UNIVERSIDAD POLITÉCNICA SALESIANA

SEDE GUAYAQUIL

CARRERA DE INGENIERÍA ELECTRÓNICA

TRABAJO DE TITULACION PREVIO A LA OBTENCIÓN DEL TÍTULO DE:

INGENIERO ELECTRÓNICO

PROYECTO TÉCNICO:

IMPLEMENTACIÓN DE UN ALGORITMO DE CONTROL PARA UN ROBOT LINE FOLLOWER USANDO RED NEURONAL

AUTOR:

LUIS DAVID LÓPEZ ALARCÓN

TUTOR:

ING BYRON LIMA, MSC.

GUAYAQUIL-ECUADOR

CERTIFICADO DE RESPONSABILIDAD Y AUTORÍA

Yo, Luis David López Alarcón, con C.I 0940348972, estudiante de la carrera de Ingeniería Electrónica de la Universidad Politécnica Salesiana, declaramos que el trabajo descrito aquí ha sido desarrollado respetando los derechos intelectuales de terceros cuyas fuentes se incorporan en la bibliografía. Los análisis realizados y conclusiones del presente trabajo son de exclusiva responsabilidad de los autores.

Guayaquil, septiembre del 2020

Luis David López Alarcón

C.I: 0940348972

CERTIFICADO DE CESIÓN DE DERECHOS DE AUTOR

A través del presente certificado, se ceden los derechos de propiedad intelectual correspondiente a este trabajo, a la Universidad Politécnica salesiana, según lo establecido por la ley de propiedad intelectual y por su normatividad institucional vigente.

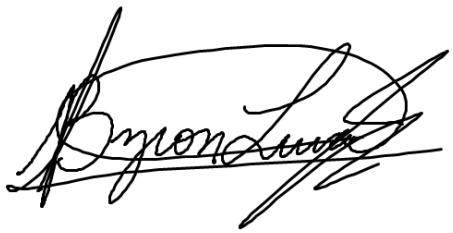
Guayaquil, septiembre del 2020

Luis David López Alarcón

C.I: 0940348972

CERTIFICADO DE DIRECCIÓN DEL TRABAJO DE TITULACIÓN

Por medio de la presente constancia que al Sr. Luis David López Alarcón ha desarrollado y elaborado satisfactoriamente el proyecto final de titulación, que se ajusta a las normas establecidas por la Universidad Politécnica Salesiana, por tanto, autorizo su presentación para los fines legales pertinentes.



Ing. Byron Lima MSc.

DIRECTOR DEL PROYECTO

AGRADECIMIENTO

Agradezco primero a Dios por darme la fuerza para seguir adelante cumpliendo cada logro propuesto, a mi familia y a mis amistades por ser parte de este camino lleno de aprendizaje, motivándome a culminar mi carrera de ingeniería.

A lo largo de esta trayectoria no ha sido sencillo este camino lleno de aprendizaje contando con personas que con el tiempo se vuelven amigos y que me respaldaban para poder continuar con mi preparación.

Agradezco al Ing. Byron Lima, Msc que con su guía y conocimientos lograron que pueda desarrollarme día a día profesionalmente.

DEDICATORIA

Se lo Dedico a Dios por darme salud y energía para seguir continuando en la carrera universitaria, también dedico esta tesis a mis padres, mis hermanas quienes no solo ahora en la universidad sino a lo largo de mi vida me han ayudado a alcanzar mis logros y me han apoyado para conseguirlos, son el soporte para siempre querer hacer algo nuevo y mejor en mi vida.

RESUMEN

Año	Alumno	Tutor de Proyecto de Titulación	Proyecto de Titulacion
2021	Luis David López Alarcón	Ing. Byron Lima, MSc.	Implementación de un algoritmo de control para un robot Line Follower usando Red Neuronal

El presente proyecto técnico de titulación, "Implementación de un algoritmo de control para un robot Line Follower usando Red Neuronal", por motivo de pandemia se lo realizo desde casa para los estudiantes de ciclos superiores especialmente en las materias sistemas inteligentes, robótica, control discreto, para poder realizar entrenamientos basados en la programación de robot Line Follower utilizando redes neuronales.

La propuesta del proyecto de titulación se basa en la Implementación de un algoritmo de control para un robot Line Follower usando Red Neuronal.

El desarrollo del proyecto consiste en una primera parte el diseño electrónico en el software Fusion 360 y verificar el respectivo funcionamiento.

Se implementa un algoritmo de control basado en la plataforma Mbed Studio con control Red neuronal que será capaz de centrar los sensores con respecto a la línea negra y recorrer toda la pista.

Esto se consigue usando una regleta de 16 sensores multiplexada y tarjeta Stm32L432KC que nos permitirá en el proceso en tiempo real sobre la posición que se encuentra los sensores con respecto a la línea negra y así obtener una salida de control que será aplicada a los motores de las ruedas para su movimiento.

El objetivo principal es que el estudiante por medio de una guía de prácticas le beneficie para el análisis y comportamiento de sistemas de control y así aplicar los conocimientos adquiridos en materias como sistemas inteligentes, control discreto, redes industriales, teoría de control, microprocesadores y electiva

Adicional, el monitoreo se lo realiza desde un PC para poder visualizar el comportamiento de nuestro robot Line Follower mientras recorre la pista.

PALABRAS CLAVES: Red Neuronal, PD difuso, PID, LabVIEW, Fusion 360, Stm32L432KC,

ABSTRACT

Year	Student	Degree Project Tutor	Technical degree project
2021	Luis David López Alarcón	Ing. Byron Lima, MSc.	Implementation of a control algorithm for a Line Follower robot using Neural Network

The present technical degree project, "Implementation of a control algorithm for a Line Follower robot using Neural Network", due to a pandemic, was carried out from home for students of higher cycles, especially in the subjects of intelligent systems, robotics, discrete control, to be able to carry out training based on the Line Follower robot programming using neural networks.

The proposal of the degree project is based on the Implementation of a control algorithm for a Line Follower robot using Neural Network.

The development of the project consists of a first part the electronic design in the Fusion 360 software and verifying the respective operation.

A control algorithm is implemented based on the Mbed Studio platform with Neural Network control that will be able to center the sensors with respect to the black line and travel the entire track.

This is achieved using a multiplexed 16-sensor strip and Stm32L432KC card that will allow us in the process in real time on the position of the sensors with respect to the black line and thus obtain a control output that will be applied to the motors of wheels for movement.

The main objective is that the student, through a practical guide, benefits him for the analysis and behavior of control systems and thus apply the knowledge acquired in subjects such as intelligent systems, discrete control, industrial networks, control theory, microprocessors and elective

Additionally, the monitoring is done from a PC to be able to visualize the behavior of our Line Follower robot while it travels the track.

KEY WORDS: Neural Network, Fuzzy PD, PID, LabVIEW, Fusion 360, Stm32L432KC,

ÍNDICE

CERTIFICADO DE RESPONSABILIDAD Y AUTORÍA II
CERTIFICADO DE CESIÓN DE DERECHOS DE AUTOR III
CERTIFICADO DE DIRECCIÓN DEL TRABAJO DE TITULACIÓN IV
AGRADECIMIENTO V
DEDICATORIA VI
RESUMEN VII
ABSTRACT VIII
ÍNDICE IX
ÍNDICE DE FIGURAS XII
ÍNDICE DE TABLAS XIV
INTRODUCCIÓN 1
1. EL PROBLEMA 2
1.1 Importancia y alcance 2
1.2 Delimitación 2
1.2.1 Delimitación temporal 2
1.2.2 Delimitación espacial 2
1.2.3 Delimitación académica 2
1.3 Objetivos 3
1.3.1 Objetivo general 3
1.3.2 Objetivos específicos 3
2. ESTADO DEL ARTE 4
2.1 Antecedentes 4
2.2 Sistemas embebidos 4
2.2.1 Stm32 Node L432kc 5
2.2.2 Mbed Studio 5
2.3 PID 5
2.4 Red Neuronal 6
2.4.1 Reseña Histórica 6
2.4.2 Definiciones de una red neuronal 7
2.4.3 Ventajas de las redes neuronales 7
2.5 Componentes del sistema 8
2.5.1 Motores de corriente continua 8
2.5.2 Controlador de Motor (TB16FN12G) 8
2.5.3 Reglador de voltaje 8

2.5.4 Regleta de Sensores 10
2.5.5 Turbina EDF 11
2.6 Módulo de comunicación BT 12
2.7 Comunicación Serial RS232 12
3. MARCO METODOLÓGICO 13
3.1 Diseño del robot Line Follower. 13
3.2 Diagrama de control 13
3.3 Esquema del robot 14
3.4 Conexiones del prototipo 14
3.4.1 Conexiones de la instrumentación 14
3.4.1.1 Sensores multiplexado 14
3.4.2 Conexiones de la comunicación 15
3.4.2.1 Bluetooth hc-05 15
3.4.2.2 Aplicativo en app inventor 15
3.4.3 Conexiones de la potencia y alimentación 16
3.4.3.1 Tarjeta controladora de motores 16
3.4.3.2 Fuente de poder 17
3.5 Controlador PID 17
3.5.1.1 Método de sintonización de ziegler nichols 17
3.6 Controlador PD Difuso 18
4. PRÁCTICAS DE LABORATORIO 20
4.1 Práctica #1 20
4.2 Práctica #2 21
4.3 Práctica #3 22
4.4 Práctica #4 23
4.5 Práctica #5 24
4.6 Práctica #6 25
4.7 Práctica #7 26
4.8 Práctica #8 27
4.9 Práctica #9 28
4.10 Práctica #10 29
5. RESULTADO 30
5.1 ANÁLISIS DE RESULTADOS. 30
5.1.1 Resultados de la práctica 1 - Diseño electrónico en Fusion 360 30
5.1.2 Resultados de la práctica 2 - Programación Mbed Studio para monitoreo de sensores. 31

5.1.3 Resultados de la práctica 3 - Programación Mbed Studio control de actuadores. 32
5.1.4 Resultados de la práctica 4 - Codificación de algoritmo para calibración de sensores. 36
5.1.5 Resultados de la práctica 5 - Control PID clásico aplicado al robot Line Follower. 38
5.1.6 Resultados de la práctica 6 - Control PID Neuronal aplicado al robot Line Follower. 39
5.1.7 Resultados de la práctica 7 - Diseño de algoritmo de entrenamiento para la Red Neuronal. 40
5.1.8 Resultados de la práctica 8 - Control PD Difuso implementado al robot Line Follower. 41
5.1.9 Resultados de la práctica 9 - Diseño de una aplicación móvil para el control del robot.42
5.1.10 Resultados de la práctica 10 - Adquisición de datos en tiempo real de la señal de control y la variable ajustada. 44
CONCLUSIONES 45
RECOMENDACIONES 46
BIBLIOGRAFÍA 47
ANEXOS 48
Anexo 1: PRÁCTICA #1 48
Anexo 2: PRÁCTICA #2 54
Anexo 3: PRÁCTICA #3 56
Anexo 4: PRÁCTICA #4 58
Anexo 5: PRÁCTICA #5 61
Anexo 6: PRÁCTICA #6 63
Anexo 7: PRÁCTICA #7 67
Anexo 8: PRÁCTICA #8 69
Anexo 9: PRÁCTICA #9 85
Anexo 10: PRÁCTICA #10 90

ÍNDICE DE FIGURAS

Figura 1. Carro Stanford en 1960 4
Figura 2. Diagrama de bloques PID 5
Figura 3. TB16FN12G en smd 8
Figura 4. Módulo TB16FN12G 8
Figura 5. Diagrama interno de un regulador de voltaje 9
Figura 6. Regulador Lineal fijo 9
Figura 7. Regulador Lineal ajustable 9
Figura 8. regulador por conmutacionLM2576 10
Figura 9. Regleta de sensores QTX 10
Figura 10. Regleta de 16 sensores multiplexado 10
Figura 11. Configuración QTR-8RC 11
Figura 12. Configuración QTR-8A 11
Figura 13. Turbina EDF 11
Figura 14. Módulos Bluetooth 12
Figura 15. Prototipo Line Follower 13
Figura 16. Diagrama de control PIDNN 13
Figura 17. Diagrama de control 14
Figura 18. Barra 16 sensores multiplexado conectado al Stm32. 14
Figura 19. Esquema de Stm32 con bluetooth 15
Figura 20. Configuración del bloque Visa Serial 15
Figura 21. Aplicativo diseñado en app inventor 16
Figura 22. Conexión de la tarjeta embebida con el driver de potencia 16
Figura 23. Batería Lipo 17
Figura 24. Código del control PID 18
Figura 25. Resultado de la Práctica 1 - Diseño de la PCB 30
Figura 26. Resultado Práctica 1 - Fabricación de la PCB 30
Figura 27. Resultado de la práctica 2 - monitoreo de 16 sensores. 31
Figura 28. Resultado de la práctica 2 - Movimiento de la regleta. 31
Figura 29. Resultado de la práctica 3 - Función para el control de los motores. 32
Figura 30. Resultado de la práctica 3 - motor derecho en alto hacia adelante. 32
Figura 31. Resultado de la práctica 3 - motor derecho en alto hacia atrás. 33
Figura 32. Resultado de la práctica 3 - motor izquierdo en alto hacia adelante. 33
Figura 33. Resultado de la práctica 3 - motor izquierdo en alto hacia atrás 34
Figura 34. Resultado de la práctica 3 - ambos motores en alto hacia adelante. 34
Figura 35. Resultado de la práctica 3 - ambos motores en alto hacia atrás. 35
Figura 36. Resultado de la práctica 3 - motor izquierdo hacia delante y motor derecho hacia atrás. 35
Figura 37. Resultado de la práctica 3 - motor izquierdo hacia atrás y motor derecho hacia adelante. 36
Figura 38. Resultado de la práctica 4 - dato de posición de la línea en 0 36
Figura 39. Resultado de la práctica 4 – dato de posición de la línea en 750. 37
Figura 40. Resultado de la práctica 4 – dato de posición de la línea en 1500. 37
Figura 41. Resultado de la práctica 5 – PID robot Line Follower. 38
Figura 42. Resultado de la práctica 5 – Oscilaciones del robot Line Follower. 38
Figura 43. Resultado de la práctica 5 –Estabilidad del robot Line Follower. 39

Figura 44. Resultado de la práctica 6 – PID Neuronal robot Line Follower. 39
Figura 45. Resultado de la práctica 6 – Antes y después de la curva. 40
Figura 46. Resultado de la práctica 7 – Entrenamiento de la Red Neuronal 40
Figura 47. Resultado de la práctica 8 – PD DIFUSO robot Line Follower. 41
Figura 48. Resultado de la práctica 8 – Estabilización de robot. 41
Figura 49. Resultado de la práctica 9 – Ejecución de la app. 42
Figura 50. Resultado de la práctica 9 - Escritura de las ganancias del PID. 42
Figura 51. Resultado de la práctica 9 - recibimiento de datos del robot Line Follower. 43
Figura 52. Resultado de la práctica 9 - Valores de calibración de los sensores. 43
Figura 53. Resultado de la práctica 10 - Señal de control y variable ajustada. 44
Figura 54. Resultado de la práctica 10 - adquisición de datos. 44
Figura 55. Fusión 360 48
Figura 56. New Electronic Library 48
Figura 57. Create new symbol 48
Figura 58. Símbolo Line 49
Figura 59. Símbolo Pin 49
Figura 60. Propiedades 49
Figura 61. Módulo TB6612FNG 49
Figura 62. Footprint 49
Figura 63. Símbolo PTH Pad 50
Figura 64. Símbolo Name 50
Figura 65. Footprint TB6612FNG 50
Figura 66. Create New Device 50
Figura 67. Add to part 50
Figura 68. Device creado 51
Figura 69. Footprint creado 51
Figura 70. Unir el footprint y el device 51
Figura 71. New Schematic 51
Figura 72. Library Manager 52
Figura 73. Abrir la librería 52
Figura 74. Device en el área de trabajo 52
Figura 75. Símbolo Net 52
Figura 76. Diseño de la PCB 53
Figura 77. Interfaz grafica 90
Figura 78. Diagrama de bloques labview. 90

ÍNDICE DE TABLAS

Tabla 1. Conexiones de regleta 16sMux..... 14
Tabla 2. Conexiones del bluetooth 15
Tabla 3. Conexiones de driver de motores 16
Tabla 4. Método de Ziegler nichols ganancia critica..... 18
Tabla 5. Reglas de control, para controlador PD Difuso 19

INTRODUCCIÓN

En este proyecto técnico de titulación es orientado para los miembros del Club de Robótica y estudiantes de la Universidad Politécnica Salesiana en donde ellos puedan aplicar los conocimientos en las diferentes materias con el enfoque a la robótica móvil de competencia, ya que en diferentes categorías competidas se ha obtenido los primeros lugares.

Actualmente en la competencia se requiere de mucho conocimientos y mejoramiento del robot que cumplan los parámetros y normas establecidas para buen el funcionamiento y desarrollo del robot Line Follower.

El prototipo consta un visualizador grafico en LabVIEW en tiempo real y una adquisición de datos de la posición en la que se encuentra el robot Line Follower, la señal de referencia para la estabilidad y la señal de salida que se obtiene al aplicar el controlador PD Neuronal, PID, PD Difuso. Se podrá observar el comportamiento que tiene nuestro robot móvil.

Para el desarrollo de los controladores anteriormente mencionados se realizarán pruebas de manera experimental con la ayuda del software Labview y Mbed Studio con la finalidad de mantenerse centrado en la línea y recorrer por toda la pista.

En el contenido del proyecto consta de la primera sección considerando el problema que se tomó de referencia para su planteamiento, como la importancia, el alcance, objetivos principales y específicos del robot Line Follower.

En la segunda sección se explica las partes teóricas del prototipo como sistemas embebidos, el software en que fue desarrollado, herramientas para la adquisición de datos, tipos de comunicación que se aplicaron en las practicas.

En la tercera sección se explica el diseño electrónico del robot tales como diagrama de control, esquema del robot, conexiones para la comunicación inalámbrica de los protocolos uart, comunicación con el bluetooth.

En la cuarta sección se encuentra el planteamiento de las practicas aplicada al robot Line Follower y en la sección de anexo se encuentra los pasos a seguir para el funcionamiento de esta.

1. EL PROBLEMA

Los estudiantes de la Carrera de Electrónica y Automatización cuentan con pocos prototipos de robótica móvil donde puedan realizar prácticas que contengan el estudio de sistemas de control automático.

Actualmente las prácticas de laboratorio utilizan simuladores para poner a prueba los diferentes modelos analizados en la catedra, lo cual limita la compresión del estudiante en cuanto a la implementación en sistemas reales.

Cuando el estudiante se encuentre en el ámbito profesional podría tener dificultades en el análisis de sistemas de control automático.

1.1 Importancia y alcance

El siguiente proyecto tiene la finalidad de que los estudiantes puedan tener un desarrollo de conocimientos referidos a sistemas de control muy utilizados en las carreras de ingeniería y en la práctica cotidiana.

A la vez se busca proporcionar a la carrera de Ingeniería Electrónica y Automatización para el desarrollo de más instrumentos que faciliten el futuro desarrollo de robots móviles por parte de los estudiantes y en un proceso conjunto con el tutor a cargo del club de robótica logren participar en eventos de importancia como es el Concurso Ecuatoriano de Robótica donde se vea finalmente reflejado el esfuerzo realizado y el conocimiento adquirido.

Los robots Line Follower son utilizados en diferentes procesos industriales que cumplen una determinada función, por ejemplo, los robots de guiado automático que se encuentra en Amazon. Con este proyecto el estudiante podrá experimentar y comprobar todos los conocimientos teóricos que le fueron impartidos durante su desarrollo académico especialmente en procesos de control.

1.2 Delimitación

1.2.1 Delimitación temporal

El proyecto técnico efectuado tuvo una duración de 6 meses a partir de la fecha de aprobación del mismo.

1.2.2 Delimitación espacial

El proyecto fue realizado para el club de robótica de la Universidad Politécnica Salesiana Sede Guayaquil, campus Centenario, pero por la situación de pandemia se realizaron las pruebas desde casa.

1.2.3 Delimitación académica

El proyecto radica en la implementación de un algoritmo de control para un robot Line Follower usando Red Neuronal. Además, se desarrollarán 10 práctica resueltas.

1.3 Objetivos

1.3.1 Objetivo general

Diseñar e implementar un algoritmo de control utilizando una red neuronal utilizando un sistema embebido para un robot Line Follower

1.3.2 Objetivos específicos

- Diseñar la estructura de un robot Line Follower que conste con los elementos necesarios para el control neuronal.
- Diseñar e implementar circuitería electrónica para integrar los diferentes periféricos de control tales como sensores, motor-reductores y controladores de motores para su funcionamiento.
- Diseñar e implementar un controlador PID clásico para el sistema Line Follower.
- Diseñar e implementar un algoritmo de control basado en Red Neuronal para el sistema Line Follower.
- Evaluar el comportamiento de los controladores planteados previamente.
- Elaborar 10 prácticas didácticas orientadas al área del control y robótica.

2. ESTADO DEL ARTE

2.1 Antecedentes

El carro de Stanford fue uno de los robots Line Follower, en 1960 fue creado para misiones de investigación lunares remotas, entre los años 1966-1970 se lo diseño para que pudiera seguir la trayectoria en una línea blanca sobre el piso (Internet Archive, 1966).

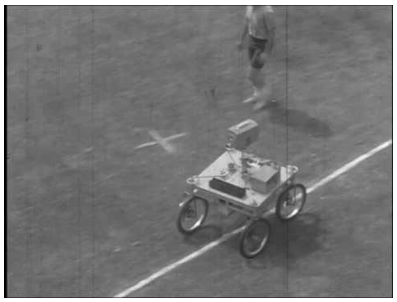


Figura 1. Carro Stanford en 1960

En la revista Wireless World se presenta otro robot seguidor de línea. Este robot sirve para explorar un cuarto, lo cual hace sin la ayuda de la línea en el piso. Sin embargo, para cargar su batería hace uso de la línea, la cual le indica el camino para poder conectarse a la energía eléctrica. Por otro lado, en 1975 la revista Everyday Electronics en su número de junio presenta un robot seguidor de línea. (Dr. Celso Márquez-Sánchez, 2019)

Por su parte, la revista Maplin Magazine, en su número de marzo de 1985 publica el robot Zero 2. Este robot era capaz de dibujar logotipos, además tenía LEDs para encender y apagar, una bocina de dos tonos y era un seguidor de línea. En 1986, se realiza la primera competencia Robotrace en Japón, en un torneo llamado "All Japan Micromouse Contest". (Dr. Celso Márquez-Sánchez, 2019)

Este torneo fue organizado por la New Technology Foundation. Esta competencia se sigue realizando hasta el día de hoy y consiste en robots seguidores de línea velocistas. La competencia de Robotrace se popularizo tanto que provoco competencias derivadas de esta, dando lugar a la competencia de Line Follower o seguidor de línea. La cual es una simplificación de la categoría Robotrace y es la que la mayoría de la gente conoce actualmente. (Dr. Celso Márquez-Sánchez, 2019)

2.2 Sistemas embebidos

Los sistemas embebidos están conformados por elementos electrónicos que permiten realizar diferentes tipos de tareas en tiempo real, la mayoría de estos elementos electrónicos se encuentran en la misma PCB (Printed Circuit Board) cuyas siglas significan Placa de Circuito impreso. Algunos sistemas embebidos permiten programar en un lenguaje ensamblador de microcontrolador mediante un compilador especifico.

