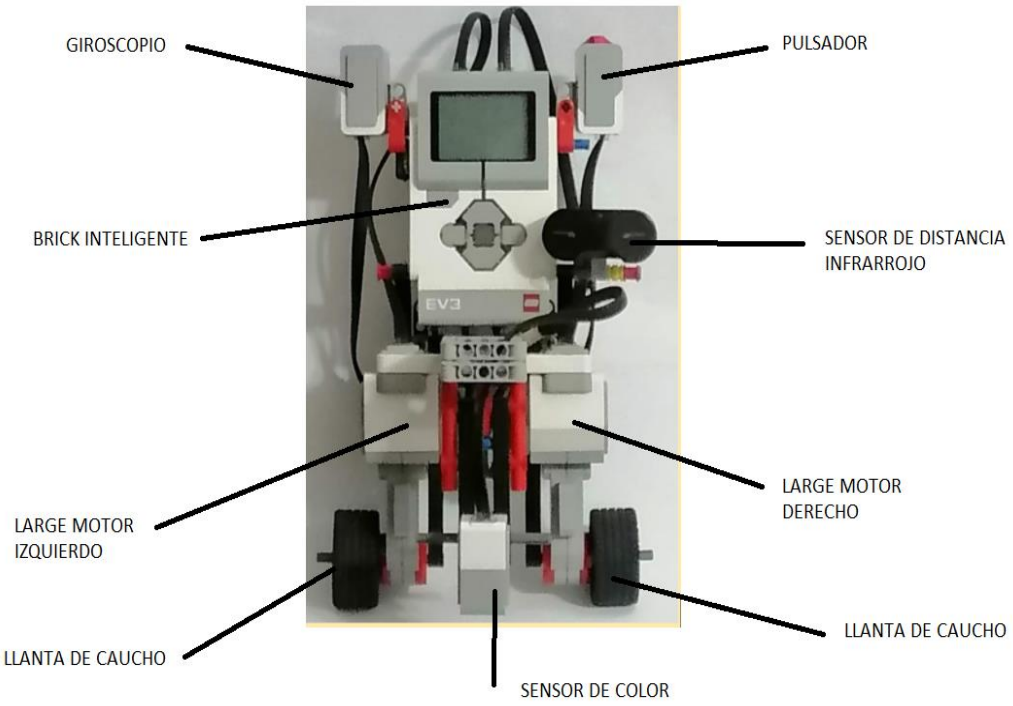


Figura 16: Foto frontal del robot balance seguidor de línea.

Figura 17: Foto lateral del robot balance seguidor de línea.

Se coloca el sensor de Gyro o giroscopio que es el encargado de medir el cambio de ángulo del robot auto balanceado seguidor de línea, se lo colocó en la parte lateral derecha del robot y un poco dirigido para la parte delantera para agregar peso a la parte de adelante del robot debido que el peso de las pilas está dirigido hacia atrás. y minimizando las vibraciones que podría afectar una mala lectura del sensor giroscopio debido que estos sensores son de gama baja son propensos a ser afectados por las vibraciones. [5]

Figura 18: Foto del sensor de Gyro de lego Mindstorms.

Se coloca un pulsador en la parte lateral izquierda del robot auto balanceado el cual nos servirá para dar inicio al robot una vez que calibre los sensores. Se lo coloca en dirección delantera para que agregue peso en dirección hacia adelante al robot para poder equilibrarlo con el peso de las pilas que están dirigidas hacia atrás. [5]

Figura 19: Foto del Pulsador del kit de Lego Mindstorms

Debido a la potencia baja que nos ofrecen los large motor del kit de Lego Mindstorms se los coloca de tal forma que el robot esté lo más simétrico posible tanto a los costados como vertical y horizontal que mantengan los mismos pesos en cada lado de este. Para nuestra estructura dicha abertura es la más óptima para que el centro de masa se encuentre lo más centrado posible y se utilizaron las llantas de tamaño estándar que ofrece Lego Mindstorms como reto de utilizar solo los componentes que ofrece solo el starter kit de lego Mindstorms. Ya que en la línea de lego si se encuentran otros tipos de llantas de mayor diámetro, pero pertenecen a otros kits. [5]

Figura 20: Foto del motor large motor y llantas del kit Lego Mindstorms

El brick se coloca en la parte superior del robot logrando que esté lo más simétrico posible tanto de forma vertical como en las direcciones adelante y atrás. se compenso el peso del brick que es dirigido hacia atrás, con los demás sensores que se lo colocaron en dirección hacia adelante

Se coloca un sensor infrarrojo de distancia para compensar el peso que generan las pilas. Este sensor se puede usar a futuro para controlar el robot mediante un control infrarrojo para futuras investigaciones. [5]

Figura 21: Foto del brick y sensor infrarrojo de distancia de Lego Mindstorms

El mayor reto fue colocar el sensor de distancia debido a que el robot debía seguir una pista compuesta por una línea negra. La pista contenía 2 puntas, una parte inclinada y dos curvas. Aunque el robot no fue propuesto para solucionar obstáculos se logra que pueda seguir diferentes pistas. para compensar el peso de las pilas se coloca el sensor de color a una distancia larga del robot, debido que por lo general el sensor se debe colocar en el centro y simétricamente al robot, pero este no fue el caso lo más óptimo fue colocarlo un poco más adelante para que los giros puedan ser lo más preciso posible y este no se salga a la hora de realizar curvas y pueda seguir la línea sin tantas complicaciones. [5]

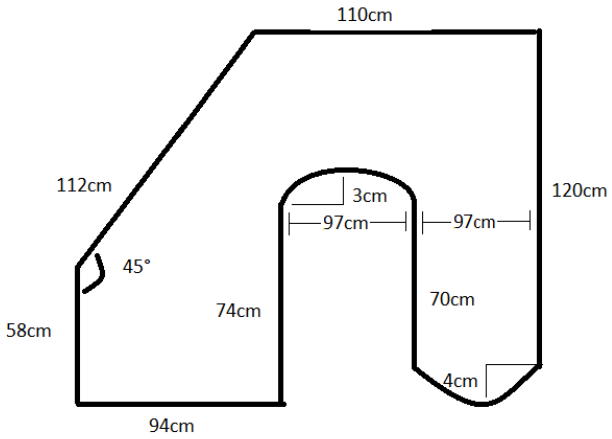


Figura 22: Foto de la pista a seguir proporcionada por el club de robótica

Figura 23: Foto del sensor de color del kit de Lego Mindstorms

3.3. Diseño para el sistema de control

A continuación, se presenta el esquema del control para que el robot pueda mantener el equilibrio del sistema carro - péndulo, luego se plantean las diferentes técnicas para el control que será utilizado en el controlador del robot

Se implementará un control difuso y un controlador PID convencional por separado para poder ver la diferencia entre estos dos controladores

Debido a problemáticas que fueron surgiendo en el transcurso del desarrollo del robot se logró observar que se nos era imposible desarrollar el controlador difuso en el software de Labview como fue propuesto por motivos que el toolkit que Lego Mindstorms ofrece para Labview hasta el momento solo tiene actualización hasta el 2016. nos encontramos con la novedad que los bloques de control difuso que ofrece LabVIEW no estaban permitidos en el toolkit se buscó implementarlo de forma matemática pero el mismo toolkit nos limita a ciertos bloques matemáticas impidiéndonos cumplir el objetivo de desarrollar el controlador difuso en el software de LabVIEW. Una de las opciones aprobadas fue realizar el controlador difuso en el lenguaje de programación Python en el cual realizamos tantos las entradas, salidas y reglas.

Debido a esta novedad el controlador PID se desarrolló en el IDE que nos ofrece Lego Mindstorms por default y se logró comunicar con una interfaz realizada en el software de LabVIEW.

Para lograr esto se tuvo que comunicar el robot auto balanceado seguidor de línea mediante vía a bluetooth a un segundo brick este está conectado vía I2C a un Arduino que es el intérprete encargado de mandar la información por vía serial al panel de control realizado en LabVIEW. [5]

Figura 24: Foto del segundo brick y el Arduino Uno

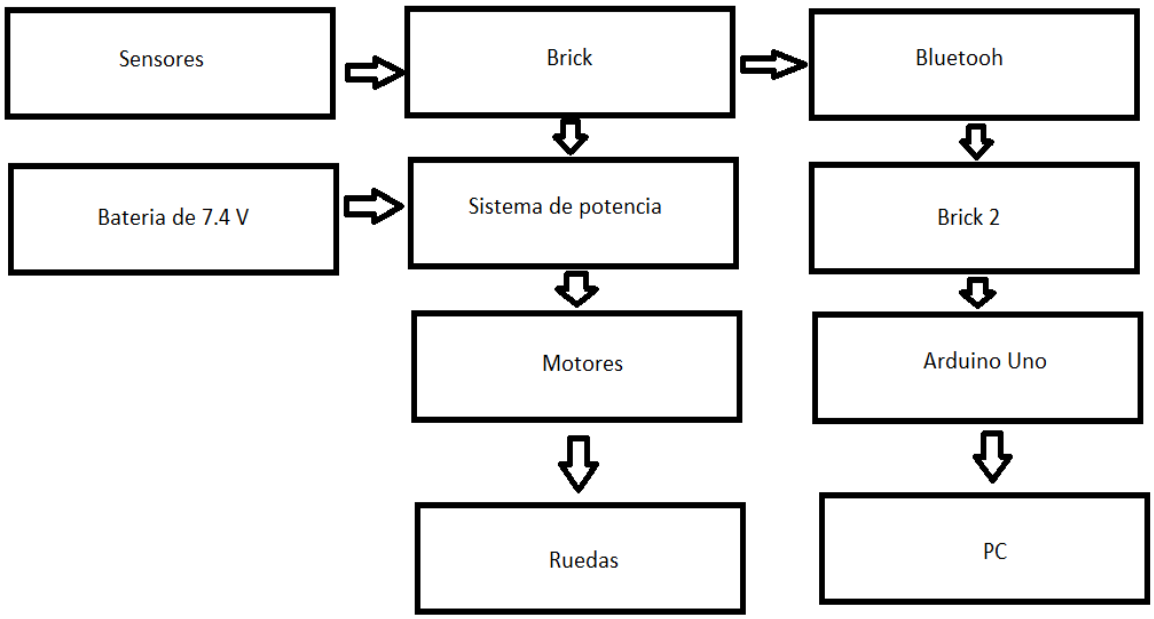


Figura 25: Diagrama de construcción del Robot

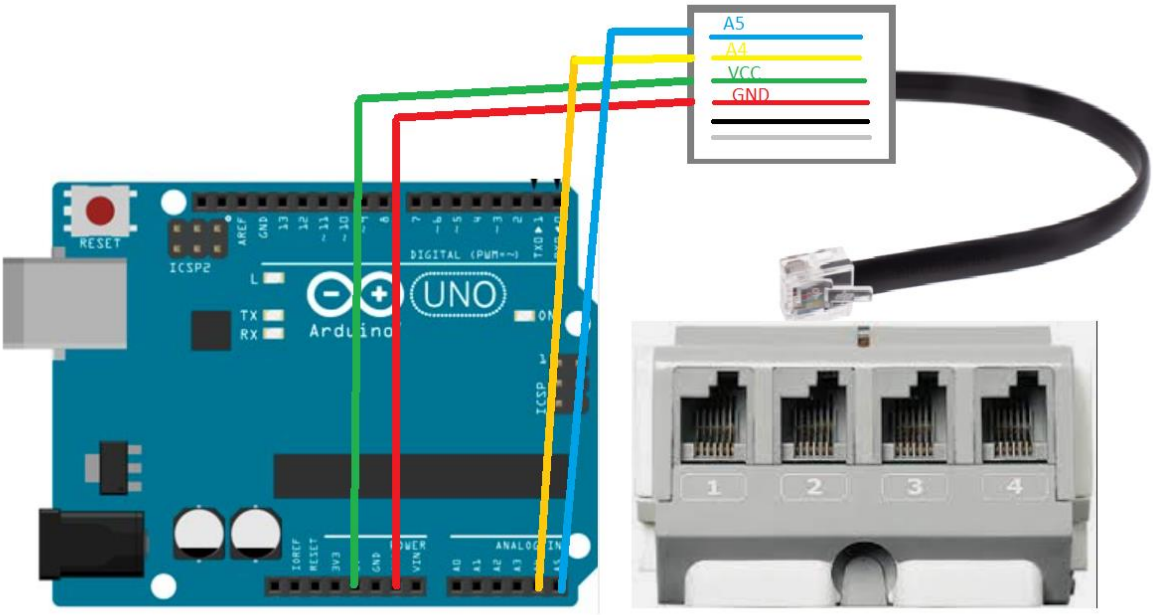


Figura 26: Diagrama de conexión del Arduino al brick

3.4. Controlador PID

Para obtener los mejores resultados al mantener el equilibrio con el robot balance se procedió a integrar un controlador. En este apartado se dará una breve descripción de este controlador. Para diseñar el control para la inclinación no se dispone de modelo matemático, En este caso para solución de este problema optamos por utilizar un controlador PID, debido que se puede sintonizar manualmente relativamente fácil. Se trata de un control proporcional-integral-derivativo, ya que es una técnica muy utilizada en el área de la robótica. Un controlador PID se encarga de calcular el resultado mediante la diferencia entre el valor medido y el valor deseado o set point, el objetivo de este controlador es minimizar este error ajustando una variable manipulable.

La fórmula del controlador la encontramos en la figura número 27. [5]

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

Figura 27: Fórmula de un controlador PID.

Dónde podemos encontrar que, “Kp” en la fórmula es la ganancia proporcional del sistema, “Ki” es la ganancia integral y “Kd” es la ganancia derivativa siendo “e” el error, donde el error es la diferencia entre el valor actual y el setpoint.

En muchos de los casos el valor kp o parte proporcional no fue suficiente para eliminar el error en el estacionario, por lo que se procede a agregar una parte integral. La parte integral es el producto entre la integral del error y la constante integral Ki, que ayudó a disminuir el error estacionario para generar un balanceo óptimo se integró también la acción derivativa la consiste en multiplicar la derivada del error por la constante derivativa Kd, así evitamos que el error se incremente corrigiendo con la misma velocidad que se produce. En la figura 28 podemos observar el diagrama de bloques del controlador PID del proyecto actual, donde la planta del sistema es el robot balance seguidor de línea, nuestra acción de control o salida del controlador PID es la activación de los motores y la salida la obtenemos del sensor de giro. [5]

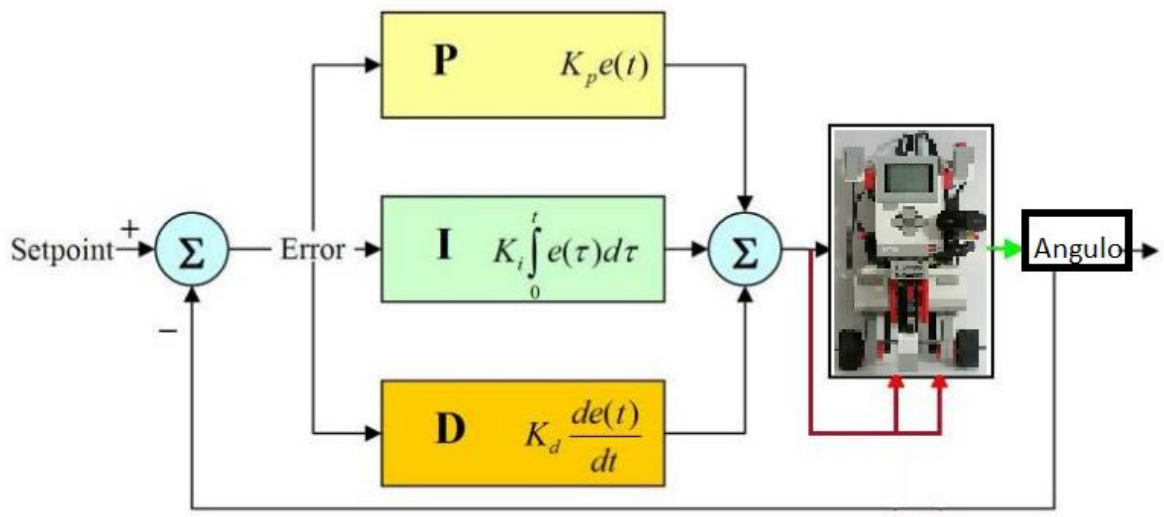


Figura 28: Diagrama de bloques PID.