Estos sistemas embebidos están orientados a reducir los costos y maximizar la confiabilidad. (semanticwebbuilder, s.f.)

2.2.1 Stm32 Node L432kc

La placa Stm32 proporciona Los dispositivos STM32L432KC son los microcontroladores de bajo consumo de energía de ultra basados en el alto rendimiento Arm ® corteza ® -M4 RISC de 32 bits operativo central a una frecuencia de hasta 80 MHz. El núcleo Cortex-M4 ofrece una sola flotante de precisión unidad de punto (FPU) que soporta todos los Arm ® instrucciones y datos tipos de tratamiento de datos de precisión simple. También implementa un conjunto completo de instrucciones DSP y una unidad de protección de memoria (MPU) que mejora la seguridad de la aplicación. (ST, s.f.)

2.2.2 Mbed Studio

Mbed Studio es un IDE gratuito para el desarrollo de bibliotecas y aplicaciones Mbed OS, que incluye todas las dependencias y herramientas que necesita en un solo paquete para que pueda crear, compilar y depurar sus programas Mbed en el escritorio. (arm mbed, s.f.)

2.3 PID

Tenemos un sistema de control con una entrada y una salida que es considerado SISO por sus siglas en ingles Single Input, Single Output. (Mazzone, 2002)

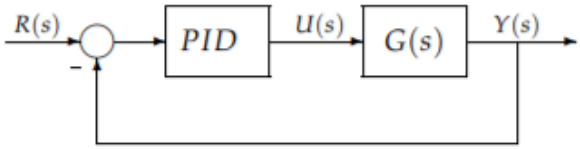


Figura 2. Diagrama de bloques PID

Los controladores PID tienen tres acciones: proporcional(P), integral(I) y derivativa(D). Estos controladores son llamados P, I, PI, PD y PID. (Mazzone, 2002)

- **Control Proporcional (P),** tiene una salida del controlador proporcional ante el error, por lo tanto, tenemos: $u(t) = KP * e(t)$ con su función de transferencia tenemos (Mazzone, 2002):

  $$Cp(s) = Kp$$

  El valor Kp es una ganancia proporcional que se ajusta, pose desempeño limitado y error en régimen permanente(off-set). (Mazzone, 2002)
- **Control Integral(I),** tiene una salida del control que es proporcional al error acumulado, esto haría que nuestro controlador sea lento, tenemos (Mazzone, 2002):

  $$u(t) = ki \int_0^t e(\tau)d\tau \qquad Ci(S) = \frac{Ki}{S}$$

  La salida del control u(t) tendrá un valor diferente a cero si la señal de error e(t) es igual a cero. Esto nos quiere decir que tendremos perturbaciones u oscilaciones. (Mazzone, 2002)

- **Control Proporcional-Integral(PI),** dada las explicaciones anteriores se define lo siguiente (Mazzone, 2002):

$$u(t) = Kp * e(t) + \frac{Kp}{Ti}\int_0^t e(\tau)d\tau$$

El valor Ti es denominado tiempo integral y esta ajustara a la acción integral. Como función de transferencia tenemos lo siguiente (Mazzone, 2002):

$$Cpi(S) = Kp * \left(1 + \frac{1}{Tis}\right)$$

Con un control proporcional, es necesario que exista error para tener una acción de control distinto de cero. Con acción integral, un error pequeño positivo siempre nos dará una acción de control creciente, y si fuera negativo la señal de control será decreciente. Este razonamiento sencillo nos muestra que el error en régimen permanente será siempre cero. (Mazzone, 2002)
Muchos controladores industriales tienen solo acción PI. Se puede demostrar que un control PI es adecuado para todos los procesos donde la dinámica es esencialmente de primer orden. Lo que puede demostrarse en forma sencilla, por ejemplo, mediante un ensayo al escalón. (Mazzone, 2002)

- **Control Proporcional-Integral-Derivativa(PID),** como salida tendremos una acción combinada de las tres ganancias individuales, donde tenemos la siguiente ecuación del controlador PID (Mazzone, 2002):

$$u(t) = Kp * e(t) + \frac{Kp}{Ti}\int_0^t e(\tau)d\tau + Kp * Td * \frac{de(t)}{dt}$$

Como función de transferencia tenemos:

$$Cpid(S) = Kp * \left(1 + \frac{1}{Tis} + Tds\right)$$

2.4 Red Neuronal

2.4.1 Reseña Histórica

1949 - Donald Hebb. Fue el primero en explicar los procesos del aprendizaje (que es el elemento básico de la inteligencia humana) desde un punto de vista psicológico, desarrollando una regla de como el aprendizaje ocurría. Aun hoy, este es el fundamento de la mayoría de las funciones de aprendizaje que pueden hallarse en una red neuronal. Su idea fue que el aprendizaje ocurría cuando ciertos cambios en una neurona eran activados. También intentó encontrar semejanzas entre el aprendizaje y la actividad nerviosa. Los trabajos de Hebb formaron las bases de la Teoría de las Redes Neuronales. (Carlos Alberto Ruiz, 2001)

1957 - Frank Rosenblatt. Comenzó el desarrollo del Perceptron. Esta es la red neuronal más antigua; utilizándose hoy en día para aplicación como identificador de patrones. Este

modelo era capaz de generalizar, es decir, después de haber aprendido una serie de patrones podía reconocer otros similares, aunque no se le hubiesen presentado en el entrenamiento. Sin embargo, tenía una serie de limitaciones, por ejemplo, su incapacidad para resolver el problema de la función OR-exclusiva y, en general, era incapaz de clasificar clases no separables linealmente. (Carlos Alberto Ruiz, 2001)

1960 - Bernard Widroff/Marcian Hoff. Desarrollaron el modelo Adaline (ADAptative LINear Elements). Esta fue la primera red neuronal aplicada a un problema real (filtros adaptativos para eliminar ecos en las líneas telefónicas) que se ha utilizado comercialmente durante varias décadas. (Carlos Alberto Ruiz, 2001)

2.4.2 Definiciones de una red neuronal

Existen numerosas formas de definir a las redes neuronales; desde las definiciones cortas y genéricas hasta las que intentan explicar más detalladamente qué son las redes neuronales. Por ejemplo (Carlos Alberto Ruiz, 2001):

- Una nueva forma de computación, inspirada en modelos biológicos. (Carlos Alberto Ruiz, 2001)
- Un modelo matemático compuesto por un gran número de elementos procesales organizados en niveles. (Carlos Alberto Ruiz, 2001)
- Un sistema de computación compuesto por un gran número de elementos simples, elementos de procesos muy interconectados, los cuales procesan información por medio de su estado dinámico como respuesta a entradas externas. (Carlos Alberto Ruiz, 2001)
- Redes neuronales artificiales son redes interconectadas masivamente en paralelo de elementos simples (usualmente adaptativos) y con organización jerárquica, las cuales intentan interactuar con los objetos del mundo real del mismo modo que lo hace el sistema nervioso biológico. (Carlos Alberto Ruiz, 2001)

2.4.3 Ventajas de las redes neuronales

Debido a su constitución y a sus fundamentos, las redes neuronales artificiales presentan un gran número de características semejantes a las del cerebro. Por ejemplo, son capaces de aprender de la experiencia, de generalizar de casos anteriores a nuevos casos, de abstraer características esenciales a partir de entradas que representan información irrelevante, etc. Esto hace que ofrezcan numerosas ventajas y que este tipo de tecnología se esté aplicando en múltiples áreas. Entre las ventajas se incluyen (Carlos Alberto Ruiz, 2001):

- Aprendizaje Adaptativo. Capacidad de aprender a realizar tareas basadas en un entrenamiento o en una experiencia inicial. (Carlos Alberto Ruiz, 2001)
- Auto-organización. Una red neuronal puede crear su propia organización o representación de la información que recibe mediante una etapa de aprendizaje. (Carlos Alberto Ruiz, 2001)
- Tolerancia a fallos. La destrucción parcial de una red conduce a una degradación de su estructura; sin embargo, algunas capacidades de la red se pueden retener, incluso sufriendo un gran daño. (Carlos Alberto Ruiz, 2001)

- Operación en tiempo real. Los cómputos neuronales pueden ser realizados en paralelo; para esto se diseñan y fabrican máquinas con hardware especial para obtener esta capacidad. (Carlos Alberto Ruiz, 2001)
- Fácil inserción dentro de la tecnología existente. Se pueden obtener chips especializados para redes neuronales que mejoran su capacidad en ciertas tareas. Ello facilitará la integración modular en los sistemas existentes. (Carlos Alberto Ruiz, 2001)

2.5 Componentes del sistema

2.5.1 Motores de corriente continua

Es un elemento eléctrico o maquina eléctrica que es capaz de convertir la energía eléctrica en mecánica, este dispositivo es uno de los inventos más versátiles de la industria que ha permitido controlar procesos, estos motores está constituido por dos elementos principales: parte fija y parte móvil. (TERCESA S.L, 2019)

- **Parte fija:** Compuesto por un electroimán producido por el campo magnético que induce la fuerza sobre la parte móvil. (TERCESA S.L, 2019)
- **Parte móvil:** Compuesto por varios espirales o bobinas denominado rotor. (TERCESA S.L, 2019)

2.5.2 Controlador de Motor (TB16FN12G)

Llamado también driver tb16fn12g, es un elemento electrónico que integra dos puentes H para así poder controlar de forma independiente a dos motores DC. Para alimentación se recomienda trabajar en un rango de 4.5v a 13.5v con una salida de corriente máxima de 3A por canal. (pololu)

Los puentes H que posee son basados en MOSFET ya que son más eficientes que los BJT. (pololu)

El controlador TB16FN12G lo podemos encontrar de dos maneras: en SMD o en un módulo donde están integrados capacitores, reguladores y resistencia, nos permitirá conectar las entradas al controlador y las salidas al motor.

Figura 3. TB16FN12G en smd

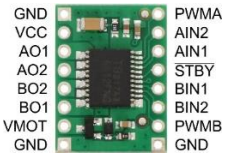


Figura 4. Módulo TB16FN12G

2.5.3 Reglador de voltaje

Es un elemento electrónico que nos permitirá elevar o reducir el voltaje de entrada sin importar los que conectes a su salida manteniendo su voltaje. Existen diferentes tipos de

reguladores para lo cual tenemos los reguladores lineales y reguladores por conmutación. (Camarillo, 2019)

- **Regulador Lineal**
  Son especialmente útiles dada su circuitería simple y facilidad de uso. Los reguladores lineales son elementos de 3 terminales, generalmente. Están basados en circuitos analógicos con realimentación que ajustan el voltaje de salida dependiendo de la señal de realimentación. (Camarillo, 2019)

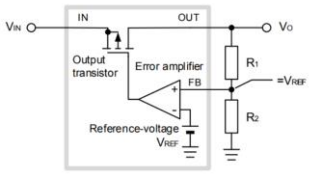


Figura 5. Diagrama interno de un regulador de voltaje

Se les puede encontrar en varias categorías, como reguladores positivos o negativos, en el caso de los reguladores negativos no hay muchas variantes. En el caso positivo los hay fijos y variables. El primer tipo ofrece un voltaje constante especificado en su matrícula; la serie 78xx y 79xx son ejemplos de este tipo de reguladores. Los variables son dispositivos que permiten ajustar el voltaje de salida mediante elementos externos, como resistores, los conocidos integrados 317 Y 337 son ejemplos de reguladores variables positivos y negativos, respectivamente. (Camarillo, 2019)

Figura 6. Regulador Lineal fijo

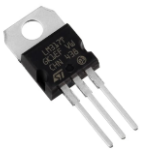


Figura 7. Regulador Lineal ajustable

La ventaja de estos reguladores es la facilidad de seleccionar los componentes para un propósito específico, además, dado que no se basan en un principio de conmutación, se les suele utilizar en aplicaciones de bajo ruido, como en comunicaciones, instrumental médico y metrología. Las desventajas, y una de las principales, es su baja eficiencia y la necesidad de un elemento adecuado para la disipación de calor. Además, esta tecnología sólo permite reducir el voltaje de entrada, no aumentarlo, como su contraparte basada en conmutación. (Camarillo, 2019)

- **Reguladores por conmutación**
  Los reguladores por conmutación son más eficientes que los lineales dado que los transistores funcionan en saturación o corte (encendido-apagado) en vez de en su región lineal (como un resistor variable), minimizando la pérdida de potencia. Además, el tamaño de las fuentes de alimentación se reduce, dado que se requiere disipar menos calor. Suponiendo una fuente con un voltaje de entrada de 12V con salida de 3.3V, un regulador de conmutación puede alcanzar una eficiencia de más del 90%, respecto al 27.5% de un regulador lineal. Esto se puede interpretar como una reducción del tamaño de cuando menos 8 veces respecto al lineal. (Camarillo, 2019)

Figura 8. regulador por conmutacionLM2576

Sin embargo, la eficiencia de estos dispositivos requiere de un cuidadoso diseño que no se presenta en los reguladores lineales, y aumenta la cantidad de componentes a elegir para obtener un diseño estable y útil, por no decir que existen una diversidad de configuraciones que pueden dar solución a un problema de diseño. Afortunadamente los proveedores de circuitos integrados diseñan sus propios modelos que reducen el número de variables a tomar en cuenta para implementar estos dispositivos, disminuyendo el tamaño de los productos y el esfuerzo para realizarlos. (Camarillo, 2019)

2.5.4 Regleta de Sensores

Está conformado por los siguientes elementos electrónicos: QRE1113, resistencias y capacitores, esta regleta de sensores pueden ser QTX Y 16 sensores multiplexados.

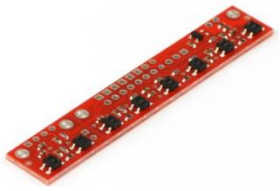


Figura 9. Regleta de sensores QTX

Figura 10. Regleta de 16 sensores multiplexado

- **QTX,** este tipo de regletas de sensores está formado por dos tipos: QTR8-RC y QTR-8A de manera digital y análoga respectivamente.

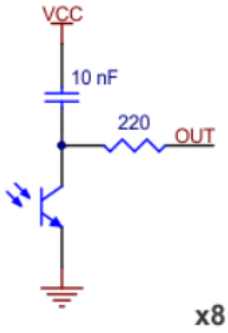


Figura 11. Configuración QTR-8RC

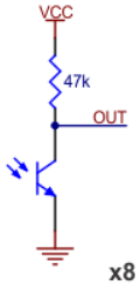


Figura 12. Configuración QTR-8A

- **Sensores multiplexados,** diseñada especialmente para seguidores de line de competencia, y solo posee 9 pines dos de ellos son de alimentación y seis para su control y lectura. Solo trabajan con 5v con una corriente de consumo aproximado 120mA

2.5.5 Turbina EDF

Este tipo de turbina son usadas en las competencias de seguidores de línea en la categoría con turbina, el beneficio que dan a los seguidores de línea al usar la turbina es darle mayor adherencia a la pista para evitar los derrapes al momento de encontrarse con una curva.

Para su alimentación requiere de un controlador llamado ESC (Electronic Speed Contoller) o variador permitiendo controlar la velocidad de giro de un motor brushlees por medio PWM (pulse-width modulation) que recibida por un microprocesador.

Figura 13. Turbina EDF

2.6 Módulo de comunicación BT

El módulo de comunicación bluetooth se puede configurar en dos modos como Master o Slaves. El modo Master permite conectarse con varios Slaves permitiendo que ellos se conecten, recibir y solicitar información de todos ellos.

Cada uno de los dispositivos presentan una dirección única asi como un nombre para identificarlo y un PIN que permitirá conectarse con el dispositivo, estos son sencillos y económicos muy accesible, los más frecuentes en el mercado son los módulos HC-05 y HC-06.

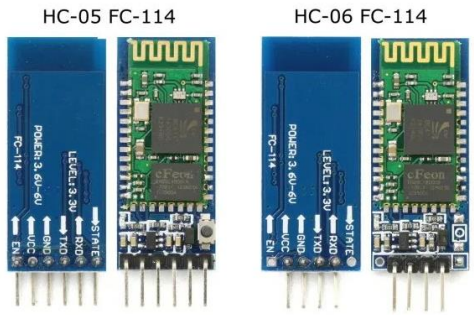


Figura 14. *Módulos Bluetooth*

2.7 Comunicación Serial RS232

La comunicación serial RS232 es un protocolo común utilizado por dispositivos y equipos usados en instrumentación. La comunicación serial puede ser utilizada para adquisición de datos, control, depuración de código, etc. (Comunicaciones Digitales)

El concepto de comunicación serial permite la transmisión - recepción bit a bit de un byte completo, este método de comunicación puede alcanzar mayores distancias. (Comunicaciones Digitales)

Por el contrario, la especificación IEEE 488 (comunicación en paralelo) determina que el largo del cable para el equipo no puede ser mayor a 20 metros, con no más de 2 metros entre cualesquier dos dispositivos; por el contrario, utilizando comunicación serial el largo del cable puede llegar a los 1200 metros. (Comunicaciones Digitales)

Típicamente, la comunicación serial se utiliza para transmitir datos en formato ASCII. (Comunicaciones Digitales)

Para realizar la comunicación se utilizan 3 líneas de transmisión: Tierra (o referencia), Transmitir, Recibir. Debido a que la transmisión es asíncrona, es posible enviar datos Por una línea mientras se reciben datos por otra. (Comunicaciones Digitales)

Las velocidades de transmisión más comunes son de 115200, 9600, y 4800. (Comunicaciones Digitales)

3. MARCO METODOLÓGICO

En esta sección daremos a conocer más a detalle cómo fue elaborado nuestro prototipo que lo implantaremos al largo de la elaboración de nuestro proyecto técnico.

3.1 Diseño del robot Line Follower.

El diseño del prototipo se lo ha realizado mediante el software Fusión 360 ya que combina el diseño electrónico y mecánico en una sola herramienta.

La parte mecánica del robot Line Follower, debemos tener en cuenta tres características para la posición de los elementos electrónicos, como es el ruido que puede llegar afectar el funcionamiento, las inercias y el centro de gravedad que deben estar bien colocados en la estructura destinada a brindarnos la movilidad y además para su construcción se debe elegir un material resistente que soporte el peso de los aparatos electrónicos.

.

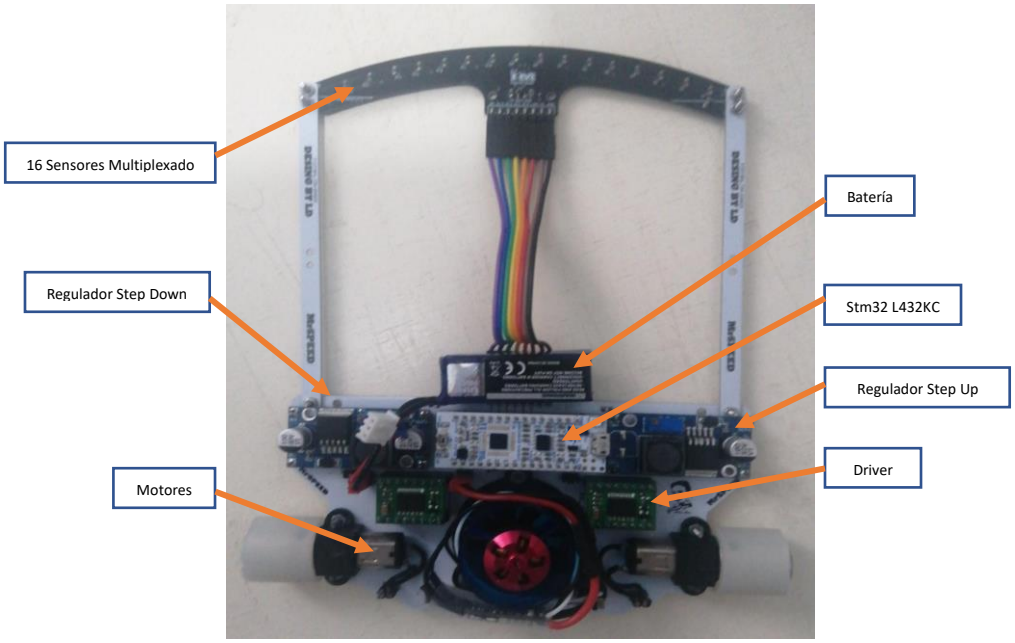


Figura 15. Prototipo Line Follower

3.2 Diagrama de control

El robot consta de un algoritmo de control pid neuronal donde la entrada del sistema o referencia es el punto 0 en la posición de los sensores conectado a un control neuronal para obtener las ganancias y aplicarlo en el pid para lograr la estabilización.

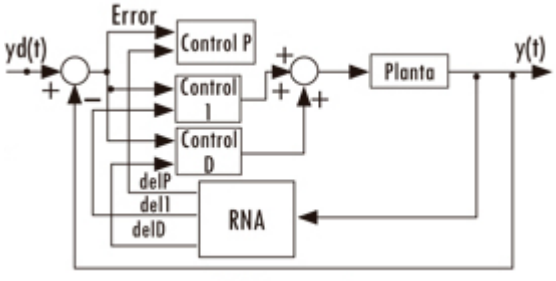


Figura 16. Diagrama de control PIDNN

3.3 Esquema del robot

El robot Line Follower consta de un controlador Stm32 conectado a una tarjeta de potencia para el control de los motores, mediante la regleta de 16 sensores multiplexado conectado al embebido se logra la estabilización mediante un control pid neuronal.

El prototipo tiene una entrada para la lectura de la comunicación serial de un bluetooth hc-05.

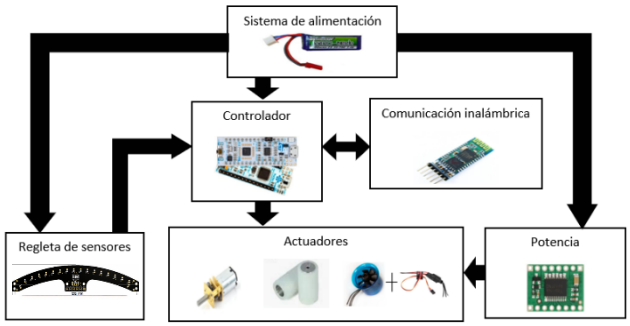


Figura 17. Diagrama de control

3.4 Conexiones del prototipo

3.4.1 Conexiones de la instrumentación

3.4.1.1 Sensores multiplexado

La regleta de 16 sensores multiplexado consta del QRE1113 es un sensor óptico infrarrojo de reflexión de corto alcance que mediante del multiplexado 74hc4067 nos va a permitir monitorear la posición en que se encuentra el robot.

Figura 18. Barra 16 sensores multiplexado conectado al Stm32.