Entre las diferentes técnicas de sintonización de un PID, las cuales generalmente requieren de un modelo de la planta y cuyo modelo no disponemos, se ha optado por la sintonización manual de las constantes del controlador, para ello se seguimos los siguientes pasos:

iniciamos con los valores ki y kd en cero, fuimos incrementando kp hasta tener oscilaciones que el robot no pueda mantenerse o encontrar el valor crítico, fijamos un valor medio en kp y procedemos a incrementar la variable de ki para reducir el error estacionario sin olvidar que un valor muy elevado de esta constante puede provocar inestabilidad. Como último incrementamos kd hasta obtener una respuesta lo suficientemente rápida ante interferencias para la sintonización manual tuvimos en cuenta los efectos de los parámetros PID cuyo efecto podemos visualizar en la figura 29. [5]

Efectos de los parámetros				
Parámetro	**Tiempo de subida**	**Sobre pico**	**Tiempo de establecimiento**	**Error en permanente**
K_p	Reduce	Aumenta	Poco cambio	Reduce
K_i	Reduce	Aumenta	Aumenta	Elimina
K_d	Indefinido	Reduce	Reduce	Nada

Figura 29: Efecto de los parámetros PID.

Teniendo en cuenta los aspectos anteriores en la figura 30 tenemos la siguiente tabla, la cual se observan algunos valores que utilizamos para afinar el PID, hasta llegar a los valores que lo hicieran funcional.

kp	Resultados
0,3	No mantenia el robot en equilibrio
1	El robot oscila mucho
5	Completamente inestable
Kd	
0.1	El robot no se balanceaba
0.01	El robot no se balanceaba
0.009	El robot se balanceaba muy bruscamente
ki	
1	El robot no se balanceaba
10	El robot no se balanceaba
20	El robot se balancea pero con el tiempo tiende a caer

Figura 30: Tabla de resultados.

Teniendo como resultado final los siguientes valores de Kp, Kd y Ki

$$\boldsymbol{Kp = 0.6, Ki = 14 \; y \; Kd = 0.005}$$

3.5. Seguidor de línea

El sensor que se utiliza para el seguimiento de trayectorias es el sensor de color del kit de lego Mindstorms ev3, en este apartado se explicará el método que se utilizó para el seguimiento de la línea. En la figura 31 se encuentra un esquema de la vista superior del robot con la posición de las ruedas y del sensor de color pintado en rojo. el objetivo es lograr que el robot pueda seguir líneas negras y trayectorias sencillas ya que estaríamos concentrándose los objetivos propuestos sin quitar el hecho que se puedan realizar futuras investigaciones en esta parte como la implementación de un control para el seguimiento de línea. [5]

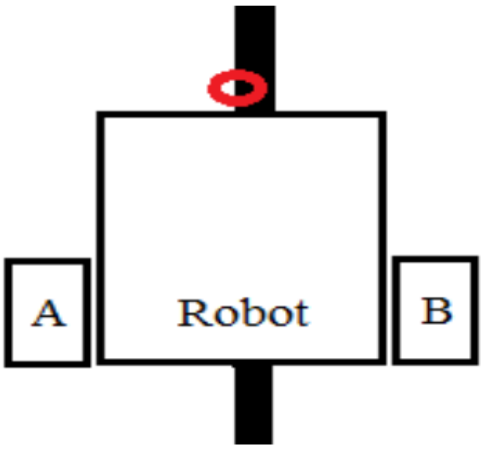


Figura 31: Esquemas robot + seguimiento de trayectoria.

Para mayor facilidad a la hora de construir la pista se ha decidido poner el suelo de color blanco con una línea negra que es la que indica la trayectoria que debe seguir el robot auto balanceado seguidor de línea. Por lo general los seguidores de línea básicos se suelen realizar con dos configuraciones.

Se suele escoger entre usar uno o dos sensores de luz o color, con la configuración de dos sensores el objetivo es que la línea se mantenga entre los dos sensores, y con un sensor el objetivo es que siga uno de los bordes de la línea conocido como sistema ON/OFF donde consiste en apagar un motor y mientras se enciende el contrario dependiendo el color a leer, también se puede implementar con más de dos sensores. En general, cuantos más sensores mejor seguimiento de la línea tendrá el robot, por cuestiones prácticas el robot lo implementamos con un sólo sensor. La causa por la que se sigue el borde de la línea es que de esta manera se sabe que cuando el sensor se encuentra en una zona de color blanco cuando este color es detectado el robot ha de girar a la derecha encendiendo un motor y apagando el otro, y cuando se encuentra en una zona de color negro el robot ha de girar a la izquierda haciendo la misma acción pero encendiendo el motor que se apagó y apagando el motor que se encontraba encendido oscilando entre estos dos colores . En la figura 32 se pueden ver los giros que debe dar el robot en función del color de la superficie donde se encuentre, el valor medio entre blanco y negro que debería ser el valor del borde de la línea será de 20 .Cuando el valor que devuelve el sensor sea mayor que 20 el robot debe girar a la

derecha ya que se encontrará en una superficie blanca, y cuando el valor se encuentre en 20 o sea menor que este debe girar a la izquierda porque se encontrará en la línea negra. Con esto se obtuvo un seguidor de líneas que realizará la trayectoria en zigzag.

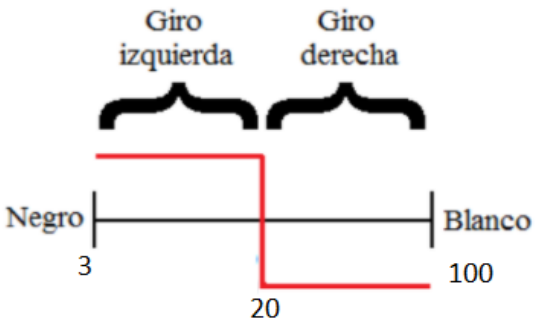


Figura 32: Giros del robot según el color.

En la figura 33 se puede observar cómo se implementa este apartado en la programación realizada en Lego Mindstorms. [5]

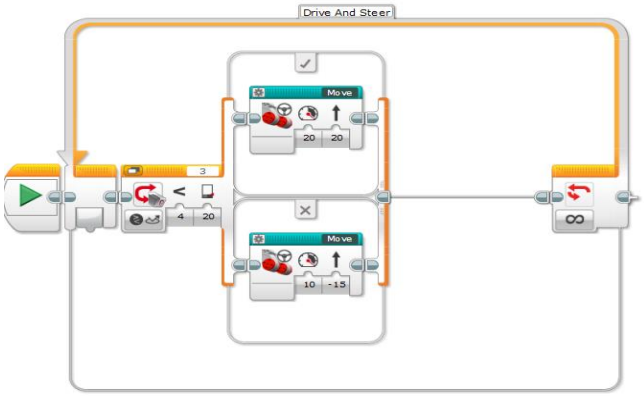


Figura 33: Giros del robot según el color.

3.6. Diseño del controlador Difuso

Para el diseño del controlador difuso, Se identifica las entradas y salidas de nuestro sistema para poder dimensionarlo correctamente. Se consideró 5 funciones de membresía para cada entrada de las cuales tenemos dos. [5]

El error la cual podemos observar en la figura 34 posee 5 funciones de membresía con sus respectivas etiquetas y se desarrolló este apartado en Python el cual podemos ver cómo es su programación en la figura 35. [5]

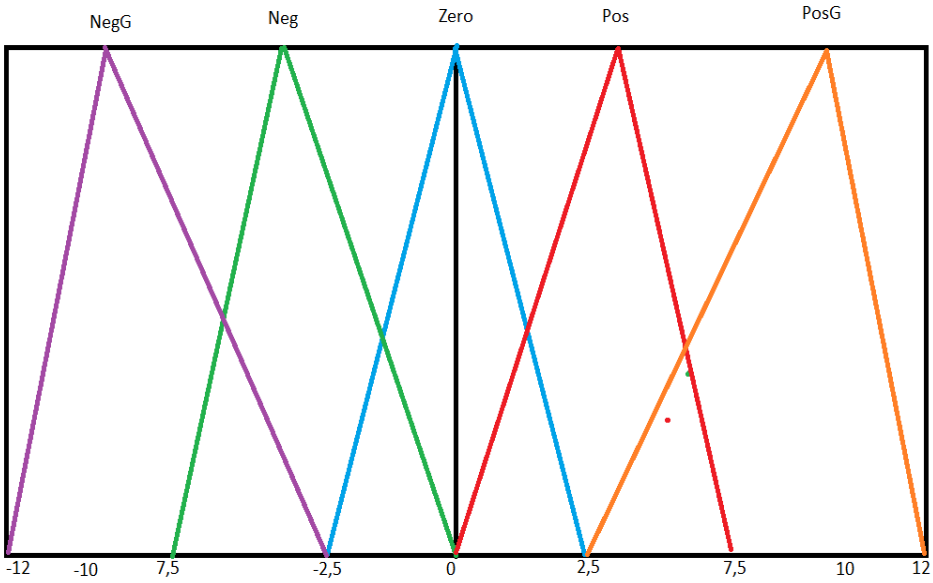


Figura 34: Funciones de membresía del error.

```
def fuzzifyErrorPosG(error):
    return trimf(error, [2.5, 10, 12])

def fuzzifyErrorPos(error):
    return trimf(error, [0, 2.5, 7.5])

def fuzzifyErrorZero(error):
    return trimf(error, [-2.5, 0, 2.5])

def fuzzifyErrorNeg(error):
    return trimf(error, [0, -2.5, -7.5])

def fuzzifyErrorNegG(error):
    return trimf(error, [-2.5, -10, -12])
```

Figura 35: Funciones de membresía del error en Python.

Se realizaron 5 funciones de membresía para la derivada del error la cual se denominó “DERROR”. esto lo podemos observar en la figura 36 y su respectiva programación en Python la cual se puede visualizar en la figura 37. [5]

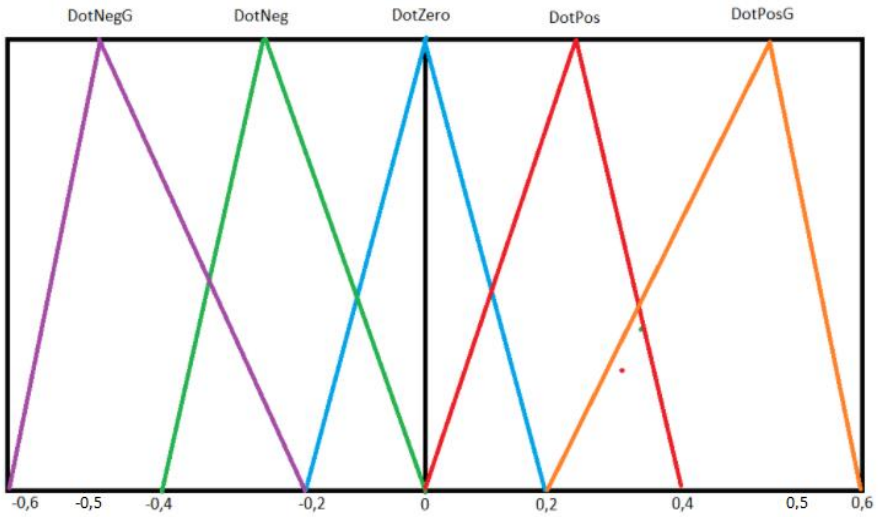


Figura 36: Funciones de membresía del DERROR.

```
def fuzzifyDerrorDotPosG(Derror):
    return trapmf(Derror, [0.2, 0.5, 0.6])

def fuzzifyDerrorDotPos(Derror):
    return trapmf(Derror, [0, 0.2, 0.4])


def fuzzifyDerrorDotZero(Derror):
    return trimf(Derror, [-0.2, 0, 0.2])


def fuzzifyDerrorDotNeg(Derror):
    return trapmf(Derror, [0, -0.2, -0.4])

def fuzzifyDerrorDotNeg(Derror:
    return trapmf(Derror, [-0.2, -0.5, -0.6])
```

Figura 37: Funciones de membresía del DERROR en Python.

Para la salida se realizaron 5 funciones de membresía la cual podemos observar en la figura 38, la programación en Python la podemos observar en la figura 39. [5]

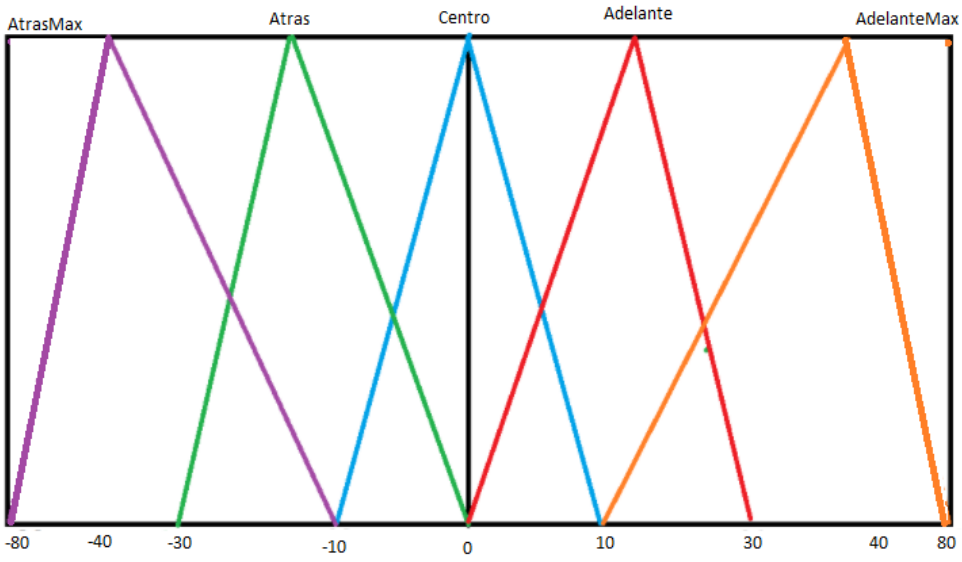


Figura 38: Funciones de membresía de la salida.

```
def fuzzifyOutputAdelanteMax():
    return getTrapmfPlots(0, 10, [10, 40,80], "left")

def fuzzifyOutputAdelante():
    return getTrimfPlots(0, 10, [0, 10, 30])


def fuzzifyOutputCentro():
    return getTrimfPlots(0, 10, [10, 0, 10])

def fuzzifyOutputAtras():
    return getTrimfPlots(0, 10, [0, -10, -30])

def fuzzifyOutputAtrasMax():
    return getTrapmfPlots(0, 10, [-10,-40, -80], "right")
```

Figura 39: Funciones de membresía de la salida en Python.