Las conexiones de la tarjeta embebida con la regleta de 16 sensores multiplexado se muestran en la tabla 1.

Regleta 16s Mux	Stm32 L432KC
GND	GND
5V	VCC
LON	PIN A6
SL0	PIN D2
SL1	PIN D3
SL2	PIN D4
SL3	PIN D5
AN	PIN A0

Tabla 1. Conexiones de regleta 16sMux

3.4.2 Conexiones de la comunicación

3.4.2.1 Bluetooth hc-05

El bluetooth hc-05 es un dispositivo con comunicación seria, tiene como función principal el enviar y recibir datos desde una aplicación creada en app inventor para el envió de datos conectado a la stm32L432kc.

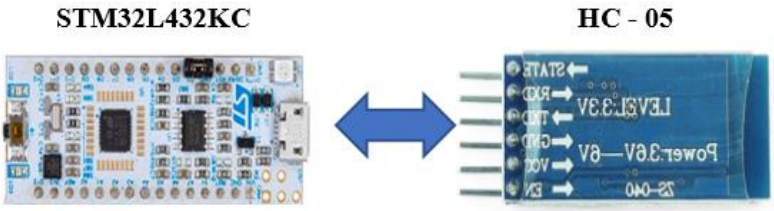


Figura 19. Esquema de Stm32 con bluetooth

En Labview la configuración para el uso del bluetooth es mediante el bloque Visa Serial donde los parámetros son conectados en el puerto COM, velocidad de transmisión de 9600, tamaño de recepción es de 8 bits.

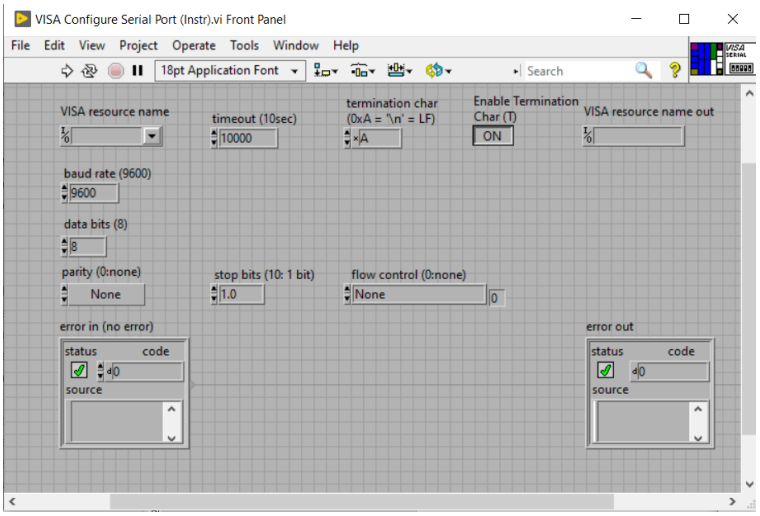


Figura 20. Configuración del bloque Visa Serial

Las conexiones de la tarjeta embebida con el modulo bluetooth se muestran en la tabla 2.

Módulo bluetooth	Stm32L432KC
VCC	VCC
GND	GND
TX	PIN D0/RX
RX	PIN D1/TX

Tabla 2. Conexiones del bluetooth

3.4.2.2 Aplicativo en app inventor

Para el control de la práctica se utiliza el aplicativo App inventor, nos permite diseñar aplicación compatible con dispositivos Android, en la figura 19 se muestra el interfaz que nos permitirá enviar los valores de las ganancias del pid mediante el botón enviar, también

nos permite guardar los valores ingresados mediante el botón sabe, posee un botón MS que nos permitirá visualizar los datos enviados y recibidos por parte del robot mediante la comunicación bluetooth.

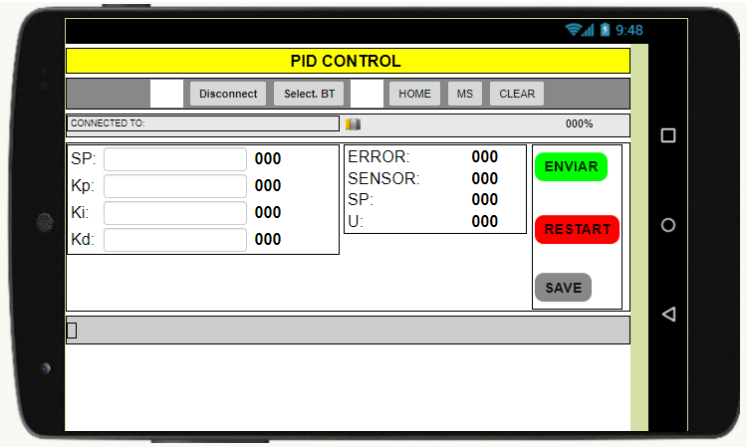


Figura 21. Aplicativo diseñado en app inventor

3.4.3 Conexiones de la potencia y alimentación

3.4.3.1 Tarjeta controladora de motores

En la sección de potencia para el control de los actuadores de tracción del robot se seleccionó el puente h o driver con un integrado TB6612FNG que no permitirá conectar y controlar dos motores de corriente continua o dos motores paso a paso.

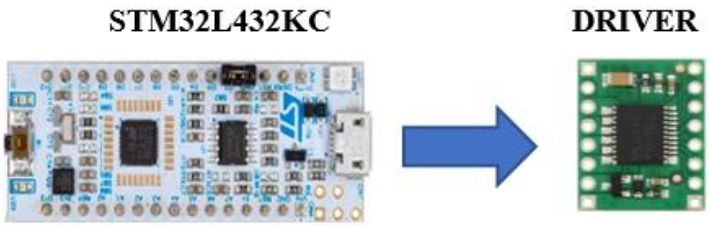


Figura 22. Conexión de la tarjeta embebida con el driver de potencia

Las conexiones de la tarjeta embebida con el driver de potencia se muestran en la tabla 3.

Driver de potencia	STM32L432KC
GND	GND
VCC	VCC
STBY	VCC
PWMA	PIN PA_3
AIN1	PIN PA_4
AIN2	PIN PA_1
BIN1	PIN D6
BIN2	PIN D11
PWMB	PIN D9

Tabla 3. Conexiones de driver de motores

3.4.3.2 Fuente de poder

Para la alimentación del robot se utiliza una batería de Lipo de 7.4v - 300mAh - 35C, según los datos del fabricante para calcular el consumo de corriente máxima es C x mA donde se multiplica la Tasa de descarga(C) por la capacidad de corriente de la batería Lipo (mA) como resultado tendremos expresado en mili Amperios.

35C x 300mA = 10.500 [mA]

Por lo tanto, nuestra batería Lipo tiene una corriente constante máxima de 10.5A

Para calcular la autonomía de batería se aplicó la siguiente formula

$$Wb = WbxIb$$

$$Wc = VbxIc$$

$$\frac{Wb}{Wc} = H$$

Dicho esto, nuestro batería suministra voltaje al regulador Step Up xl6009, su consumo de corriente máxima es de 0.80A y el regulador Step Down lm2596, su consumo de corriente máxima es de 0.75A, respectivamente suministrara la parte potencia y control

Aplicamos la fórmula:

$$Wb = 7.4[V]x10.5[A] = 77.7W$$

$$Wb = 7.4[V]x1.5[A] = 11.1W$$

$$\frac{77.7Wh}{11.1W} = 7Hx0.2 = 1.4h$$

Según los datos del fabricante indica que la batería no puede estar totalmente descarga ya disminuiría el tiempo de vida útil de la misma, se recomienda que este el 20% descargado.

Batería de 7.4v 300mAh

Figura 23. Batería Lipo

3.5 Controlador PID

3.5.1.1 Método de sintonización de ziegler nichols

El método Ziegler Nichols permite la sintonización de controladores PID basándose en las respuestas empíricas o en el valor Kp que produce la estabilidad marginal cuando solo

se usa la acción de control proporcional. Si la planta es tan complicada que no es fácil obtener un modelo matemático, tampoco es posible un método analítico para el diseño de un control PID lo cual se debe recurrir a procedimientos experimentales para la sintonía de los controladores PID (Ogata, 2010).

En el prototipo tiene como sistema de lazo cerrado donde la entrada es 6cm a estabilizar, la planta es el prototipo, la retroalimenta es la regleta de sensores y como salida los actuadores

Este método se fija las variables integrales y derivativas en cero, usando solo la acción proporcional se incrementa Kp hasta que el sistema presente oscilaciones sostenidas caso contrario no se puede aplicar este método. Así la ganancia critica Kc y el periodo de oscilación Tc en segundos (Ogata, 2010).

Después de haber obtenido estos dos valores, se puede calcular las ganancias del controlador PID con la acción proporcional(P), proporcional e integral(PI), proporcional y derivativa(PD) o proporcional integral y derivativo(PID) (Ogata, 2010).

Tipo de controlador	Kp	Ti	Td
P	0.5*Kc		
PI	0.45*Kc	0.01*Tc	
PID	0.25*Kc	0.001*Tc	5.5*Tc

Tabla 4. Método de Ziegler nichols ganancia critica.

En el siguiente comando de código se creó una función PID aplicando el método de ziegler nichols.

```
149 void pid() {
150   errorActual = (position) - Sp;
151   errorDerivativo = errorActual - errorPasado;
152   errorIntegral = error1 + error2 + error3 + error4 + error5 + error6;
153   errorPasado = errorActual;
154   error6 = error5;
155   error5 = error4;
156   error4 = error3;
157   error3 = error2;
158   error2 = error1;
159   error1 = errorActual;
160   u = ( errorActual * KP ) + ( errorDerivativo * KD ) + (errorIntegral * KI) ;
161 }
```

Figura 24. Código del control PID

3.6 Controlador PD Difuso

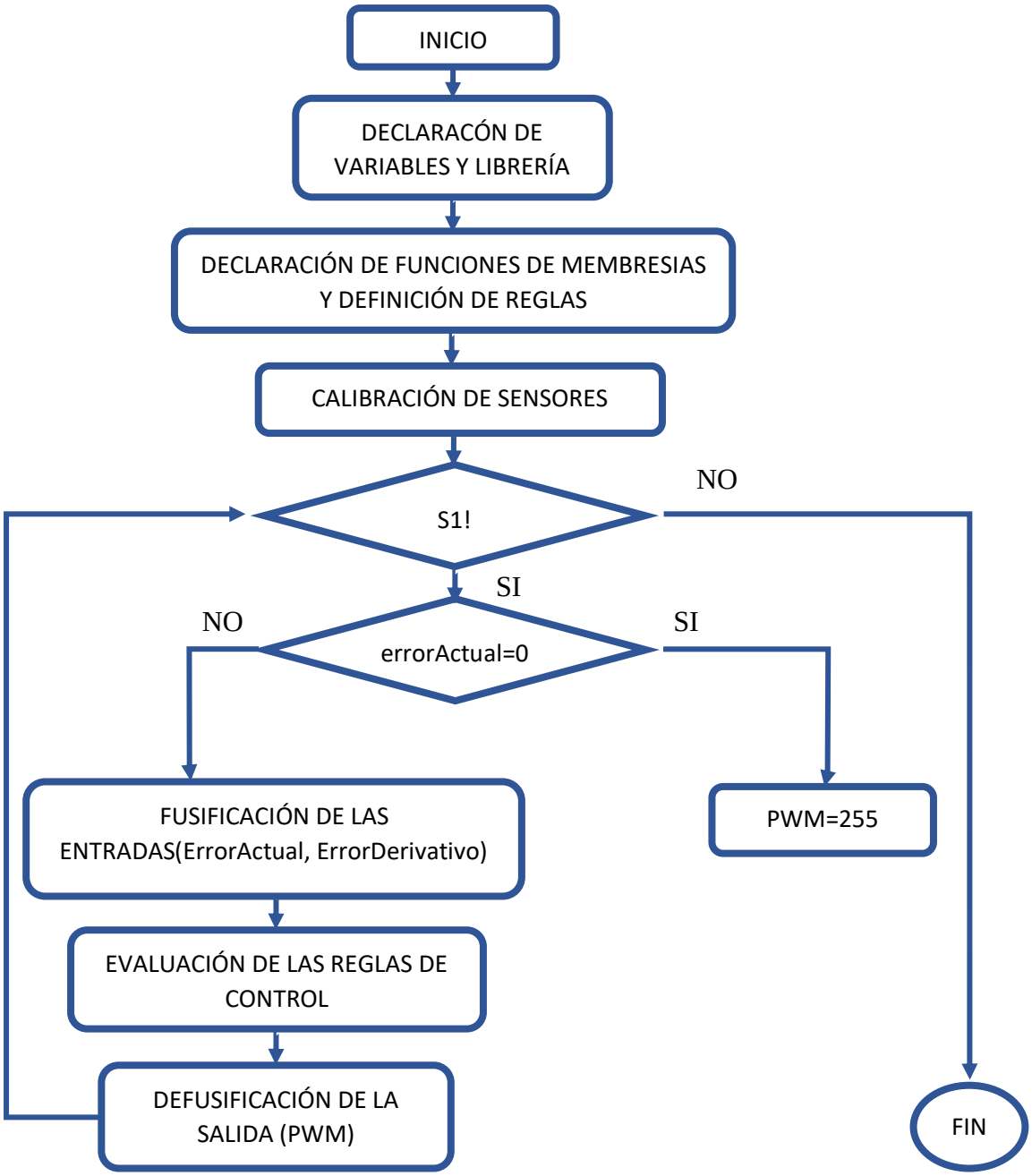
Mediante las siguientes líneas de comando se crearon una función FLC donde se establecen las reglas de correspondencia como se muestra en la siguiente tabla 5.

$\Delta e(k)$	**NG**	**NM**	**NG**	**Z**	**PG**	**PM**	**PG**
NG	NG	NG	NG	NG	NM	NP	Z
NM	NG	NG	NM	NM	NP	Z	Z
NP	NG	NM	NM	NP	Z	Z	PP
Z	NM	NP	NP	Z	PP	PP	PM
PP	NP	Z	Z	PP	PM	PM	PG
PM	Z	Z	PP	PM	PM	PG	PG
PG	Z	PP	PM	PG	PG	PG	PG

Tabla 5. *Reglas de control, para controlador PD Difuso*

En el anexo 8 se encuentra detallado el algoritmo de la creación de las reglas difusas.

En la siguiente figura se obtiene un diagrama de flujo para la ejecución de controlador PD Difuso.



4. PRÁCTICAS DE LABORATORIO

4.1 Práctica #1

UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR	GUÍA DE PRÁCTICA DE LABORATORIO
CARRERA: Ingeniería Electrónica	ASIGNATURA: Informática Industrial, Sistemas Inteligentes, Diseño Electrónico, Robótica Móvil.
NÚMERO DE PRÁCTICA: 1	**TITULO DE PRÁCTICA**: "Diseño electrónico en Fusion 360"

OBJETIVOS:

- Diseñar circuito esquemático y tarjeta electrónica.
- Conocer los componentes del robot Line Follower.
- Crear footprint de los elementos a usarse.

PLANTEAMIENTO DEL PROBLEMA:

Se desea que el robot Line Follower cumpla con las siguientes especificaciones:

- Un controlador Stm32 núcleo l432kc la cual nos va permitir programar el algoritmo de control creado.
- Se debe contar con dos drivers tb16fn12g que nos permitirá darle la motricidad al robot.
- El robot Line Follower de contar con dos reguladores por conmutación Step Up y Step Down así poder controlar el voltaje a los motores y alimentar a cada elemento utilizado en el diseño electrónico.
- Debe considerar dos switch; el primer switch para el encendido y apagado del robot y el segundo switch para el encendido y apagado de la turbina.
- Contará con dos pulsadores para la calibración y arranque del robot.

4.2 Práctica #2

UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR	GUÍA DE PRÁCTICA DE LABORATORIO
CARRERA: Ingeniería Electrónica	ASIGNATURA: Informática Industrial, Sistemas Inteligentes, Diseño Electrónico, Robótica Móvil.
NÚMERO DE PRÁCTICA: 2	**TITULO DE PRÁCTICA**: "Programación Mbed Studio para monitoreo de sensores."

OBJETIVOS:

- Reconocer los pines embebidos.
- Realizar una conexión entre los sensores multiplexados y la tarjeta embebida.
- Normalizar la entrada análoga y monitoreo de datos en puerto serial.
- Ubicar en diferentes posiciones la regleta de sensores.
- Implementar un código de programación para realizar la iteración de sensores.

PLANTEAMIENTO DEL PROBLEMA:

Se desea que la práctica cumpla lo siguiente:

- Se debe usar el programa Mbed Studio.
- Se requiere hacer el monitoreo de sensores considerando que se está usando una regleta multiplexada 4 a 1 tipo analógica en donde nos permitirá leer los 16 sensores.
- La regleta multiplexada nos va permitir al robot en qué posición esta la línea negra con fondo blanco.
- En el monitor serial se debe de visualizar el valor analógico que tiene cada sensor.
- Colocar los sensores en un fondo blanco y mostrar los valores obtenidos.
- Colocar los sensores en la línea negra y mostrar los valores obtenidos.

4.3 Práctica #3

UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR	GUÍA DE PRÁCTICA DE LABORATORIO
CARRERA: Ingeniería Electrónica	ASIGNATURA: Informática Industrial, Sistemas Inteligentes, Diseño Electrónico, Robótica Móvil.
NÚMERO DE PRÁCTICA: 3	**TITULO DE PRÁCTICA**: "Programación Mbed Studio control de actuadores"

OBJETIVOS:

- Identificar los pines de entrada y salida en el driver tb16fn12g.
- Implementar un algoritmo para el control de los motores.

PLANTEAMIENTO DEL PROBLEMA:

Se desea que la práctica cumpla lo siguiente:

- Se debe usar el programa Mbed Studio.
- Se requiere hacer el control de actuadores mediante una función de programación para el movimiento de los motores.
- En la tabla se muestra la lógica que deberá tener para el movimiento de los motores.

AIN1	AIN2	BIN1	BIN2	M1	M2
1	0	1	0	255	255
0	1	1	0	-255	255
1	0	0	1	255	-255
0	1	0	1	-255	-255

4.4 Práctica #4

UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR	GUÍA DE PRÁCTICA DE LABORATORIO
CARRERA: Ingeniería Electrónica	ASIGNATURA: Informática Industrial, Sistemas Inteligentes, Diseño Electrónico, Robótica Móvil.
NÚMERO DE PRÁCTICA: 4	**TITULO DE PRÁCTICA**: "Codificación de algoritmo para calibración de sensores"

OBJETIVOS:

- Reconocer los pines embebidos.
- Realizar una conexión entre los sensores multiplexados y la tarjeta embebida.
- Implementar un algoritmo de calibración para una regleta de sensores multiplexada.
- Ubicar en diferentes posiciones la regleta de sensores.

PLANTEAMIENTO DEL PROBLEMA

Se desea que la práctica cumpla lo siguiente:

- Se debe usar el programa Mbed Studio.
- Se requiere hacer la calibración de los sensores considerando que se está usando una regleta multiplexada 4 a 1 tipo analógica en donde nos permitirá leer los 16 sensores.
- La regleta multiplexada nos va permitir al robot en qué posición esta la línea negra con fondo blanco.
- En el monitor serial se debe de visualizar el valor analógico que tiene cada sensor. Se debe hacer un barrido en la línea y ver qué valor tiene en el color negro y color blanco.
- El robot Line Follower para poder calibrarlo deberá ubicarlo primero en el fondo blanco y con un pulsador dar la orden para que registre los valores obtenido, luego ubicarlo en la línea negra y con el mismo pulsador dar la siguiente orden para que registre los valores, como último pulso deberá mostrar los valores de cada sensor calibrado.
- Se deberá crear una variable de tipo **long int position** que deberá estar en un rango de 0 a 1500, con esto el robot estará listo para la aplicación de los diferentes controladores lógicos a utilizarse.

4.5 Práctica #5

UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR	GUÍA DE PRÁCTICA DE LABORATORIO
CARRERA: Ingeniería Electrónica	ASIGNATURA: Informática Industrial, Sistemas Inteligentes, Diseño Electrónico, Robótica Móvil.
NÚMERO DE PRÁCTICA: 5	**TITULO DE PRÁCTICA**: "Control PID clásico aplicado al robot Line Follower"

OBJETIVOS:

- Identificar las entradas y salidas de la tarjeta embebida.
- Realizar una conexión entre los sensores multiplexados y la tarjeta embebida.
- Implementar un controlador PID para controlar la posición del robot Line Follower.
- Ubicar en diferentes posiciones la regleta de sensores.

PLANTEAMIENTO DEL PROBLEMA

Se desea que la práctica cumpla lo siguiente:

- Se debe usar el programa Mbed Studio.
- Se requiere hacer la calibración de los sensores considerando que se está usando una regleta multiplexada 4 a 1 tipo analógica en donde nos permitirá leer los 16 sensores.
- La regleta multiplexada nos va permitir al robot en qué posición esta la línea negra con fondo blanco.
- En el monitor serial se debe de visualizar el valor analógico que tiene cada sensor. Se debe hacer un barrido en la línea y ver qué valor tiene en el color negro y color blanco.
- El robot Line Follower para poder calibrarlo deberá ubicarlo primero en el fondo blanco y con un pulsador dar la orden para que registre los valores obtenido, luego ubicarlo en la línea negra y con el mismo pulsador dar la siguiente orden para que registre los valores, como último pulso deberá mostrar los valores de cada sensor calibrado.
- Se deberá crear una variable de tipo **long int position** que deberá estar en un rango de 0 a 12 cm, con esto el robot estará listo para la aplicación de los diferentes controladores lógicos a utilizarse.
- Debe encontrar las ganancias para el control PID.

4.6 Práctica #6

UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR	GUÍA DE PRÁCTICA DE LABORATORIO
CARRERA: Ingeniería Electrónica	ASIGNATURA: Informática Industrial, Sistemas Inteligentes, Diseño Electrónico, Robótica Móvil.
NÚMERO DE PRÁCTICA: 6	**TITULO DE PRÁCTICA**: "Control PID Neuronal aplicado al robot Line Follower"

OBJETIVOS:

- Identificar las entradas y salidas de la tarjeta embebida.
- Realizar una conexión entre los sensores multiplexados y la tarjeta embebida.
- Implementar un controlador PID Neuronal para controlar la posición del robot Line Follower.
- Ubicar en diferentes posiciones la regleta de sensores.

PLANTEAMIENTO DEL PROBLEMA

- Se debe usar el programa Mbed Studio.
- Se requiere hacer la calibración de los sensores considerando que se está usando una regleta multiplexada 4 a 1 tipo analógica en donde nos permitirá leer los 16 sensores.
- La regleta multiplexada nos va permitir al robot en qué posición esta la línea negra con fondo blanco.
- El robot Line Follower para poder calibrarlo deberá ubicarlo primero en el fondo blanco y con un pulsador dar la orden para que registre los valores obtenido, luego ubicarlo en la línea negra y con el mismo pulsador dar la siguiente orden para que registre los valores, como último pulso deberá mostrar los valores de cada sensor calibrado.
- Se deberá crear una variable de tipo **long int position** que deberá estar en un rango de 0 a 12 cm, con esto el robot estará listo para la aplicación de los diferentes controladores lógicos a utilizarse.
- Seguir las indicación para la implementación del control PID Neuronal que se encuentra en el anexo 6.