En el controlador se plantean 25 reglas donde se designan funciones de pertenencias triangulares la cual podemos observar en la figura 40. [5]

		DERROR				
		DotNegG	DotNeg	DotZero	DotPos	DotPos
ERROR	NegG	AtrasMax	AtrasMax	AtrasMax	Atras	Centro
	Neg	AtrasMax	AtrasMax	Atras	Centro	Adelante
	Zero	AtrasMax	Atras	Centro	Adelante	AdelanteMax
	Pos	Atras	Centro	Adelante	AdelanteMax	AdelanteMax
	PosG	Centro	Adelante	AdelanteMax	AdelanteMax	AdelanteMax

Figura 40: Reglas del controlador difuso.

4 RESULTADOS

4.1 Prueba y evaluación del Controlador PID

Luego de visualizar el recorrido completo del prototipo, se puede verificar la tabla proporcionada por el programa realizado en LabVIEW y sus tendencias. En la siguiente figura, podemos entender el comportamiento de los controladores PID y sus respectivos ángulos de inclinación a lo largo del tiempo.

En el mejor recorrido robot tardo 5 minutos con 10 segundos en recorrer la pista entera con el controlador PID.

Figura 41: Visualización del cambio de ángulo en la línea del tiempo.

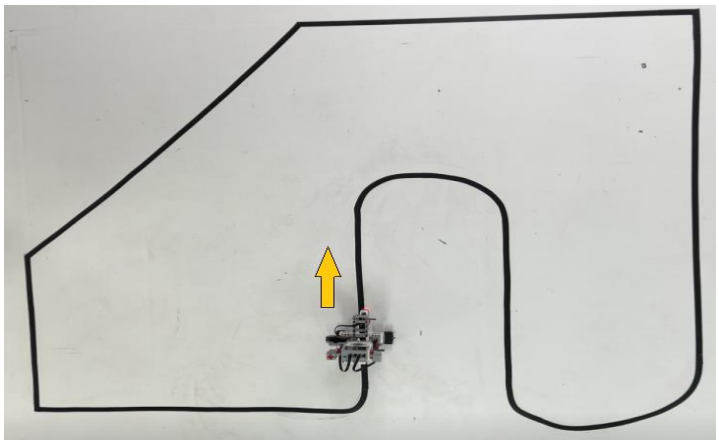


Figura 42: Visualización del robot recorriendo la pista con el controlador PID.

4.2 Prueba y evaluación del Controlador DIFUSO

Se procedió a realizar el siguiente recorrido completo del prototipo, se realiza el mismo procedimiento de validar la salida del ángulo con este controlador. En la siguiente figura, podemos entender el comportamiento del controlador DIFUSO debido a problemáticas mencionadas antes el interfaz solo se puede obtener la información del tiempo recorrido que en este caso fue de 5 minutos y 15 minutos

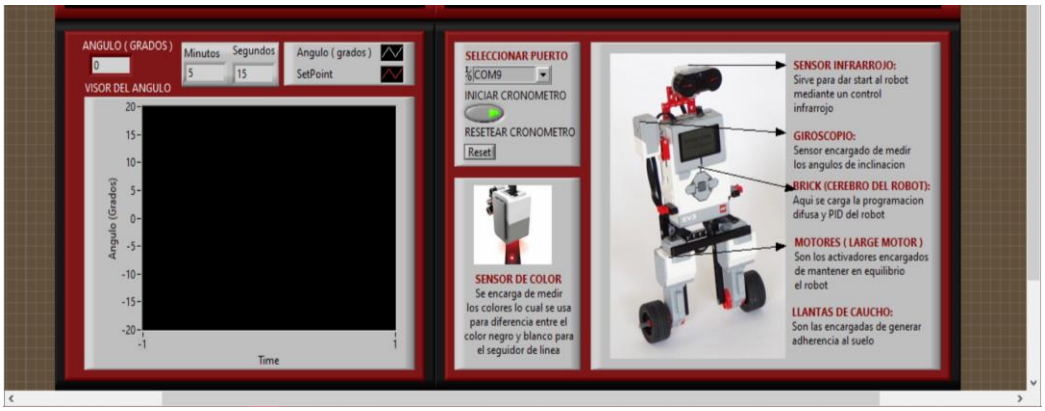


Figura 43: Tiempo transcurrido del robot con el controlador difuso.

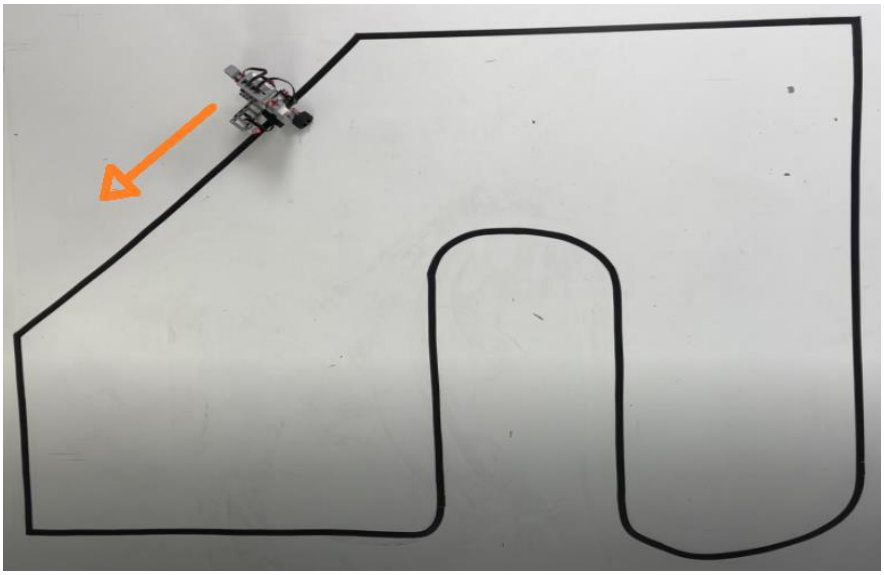


Figura 44: Robot recorriendo la pista con el controlador difuso.

El controlador PID implementado presenta una ventajas y respuesta mucho más rápida que el controlador difuso para lograr una estabilización con los kits de Lego Mindstorms.

Otra de las ventajas del PID ante el controlador difuso es la respuesta a perturbaciones y el tiempo de asentamiento al estabilizarse por eso el recorrido con el controlador PID es más rápido que el controlador difuso

Las posibles desventajas del controlador PID ante el controlador difuso es cuando este se destina a aplicaciones que requieren una mayor precisión donde puede presentar problemas con su tiempo de respuesta y su estabilidad.

5. CONCLUSIONES Y RECOMENDACIONES

5.1 Conclusiones

1. Los kits de Lego Mindstorms suelen tener ciertas limitantes a la hora de implementarles controladores inteligentes
2. Al añadir perturbaciones el PID presenta mayor respuesta y rechazo a la misma, mientras que el difuso presenta una tardía estabilización y se muestra en su salida más picos oscilatorios con esto podemos concluir que la respuesta en este caso específico va mejor representada por el controlador PID que por el DIFUSO.
3. Las reglas del controlador difuso fueron elaboradas a partir de pruebas en tiempo real realizadas con el robot balance seguidor de línea, dichas pruebas ayudaron para hallar los rangos para que el robot pueda funcionar de la manera más óptima posible
4. La visualización de los parámetros en su mayoría requerirá un elemento que haga de intérprete entre el brick de Lego Mindstorms y un software externo

5.2 Recomendaciones

1. Recordar que al enviar datos en tiempo real se debe considerar el tiempo de transmisión y en este caso en particular debido a la comunicación solo se muestra un dato.
2. Siempre considerar que al trabajar con sensores ópticos para aplicación en seguidores de líneas se debe considerar la distancia que mantiene con respecto a la superficie el sensor y que esta varía acorde a la inclinación.
3. Para mayor precisión en un controlador difuso se deben realizar varias pruebas.
4. Continuar con la investigación, pero sin restricciones del bloque lego por el contrario debería realizarse mediante placa de Arduino para mejores resultados.

Bibliografía

[1] Google, «https://www.google.com.ec/maps/place/Universidad+Polit%C3%A9cnica+Salesiana+-+Guayaquil/@-2.221151,-79.8935438,16.33z/data=!4m9!1m2!2m1!1suniversidad+politecnica+salesiana+de+guayaquil!3m5!1s0x902d6e4fced73235:0xb76f5008ec6c4345!8m2!3d-2.2201494!4d-79.886,» Universidad Salesiana, 2020. [En línea].

[2] LEGO, «LEGOeducation.com/MINDSTORMS,» 2016. [En línea]. Available: https://le-www-live-s.legocdn.com/sc/media/files/user-guides/ev3/ev3_user_guide_esmx-6ac740d3cdd578cc6a52d10d7d173da9.pdf. [Último acceso: 2021].

[3] iProfesional, «www.iProfesional.com,» Noviembre 2020. [En línea]. Available: https://www.iprofesional.com/tecnologia/326651-que-es-el-bluetooth-y-para-que-sirve.

[4] C. Idict, «ECURED,» Agosto 2019. [En línea]. Available: https://www.ecured.cu/L%C3%B3gica_difusa.

[5] P. Lora, «Universitat Politécnica de Catalunya,» Marzo 2015. [En línea]. Available: https://upcommons.upc.edu/bitstream/handle/2117/78036/Memoria.pdf

[6] P. Nacional, «DSpace Repository,» Noviembre 2018. [En línea]. Available: https://tesis.ipn.mx/bitstream/handle/123456789/14336/Tesis%20LabVIEW-.

[7] A. Robledano, «Open Webinars,» Septiembre 2019. [En línea]. Available: https://openwebinars.net/blog/que-es-python/.

[8] A. Arduino, «Aprendiendo Arduino,» Junio 2018. [En línea]. Available: https://aprendiendoarduino.wordpress.com/2016/12/11/ide-arduino/..

[9] Y. Fernandez, «XATAKA,» Marzo 2019. [En línea]. Available: https://www.xataka.com/basics/que-arduino-como-funciona-que-puedes-hacer-uno.

[10] V. Studio, «Microsoft,» Enero 2022. [En línea]. Available: https://visualstudio.microsoft.com/es/..

[11] M. -. D. No, «Exploradores,» Enero 2017. [En línea]. Available: https://www.esploradores.com/python_y_micropython_que_son.

[12] A. A. DUARTE, CONTROL DE UN ROBOT AUTÓNOMO TIPO PÉNDULO, La Paz, México: TECNOLÓGICO NACIONAL DE MÉXICO.

[13] L. Valk, «RobotSquare,» Julio 2014. [En línea]. Available: http://robotsquare.com/2014/07/01/tutorial-ev3-self-balancing-robot/.

7 ANEXOS

7.1 Programación del controlador PID realizado en Lego Mindstorms

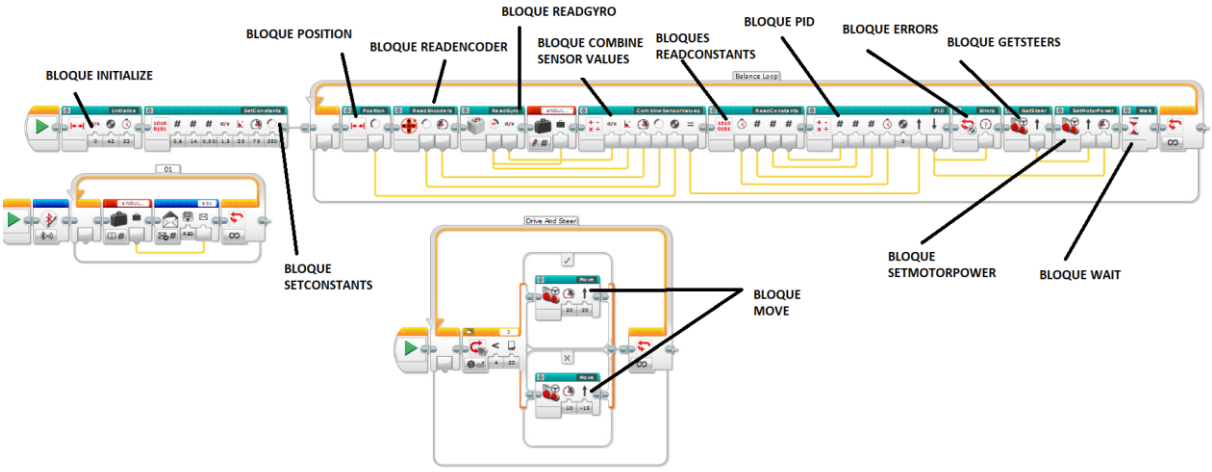


Figura 45: Programación controlador PID en lego mindstorms.

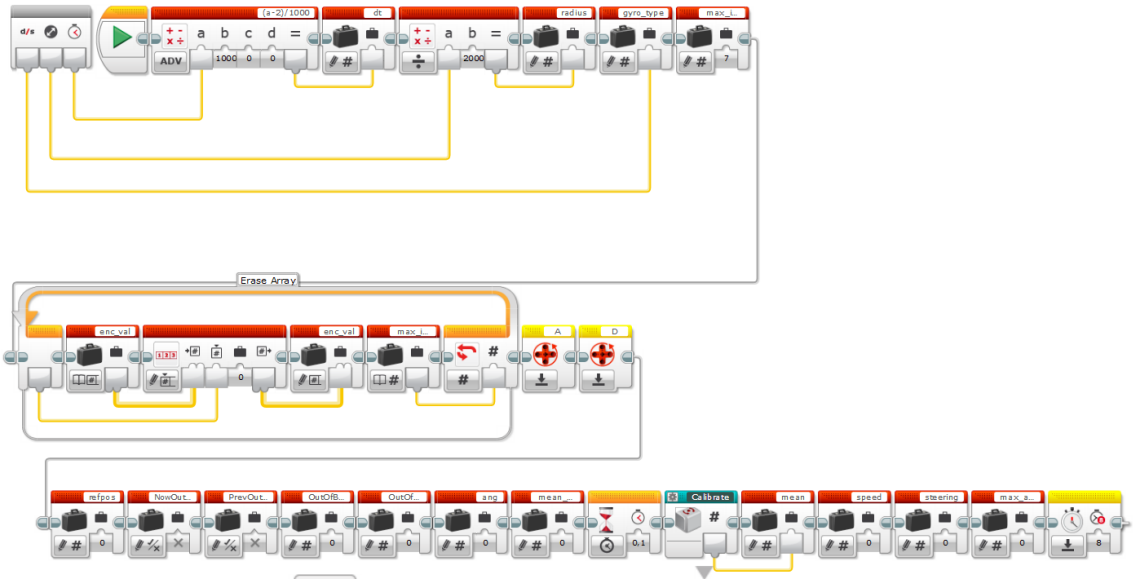


Figura 46: Bloque Initialize.