4.7 Práctica #7

UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR	GUÍA DE PRÁCTICA DE LABORATORIO
CARRERA: Ingeniería Electrónica	ASIGNATURA: Informática Industrial, Sistemas Inteligentes, Diseño Electrónico, Robótica Móvil.
NÚMERO DE PRÁCTICA: 7	**TITULO DE PRÁCTICA**: "Diseño de algoritmo de entrenamiento para la Red Neuronal"

OBJETIVOS:

- Identificar las entradas y salidas del sistema
- Diseñar un algoritmo de entrenamiento en la Tarjeta Embebida.

PLANTEAMIENTO DEL PROBLEMA

Se desea que la práctica cumpla lo siguiente:

- Se debe usar el programa Mbed Studio.
- Se requiere hacer el entrenamiento la siguiente compuerta lógica XOR.
- A continuación, la tabla lógica XOR

A	B	Q
1	1	0
1	0	1
0	1	1
0	0	0

4.8 Práctica #8

UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR	GUÍA DE PRÁCTICA DE LABORATORIO
CARRERA: Ingeniería Electrónica	ASIGNATURA: Informática Industrial, Sistemas Inteligentes, Diseño Electrónico, Robótica Móvil.
NÚMERO DE PRÁCTICA: 8	**TITULO DE PRÁCTICA**: "Control PD Difuso implementado al robot Line Follower."

OBJETIVOS:

- Identificar las entradas y salidas de la tarjeta embebida.
- Realizar una conexión entre los sensores multiplexados y la tarjeta embebida.
- Implementar un controlador PD Difuso para controlar la posición del robot Line Follower.
- Ubicar en diferentes posiciones la regleta de sensores.

PLANTEAMIENTO DEL PROBLEMA

Se desea que la práctica cumpla lo siguiente:

- Se debe usar el programa Mbed Studio.
- Se requiere hacer la calibración de los sensores considerando que se está usando una regleta multiplexada 4 a 1 tipo analógica en donde nos permitirá leer los 16 sensores.
- La regleta multiplexada nos va permitir al robot en qué posición esta la línea negra con fondo blanco.
- El robot Line Follower para poder calibrarlo deberá ubicarlo primero en el fondo blanco y con un pulsador dar la orden para que registre los valores obtenido, luego ubicarlo en la línea negra y con el mismo pulsador dar la siguiente orden para que registre los valores, como último pulso deberá mostrar los valores de cada sensor calibrado.
- Se deberá crear una variable de tipo **long int position** que deberá estar en un rango de 0 a 12cm, con esto el robot estará listo para la aplicación de los diferentes controladores lógicos a utilizarse.
- Seguir las indicación para la implementación del control PD Difuso que se encuentra en el anexo 8.

4.9 Práctica #9

UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR	GUÍA DE PRÁCTICA DE LABORATORIO
CARRERA: Ingeniería Electrónica	ASIGNATURA: Informática Industrial, Sistemas Inteligentes, Diseño Electrónico, Robótica Móvil.
NÚMERO DE PRÁCTICA: 9	**TITULO DE PRÁCTICA**: "Diseño de una aplicación móvil para el control del robot."

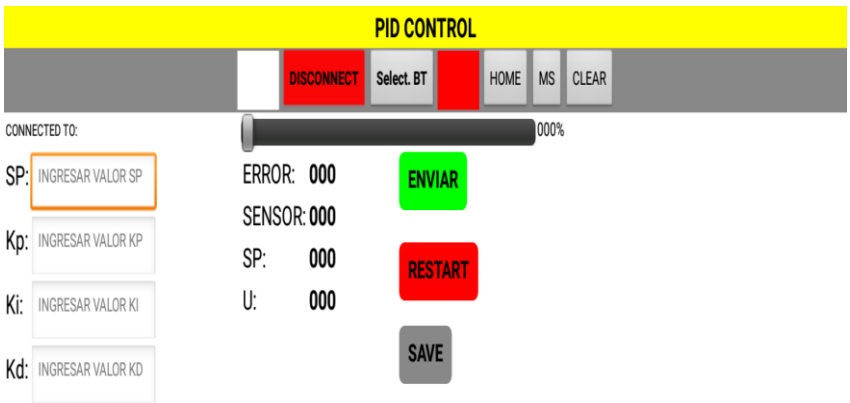
OBJETIVOS:

- Configurar el robot Line Follower para recepción de datos seriales mediante el protocolo uart.
- Realizar la conexión entre el robot Line Follower y una aplicación Android.
- Enviar las ganancias del controlador PID mediante una aplicación Android.

PLANTEAMIENTO DEL PROBLEMA

Se desea que el robot Line Follower cumpla con las siguientes especificaciones:

- Realizar un algoritmo lógico que permita al robot Line Follower adquirir las ganancias del controlador PID mediante una aplicación Android.
- Desarrollar un algoritmo lógico para un controlador PID.
- En la figura se muestra el diseño de la aplicación Android propuesto para esta práctica.
- Seguir las indicaciones para la implementación de la aplicación que se encuentra en el anexo 9.



4.10 Práctica #10

UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR	GUÍA DE PRÁCTICA DE LABORATORIO
CARRERA: Ingeniería Electrónica	ASIGNATURA: Informática Industrial, Sistemas Inteligentes, Diseño Electrónico, Robótica Móvil.
NÚMERO DE PRÁCTICA: 10	**TITULO DE PRÁCTICA**: "Adquisición de datos en tiempo real de la señal de control y la variable ajustada."

OBJETIVOS:

- Identificar las entradas y salidas de la tarjeta embebida.
- Realizar una conexión entre los sensores multiplexados y la tarjeta embebida.
- Implementar un controlador PID para controlar la posición del robot Line Follower.
- Ubicar en diferentes posiciones la regleta de sensores.
- Realizar la adquisición de datos mediante el software LABVIEW.

PLANTEAMIENTO DEL PROBLEMA

Se desea que el robot Line Follower cumpla con las siguientes especificaciones:

- Desarrollar una aplicación en el programa labview donde se adquiera los datos en tiempo real entre la señal de control y la variable ajustada.
- En la figura se muestra el diseño propuesto para la práctica.
- Seguir las indicaciones para la implementación de la adquisición de datos en tiempo real de la señal de control y la variable ajustada que se encuentra en el anexo 10.

5. RESULTADO

5.1 ANÁLISIS DE RESULTADOS.

5.1.1 Resultados de la práctica 1 - Diseño electrónico en Fusion 360

En la figura 23 se muestra el resultado de la practica 1 la elaboración del diseño de la PCB.

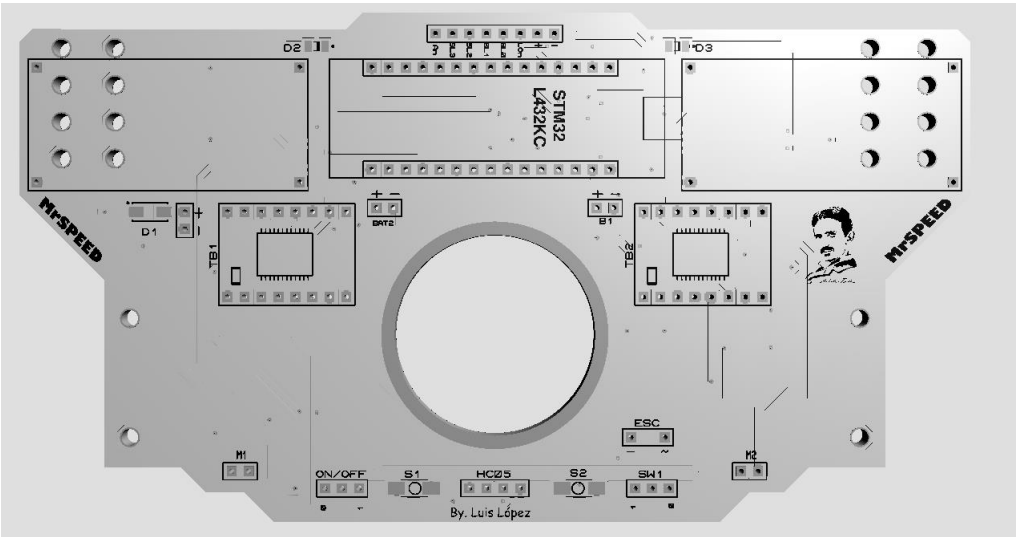


Figura 25. Resultado de la Práctica 1 - Diseño de la PCB

Una vez elaborado el diseño se envió a fabricar la PCB de nuestro prototipo en la figura 24 se muestra el resultado de la misma.

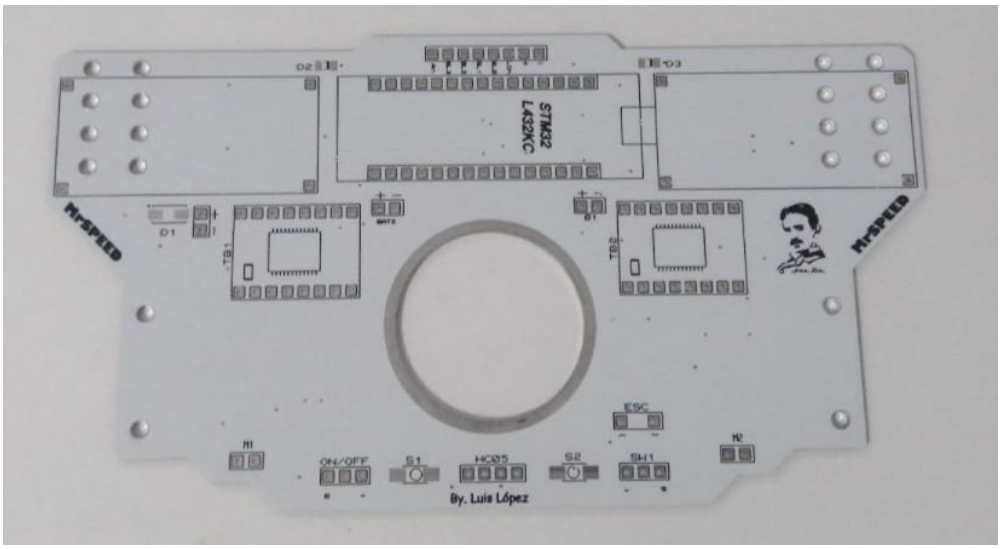


Figura 26. Resultado Práctica 1 - Fabricación de la PCB

5.1.2 Resultados de la práctica 2 - Programación Mbed Studio para monitoreo de sensores.

Al compilar el código en la tarjeta embebida STM32L432KC se tiene como resultado el monitoreo de los 16 sensores multiplexado en donde el dato 0 nos indica que se encuentra en la zona blanca y el dato mayor 0 nos indica que se encuentra en la línea.

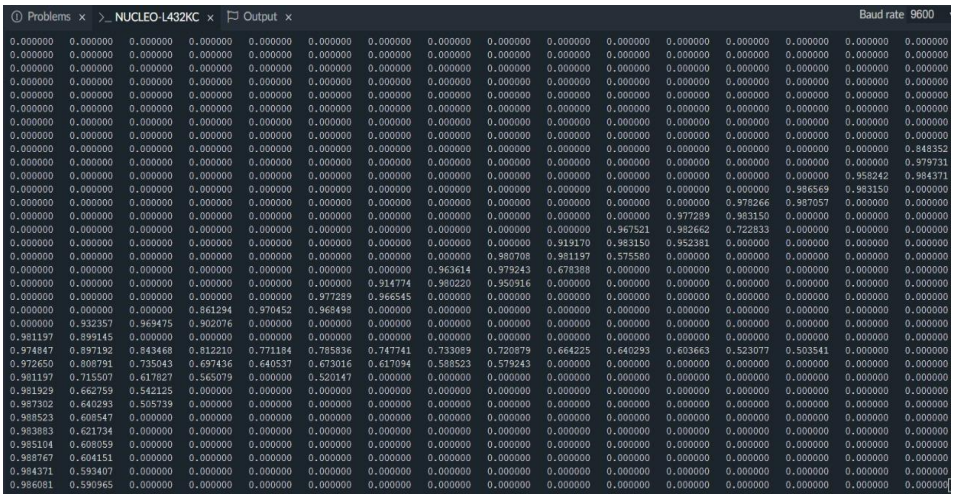


Figura 27. Resultado de la práctica 2 - monitoreo de 16 sensores.

Para ser visualizado estos datos se realizó al robot primero el movimiento de los sensores hacia la derecha y luego hacia la izquierda como se muestra en la figura 26.

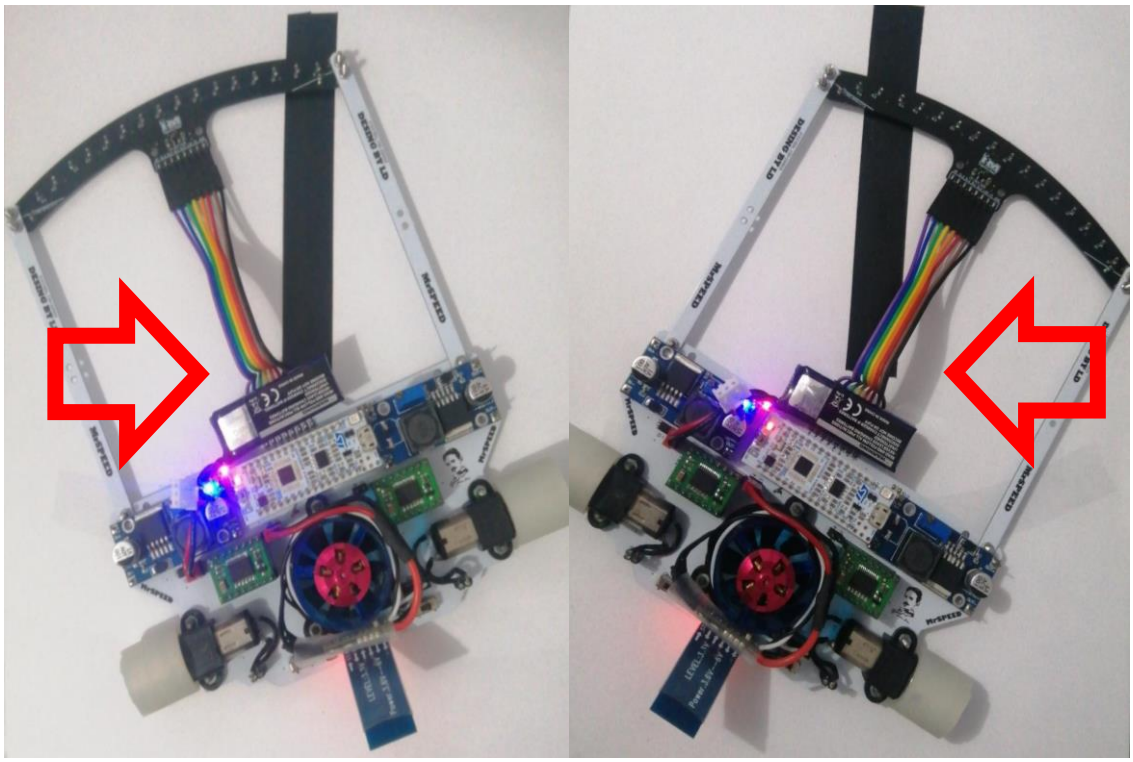


Figura 28. Resultado de la práctica 2 - Movimiento de la regleta.

5.1.3 Resultados de la práctica 3 - Programación Mbed Studio control de actuadores.

Al compilar el código en la tarjeta embebida STM32L432KC se tiene como resultado el control de los motores mediante la siguiente función creada, los valores que se ingresan de 0….1 es el datos que controla el PWM de nuestra salida, si el valor es negativo automáticamente cambia de estado los pines que controlan el sentido de los motores como se muestra en la figura 27.

```
58 void motors(double speedL, double speedR)
59 {
60     if(speedL>0) {
61         AIN1=1;
62         AIN2=0;
63     } else {
64         AIN1=0;
65         AIN2=1;
66         speedL=speedL*-1;
67     }
68     if(speedR>0) {
69         BIN1 = 1;
70         BIN2 = 0;
71     } else {
72         BIN1 = 0;
73         BIN2 = 1;
74         speedR=speedR*-1;
75     }
```

Figura 29. Resultado de la práctica 3 - Función para el control de los motores.

Condición 1: motors (0,0.5) el motor izquierda se encuentra desactivado y el motor derecho en alto hacia adelante.

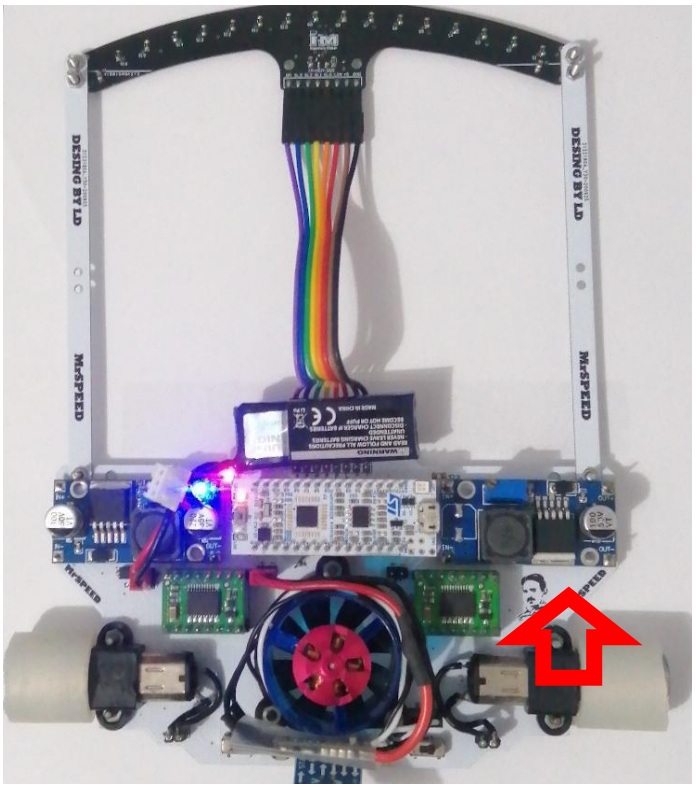


Figura 30. Resultado de la práctica 3 - motor derecho en alto hacia adelante.

Condición 2: motors (0,-0.5) el motor izquierda se encuentra desactivado y el motor derecho en alto hacia atrás.

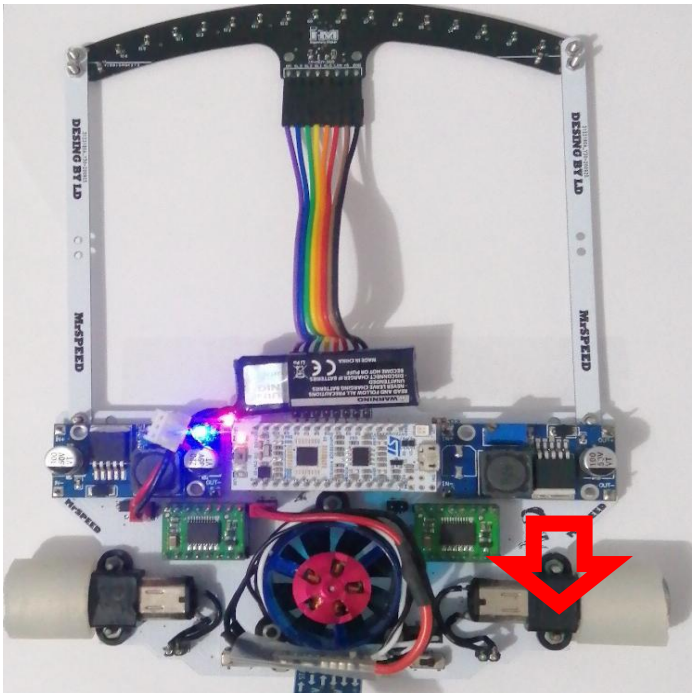


Figura 31. Resultado de la práctica 3 - motor derecho en alto hacia atrás.

Condición 3: motors (0.5,0) el motor derecho se encuentra desactivado y el motor izquierdo en alto hacia adelante.

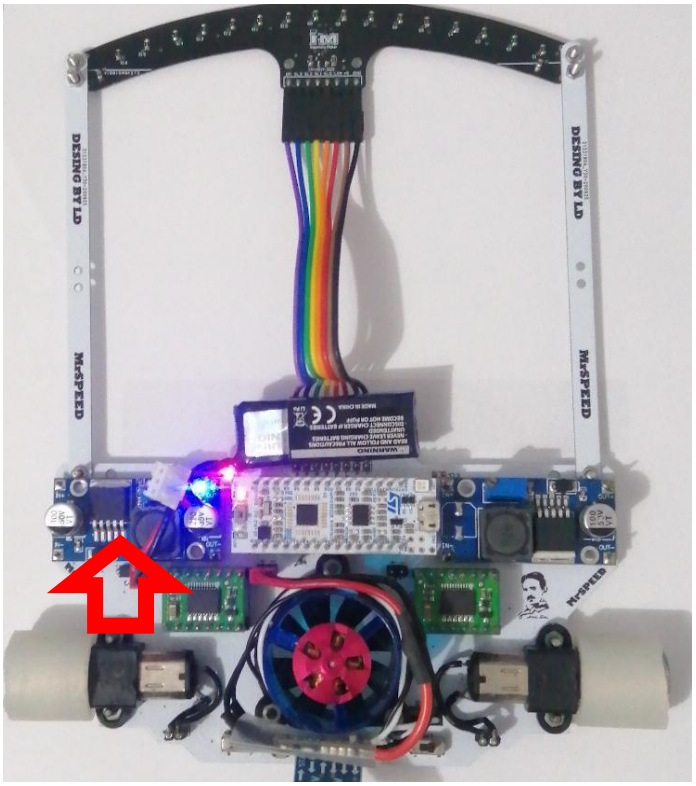


Figura 32. Resultado de la práctica 3 - motor izquierdo en alto hacia adelante.

Condición 4: motors (- 0.5,0) el motor derecho se encuentra desactivado y el motor izquierdo en alto hacia atrás.