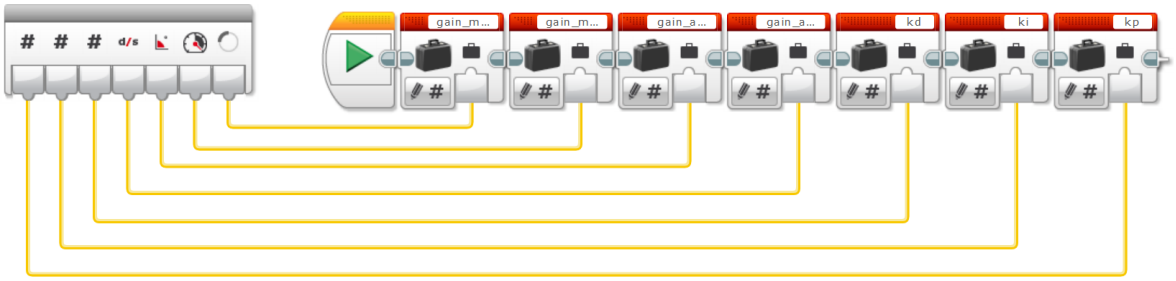


Figura 47: Bloque SetConstants.

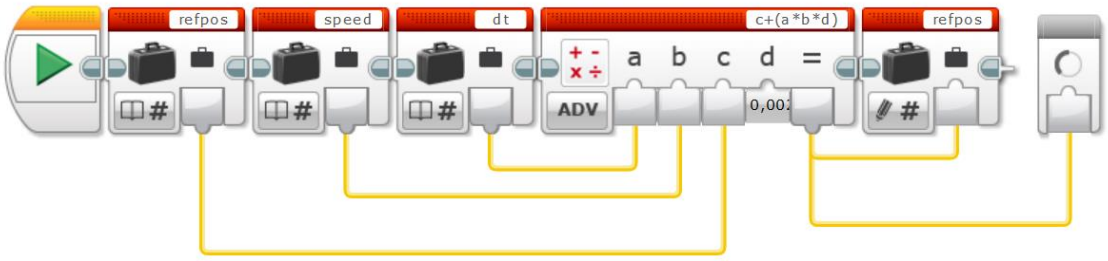


Figura 48: Bloque Position.

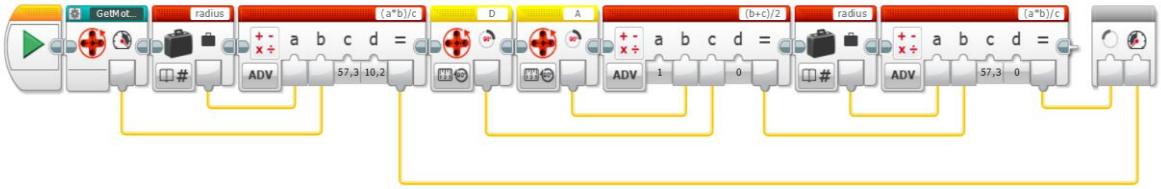


Figura 49: Bloque ReadEncoder.

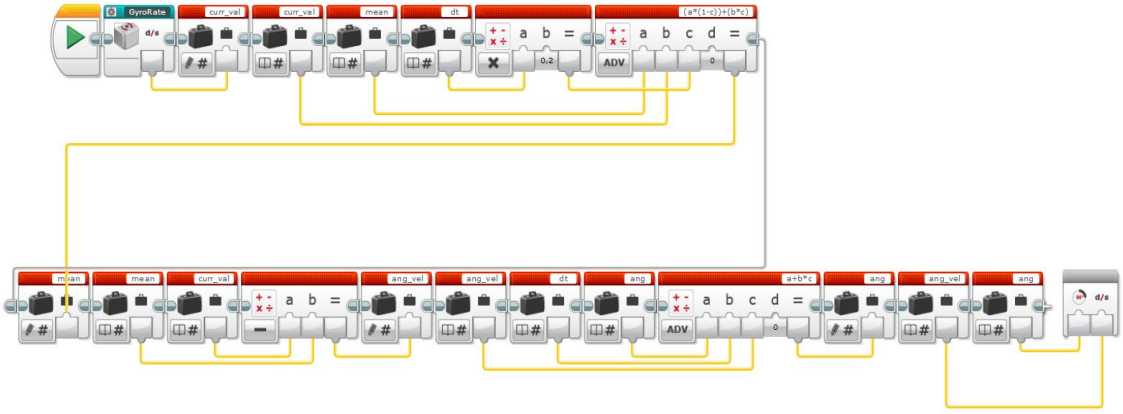


Figura 50: Bloque ReadGyro.