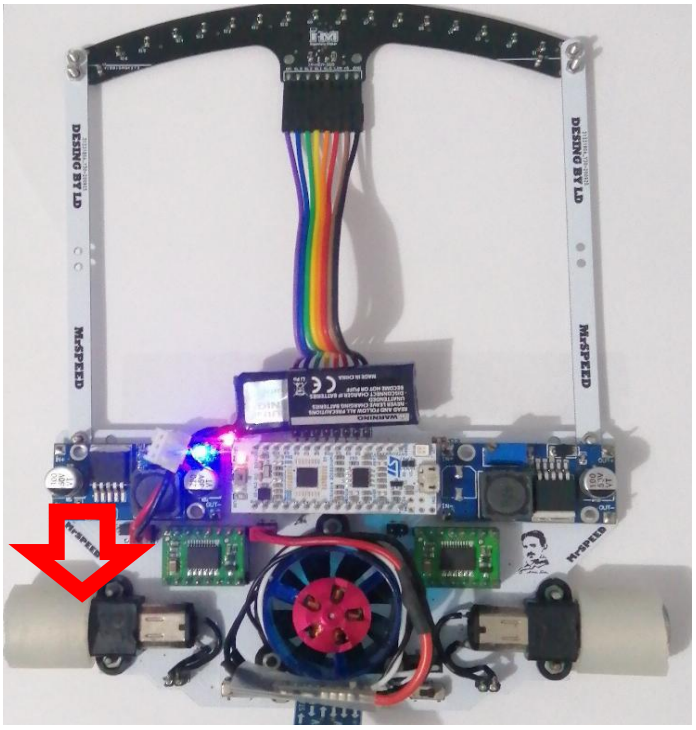


Figura 33. Resultado de la práctica 3 - motor izquierdo en alto hacia atrás.

Condición 5: motors (0.5,0.5) el motor derecho y el motor izquierdo en alto hacia adelante.

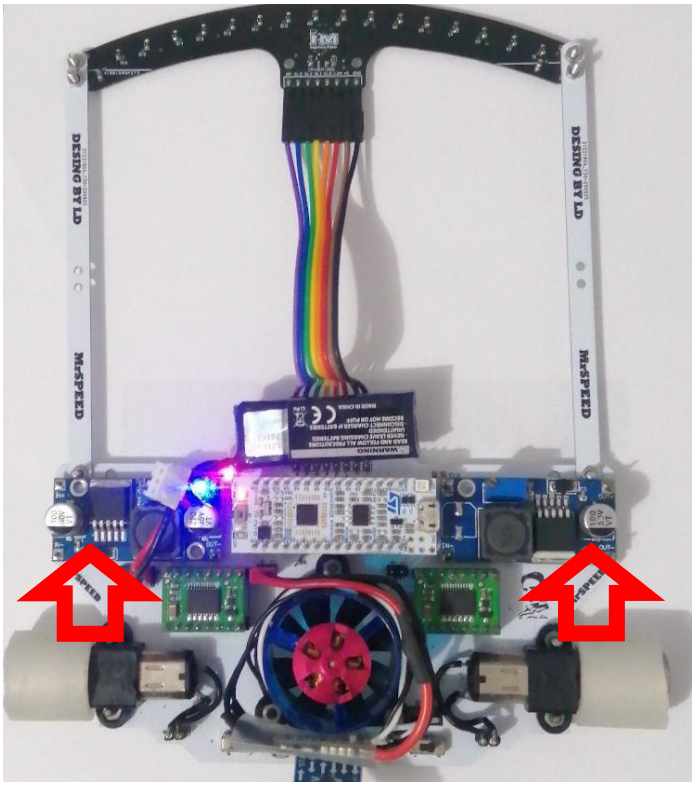


Figura 34. Resultado de la práctica 3 - ambos motores en alto hacia adelante.

Condición 6: motors (-0.5,-0.5) el motor derecho y el motor izquierdo en alto hacia atrás.

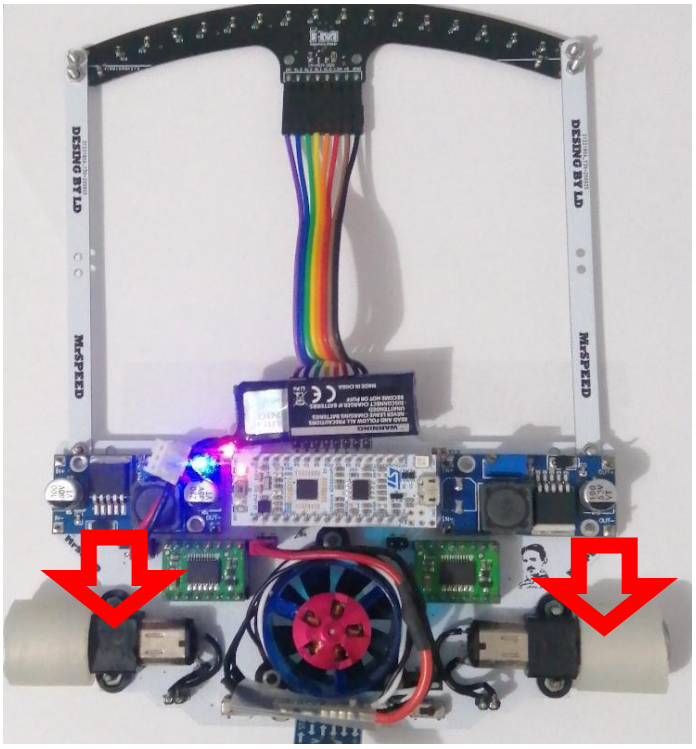


Figura 35. Resultado de la práctica 3 - ambos motores en alto hacia atrás.

Condición 7: motors (0.5,-0.5) el motor izquierdo en alto hacia adelante y el motor derecho en alto hacia atrás.

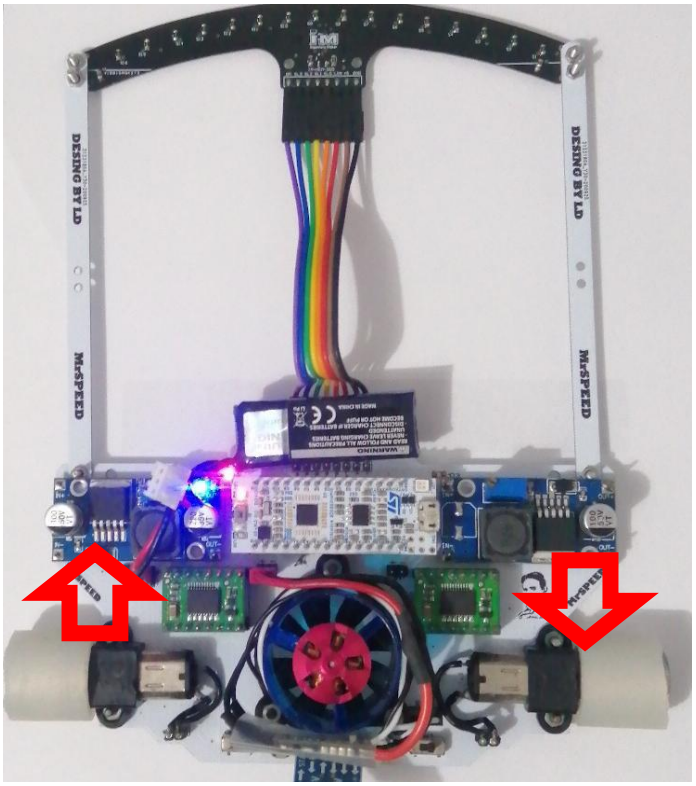


Figura 36. Resultado de la práctica 3 - motor izquierdo hacia delante y motor derecho hacia atrás.

Condición 8: motors (-0.5,0.5) el motor izquierdo en alto hacia atrás y el motor derecho en alto hacia adelante.

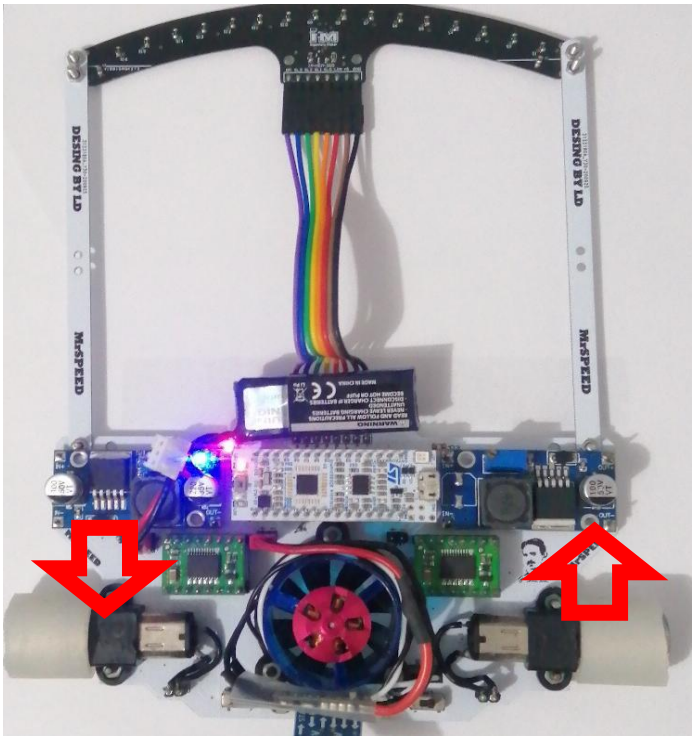


Figura 37. Resultado de la práctica 3 - motor izquierdo hacia atrás y motor derecho hacia adelante.

5.1.4 Resultados de la práctica 4 - Codificación de algoritmo para calibración de sensores.

Al compilar el código en la tarjeta embebida STM32L432KC se tiene como resultado la calibración de los 16 sensores multiplexado. Se va obtener un solo dato que va de 0 a 1500 que nos indicará en qué posición de la línea se encuentra nuestra regleta de sensores.

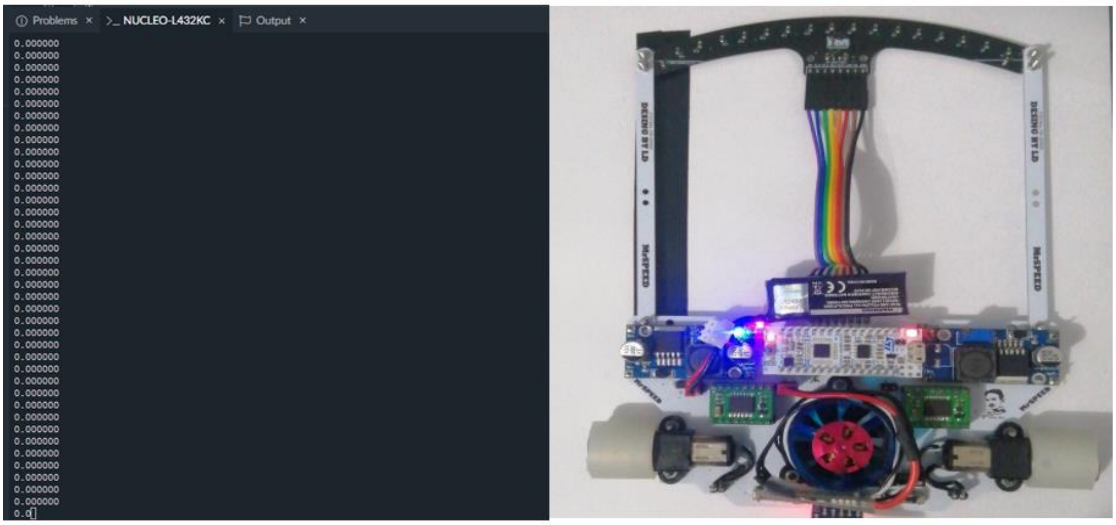


Figura 38. Resultado de la práctica 4 - dato de posición de la línea en 0.

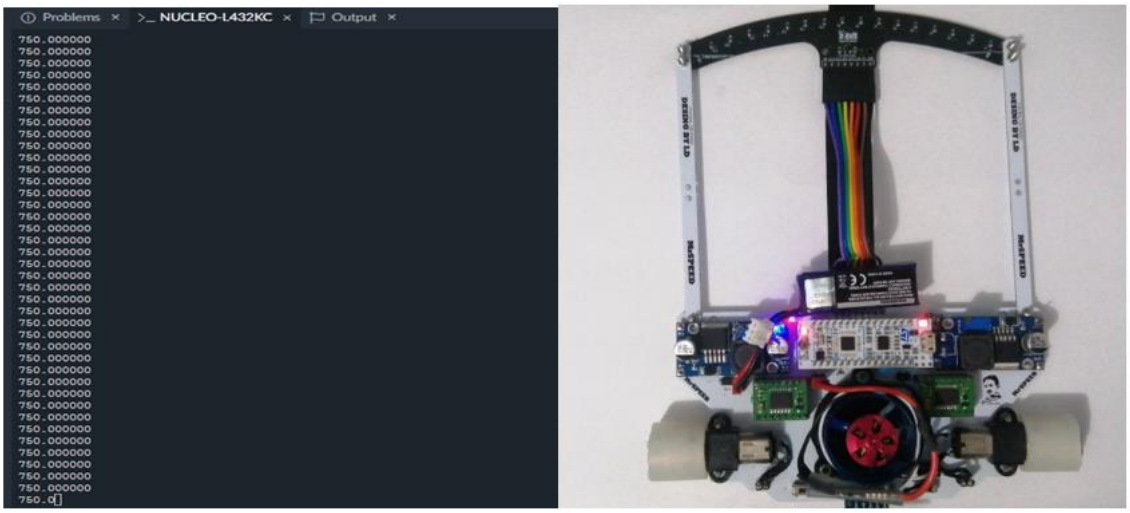


Figura 39. Resultado de la práctica 4 – dato de posición de la línea en 750.

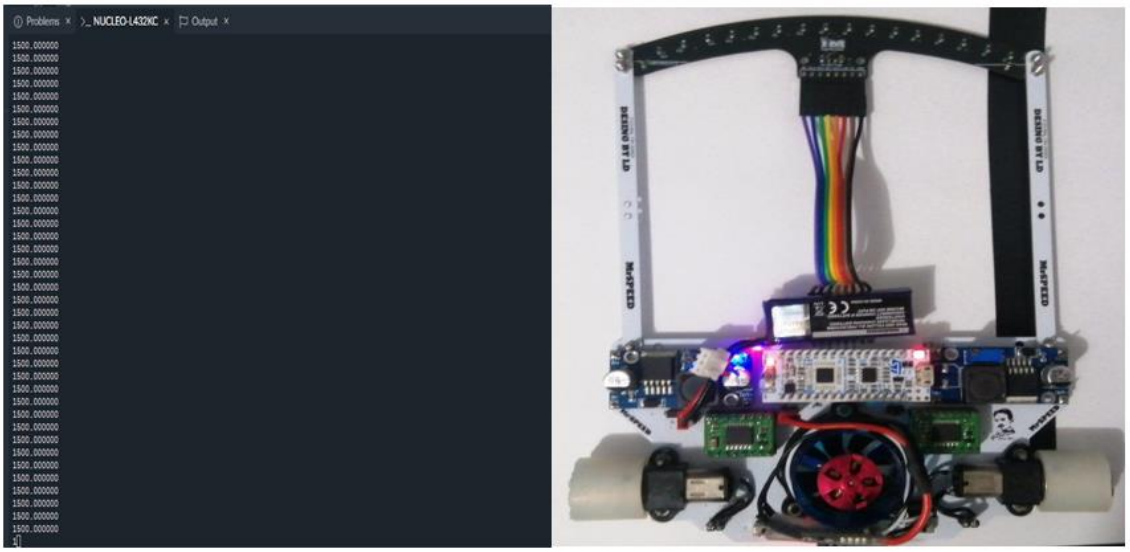


Figura 40. Resultado de la práctica 4 – dato de posición de la línea en 1500.

5.1.5 Resultados de la práctica 5 - Control PID clásico aplicado al robot Line Follower.

En el desarrollo de la sexta practica se desarrolló en el labview una gráfica virtual para visualizar el comportamiento del controlador PID clásico como se muestra en la figura 39, el robot al encontrarse una curva mediante el controlador aplicado se corrige buscando nuevamente en la línea.

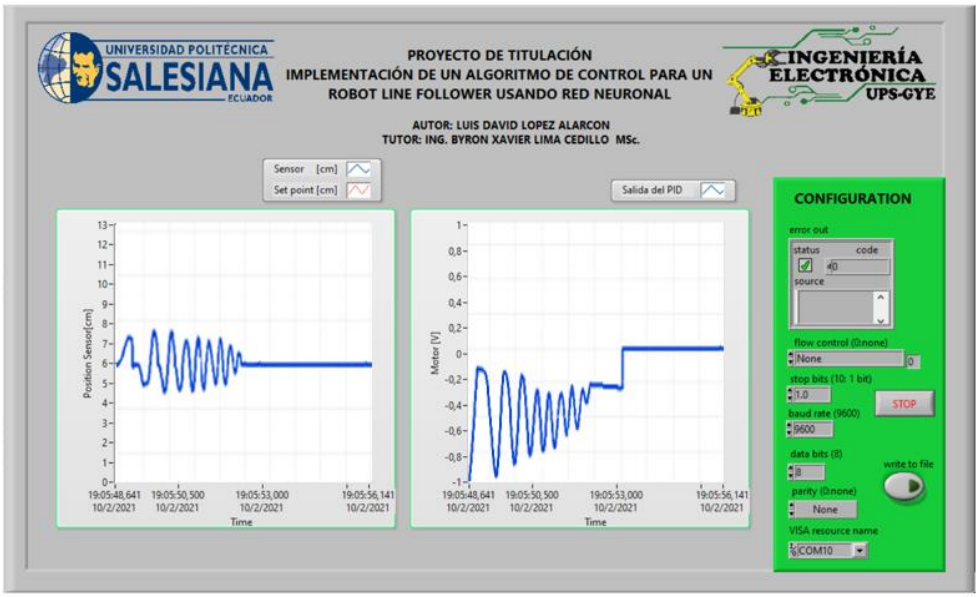


Figura 41. Resultado de la práctica 5 – PID robot Line Follower.

Al ejecutar el microcontrolador el robot comienza realizar oscilaciones como se muestra en la figura 40 hasta lograr estabilizarse cumpliendo con el objetivo de no salirse de la línea negra

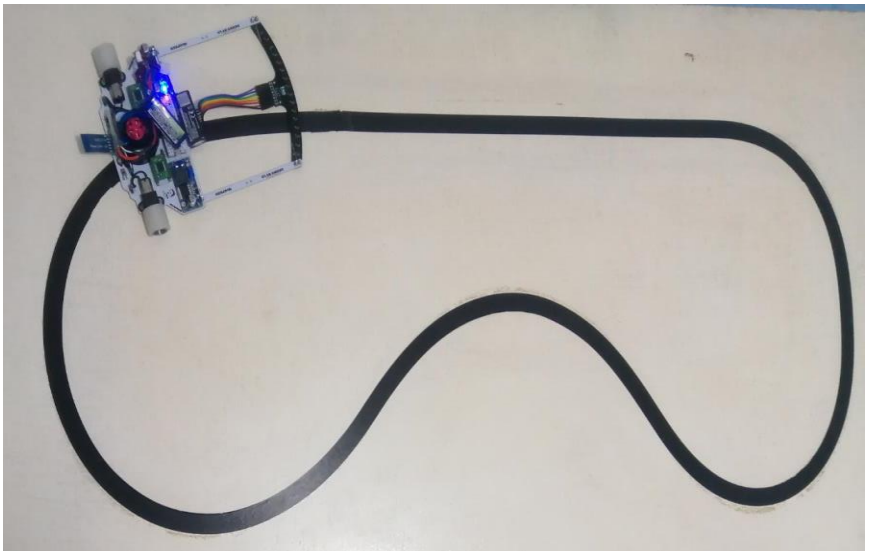


Figura 42. Resultado de la práctica 5 – Oscilaciones del robot Line Follower.

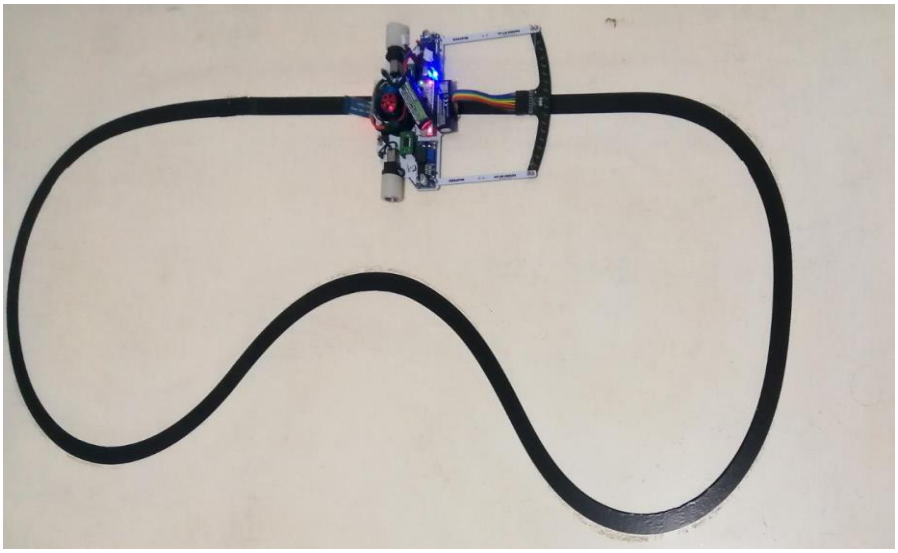


Figura 43. Resultado de la práctica 5 –Estabilidad del robot Line Follower.

5.1.6 Resultados de la práctica 6 - Control PID Neuronal aplicado al robot Line Follower.

En el desarrollo de la sexta practica se desarrolló en el labview una gráfica virtual para visualizar el comportamiento del controlador PD Difuso como se muestra en la figura 42, el robot al encontrarse una curva mediante el controlador aplicado se corrige buscando nuevamente en la línea.

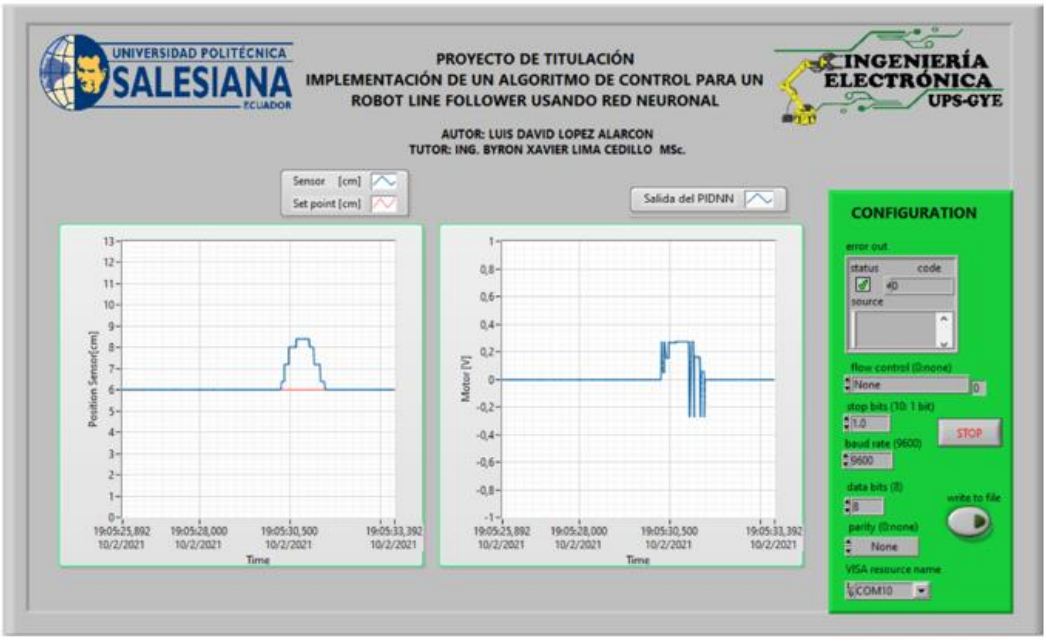


Figura 44. Resultado de la práctica 6 – PID Neuronal robot Line Follower.

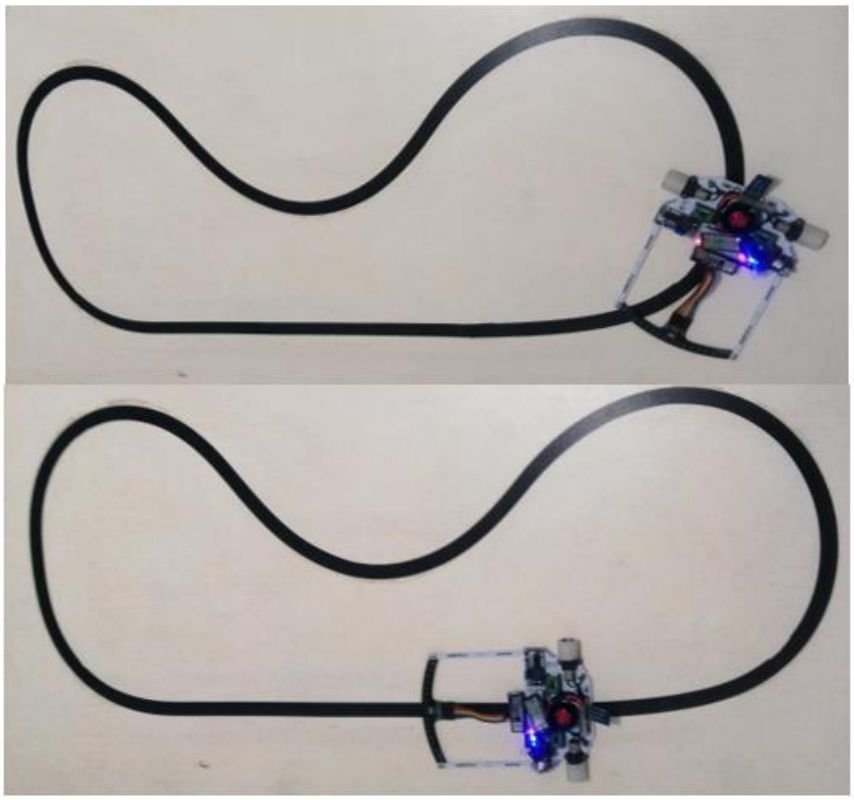


Figura 45. Resultado de la práctica 6 – Antes y después de la curva.

5.1.7 Resultados de la práctica 7 - Diseño de algoritmo de entrenamiento para la Red Neuronal.

Mediante el software MATLAB se realizó el entrenamiento de la red para obtener los pesos dando como resultado en la figura

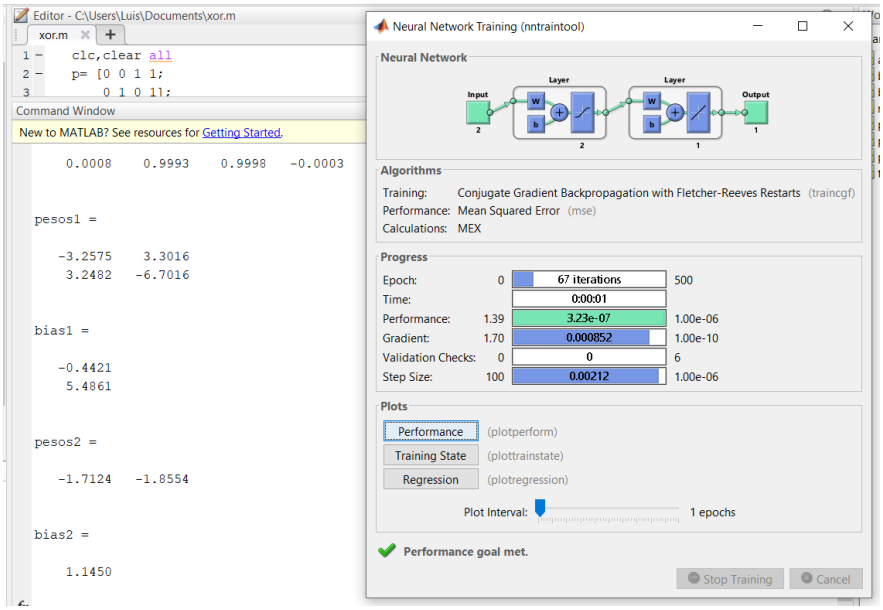


Figura 46. Resultado de la práctica 7 – Entrenamiento de la Red Neuronal

5.1.8 Resultados de la práctica 8 - Control PD Difuso implementado al robot Line Follower.

En el desarrollo de la octava practica se desarrolló en el labview una gráfica virtual para visualizar el comportamiento del controlador PD Difuso como se muestra en la figura 44.

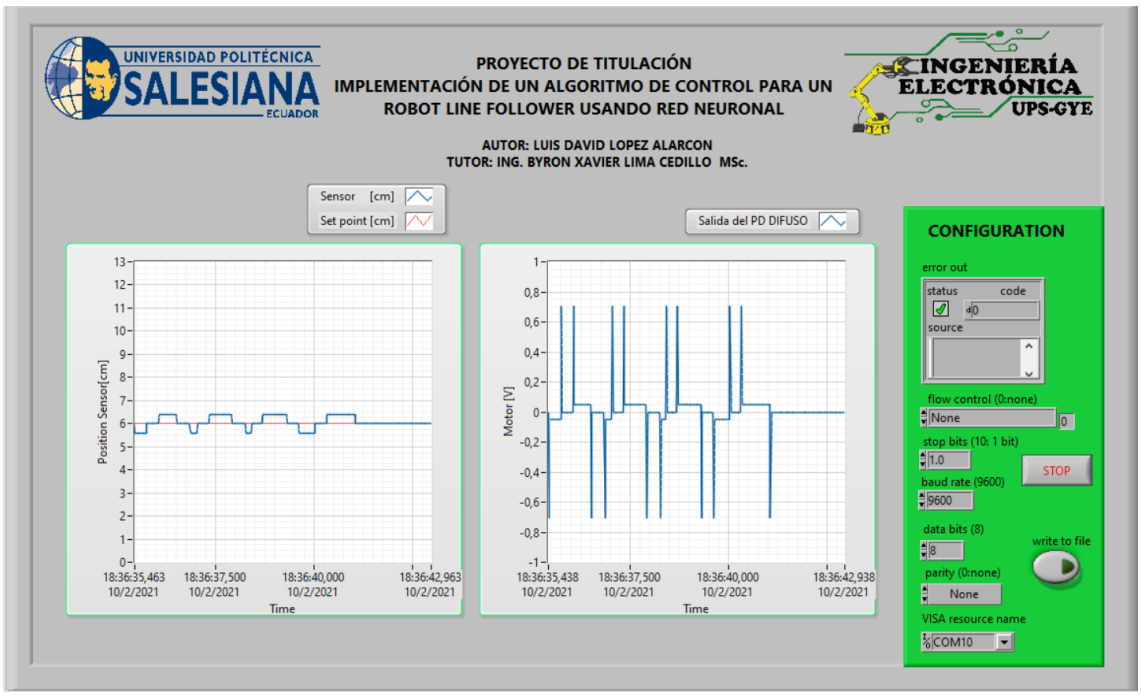


Figura 47. Resultado de la práctica 8 – PD DIFUSO robot Line Follower.

En la siguiente figura 45 se observa la estabilización del robot mediante el PD DIFUSO.

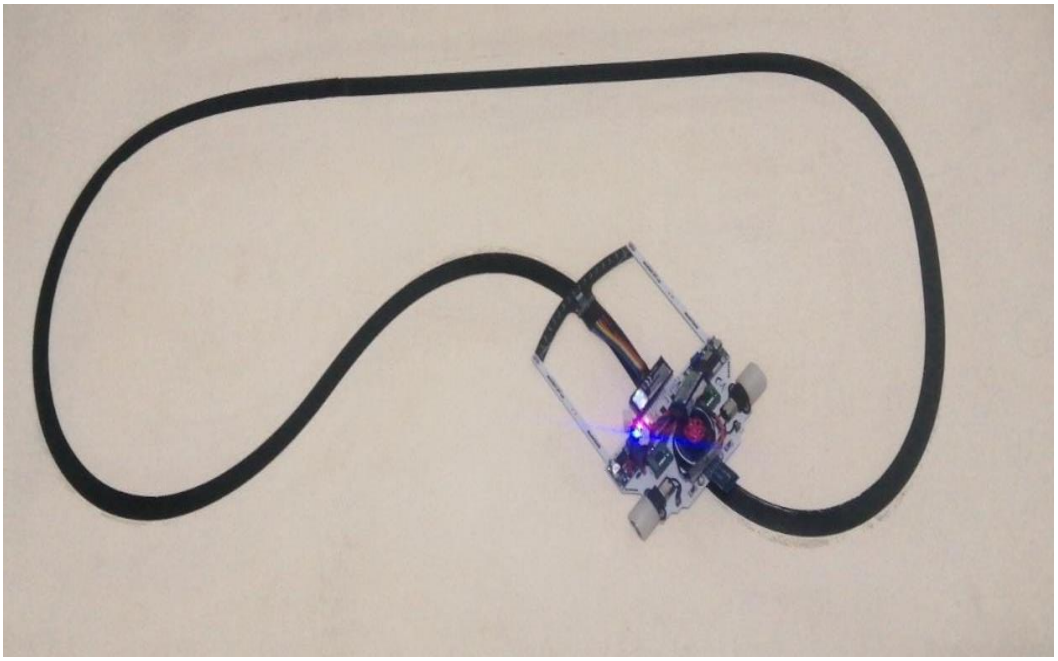


Figura 48. Resultado de la práctica 8 – Estabilización de robot.

5.1.9 Resultados de la práctica 9 - Diseño de una aplicación móvil para el control del robot.

Se debe ejecutar la aplicación PID CONTROL creada, el cual se envía las ganancias del pid para la estabilización del robot Line Follower.

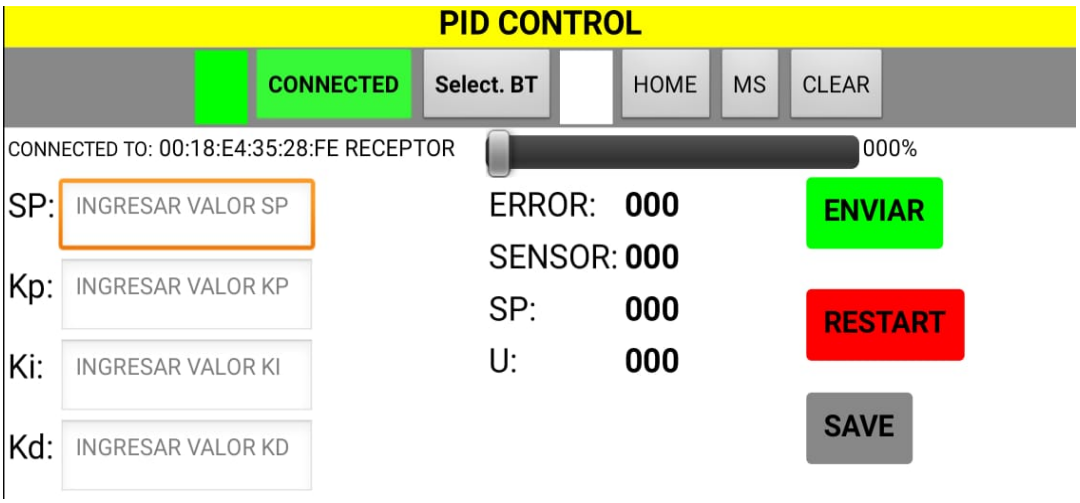


Figura 49. Resultado de la práctica 9 – Ejecución de la app.

A continuación, se escribe las ganancias del pid para ser enviado posteriormente.

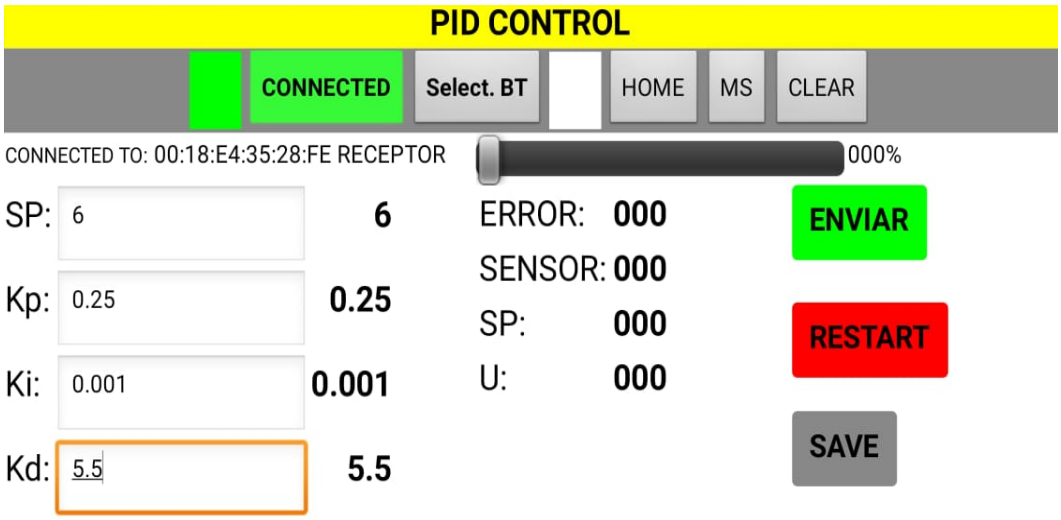


Figura 50. Resultado de la práctica 9 - Escritura de las ganancias del PID.

En la figura 48, se muestra la comunicación que hay entre la aplicación y el robot Line Follower al recibir los valores enviados atreves de la app.

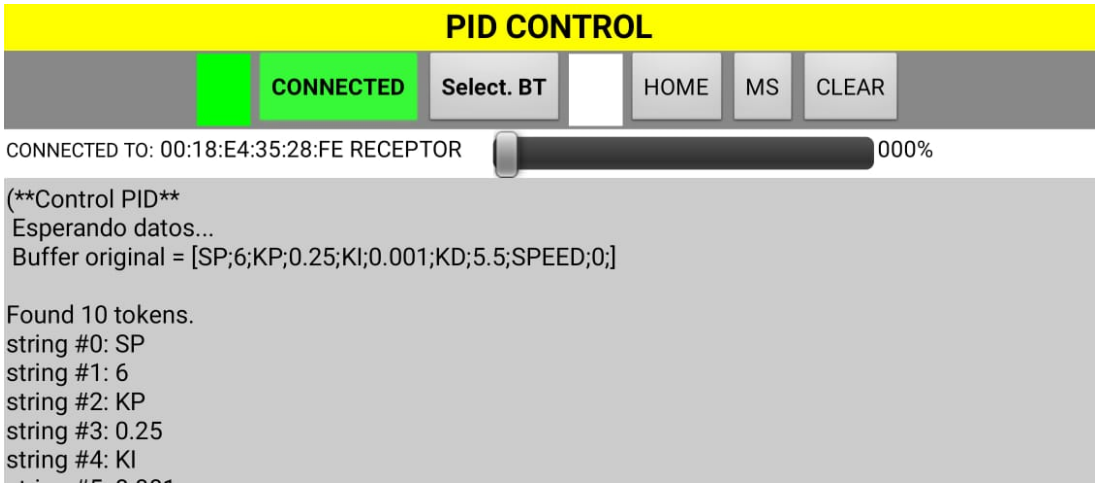


Figura 51. Resultado de la práctica 9 - recibimiento de datos del robot Line Follower.

Ahora en la siguiente figura 49, se muestra los valores de la calibración de los sensores listo para ser poner puesta en marcha al robot Line Follower.

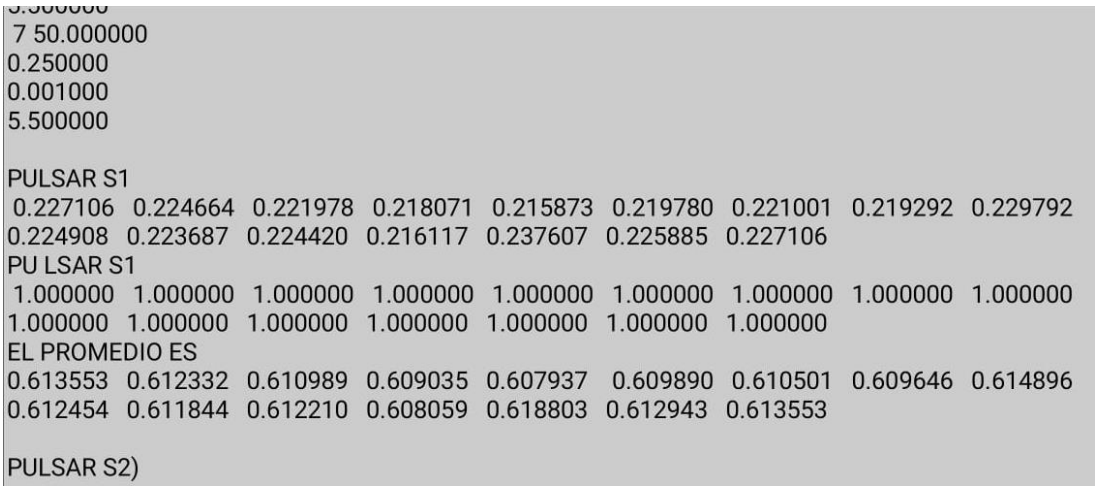


Figura 52. Resultado de la práctica 9 - Valores de calibración de los sensores.

5.1.10 Resultados de la práctica 10 - Adquisición de datos en tiempo real de la señal de control y la variable ajustada.

Al compilar el código en la tarjeta embebida STM32L432KC se tiene como resultado la estabilización del robot Line Follower visualizado en una interfaz gráfica como se muestra en la figura 50.

Figura 53. Resultado de la práctica 10 - Señal de control y variable ajustada.

En la interfaz gráfica cuenta con **Write to file** que nos permitirá guardar los datos en un archivo de formato **.txt** como se muestra en la figura 51.

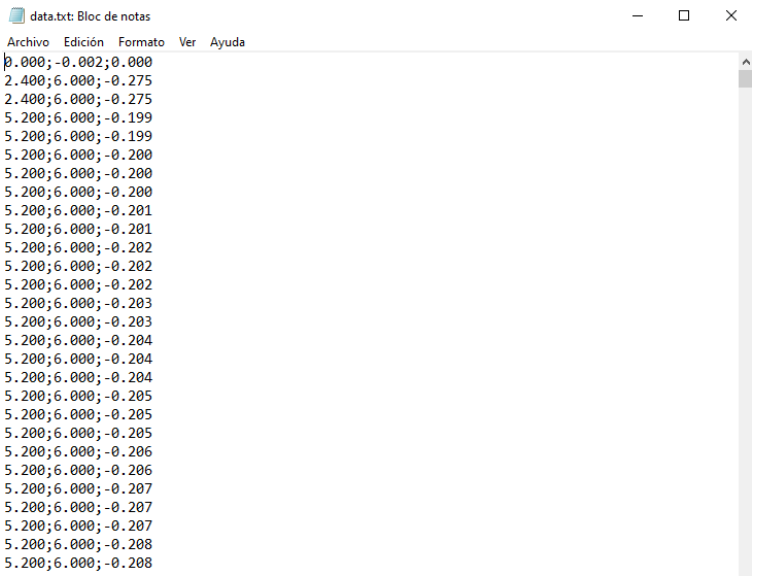


Figura 54. Resultado de la práctica 10 - adquisición de datos.

CONCLUSIONES

- Se diseñó un instrumento virtual en labview donde nos permite monitorear la salida y entrada que tiene cada algoritmo de control aplicado al robot Line Follower.
- Se diseñó un algoritmo para la adquisición de datos en tiempo real en donde se podrá visualizar el comportamiento de la señal de salida de los diferentes tipos de controladores planteado en las prácticas en nuestro prototipo.
- Se realizó un manual de 10 prácticas para el robot Line Follower controlado con una tarjeta stm32l432kc para complementar la catedra en diferentes asignaturas de la carrera.
- El controlador pid es más viable usar en estos tipos de robot ya que al no usar utilizar de cálculos matemáticos complejos no consumiría muchos recursos de la stm32, en comparación con el PD difuso el cual posee mucho más línea de código, cálculos, declaración de variables.
- Se diseñó una aplicación personal mediante app inventor y se la en lazo con la stm32 mediante el bluetooth hc-05 que nos permitirá el envío y recepción de datos.
- Con el uso del software Hercules se pudo verificar mediante puerto serial el correcto funcionamiento de la programación, tales como la calibración de los sensores, monitorear las salidas que posee cada controlador aplicado al prototipo.

RECOMENDACIONES

- Revisar las conexiones que posee cada parte del robot al momento de realizar el ensamblaje.
- Verificar el estado en que se encuentre la batería antes de cualquier manipulación.
- Revisar que el robot debe estar bien asegurado con sus tornillos que sujetan a la regleta de sensores y a los motores.
- Antes de cada competencia realizar una revisión a las cajas reductoras que poseen cada motor ya que puede existir un desgaste en los engranes disminuyendo la velocidad y movimiento del robot.
- Como mejora del robot se recomienda llevar a electrónica smd a los módulos Step Up, Step Down, driver de potencia, logrando así que el peso disminuya y sea favorable para el robot.
- Como trabajo futuro se podría implementar encoder a los motores para así obtener la odometria aplicado al robot Line Follower.
- Realizar una adaptación para cualquier tipo de tarjeta embebida quedando la tarjeta del prototipo genérica.
- Implementar nuevos controladores robusto en el sistema embebido stm32 como controladores adaptativos, neuro difusos y realizar comparaciones con los controladores de aplicados en el presente proyecto.

BIBLIOGRAFÍA

Alvarado, R. C., Mendoza, A., Solorzano, G., & Ponce, J. (s.f.).