Figura 51: Bloque ReadGyro opción 1

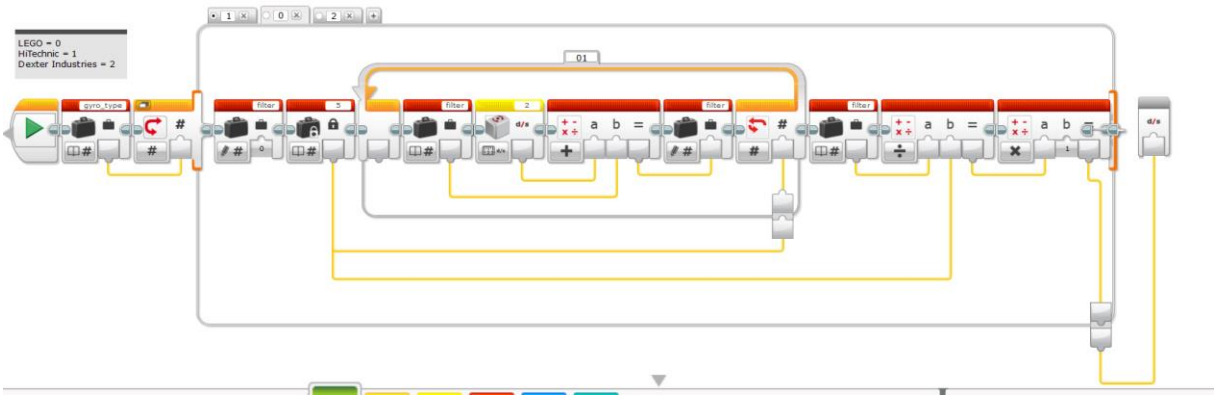


Figura 52: Bloque ReadGyro opción 0

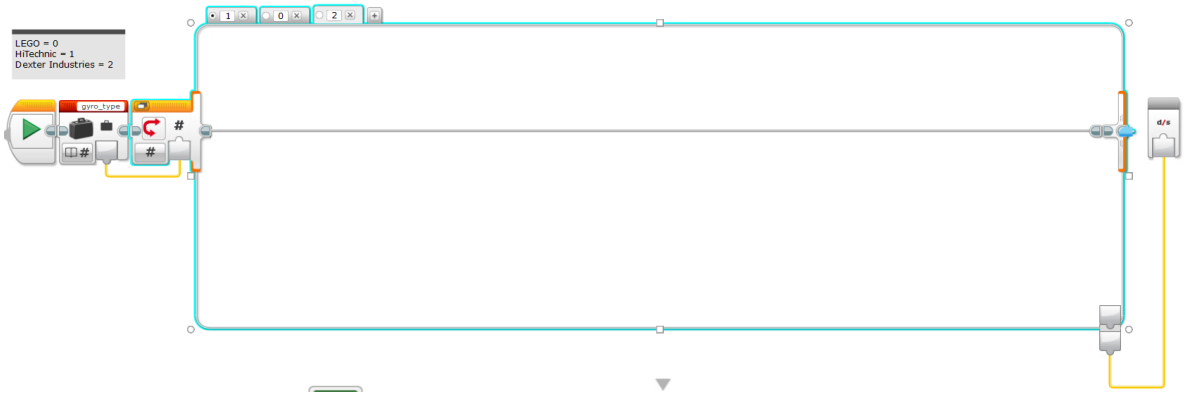


Figura 53: Bloque ReadGyro opción 2

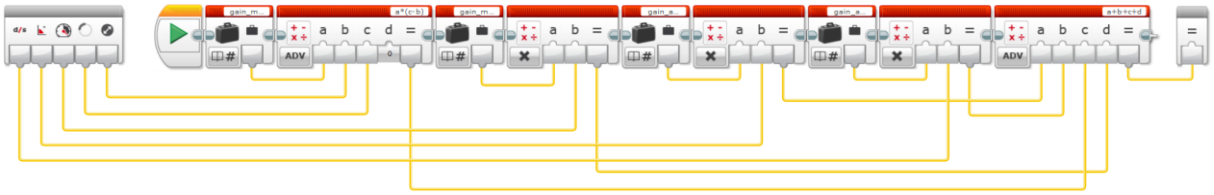


Figura 54: Bloque CombineSensorValues

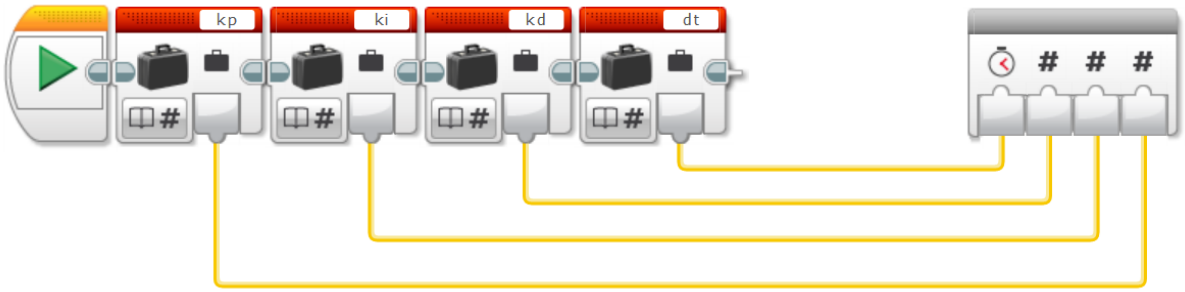


Figura 55: Bloque ReadConstants

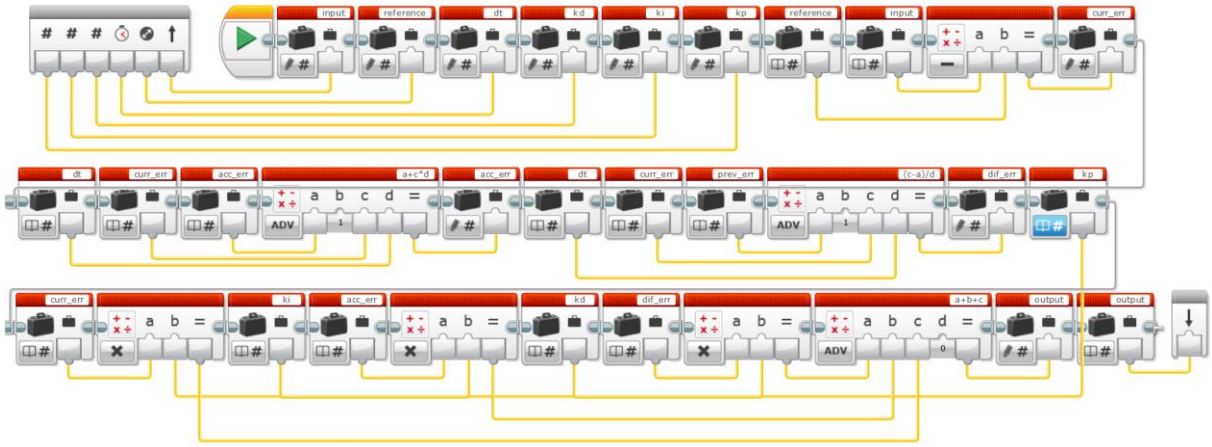


Figura 56: Bloque PID

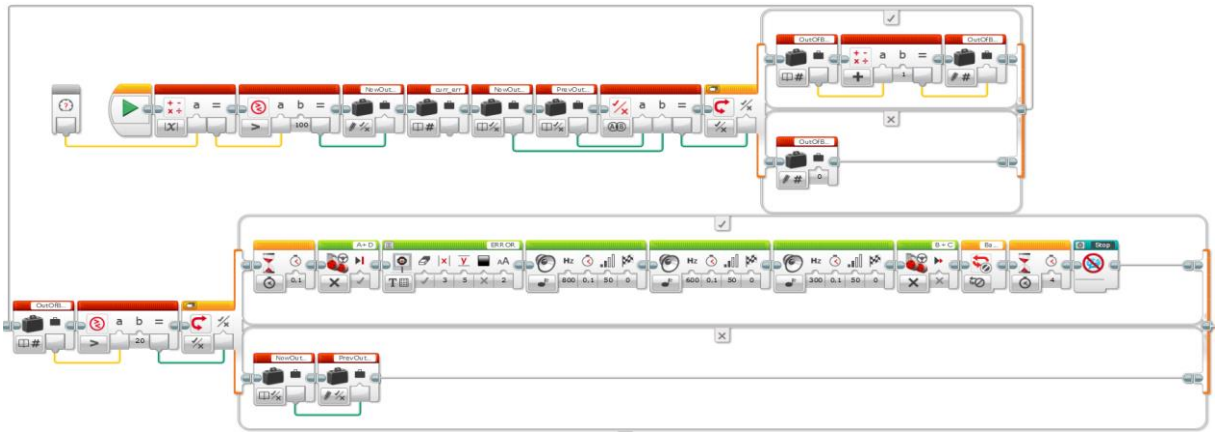


Figura 57: Bloque Errors

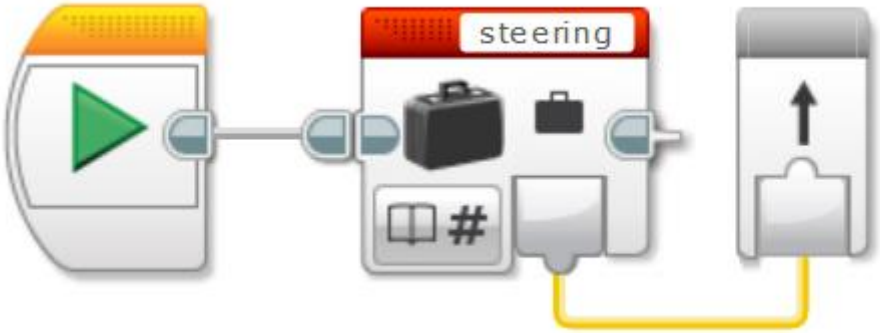


Figura 58: Bloque GetSteer

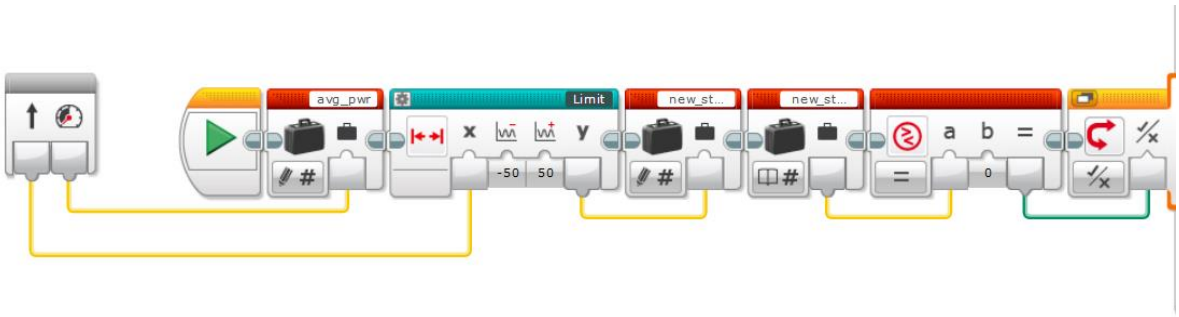


Figura 59: Bloque SetMotorPower parte a

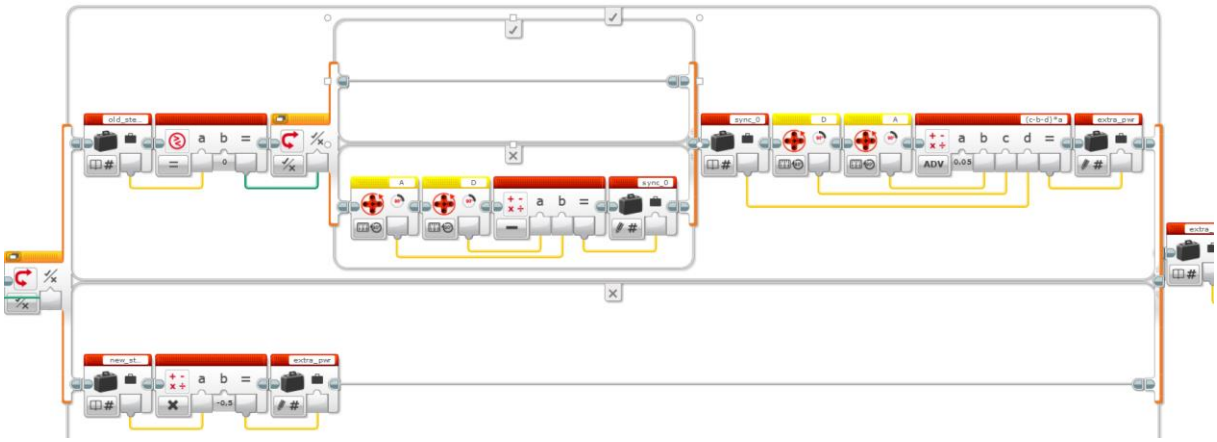


Figura 60: Bloque SetMotorPower parte b

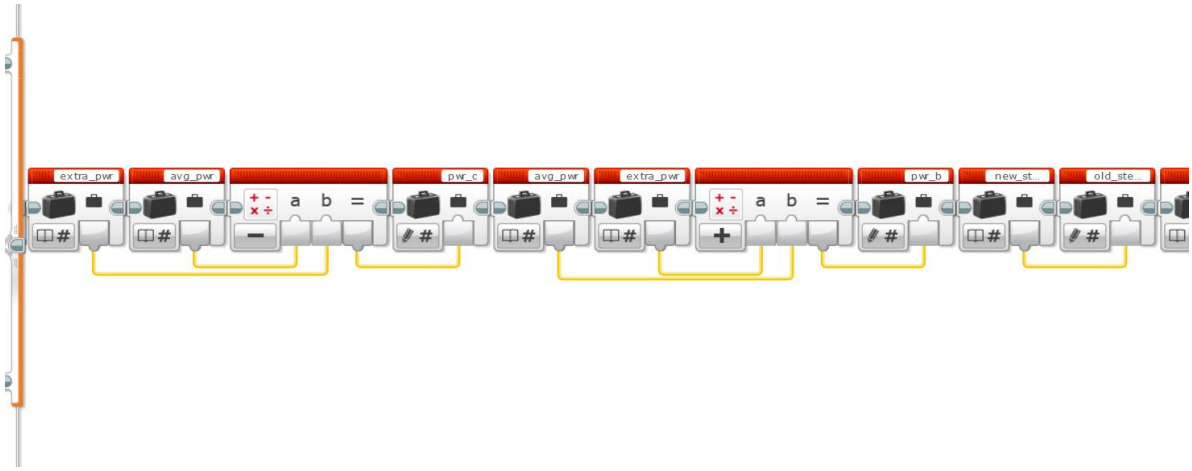


Figura 61: Bloque SetMotorPower parte c

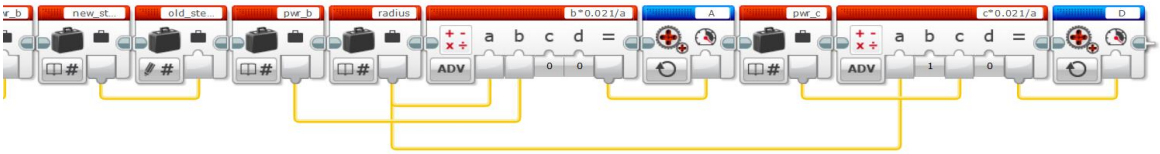


Figura 62: Bloque SetMotorPower parte final

Figura 63: Bloque Wait

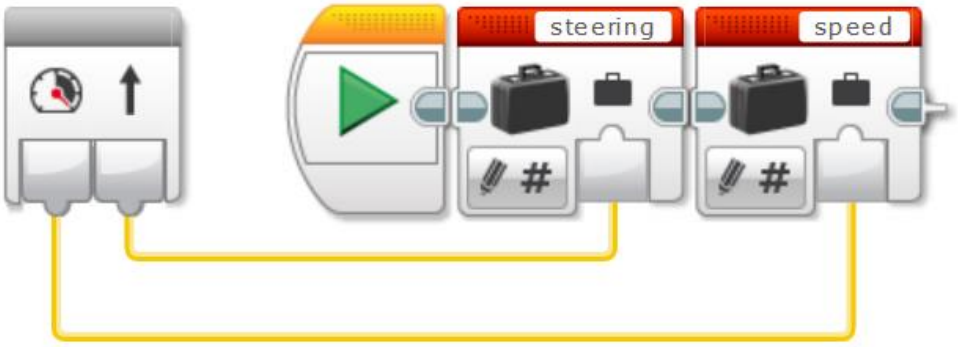


Figura 64: Bloque Move

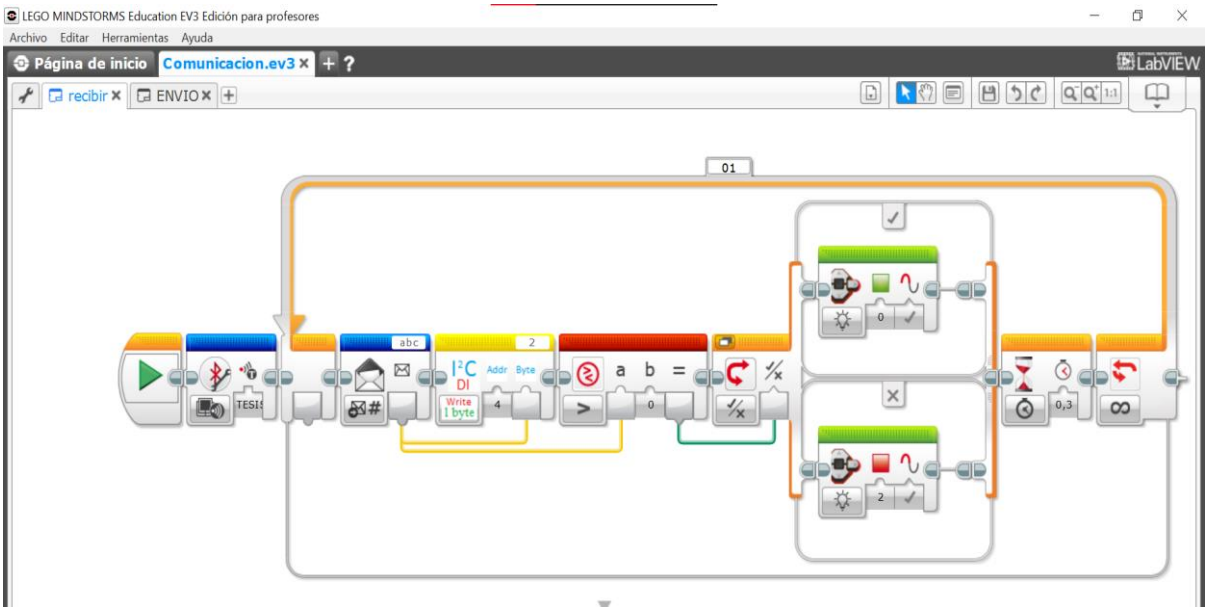


Figura 65: Conexión del segundo brick al Arduino

7.2 Programación del controlador difuso en Python

main.py

```
#!/usr/bin/env python3
# importacion de las librerias
########

########
from fuzzy_logic import *
import time
import sys
from collections import deque
import ev3dev.ev3 as ev3
from ev3dev.ev3 import *
import parameters
import importlib
import json

##############################################################################
## File I/O functions
##############################################################################

# Function for fast reading from sensor files
#Función para lectura rápida de archivos de sensores
# seek :  sirve para darle prioridad a un archivo
# decode :
# strip : devuelve una copia de la cadena
def FastRead(infile):
    infile.seek(0)
    return(int(infile.read().decode().strip()))

# Function for fast writing to motor files
# funcion para escritura rapida en archivos de motor
# truncate cambia el tamaño del archivo al numero dado de bytes
# write : sobrescribira cualquier contenido existente
```

```
# flush : borra el bufer interno del archivo
def FastWrite(outfile,value):
    outfile.truncate(0)
    outfile.write(str(int(value)))
    outfile.flush()

# Debug print
# impresion de depuracion
# print imprime el mensaje especificado en la pantalla
def eprint(*args, **kwargs):
    print(*args, file=sys.stderr, **kwargs)

# Function to set the duty cycle of the motors
# funcion para configurar el ciclo de trabajo de los motores
# La funcion round devuelve un numero de punto flotante que es una version redondeada del numero especificado
# la funcion min devuelve el elemento con el valor mas bajo
#redondea el valor colocado en duty * la compensacion de un voltaje

# si dutyint es mayor a 0 escoge el valor mas pequeño entre 100 y la suma de dutyint y friccion

# si dutyint es menor es resta

def SetDuty(motorDutyFileHandle, duty):
    # Compansate for nominal voltage and round the input
    dutyInt = int(round(duty*voltageCompensation))

    # Add or subtract offset and clamp the value between -100 and 100
    #sumar o restar comprensacion y fijar el valor entre -100 y 100
    if dutyInt > 0:
        dutyInt = min(100, dutyInt + frictionOffset)
    elif dutyInt < 0:
```

```
        dutyInt = max(-100, dutyInt - frictionOffset)

    # Apply the signal to the motor
    # aplica la señal del motor
    FastWrite(motorDutyFileHandle, dutyInt)


################################################################################
## One-time Hardware setup
## configuracion de hardware unica
################################################################################

# powersupply = lee la fuente de alimentacion
# buttons = lee un boton
# EV3 Brick
powerSupply = ev3.PowerSupply()
buttons = ev3.Button()
cl = ColorSensor()
cl2 = ColorSensor()
#cl.mode='COL-COLOR'
cl = ev3.ColorSensor(address="ev3-ports:in3")
cl2 =  ev3.ColorSensor(address="ev3-ports:in4")




# Gyro Sensor setup. Possibly make this more generic to better support more sensors.
# configuracion del sensor giroscopico posiblemente haga esto mas genericopara soportar mejor mas sensores
```

```
94
95    # Try : le permite probar un bloque de codigo en busca de errores
96    # except : le permite manejar el error
97
98    #Establece el modo de velocidad de giro si esta conectado
99    # asume que se adjunta un giroscopio diferente si no encuentra el giroscopio lego
100   try: # Set LEGO gyro to Gyro Rate mode, if attached
101       gyroSensor          = ev3.GyroSensor()
102       gyroSensor.mode     = gyroSensor.MODE_GYRO_RATE
103       gyroType            = 'LEGO-EV3-Gyro'
104   except: # Assume HiTechnic Gyro is attached if LEGO Gyro not found
105       gyroSensor          = ev3.Sensor(address="ev3-ports:in2")
106       gyroType            = 'HITECHNIC-NXT-Gyro'
107
108   # Open gyro rate sensor value file
109   # abrir archivo de valor del sensor de velocidad del giroscopio
110   # open : abra un archivo e imprima el contenido
111   gyroSensorValueRaw  = open(gyroSensor._path + "/value0", "rb")
112
113
114   # Configuracion del sensor tactil
115   # Touch Sensor setup
116   touchSensor         = ev3.TouchSensor()
117   touchSensorValueRaw = open(touchSensor._path + "/value0", "rb")
118
119
120   # configuracion del sensor infrarrojo
121   # IR Buttons setup
122   #irRemote           = ev3.InfraredSensor()
```

```
123   #irRemote.mode     = irRemote.MODE_IR_REMOTE
124   #irRemoteValueRaw  = open(irRemote._path + "/value0", "rb")
125   #irRemote = ev3.Sensor(address="ev3-ports:in1")
126   #touchSensor         = ev3.InfraredSensor()
127   #touchSensor.mode  = irRemote.MODE_IR_REMOTE
128   #touchSensorValueRaw = open(irRemote._path + "/value0", "rb")
129   #buttonCode = FastRead(irRemoteValueRaw)
130
131   # configuracion de los motores en el puerto b y D
132   # Configure the motors
133   motorLeft  = ev3.LargeMotor('outD')
134   motorRight = ev3.LargeMotor('outB')
135
136   ##########################################################################
137   ## Outer Loop (Uses Touch Sensor to easily start and stop "Balancing Loop")
138   ##########################################################################
139   #Bucle exterior ( utiliza un sensor tactil para iniciar y detener facilmente el bucle de equilibrio)
140
141   while True:
142
143       ######################################################################
144       ## Read/reload Parameters
145       ## leer y recargar parametros
146       ######################################################################
147
148       # Reload parameters class
149       # importa la libreria parametros
150       importlib.reload(parameters)
151       powerParameters = parameters.Power()
```

```
    gyroParameters  = parameters.Gyro(gyroType)
    motorParameters = parameters.Motor()
    gains           = parameters.Gains()
    timing          = parameters.Timing()

    # Define Math constants and conversions
    # definir conversiones y constantes matematicas
    # The number of radians in a degree/el numero de radianes en un grado
    radiansPerDegree              = 3.14159/180
    # Rate in radians/sec per gyro output unit/tasa en radianes/s por unidad de salida de giro
    radiansPerSecondPerRawGyroUnit = gyroParameters.degPerSecondPerRawGyroUnit*radiansPerDegree
    # Angle in radians per motor encoder unit/angulo en radianes por unidad de codificador de motor
    radiansPerRawMotorUnit         = motorParameters.degPerRawMotorUnit*radiansPerDegree
    # Actual speed in radians/sec per unit of motor speed /velocidad real en radianes/s por unidad de velocidad del m
    radPerSecPerPercentSpeed       = motorParameters.RPMperPerPercentSpeed*6*radiansPerDegree


    #leer el voltaje de la bateria
    # Read battery voltage
    # de aqui sale voltageCompensation que se encuentra en setDuty
    voltageIdle = powerSupply.measured_volts
    voltageCompensation = powerParameters.voltageNominal/voltageIdle

    # Offset to limit friction deadlock
    # comprensacion para limitar el punto muerto por friccion
    frictionOffset = int(round(powerParameters.frictionOffsetNominal*voltageCompensation))

    #Timing settings for the program
    #ajustes de tiempo para el programa
```

```
    # Time of each loop, measured in seconds/tiempo de cada bucle medido en segundos
    loopTimeSec             = timing.loopTimeMiliSec/1000
    # Loop counter, starting at 0 / contandor de bucle comenzando en 0
    loopCount               = 0

    # A deque (a fifo array) which we'll use to keep track of previous motor positions, which we can use to calculate
    # Una deque ( una matriz fifo )que usaremos para realizar un seguimiento de las posiciones anteriores del motor
    #que podemos usar para calcular la tasa de cambio
    # deque : deque se prefiere a la lista en los casos en los que necesitamos operaciones de adicion y extraccion
    # mas rapidas desde ambos extremos del contenedor
    motorAngleHistory = deque([0],timing.motorAngleHistoryLength)

    # The rate at which we'll update the gyro offset (precise definition given in docs)
    # la velocidad a la que actualizamos el desplazamiento del giroscopo ( definicion precisa dada en los documentos)
    gyroDriftCompensationRate      = timing.gyroDriftCompensationFactor*loopTimeSec*radiansPerSecondPerRawGyroUnit

    ################################################################################
    ## Hardware (Re-)Config
    #configuracion de hardware
    ################################################################################

    # Reset the motors
    # reiniciar los motores
    motorLeft.reset()                   # Reset the encoder / resetea los motores
    motorRight.reset()
    motorLeft.run_direct()              # Set to run direct mode / configurado para ejecutar el modo directo
    motorRight.run_direct()