Bureau, M. (s.f.). *Motorpasion Mexico*. Obtenido de https://www.motorpasion.com.mx/seguridad/la-importancia-del-inflado-de-las-llantas

Camarillo, A. (29 de julio de 2019). Obtenido de 330ohms: https://blog.330ohms.com/2019/07/29/que-es-un-regulador-de-voltaje/

Carlos Alberto Ruiz, M. S. (Marzo de 2001). Obtenido de https://www.frro.utn.edu.ar/repositorio/catedras/quimica/5_anio/orientadora1/monograias/matich-redesneuronales.pdf

Catherin Marin Quintero, A. G. (s.f.). *overblog*. Obtenido de http://redesindustriales.over-blog.com/2014/10/redes-industriales-0.html

Catherin Marin Quintero, A. G. (s.f.). *Overblog*. Obtenido de http://redesindustriales.over-blog.com/2014/10/redes-industriales-0.html

Comunicaciones Digitales. (s.f.). Obtenido de http://www.itq.edu.mx/carreras/IngElectronica/archivos_contenido/Apuntes%20de%20materias/ETD1022_Microcontroladores/4_SerialCom.pdf

Dr. Celso Márquez-Sánchez, D. J.-S.-G. (1 de mayo de 2019). *boletin upiita*. Obtenido de http://www.boletin.upiita.ipn.mx/index.php/ciencia/813-cyt-numero-72/1677-robots-seguidores-de-linea-generalidades

Mazzone, V. (maro de 2002). Obtenido de https://www-eng.newcastle.edu.au/~jhb519/teaching/caut1/Apuntes/PID.pdf

Michellin. (s.f.). Obtenido de https://www.michelintruck.com/reference-materials/manuals-bulletins-and-warranties/load-and-inflation-tables/#/

Moncada, L. (s.f.). *plantscontrol*. Obtenido de http://plantscontrol.blogspot.com/2012/02/6_13.html

Ogata, K. (2010). *Ingenieria de contol moderna*. Obtenido de https://www.u-cursos.cl/usuario/78303fe04da8e4eb340eaee09f1840b2/mi_blog/r/Ingenieria_de_Control_Moderna_Ogata_5a_ed.pdf

paz, A. (s.f.). *Monografias*. Obtenido de https://www.monografias.com/trabajos53/topologias-red/topologias-red.shtml

Relief, L. W. (s.f.). Obtenido de http://infocafes.com/portal/wp-content/uploads/2017/02/19_Guia_8_Beneficiado.pdf

semanticwebbuilder. (s.f.). Obtenido de http://www.semanticwebbuilder.org.mx/es_mx/swb/Sistemas_Embebidos_Innovando_hacia_los_Sistemas_Inteligentes_

ANEXOS

Anexo 1: PRÁCTICA #1

Se debe contar instalado el programa Fusion 360 para realizar el diseño del robot Line Follower.

Figura 55. Fusión 360

Paso 1

Nos dirigimos a **File** y le damos click en **New Electronic Library.**

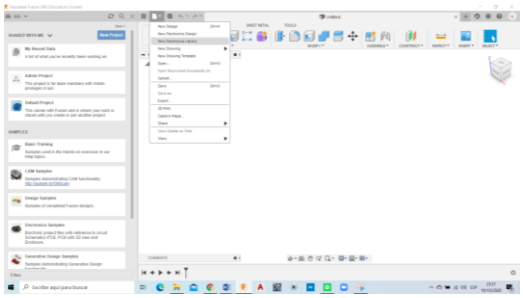


Figura 56. New Electronic Library

Paso 2

Crear los siguientes elementos a usarse en el diseño del robot Line Follower.

- Módulo Tb16fn12g.
- Regulador por conmutación Step Down y Step UP.
- Controlador Stm32 núcleo l432kc.

Paso 3

Nos dirigimos a **create new symbol** y le asignamos el nombre tb16fn12g y le damos **Ok.**

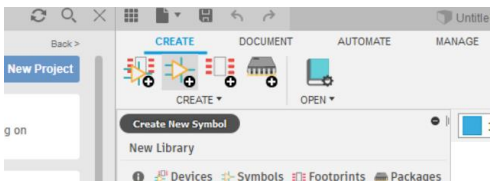


Figura 57. Create new symbol

Usamos el símbolo Line para hacer el símbolo del módulo.

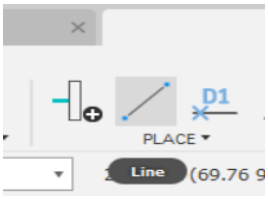


Figura 58. Símbolo Line

Usamos el símbolo **Pin** para ubicarle los 8 pines en cada lado que lleva el modulo.

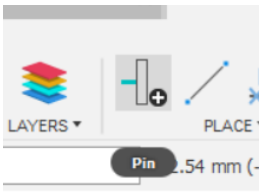


Figura 59. Símbolo Pin

Le damos click en cada pin y elegimos propiedades para ubicarle el nombre que lleva cada pin.

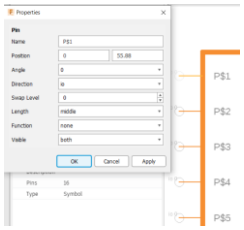


Figura 60. Propiedades

Obtendremos de esa manera ya creado nuestro símbolo del módulo.

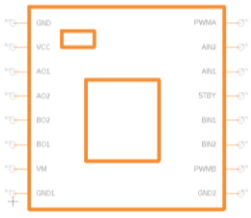


Figura 61. Módulo TB6612FNG

Ahora creamos nuestro footprint y le ponemos el mismo nombre del símbolo.

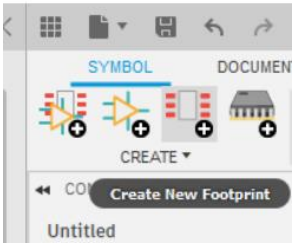


Figura 62. Footprint

Usamos el siguiente símbolo que nos permitirá realizar el path del módulo considerar que la distancia entre path y path es de 100mil y le asignamos el nombre a cada path creado.

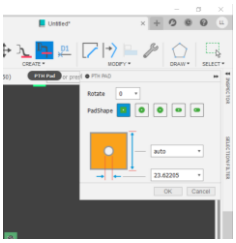


Figura 63. Símbolo PTH Pad

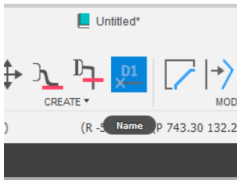


Figura 64. Símbolo Name

Finalmente tenemos footprint creado.

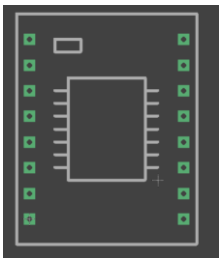


Figura 65. Footprint TB6612FNG

Ahora debemos unir nuestro símbolo y footprint creado para culminar con la creación de la librería. Le damos clic en **Create New Device** y le asignamos el mismo nombre.

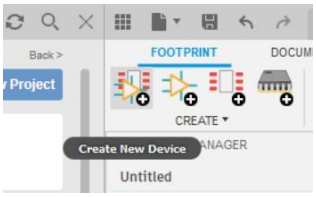


Figura 66. Create New Device

Buscamos nuestro símbolo creado dándole click en **add to part.**

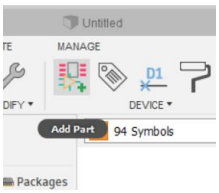


Figura 67. Add to part

Una vez encontrado le damos click en ok y lo ubicamos en nuestra área de trabajo.

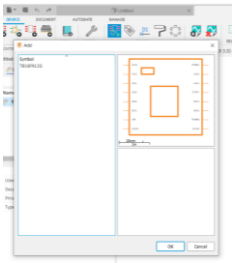


Figura 68. Device creado

Le damos click en New y seleccionamos **add local package** así poder enlazar ambos elementos creados.

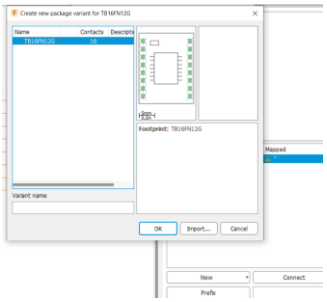


Figura 69. *Footprint creado*

Ahora le damos clic en **connect** y pareamos cada pin con su respectivo pad.

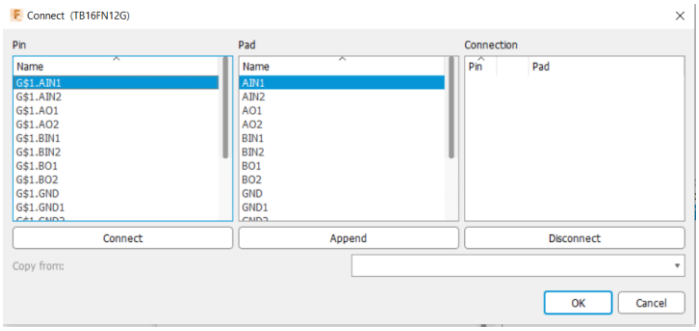


Figura 70. Unir el footprint y el device

Exportamos nuestro dispositivo creado para poder utilizarlo en programa dándole click en **File** y seleccionamos **export,** ubicamos el nombre y la ruta del archivo para exportar.

Paso 4

Nos dirigimos a **File** y le damos click en **New Electronic Design** y le damos click en **New Schematic.**

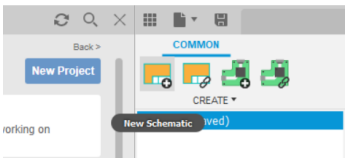


Figura 71. New Schematic

Le damos clic en **add to part** y click en **open library manager** buscaremos nuestro modulo creado y lo ubicamos en nuestra área de trabajo.

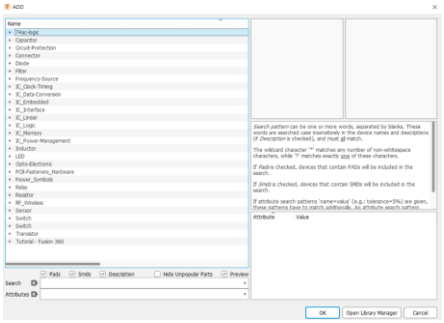


Figura 72. Library Manager

Encontramos nuestra librería creada ahora si podemos usarla.

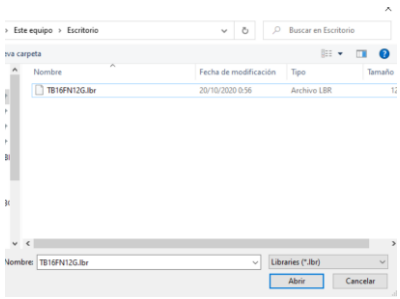


Figura 73. Abrir la librería

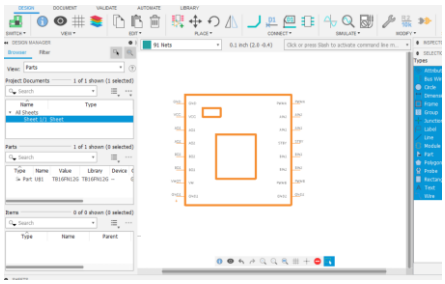


Figura 74. Device en el área de trabajo

Paso 5

Repetimos el Paso 3 para la elaboración de los reguladores y el controlador.

Paso 6

Con la herramienta **Net** procedemos hacer las conexiones de nuestro robot.

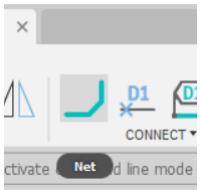


Figura 75. Símbolo Net

Paso 7

Finalmente tenemos listo nuestra PCB.

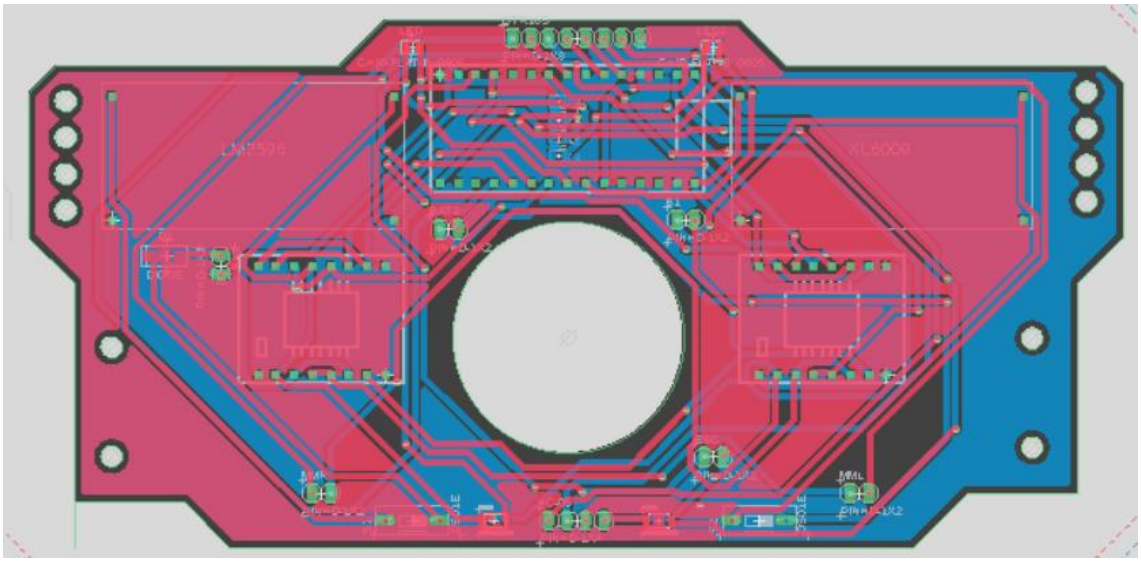


Figura 76. Diseño de la PCB

Anexo 2: PRÁCTICA #2
Paso1

Debemos declarar nuestras variables como entradas y salida

```
//ENTRADA ANALOGICA
AnalogIn An(A0);
//SALIDA PARA EL MUX
DigitalOut SL_0(D2);
DigitalOut SL_1(D3);
DigitalOut SL_2(D4);
DigitalOut SL_3(D5);
```

Paso 2

Debemos revisar la tabla lógica del mux 74hc4067 y realizar la programación para el sensor#1

TRUTH TABLE

S0	S1	S2	S3	$\bar{E}$	SELECTED CHANNEL
X	X	X	X	1	None
0	0	0	0	0	0
1	0	0	0	0	1
0	1	0	0	0	2
1	1	0	0	0	3
0	0	1	0	0	4
1	0	1	0	0	5
0	1	1	0	0	6
1	1	1	0	0	7
0	0	0	1	0	8
1	0	0	1	0	9
0	1	0	1	0	10
1	1	0	1	0	11
0	0	1	1	0	12
1	0	1	1	0	13
0	1	1	1	0	14
1	1	1	1	0	15

H= High Level
L= Low Level
X= Don't Care

```
int main(){
  while(1) {
      SL_0=0;SL_1=0;SL_2=0;SL_3=0; sensores[0] = An.read();
        }
}
```

Paso 3

Ahora realizamos la combinación lógica para los 16 sensores

```
int main(){
  while(1) {
```

```
        SL_0=1;SL_1=0;SL_2=0;SL_3=0; sensores[1] = An.read();
        SL_0=0;SL_1=1;SL_2=0;SL_3=0; sensores[2] = An.read();
        SL_0=1;SL_1=1;SL_2=0;SL_3=0; sensores[3] = An.read();
        SL_0=0;SL_1=0;SL_2=1;SL_3=0; sensores[4] = An.read();
        SL_0=1;SL_1=0;SL_2=1;SL_3=0; sensores[5] = An.read();
        SL_0=0;SL_1=1;SL_2=1;SL_3=0; sensores[6] = An.read();
        SL_0=1;SL_1=1;SL_2=1;SL_3=0; sensores[7] = An.read();
        SL_0=0;SL_1=0;SL_2=0;SL_3=1; sensores[8] = An.read();
        SL_0=1;SL_1=0;SL_2=0;SL_3=1; sensores[9] = An.read();
        SL_0=0;SL_1=1;SL_2=0;SL_3=1; sensores[10] = An.read();
        SL_0=1;SL_1=1;SL_2=0;SL_3=1; sensores[11] = An.read();
        SL_0=0;SL_1=0;SL_2=1;SL_3=1; sensores[12] = An.read();
        SL_0=1;SL_1=0;SL_2=1;SL_3=1; sensores[13] = An.read();
        SL_0=0;SL_1=1;SL_2=1;SL_3=1; sensores[14] = An.read();
        SL_0=1;SL_1=1;SL_2=1;SL_3=1; sensores[15] = An.read();}
          for(int i=0;i<16;i++){
          sensores[i] = An.read();
          printf("%f   ",sensores[i]);
          }
          printf("\n");
}
}
```

Anexo 3: PRÁCTICA #3
Paso 1

Verificar cuales son los pines conectados entre el módulo TB16FN12G y el controlador stm32l432kc.

Paso 2

Ahora realizamos la programación para controlar la velocidad a los motores.

```
//SALIDAS PARA EL TB16FN12G
PwmOut     PWMB(D9);
DigitalOut BIN2(D11);
DigitalOut BIN1(D6);
DigitalOut AIN2(PA_1);
DigitalOut AIN1(PA_4);
PwmOut     PWMA(PA_3);
double speedL=0;
double speedR=0;
int main(){
   while(1) {
   if(speedL>0) {
      AIN1=1;
      AIN2=0;
   } else {
      AIN1=0;
      AIN2=1;
      speedL=speedL*-1;
   }
   if(speedR>0) {
      BIN1 = 1;
      BIN2 = 0;
   } else {
      BIN1 = 0;
      BIN2 = 1;
```

```
        speedR=speedR*-1;
    }
    PWMA.write(speedL);
    PWMB.write(speedR);
    }
}
```

Anexo 4: PRÁCTICA #4

Paso 1

Se debe realizar los pasos de la práctica #1 hasta la práctica #3

Paso 2

Declarar la variable del pulsador como entrada

```
//ENTRADAS DE BOTONES
InterruptIn event(D12);
DigitalIn S1(D13);
```

Paso 3

Utilizaremos la siguiente variable para usar en la calibración

```
float digital[16];
float lectura_fondo[16];
float lectura_linea[16];
float umbral[16];
float sumap, suma, pos, poslast, position;
```

Paso 4

Creamos las nuevas funciones para el proceso de calibracion

```
void fondos(){
    for(int i=0;i<16;i++){
        SL_0=i&0x01;
        SL_1=i&0x02;
        SL_2=i&0x04;
        SL_3=i&0x08;
        lectura_fondo[i] = An.read();
        printf("%f   ",lectura_fondo[i]);
    }
    printf("\n");
}
void lineas(){
    for(int i=0;i<16;i++){
        SL_0=i&0x01;
```

```
        SL_1=i&0x02;
        SL_2=i&0x04;
        SL_3=i&0x08;
        lectura_linea[i] = An.read();
        printf("%f   ",lectura_linea[i]);
    }
    printf("\n");
}
void promedio(){
    for(int i=0;i<16;i++){
        umbral[i]=(lectura_fondo[i]+lectura_linea[i])/2;
        printf("%f   ",umbral[i]);
    }
    printf("\n");
}
```

Paso 5

Usamos la ecuacion de la media ponderada para establecer el rango entre 0 a 1500

$$MP = \frac{p1X1 + p2X2 + \cdots + pnXn}{p1 + p2 + \cdots pn}$$

Siendo(X1, X2, …Xn) el conjunto de datos y (p1, p2, … pn) los pesos

Paso 6

Ahora realizamos el codigo para la ecuacion de la media ponderada

```
float lectura(void){
sumap =(1500*digital[0]+1400*digital[1]+1300*digital[2]+1200*digital[3]+1100*digital[4]+1000*digital[5]+900*digital[6]+800*digital[7]+700*digital[8]+600*digital[9]+500*digital[10]+400*digital[11]+300*digital[12]+200*digital[13]+100*digital[14]+0*digital[15]);
suma =(digital[0]+digital[1]+digital[2]+digital[3]+digital[4]+digital[5]+digital[6]+digital[7]+digital[8]+digital[9]+digital[10]+digital[11]+digital[12]+digital[13]+digital[14]+digital[15]);
pos=(sumap/suma);
```

```
return pos;
}
```

Paso 7

Realizaremos la programacion dentro del main cumpliendo el proceso de calibración

```
int main(){
   led_1=1;
   printf("\nPULSAR S1\n");
   while(!S1);
   for(int i=0;i<iteracion;i++){
      fondos();
      led_1 = !led_1;
   }
   led_1=0;
   led_2=1;
   printf("PULSAR S1\n");
   while(!S1);
   for(int i=0;i<iteracion;i++){
      lineas();
      led_2 = !led_2;
   }
   printf("EL PROMEDIO ES\n");
   promedio();
   printf("PULSAR S1\n");
   while(!S1);
   led_2=0;
   while(1) {
      position = lectura();
      printf("%f\n", position);
   }
}
```

Anexo 5: PRÁCTICA #5

Paso 1

Se debe realizar los pasos de la práctica #1 hasta la práctica #4

Paso 2

Declarar las siguientes variables a utilizar

```
float errorActual = 0;
float errorIntegral = 0;
float errorDerivativo = 0;
float errorPasado = 0;
float Vmax = 0.3;
float Sp =750;
float Kp =0.0003;
float Ki =0.000002;
float Kd =0.04;
```

Paso 4

Creamos las nuevas funciones para el control PID de nuestro robot Line Follower

```
void PIDvar(float kp, float kd, float ki) {
   KP = kp;
   KD = kd;
   KI = ki;}
float PID(int POS, int setpoint, float lim) {
   errorActual = ((int)POS) - setpoint;
   errorIntegral += KI*errorActual;
   errorDerivativo = errorActual - errorPasado;
   errorPasado = errorActual;
   float U = (KP*errorActual) + errorIntegral + (KD*errorDerivativo);
   if (U > lim)
      U = lim;
   else if (U < -lim)
      U = -lim;
   }
```

Paso 5

Realizaremos la programacion dentro del main para llamar a nuestra funcion PID

```
int main(){
   led_1=1;
   BLUETOOTH.printf("\nPULSAR S1\n");
   while(!S1);
   for(int i=0;i<iteracion;i++){
      fondos();
      led_1 = !led_1;
   }
   led_1=0;
   led_2=1;
   BLUETOOTH.printf("PULSAR S1\n");
   while(!S1);
   for(int i=0;i<iteracion;i++){
      lineas();
      led_2 = !led_2;
   }
   BLUETOOTH.printf("EL PROMEDIO ES\n");
   promedio();
   BLUETOOTH.printf("PULSAR S2\n");
   while(!S2);
   while(!S1){
      position = lectura();
      PIDvar(Kp, Kd, Ki);
      float u = PID(position, Sp, Vmax);
      if (position==750 && u<=0.05f){u=0;}
      BLUETOOTH.printf("%1.0f %1.3f\n", position,u);
      motors(Vmax - Power, Vmax + Powe);
   }}
```

Anexo 6: PRÁCTICA #6

Paso 1

Se debe realizar los pasos de la práctica #1 hasta la práctica #4

Paso 2

Declarar las siguientes variables a utilizar

```
float Vmax = 70;
float Sp =750;
float Error[7] = {0,0,0,0,0,0,0};        // Error vector
float ErrorI[1] ={0};
float ErrorD[1] ={0};
float ErrorL[1] ={0};
float u[2] = {0,0};              // Control vector
float w[3] = {0.25,0.001,5.5};          // NN weights
```