```

```
209     # Open sensor files for (fast) reading
210     # Abrir archivos de sensor para lectura rapida
211     motorEncoderLeft    = open(motorLeft._path + "/position", "rb")
212     motorEncoderRight   = open(motorRight._path + "/position", "rb")
213
214     # Open motor files for (fast) writing
215     # abrir archivos de motor para escritura rapida
216     motorDutyCycleLeft = open(motorLeft._path + "/duty_cycle_sp", "w")
217     motorDutyCycleRight= open(motorRight._path + "/duty_cycle_sp", "w")
218
219     ################################################################################
220     ## Definitions and Initialization variables
221     # definiciones y varuables de inicializacion
222     ################################################################################
223
224     # Reset variables representing physical signals
225     # restablecer las variables que representan la señal fisica
226     motorAngleRaw              = 0
227     motorAngleReference        = 0
228     motorAngleError            = 0
229     motorAngleErrorAccumulated = 0
230     motorAngularSpeed          = 0
231     motorAngularSpeedReference = 0
232     motorAngularSpeedError     = 0
233     motorDutyCycle             = 0
234     gyroRateRaw                = 0
235     gyroRate                   = 0
236     gyroEstimatedAngle         = 0
237     gyroOffset                 = 0
238
```

```
    # imprimir instrucciones de inicio y parada
    # Print start and stop instructions
    eprint("Hold robot upright. Press Touch Sensor to start. Or any button to exit.")

    # Wait for Touch Sensor or any Button Press
    # esperar el sensor tactil o cualquier botor
    while not touchSensor.is_pressed and not buttons.any():
        time.sleep(0.01)

    #while not buttonCode == 0 and not buttons.any():
    # time.sleep(0.01)

    # si se presiono alguno de los botones , salga del programa rompiendo el bucle externo
    # If any of the buttons was pressed, exit the program by breaking the outer loop
    if buttons.any():
        break

    # de lo contrario , si fue el sensor tactil , espere a que se suelte y procesa a la calibracion y balanceo
    # Otherwise, if it was the Touch Sensor, wait for release and proceed to calibration and balancing
    while touchSensor.is_pressed:
        time.sleep(0.01)

    #while buttonCode > 0 :
     #time.sleep(0.01)

    ##########################################################################
    ## Create empty datalogs
    # crear registros de datos vacios
    ##########################################################################
```

```
    datalog = {
        'timeStart' : time.strftime("UTC: %Y-%m-%d-%H:%M:%S"),
        'tLoopStart' : [],
        'gyroRate' : [],
        'gyroEstimatedAngle' : [],
        'motorAngle' : [],
        'motorAngleError' : [],
        'motorAngularSpeed' : [],
        'motorAngleErrorAccumulated' : [],
        'motorDutyCycle' : [],
        'gyroOffset': []
    }

    ##########################################################################
    ## Calibrate Gyro
    # calibrar sensor
    ##########################################################################

    eprint("-------\nCalibrating\n-------")

    #As you hold the robot still, determine the average sensor value of 100 samples
    # mientras sostiene el robot quieto determine el valor promedio del sensor de 100 muestras
    gyroRateCalibrateCount = 100
    for i in range(gyroRateCalibrateCount):
        gyroOffset = gyroOffset + FastRead(gyroSensorValueRaw)
        time.sleep(0.01)
    gyroOffset = gyroOffset/gyroRateCalibrateCount

    # Print calibration result
    eprint("GyroOffset: ",gyroOffset)
```

```
298     eprint("GyroOffset: ",gyroOffset)
299     eprint("-----------\nGo!\n-----------")
300
301     ################################################################################
302     ## Balancing Loop
303     ## bucle de balanceo
304     ################################################################################
305
306     # Remember start time
307     # recuerda la hora de inicio
308     tProgramStart = time.time()
309
310     # Initial fast read touch sensor value
311     # valor inicial del sensor tactil de lectura rapida
312     touchSensorPressed = False
313
314
315
316
317     # Keep looping until Touch Sensor is pressed again
318     # siga girando hasta que se presione de nuevo el sensor tactil
319
320     #while not touchSensorPressed:
321     while not touchSensorPressed:
322         eprint(" DENTRO DEL BUCLE DE SEGUIDOR DE LINEA")
323         ########################################################################
324         ##  Loop info
325         # informacion de bucle
326         ########################################################################
```

```
        loopCount = loopCount + 1
        tLoopStart = time.time() - tProgramStart

        ###############################################################
        ##
        ##  Driving and Steering. Modify this section as you like to
        ##  make your segway go anywhere!
        ##
        ##  To begin, uncomment one of the examples below, and modify
        ##  from there
        ##
        ##  conduccion y direccion , modifique esta seccion como desse para que su segway vaya a cualquier parte
        ##  para comenzar , desactive uno de los ejemplos a continuacion y modifique desde aali
        ##
        ###############################################################

        # Example 1: Doing nothing: just balance in place:
        # no hace nada : solo mantener el equilibrio en su lugar



        speed    = 0
        steering = 0



        # Example 2: Control speed and steering based on the IR Remote
        # controlar la velocidad y la direccion con el mando a distancia por infrarrojos
        # buttonCode = FastRead(irRemoteValueRaw)

```

```
        speed_max = 20
        steer_max_right = 8


        '''
        if cl.value()>20 and cl2.value()>20:
            eprint('negro')
            speed    =   speed_max
            steering =  0
        if cl.value()<20 and cl2.value()>20:
            eprint('blanco')
            speed    =  0
            steering =  -steer_max_right
        if cl.value()>20 and cl2.value()<20:
            eprint('blanco')
            speed    =  0
            steering =  steer_max_right
        else:
            speed    = 0
            steering = 0
        '''


      # if(buttonCode == 0):



```

```
385         if cl.value()>0 and cl.value()<21:
386             eprint('negro')
387             speed    =  10
388             steering =  -8
389         if cl.value()> 18:
390             eprint('blanco')
391             speed    =  3
392             steering =  6
393 
394 
395         '''
396         if(buttonCode == 5):
397             speed    =  speed_max
398             steering =  0
399         elif (buttonCode == 6):
400             speed    =  0
401             steering = -steer_max_right
402         elif (buttonCode == 7):
403             speed    =  0
404             steering = steer_max_right
405         elif (buttonCode == 8):
406             speed    = -speed_max
407             steering = 0
408         else:
409             speed    = 0
410             steering = 0
411         '''
412 
413         ######################################################################
```

```
414         ##  Reading the Gyro.
415         ## leyendo el giroscopio
416         ################################################################
417         gyroRateRaw = FastRead( gyroSensorValueRaw)
418         gyroRate = (gyroRateRaw - gyroOffset)*radiansPerSecondPerRawGyroUnit
419 
420         ################################################################
421         ##  Reading the Motor Position
422         ## leer la posicion del motor
423         ################################################################
424 
425         motorAngleRaw = (FastRead(motorEncoderLeft) + FastRead(motorEncoderRight))/2
426         motorAngle = motorAngleRaw*radiansPerRawMotorUnit
427 
428         motorAngularSpeedReference = speed*radPerSecPerPercentSpeed
429         motorAngleReference = motorAngleReference + motorAngularSpeedReference*loopTimeSec
430 
431         motorAngleError = motorAngle - motorAngleReference
432 
433         ################################################################
434         ##  Computing Motor Speed
435         ## calcular la velocidad del motor
436         ################################################################
437 
438         motorAngularSpeed = (motorAngle - motorAngleHistory[0])/(timing.motorAngleHistoryLength*loopTimeSec)
439         motorAngularSpeedError = motorAngularSpeed# - motorAngularSpeedReference # Uncommenting this leads to a raher
440         #sin comentar esto , conduce a un cambio brusco mas raro en la velocidad cuando se usa el control remoto
441         #asi que lo dejare comentado hasta que agregue algun codigo que aumente
442         #gradualmente esta referencia cuando se presiona o se despreciona un boton
443         motorAngleHistory.append(motorAngle)
```

```
444 
445         ################################################################
446         ##  Computing the motor duty cycle value
447         ## Calcular el valot del ciclo de trabajo del motor
448         ################################################################
449 
450         motorDutyCycle =( gains.GyroAngle  * gyroEstimatedAngle
451                         + gains.GyroRate   * gyroRate
452                         + gains.MotorAngle * motorAngleError
453                         + gains.MotorAngularSpeed * motorAngularSpeedError
454                         + gains.MotorAngleErrorAccumulated * motorAngleErrorAccumulated)
455 
456         ################################################################
457         ##  Apply the signal to the motor, and add steering
458         # aplique la señal al motor y agregue direccion
459         ################################################################
460 
461         SetDuty(motorDutyCycleRight, motorDutyCycle + steering)
462         SetDuty(motorDutyCycleLeft , motorDutyCycle - steering)
463 
464         ################################################################
465         ##  Update angle estimate and Gyro Offset Estimate
466         ## actualizar la estimacion del angulo y la estimacion del desplazamiento del giroscopio
467         ################################################################
468 
469         gyroEstimatedAngle = gyroEstimatedAngle + gyroRate*loopTimeSec
470         gyroOffset = (1-gyroDriftCompensationRate)*gyroOffset+gyroDriftCompensationRate*gyroRateRaw
471 
472         ################################################################
473         ##  Update Accumulated Motor Error
```

```
474         ## actualizar error acumulado del motor
475         ######################################################################
476 
477         motorAngleErrorAccumulated = motorAngleErrorAccumulated + motorAngleError*loopTimeSec
478 
479         ######################################################################
480         ##  Read the touch sensor (the kill switch)
481         # leer el sensor tactil (el interruptor de apagado)
482         ######################################################################
483 
484        # touchSensorPressed = FastRead(touchSensorValueRaw)
485 
486         ######################################################################
487         ##  Append data datalog
488         ## añadir registro de datos
489         ######################################################################
490 
491         datalog['tLoopStart']                .append(tLoopStart)
492         datalog['gyroRate']                  .append(gyroRate)
493         datalog['gyroEstimatedAngle']        .append(gyroEstimatedAngle)
494         datalog['motorAngle']                .append(motorAngle)
495         datalog['motorAngleError']           .append(motorAngleError)
496         datalog['motorAngularSpeed']         .append(motorAngularSpeed)
497         datalog['motorAngleErrorAccumulated'].append(motorAngleErrorAccumulated)
498         datalog['motorDutyCycle']            .append(motorDutyCycle)
499         datalog['gyroOffset']                .append(gyroOffset)
500 
501         ######################################################################
502         ##  Busy wait for the loop to complete
503         ## ocupada espera a que el bucle se complete
```

```
504         ##################################################################
505
506         while(time.time()-tProgramStart - tLoopStart <  loopTimeSec):
507             time.sleep(0.0001)
508
509     ############################################################################
510     ##
511     ## Closing down & Cleaning up
512     ## Cerrando y limpiando
513     ##
514     ############################################################################
515
516     # Loop end time, for stats
517     # tiempo de finalizacion del bucle , para estadisticas
518     tProgramEnd = time.time()
519
520     # Turn off the motors
521     # apaga los motores
522     FastWrite(motorDutyCycleLeft ,0)
523     FastWrite(motorDutyCycleRight,0)
524
525     # Wait for the Touch Sensor to be released
526     # espere a que se suelte el sensor tactil
527    # while touchSensor.is_pressed:
528     #    time.sleep(0.01)
529
530     # Calculate loop time
531     # calcular el tiempo de ciclo
532     tLoop = (tProgramEnd - tProgramStart)/loopCount
533     eprint("Loop time:", tLoop*1000,"ms")
```

```
534
535     # Write datalog file
536     #excribir archivos de registro de datos
537     with open('datalog.txt', 'w') as f:
538         f.write(json.dumps(datalog))
539
540     # Print a stop message
541     # imprimir un mensaje de parada
542     eprint("----------\nStopped\n--------")
543
544
545 def main():
546
547
548
549
550
551
552     salida = int (250)
553     angulo = gyro_sensor.angle()
554     ev3.screen.print(angulo)
555
556     currentTemp = float(angulo)
557
558
559     prevError = targetTemp - prevTemp
560     currentError = targetTemp - currentTemp
561
562     error = currentError
```

```
595         # RULE 1
596         rules[0][0] = min(fuzzifiedErrorNeg, fuzzifiedErrorDotNeg)
597         # RULE 2
598         rules[0][1] = min(fuzzifiedErrorZero, fuzzifiedErrorDotNeg)
599         # RULE 3
600         rules[0][2] = min(fuzzifiedErrorPos, fuzzifiedErrorDotNeg)
601         # RULE 4
602         rules[1][0] = min(fuzzifiedErrorNeg, fuzzifiedErrorDotZero)
603         # RULE 5
604         rules[1][1] = min(fuzzifiedErrorZero, fuzzifiedErrorDotZero)
605         # RULE 6
606         rules[1][2] = min(fuzzifiedErrorPos, fuzzifiedErrorDotZero)
607         # RULE 7
608         rules[2][0] = min(fuzzifiedErrorNeg, fuzzifiedErrorDotPos)
609         # RULE 8
610         rules[2][1] = min(fuzzifiedErrorZero, fuzzifiedErrorDotPos)
611         # RULE 9
612         rules[2][2] = min(fuzzifiedErrorPos, fuzzifiedErrorDotPos)
613         return rules
614 
615     def fuzzifyErrorPosG(error):
616             return trimf(error, [2.5, 10, 12])
617 
618     def fuzzifyErrorPos(error):
619             return trimf(error, [0, 2.5, 7.5])
620 
621     def fuzzifyErrorZero(error):
622             return trimf(error, [-2.5, 0, 2.5])
623 
624     def fuzzifyErrorNeg(error):
```

```
625         return trimf(error, [0, -2.5, -7.5])
626
627     def fuzzifyErrorNegG(error):
628         return trimf(error, [-2.5, -10, -12])
629
630
631     def fuzzifyDerrorDotPosG(Derror):
632         return trapmf(Derror, [0.2, 0.5, 0.6])
633
634     def fuzzifyDerrorDotPos(Derror):
635         return trapmf(Derror, [0, 0.2, 0.4])
636
637
638     def fuzzifyDerrorDotZero(Derror):
639         return trimf(Derror, [-0.2, 0, 0.2])
640
641
642     def fuzzifyDerrorDotNeg(Derror):
643         return trapmf(Derror, [0, -0.2, -0.4])
644
645     def fuzzifyDerrorDotNeg(Derror:
646         return trapmf(Derror, [-0.2, -0.5, -0.6])
647
648
649
650     def fuzzifyOutputAdelanteMax():
651         return getTrapmfPlots(0, 10, [10, 40,80], "left")
652
653     def fuzzifyOutputAdelante():
654        return getTrimfPlots(0, 10, [0, 10, 30])
```

```
656
657     def fuzzifyOutputCentro():
658         return getTrimfPlots(0, 10, [10, 0, 10])
659
660     def fuzzifyOutputAtras():
661         return getTrimfPlots(0, 10, [0, -10, -30])
662
663     def fuzzifyOutputAtrasMax():
664         return getTrapmfPlots(0, 10, [-10,-40, -80], "right")
665
666
667
668     def fisAggregation(rules, pcc, pcnc, pch):
669             result = [0] * 20
670             for rule in range(len(rules)):
671                 for i in range(10):
672                     if rules[rule][0] > 0 and i < 3:
673                         result[i] = min(rules[rule][0], pcc[i])
674                     if rules[rule][1] > 0 and i > 2 and i < 5:
675                         result[i] = min(rules[rule][1], pcnc[i])
676                     if rules[rule][2] > 0 and i > 4 and i < 10:
677                         result[i] = min(rules[rule][2], pch[i])
678             return result
679
680     if __name__ == "__main__":
681             main()
682
```

7.3 interfaz realizada en el software de LabVIEW

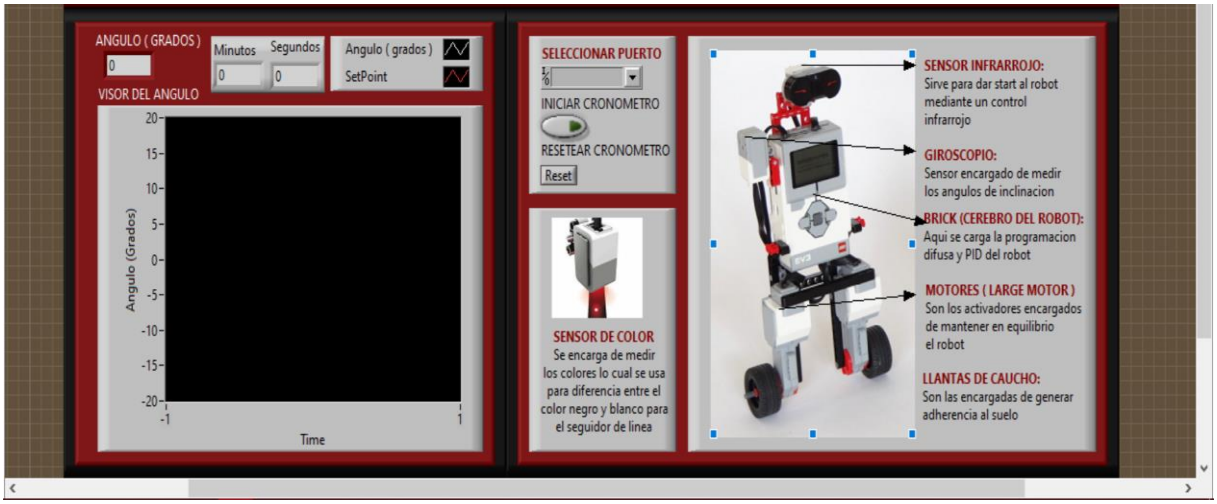


Figura 66: Interfaz para la lectura de parámetros realizada en LabVIEW

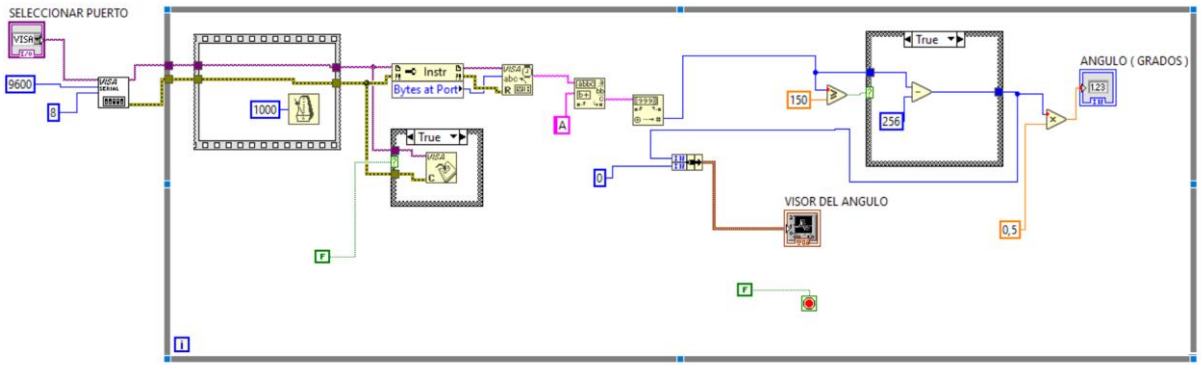


Figura 67: Diagrama de Bloque de la lectura de datos opcion true

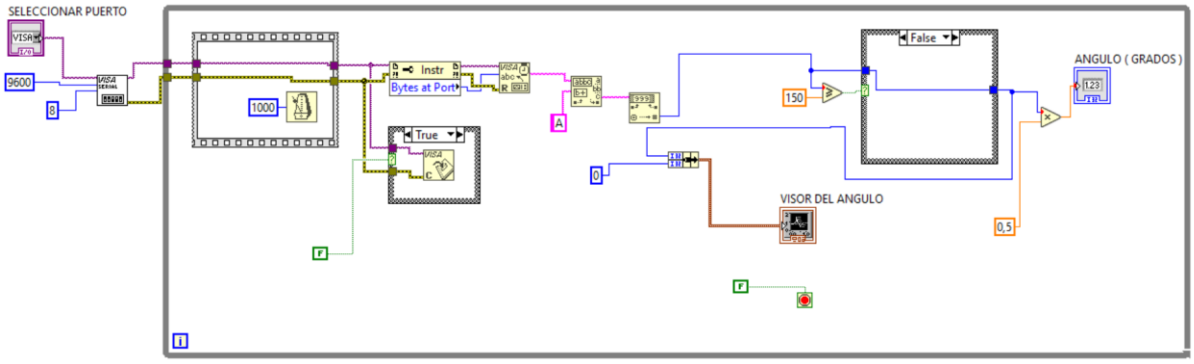


Figura 68: Diagrama de Bloque de la lectura de datos opción False

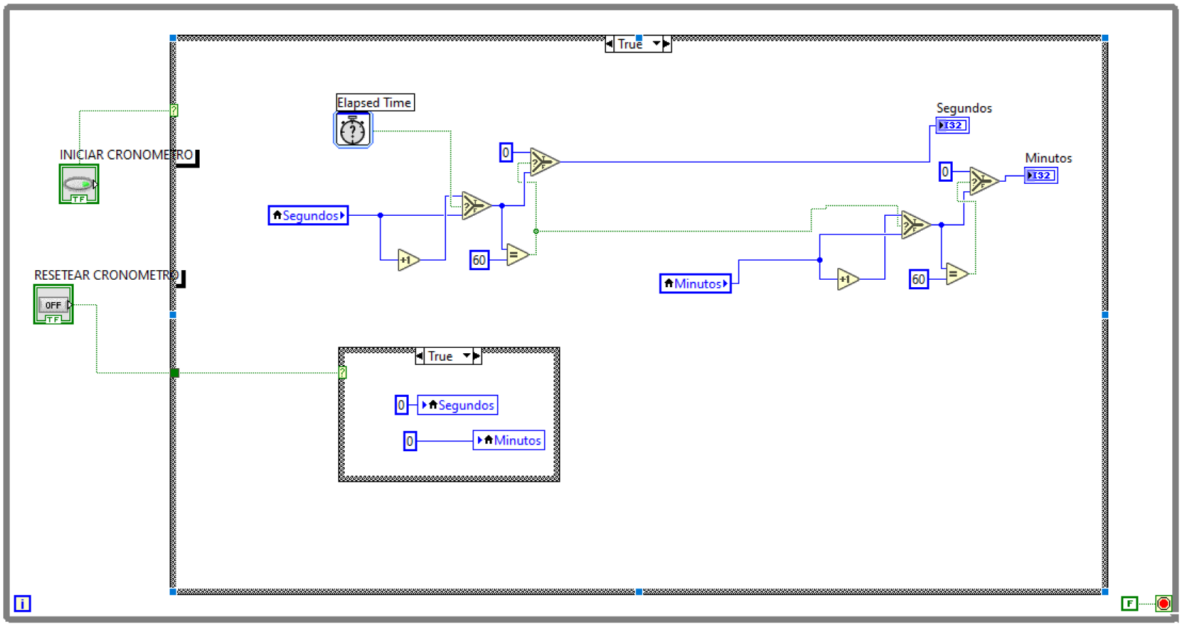


Figura 69: Diagrama de Bloque del reloj option true parte a

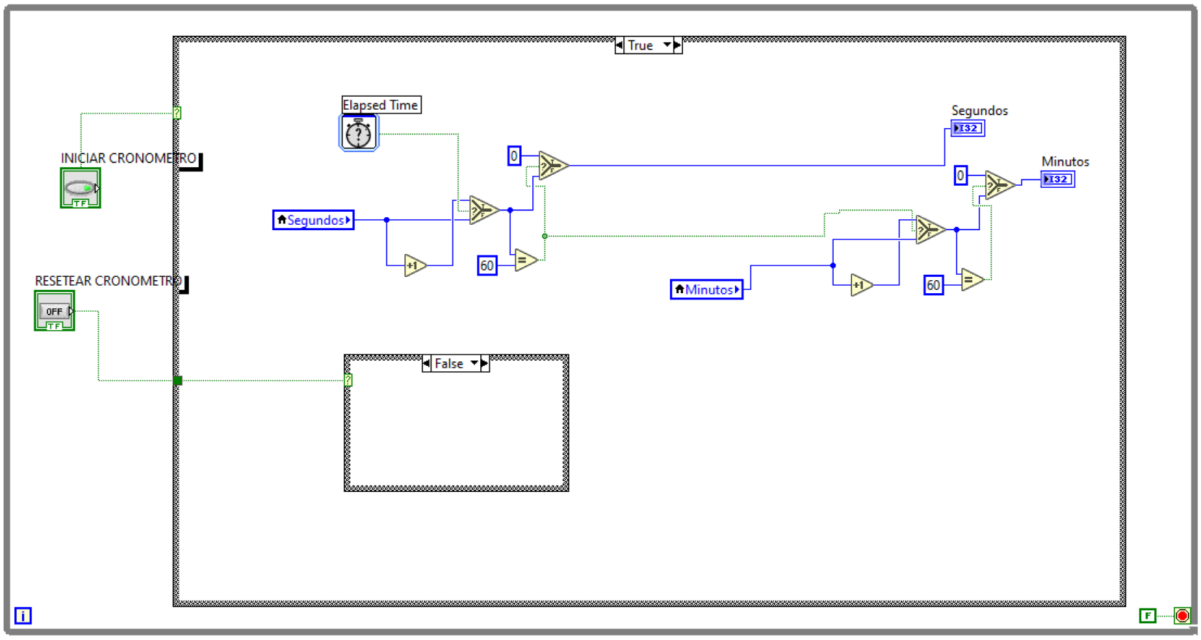


Figura 70: Diagrama de Bloque del reloj option true parte b

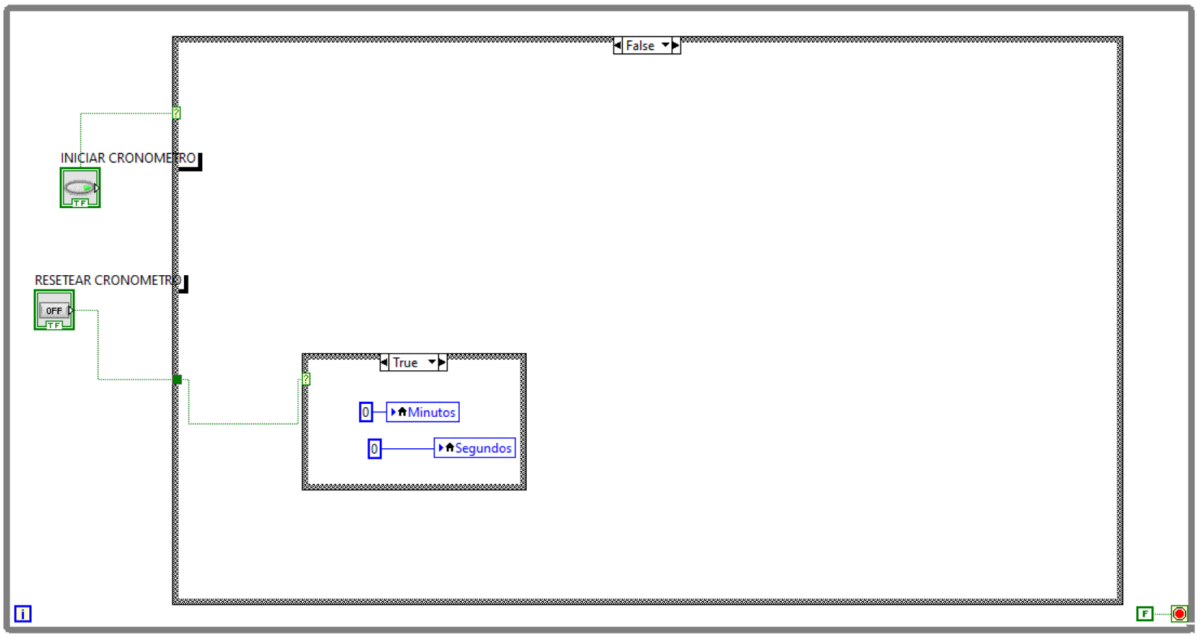


Figura 71: Diagrama de Bloque del reloj option false parte a

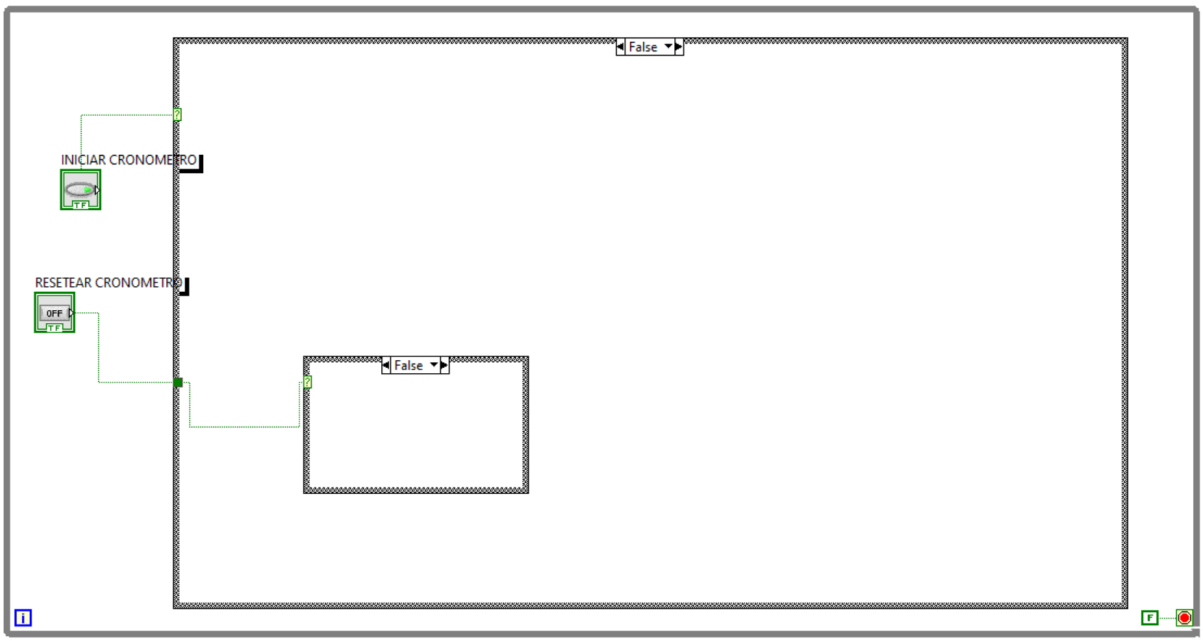
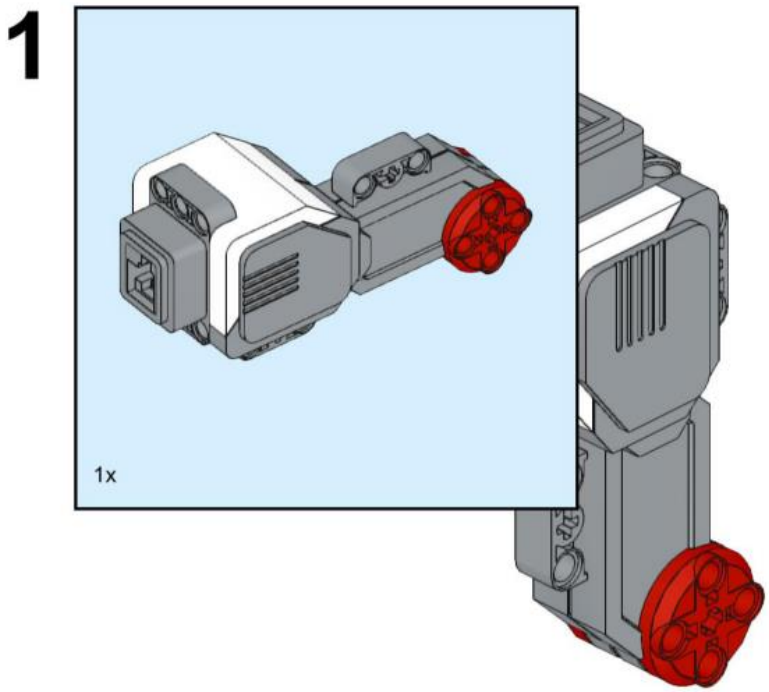


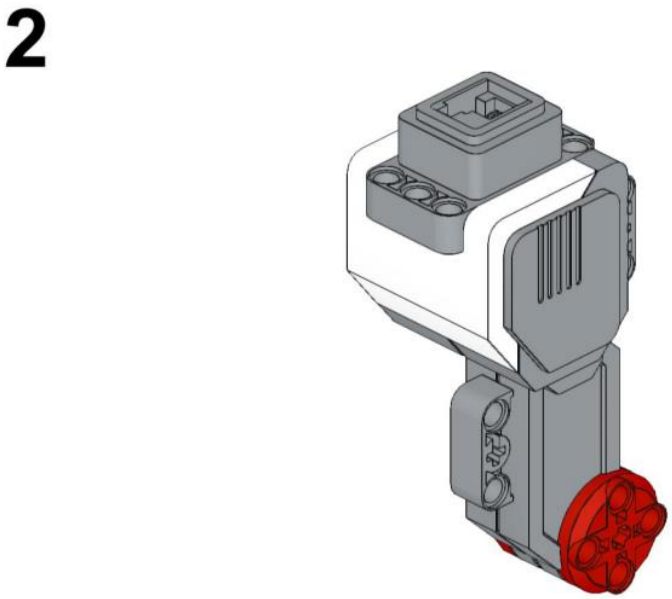
Figura 72: Diagrama de Bloque del reloj option false parte b

7.4 Programación en Arduino para comunicar con la interfaz

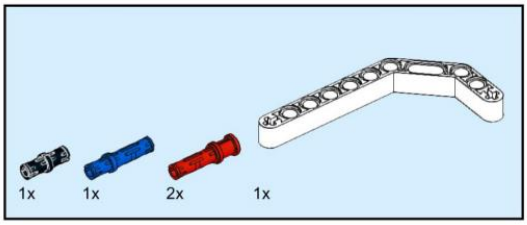
```
#include <Wire.h>
#include <Servo.h>

#define SLAVE_ADDRESS 0x04
#define SERVO_PIN       8

Servo srv;

void setup()
{
   Serial.begin(9600);
   Wire.begin(SLAVE_ADDRESS);
   Wire.onReceive(receiveData);
   srv.attach(SERVO_PIN);
}

void loop()
{
 /* String A7 = String ( 7 , DEC );
  String Datos = String("A"+A7);

   Serial.print(Datos);*/
}

void receiveData(int byteCount)
{
   while(Wire.available()>0)
   {

  String A7 = String ( Wire.read() , DEC );
  String Datos = String("A"+A7);
  Serial.print(Datos);

    // Serial.print(Wire.read());
     //srv.write(Wire.read());
   }


}
```

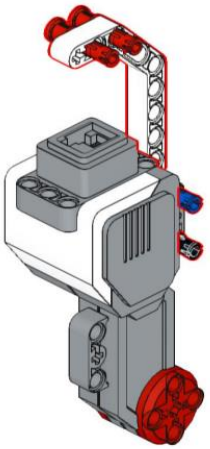
7.5 Instrucciones de construcción del robot Balance seguidor de línea



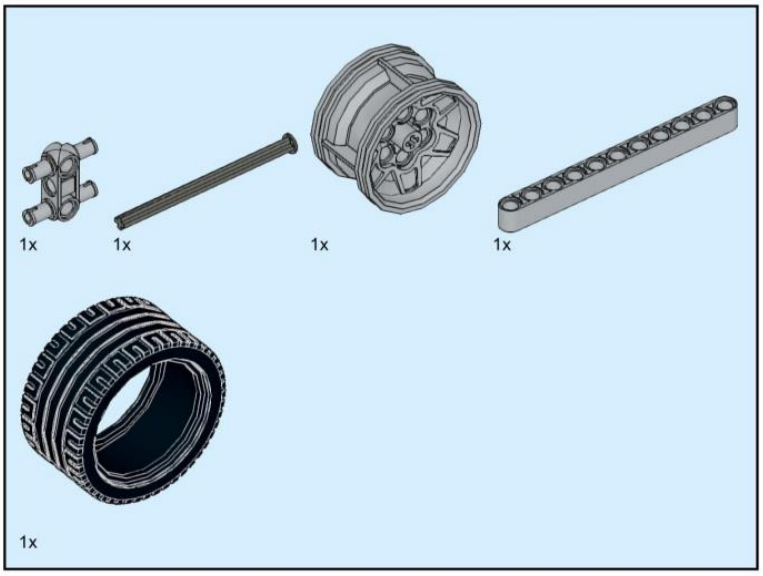


3

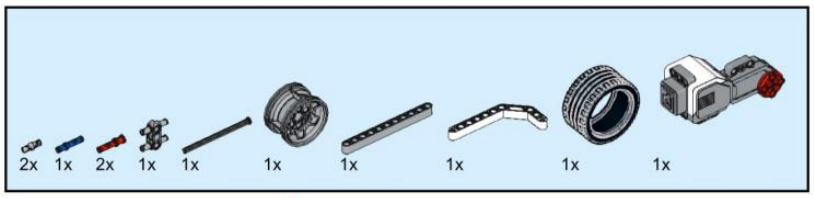


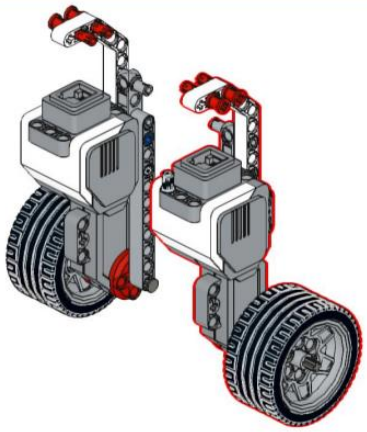


4

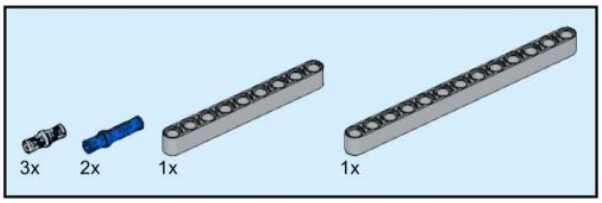


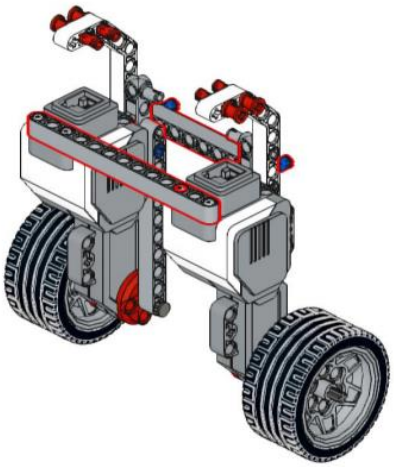
5



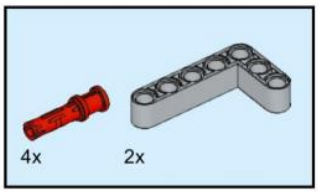


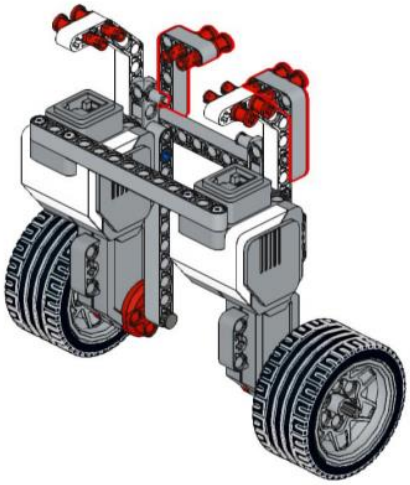
6



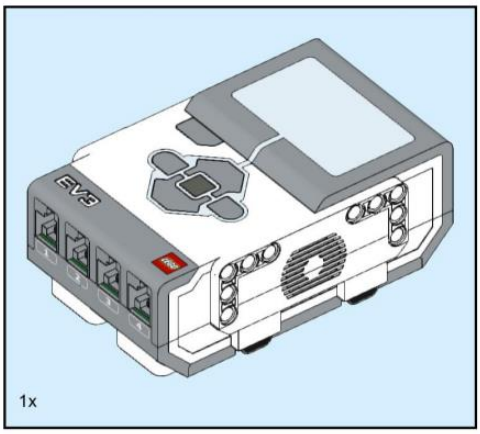


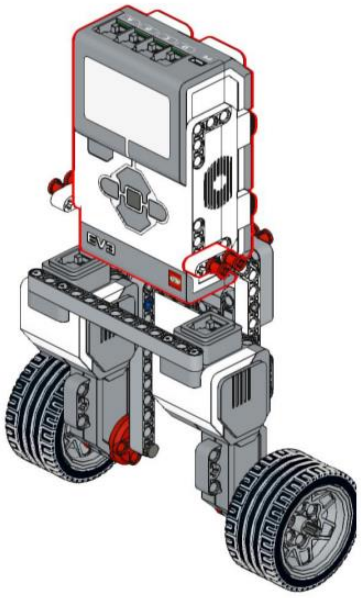
7



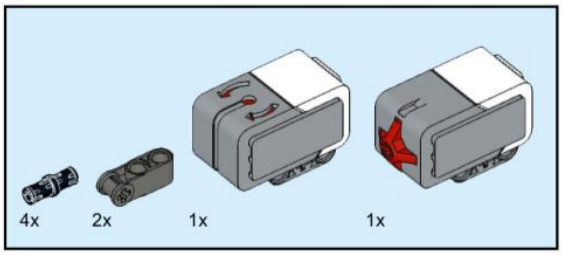


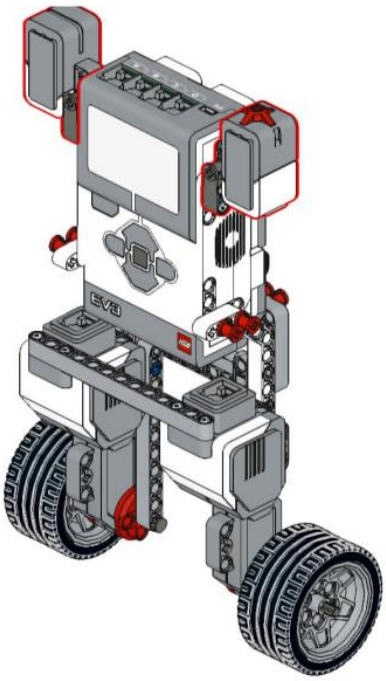
8



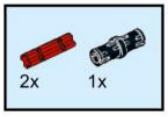


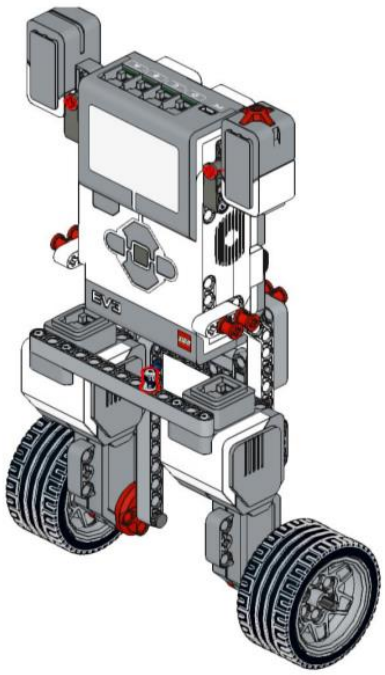
9



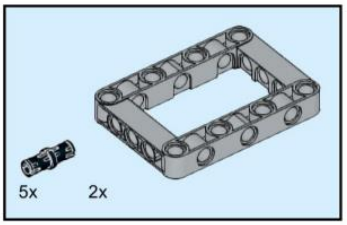


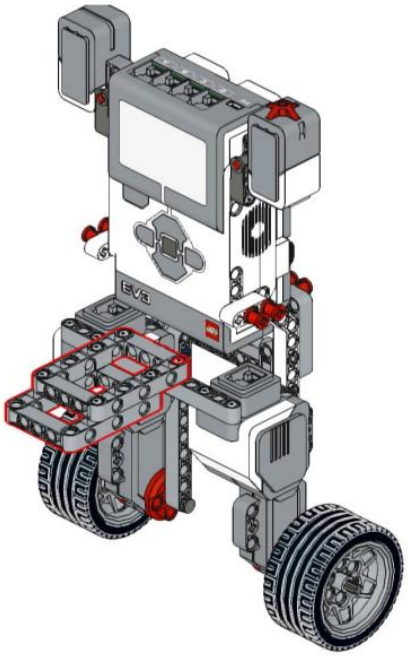
10





11





12