Paso 3

Creamos las nuevas funciones para el control PID Neuronal de nuestro robot Line Follower

```
float fmap( float x, float in_min, float in_max, float out_min, float out_max)
{
   return (x - in_min) * (out_max- out_min) / (in_max - in_min) + out_min;
}

int sat(int s, int lb, int ub)      // Saturation function
{
   if (s >= ub) return ub;
   if (s <= lb) return lb;
   return s;
}

int pidnn(int e0, int e1, int e2, int u1)
{
 const float eta = 2e-9;              // Learning factor
```

```
  int u = eta + w[0]*e0 + w[1]*e1 + w[2]*e2;
  w[0] = e0 * e0 * eta + w[0];
  w[1] = e0 * e2 * eta + w[1];
  w[2] = e0 * e1 * eta/10 + w[2];
  return sat(u, -255, 255);
}

void pid() {
  Error[0] = position - Sp;       // Error calculation
  ErrorI[0] = Error[1]+Error[2]+Error[3]+Error[4]+Error[5]+Error[6];
  ErrorD[0] = Error[0] - ErrorL[0];              // Shifting operation
  ErrorL[0] = Error[0];
  Error[6] = Error[5];
  Error[5] = Error[4];
  Error[4] = Error[3];
  Error[3] = Error[2];
  Error[2] = Error[1];
  Error[1] = Error[0];
  u[0] = pidnn(Error[0], ErrorI[0], ErrorD[0], u[1]);
  u[1] = u[0];
  u[0] = fmap(u[0],-255,255,-1,1);
  if ( u[0] > Vmax ) u[0] = Vmax;
  else if ( u[0] < -Vmax ) u[0] = -Vmax;
  ( u[0] < 0 ) ?
  motors(Vmax + u[0], Vmax) : motors(Vmax, Vmax - u[0]);
}
```

Paso 4

Realizaremos la programacion dentro del main que nos permitira ejecutar las funciones creadas

```
int main(){
   Vmax = fmap(Vmax,0,255,0,1.0);
```

```
BLUETOOTH.baud(9600);
BLUETOOTH.format(8,Serial::None,1);
led_1=1;
BLUETOOTH.printf("%f\n", Vmax);
BLUETOOTH.printf("\nPULSAR S1\n");
while(!S1);
fondos(iteracion);
BLUETOOTH.printf("\nPULSAR S1\n");
while(!S1);
lineas(iteracion);
BLUETOOTH.printf("\nEL PROMEDIO ES\n");
promedio();
BLUETOOTH.printf("\nPULSAR S2");
while(!S2);
while(!S1){
    position = lectura();
    pid();
    cleanBuffer(bufferenvio,100);
    char buf1[10];
    char buf2[10];
    char buf3[10];
    char buf4[10];
    char buf5[10];
    char buf6[10];
    sprintf(buf1,"%1.0f",position);
    sprintf(buf2,"%1.0f",Sp);
    sprintf(buf3,"%1.5f",u[0]);
    sprintf(buf4,"%1.5f",w[0]);
    sprintf(buf5,"%1.5f",w[1]);
    sprintf(buf6,"%1.5f",w[2]);
```

```
        strcat (bufferenvio,buf1);
        strcat (bufferenvio,";");
        strcat (bufferenvio,buf2);
        strcat (bufferenvio,";");
        strcat (bufferenvio,buf3);
        strcat (bufferenvio,";");
        strcat (bufferenvio,buf4);
        strcat (bufferenvio,";");
        strcat (bufferenvio,buf5);
        strcat (bufferenvio,";");
        strcat (bufferenvio,buf6);
        strcat (bufferenvio,";");
        BLUETOOTH.printf("%s\r\n", bufferenvio);
    }
    wait(1);
    while(1) {
        motors(0,0);
    }
}
```

Anexo 7: PRÁCTICA #7

Paso 1

Declarar los pines de entrada y salida a usarse

```
DigitalIn S2(D12);
DigitalIn S1(D13);
DigitalOut led_1(PA_5); //A4
```

Paso 2

Declarar las siguientes variables a utilizar

```
int p1,p2;
float n1,n2,n3;
float a1,a2;
float e=2.7192818;
```

Paso 3

Crear las funciones de activación de la salida de la red neuronal

```
float tansig(float x){
 float a;
 a=(pow(e,x)-pow(e,-x))/(pow(e,x)+pow(e,-x));
 return a;
}
```

Paso 4

Mediante Matlab obtener los pesos y bias de la red entrenada para ser usada en la Stm32

```
clc,clear all
p= [0 0 1 1;
   0 1 0 1];
t=[0 1 1 0];
net=newff(minmax(p),[2 1],{'tansig','purelin'},'traincgf');
net=init(net);
net.trainParam.epochs=500;
net.trainparam.goal=1e-6;
net=train(net,p,t);
```

```
a=sim(net,p)
pesos1=net.iw{1,1}
bias1=net.b{1,1}
pesos2=net.lw{2,1}
bias2=net.b{2,1}
```

Paso 5

Realizaremos la programacion dentro del main que nos permitira ejecutar las funciones creadas

```
int main()
{
   while (1) {
      p1= S1.read();
      p2= S2.read();
      printf("%d ",p1);
      printf("%d ",p2);

      n1=p1*(-2.3879)+p2*(-3.5129)+(4.4905);
      n2=p1*(1.5427)+p2*(3.8616)+(-0.2902);
      a1=tansig(n1);
      a2=tansig(n2);
      n3=a1*(0.6100)+a2*(0.8978)+(-0.3553);
      led_1.write(n3);
      printf("%f\n",n3);
      wait(1);
   }
}
```

Anexo 8: PRÁCTICA #8

Paso 1

Se debe realizar los pasos de la práctica #1 hasta la práctica #4

Paso 2

Declarar las siguientes variables a utilizar

```
float errorActual = 0;
float errorIntegral = 0;
float errorDerivativo = 0;
float errorPasado = 0;
float Vmax = 180;
float Sp =750;
float KP =0.0;
float KI =0.001;
float KD =0.0;
float error1 = 0;
float error2 = 0;
float error3 = 0;
float error4 = 0;
float error5 = 0;
float error6 = 0;
char bufferenvio[100];
Fuzzy *fuzzy = new Fuzzy();
```

Paso 3

Creamos las nuevas funciones para el control PD Difuso de nuestro robot Line Follower

```
void pid() {
  errorActual = (position) - Sp;
  errorDerivativo = errorActual - errorPasado;
  errorIntegral = error1 + error2 + error3 + error4 + error5 + error6;
  errorPasado = errorActual;
  error6 = error5;
  error5 = error4;
```

```
  error4 = error3;
  error3 = error2;
  error2 = error1;
  error1 = errorActual;
  u = ( errorActual * KP ) + ( errorDerivativo * KD ) + (errorIntegral * KI) ;
  u = fmap(u,-255,255,-1.0,1.0);
  if ( u > Vmax ) u = Vmax;
  else if ( u < -Vmax ) u = -Vmax;
  ( u < 0 ) ?
  motors(Vmax + u, Vmax) : motors(Vmax, Vmax - u);
}

void FLC()
{
    FuzzyInput *e = new FuzzyInput(1);
    FuzzySet *eNG = new FuzzySet( -2, -1, -0.8, -0.4);
    e->addFuzzySet(eNG);
    FuzzySet *eNP = new FuzzySet( -0.8, -0.4, -0.4, 0);
    e->addFuzzySet(eNP);
    FuzzySet *eZ = new FuzzySet( -0.4, 0, 0, 0.4);
    e->addFuzzySet(eZ);
    FuzzySet *ePP = new FuzzySet( 0, 0.4, 0.4, 0.8);
    e->addFuzzySet(ePP);
    FuzzySet *ePG = new FuzzySet( 0.4, 0.8, 1, 2);
    e->addFuzzySet(ePG);
    fuzzy->addFuzzyInput(e);

    FuzzyInput *de = new FuzzyInput(2);
    FuzzySet *deNG = new FuzzySet( -2, -1, -0.8, -0.4);
    de->addFuzzySet(deNG);
```

```
FuzzySet *deNP = new FuzzySet( -0.8, -0.4, -0.4, 0);
de->addFuzzySet(deNP);
FuzzySet *deZ = new FuzzySet( -0.4, 0, 0, 0.4);
de->addFuzzySet(deZ);
FuzzySet *dePP = new FuzzySet( 0, 0.4, 0.4, 0.8);
de->addFuzzySet(dePP);
FuzzySet *dePG = new FuzzySet( 0.4, 0.8, 1, 2);
de->addFuzzySet(dePG);
fuzzy->addFuzzyInput(de);

FuzzyOutput *u = new FuzzyOutput(1);
FuzzySet *uNG = new FuzzySet( -2, -1, -0.8, -0.4);
u->addFuzzySet(uNG);
FuzzySet *uNP = new FuzzySet( -0.8, -0.4, -0.4, 0);
u->addFuzzySet(uNP);
FuzzySet *uZ = new FuzzySet( -0.4, 0, 0, 0.4);
u->addFuzzySet(uZ);
FuzzySet *uPP = new FuzzySet( 0, 0.4, 0.4, 0.8);
u->addFuzzySet(uPP);
FuzzySet *uPG = new FuzzySet( 0.4, 0.8, 1, 2);
u->addFuzzySet(uPG);
fuzzy->addFuzzyOutput(u);

FuzzyOutput *u2 = new FuzzyOutput(2);
FuzzySet *u2NG = new FuzzySet( -2, -1, -0.8, -0.4);
u2->addFuzzySet(u2NG);
FuzzySet *u2NP = new FuzzySet( -0.8, -0.4, -0.4, 0);
u2->addFuzzySet(u2NP);
FuzzySet *u2Z = new FuzzySet( -0.4, 0, 0, 0.4);
u2->addFuzzySet(u2Z);
```

```
    FuzzySet *u2PP = new FuzzySet( 0, 0.4, 0.4, 0.8);
    u2->addFuzzySet(u2PP);
    FuzzySet *u2PG = new FuzzySet( 0.4, 0.8, 1, 2);
    u2->addFuzzySet(u2PG);
    fuzzy->addFuzzyOutput(u2);
////////////////////////////////////reglas para encontrar Kp-Kd////////////////////////////////////////////
// ------------------------------------ Rules (1 - 5) ------------------------------------------
    // Building FuzzyRule "IF error = big negative and derror = big negative THEN control = big negative"
    FuzzyRuleAntecedent *ifeNGAnddeNG = new FuzzyRuleAntecedent(); // Instantiating a FuzzyRuleAntecedent objects
    ifeNGAnddeNG->joinWithAND(eNG,deNG); // Creating a FuzzyRuleAntecedent with just a single FuzzySet
    FuzzyRuleConsequent *thenuNG1 = new FuzzyRuleConsequent(); // Instantiating a FuzzyRuleConsequent objects
    thenuNG1->addOutput(uNG); // Including a FuzzySet to this FuzzyRuleConsequent
    FuzzyRule *fuzzyRule01 = new FuzzyRule(1, ifeNGAnddeNG, thenuNG1); // Instantiating a FuzzyRule objects
    fuzzy->addFuzzyRule(fuzzyRule01); // Including the FuzzyRule into Fuzzy

    // Building FuzzyRule "IF error = big negative and derror = small negative THEN control = big negative"
    FuzzyRuleAntecedent *ifeNGAnddeNP = new FuzzyRuleAntecedent(); // Instantiating a FuzzyRuleAntecedent objects
    ifeNGAnddeNP->joinWithAND(eNG,deNP); // Creating a FuzzyRuleAntecedent with just a single FuzzySet
    FuzzyRuleConsequent *thenuNG2 = new FuzzyRuleConsequent(); // Instantiating a FuzzyRuleConsequent objects
    thenuNG2->addOutput(uNG); // Including a FuzzySet to this FuzzyRuleConsequent
    FuzzyRule *fuzzyRule02 = new FuzzyRule(2, ifeNGAnddeNP, thenuNG2); // Instantiating a FuzzyRule objects
    fuzzy->addFuzzyRule(fuzzyRule02); // Including the FuzzyRule into Fuzzy
```

```
    // Building FuzzyRule "IF error = big negative and derror = zero THEN control = small negative"

    FuzzyRuleAntecedent *ifeNGAnddeZ = new FuzzyRuleAntecedent(); // Instantiating a FuzzyRuleAntecedent objects

    ifeNGAnddeZ->joinWithAND(eNG,deZ); // Creating a FuzzyRuleAntecedent with just a single FuzzySet

    FuzzyRuleConsequent *thenuNP1 = new FuzzyRuleConsequent(); // Instantiating a FuzzyRuleConsequent objects

    thenuNP1->addOutput(uNP); // Including a FuzzySet to this FuzzyRuleConsequent

    FuzzyRule *fuzzyRule03 = new FuzzyRule(3, ifeNGAnddeZ, thenuNP1); // Instantiating a FuzzyRule objects

    fuzzy->addFuzzyRule(fuzzyRule03); // Including the FuzzyRule into Fuzzy


    // Building FuzzyRule "IF error = big negative and derror = small positive THEN control = small negative"

    FuzzyRuleAntecedent *ifeNGAnddePP = new FuzzyRuleAntecedent(); // Instantiating a FuzzyRuleAntecedent objects

    ifeNGAnddePP->joinWithAND(eNG,dePP); // Creating a FuzzyRuleAntecedent with just a single FuzzySet

    FuzzyRuleConsequent *thenuNP2 = new FuzzyRuleConsequent(); // Instantiating a FuzzyRuleConsequent objects

    thenuNP2->addOutput(uNP); // Including a FuzzySet to this FuzzyRuleConsequent

    FuzzyRule *fuzzyRule04 = new FuzzyRule(4, ifeNGAnddePP, thenuNP2); // Instantiating a FuzzyRule objects

    fuzzy->addFuzzyRule(fuzzyRule04); // Including the FuzzyRule into Fuzzy


    // Building FuzzyRule "IF error = big negative and derror = big positive THEN control = zero"

    FuzzyRuleAntecedent *ifeNGAnddePG = new FuzzyRuleAntecedent(); // Instantiating a FuzzyRuleAntecedent objects

    ifeNGAnddePG->joinWithAND(eNG,dePG); // Creating a FuzzyRuleAntecedent with just a single FuzzySet

    FuzzyRuleConsequent *thenuZ1 = new FuzzyRuleConsequent(); // Instantiating a FuzzyRuleConsequent objects

    thenuZ1->addOutput(uZ); // Including a FuzzySet to this FuzzyRuleConsequent
```

```
    FuzzyRule *fuzzyRule05 = new FuzzyRule(5, ifeNGAnddePG, thenuZ1); // Instantiating a FuzzyRule objects

    fuzzy->addFuzzyRule(fuzzyRule05); // Including the FuzzyRule into Fuzzy


    // ----------------------------------- Rules (6 - 10) ----------------------------------------

    // Building FuzzyRule "IF error = small negative and derror = big negative THEN control = big negative"

    FuzzyRuleAntecedent *ifeNPAnddeNG = new FuzzyRuleAntecedent(); // Instantiating a FuzzyRuleAntecedent objects

    ifeNPAnddeNG->joinWithAND(eNP,deNG); // Creating a FuzzyRuleAntecedent with just a single FuzzySet

    FuzzyRuleConsequent *thenuNG6 = new FuzzyRuleConsequent(); // Instantiating a FuzzyRuleConsequent objects

    thenuNG6->addOutput(uNG); // Including a FuzzySet to this FuzzyRuleConsequent

    FuzzyRule *fuzzyRule06 = new FuzzyRule(6, ifeNPAnddeNG, thenuNG6); // Instantiating a FuzzyRule objects

    fuzzy->addFuzzyRule(fuzzyRule06); // Including the FuzzyRule into Fuzzy


    // Building FuzzyRule "IF error = small negative and derror = small negative THEN control = small negative"

    FuzzyRuleAntecedent *ifeNPAnddeNP = new FuzzyRuleAntecedent(); // Instantiating a FuzzyRuleAntecedent objects

    ifeNPAnddeNP->joinWithAND(eNP,deNP); // Creating a FuzzyRuleAntecedent with just a single FuzzySet

    FuzzyRuleConsequent *thenuNP7 = new FuzzyRuleConsequent(); // Instantiating a FuzzyRuleConsequent objects

    thenuNP7->addOutput(uNP); // Including a FuzzySet to this FuzzyRuleConsequent

    FuzzyRule *fuzzyRule07 = new FuzzyRule(7, ifeNPAnddeNP, thenuNP7); // Instantiating a FuzzyRule objects

    fuzzy->addFuzzyRule(fuzzyRule07); // Including the FuzzyRule into Fuzzy


    // Building FuzzyRule "IF error = small negative and derror = zero THEN control = small negative"
```

```
    FuzzyRuleAntecedent *ifeNPAnddeZ = new FuzzyRuleAntecedent(); // Instantiating a FuzzyRuleAntecedent objects

    ifeNPAnddeZ->joinWithAND(eNP,deZ); // Creating a FuzzyRuleAntecedent with just a single FuzzySet

    FuzzyRuleConsequent *thenuNP8 = new FuzzyRuleConsequent(); // Instantiating a FuzzyRuleConsequent objects

    thenuNP8->addOutput(uNP); // Including a FuzzySet to this FuzzyRuleConsequent

    FuzzyRule *fuzzyRule08 = new FuzzyRule(8, ifeNPAnddeZ, thenuNP8); // Instantiating a FuzzyRule objects

    fuzzy->addFuzzyRule(fuzzyRule08); // Including the FuzzyRule into Fuzzy


    // Building FuzzyRule "IF error = small negative and derror = small positive THEN control = zero"

    FuzzyRuleAntecedent *ifeNPAnddePP = new FuzzyRuleAntecedent(); // Instantiating a FuzzyRuleAntecedent objects

    ifeNPAnddePP->joinWithAND(eNP,dePP); // Creating a FuzzyRuleAntecedent with just a single FuzzySet

    FuzzyRuleConsequent *thenuZ9 = new FuzzyRuleConsequent(); // Instantiating a FuzzyRuleConsequent objects

    thenuZ9->addOutput(uZ); // Including a FuzzySet to this FuzzyRuleConsequent

    FuzzyRule *fuzzyRule09 = new FuzzyRule(9, ifeNPAnddePP, thenuZ9); // Instantiating a FuzzyRule objects

    fuzzy->addFuzzyRule(fuzzyRule09); // Including the FuzzyRule into Fuzzy


    // Building FuzzyRule "IF error = small negative and derror = big positive THEN control = small positive"

    FuzzyRuleAntecedent *ifeNPAnddePG = new FuzzyRuleAntecedent(); // Instantiating a FuzzyRuleAntecedent objects

    ifeNPAnddePG->joinWithAND(eNP,dePG); // Creating a FuzzyRuleAntecedent with just a single FuzzySet

    FuzzyRuleConsequent *thenuPP10 = new FuzzyRuleConsequent(); // Instantiating a FuzzyRuleConsequent objects

    thenuPP10->addOutput(uPP); // Including a FuzzySet to this FuzzyRuleConsequent

    FuzzyRule *fuzzyRule10 = new FuzzyRule(10, ifeNPAnddePG, thenuPP10); // Instantiating a FuzzyRule objects
```

```
    fuzzy->addFuzzyRule(fuzzyRule10); // Including the FuzzyRule into Fuzzy


//-------------------------------------------------------REGLA 11-15-----------------------------
-----------------------

// Building FuzzyRule "IF error = small negative and derror = big negative THEN control
= big negative"

    FuzzyRuleAntecedent *ifeZAnddeNG = new FuzzyRuleAntecedent(); // Instantiating
a FuzzyRuleAntecedent objects

    ifeZAnddeNG->joinWithAND(eZ,deNG); // Creating a FuzzyRuleAntecedent with
just a single FuzzySet

    FuzzyRuleConsequent *thenuNP11= new FuzzyRuleConsequent(); // Instantiating a
FuzzyRuleConsequent objects

    thenuNP11->addOutput(uNP); // Including a FuzzySet to this FuzzyRuleConsequent

    FuzzyRule *fuzzyRule11 = new FuzzyRule(11, ifeZAnddeNG, thenuNP11); //
Instantiating a FuzzyRule objects

    fuzzy->addFuzzyRule(fuzzyRule11); // Including the FuzzyRule into Fuzzy


    // Building FuzzyRule "IF error = small negative and derror = big negative THEN
control = big negative"

    FuzzyRuleAntecedent *ifeZAnddeNP = new FuzzyRuleAntecedent(); // Instantiating
a FuzzyRuleAntecedent objects

    ifeZAnddeNP->joinWithAND(eZ,deNP); // Creating a FuzzyRuleAntecedent with just
a single FuzzySet

    FuzzyRuleConsequent *thenuNP12= new FuzzyRuleConsequent(); // Instantiating a
FuzzyRuleConsequent objects

    thenuNP12->addOutput(uNP); // Including a FuzzySet to this FuzzyRuleConsequent

    FuzzyRule *fuzzyRule12 = new FuzzyRule(12, ifeZAnddeNP, thenuNP12); //
Instantiating a FuzzyRule objects

    fuzzy->addFuzzyRule(fuzzyRule12); // Including the FuzzyRule into Fuzzy


    // Building FuzzyRule "IF error = small negative and derror = big negative THEN
control = big negative"

    FuzzyRuleAntecedent *ifeZAnddeZ = new FuzzyRuleAntecedent(); // Instantiating a
FuzzyRuleAntecedent objects
```

```
    ifeZAnddeZ->joinWithAND(eZ,deZ); // Creating a FuzzyRuleAntecedent with just a single FuzzySet

    FuzzyRuleConsequent *thenuZ13= new FuzzyRuleConsequent(); // Instantiating a FuzzyRuleConsequent objects

    thenuZ13->addOutput(uZ); // Including a FuzzySet to this FuzzyRuleConsequent

    FuzzyRule *fuzzyRule13 = new FuzzyRule(13, ifeZAnddeZ, thenuZ13); // Instantiating a FuzzyRule objects

    fuzzy->addFuzzyRule(fuzzyRule13); // Including the FuzzyRule into Fuzzy


    // Building FuzzyRule "IF error = small negative and derror = big negative THEN control = big negative"

    FuzzyRuleAntecedent *ifeZAnddePP = new FuzzyRuleAntecedent(); // Instantiating a FuzzyRuleAntecedent objects

    ifeZAnddePP->joinWithAND(eZ,dePP); // Creating a FuzzyRuleAntecedent with just a single FuzzySet

    FuzzyRuleConsequent *thenuPP14= new FuzzyRuleConsequent(); // Instantiating a FuzzyRuleConsequent objects

    thenuPP14->addOutput(uPP); // Including a FuzzySet to this FuzzyRuleConsequent

    FuzzyRule *fuzzyRule14 = new FuzzyRule(14, ifeZAnddePP, thenuPP14); // Instantiating a FuzzyRule objects

    fuzzy->addFuzzyRule(fuzzyRule14); // Including the FuzzyRule into Fuzzy


    // Building FuzzyRule "IF error = small negative and derror = big negative THEN control = big negative"

    FuzzyRuleAntecedent *ifeZAnddePG = new FuzzyRuleAntecedent(); // Instantiating a FuzzyRuleAntecedent objects

    ifeZAnddePG->joinWithAND(eZ,dePG); // Creating a FuzzyRuleAntecedent with just a single FuzzySet

    FuzzyRuleConsequent *thenuPP15= new FuzzyRuleConsequent(); // Instantiating a FuzzyRuleConsequent objects

    thenuPP15->addOutput(uPP); // Including a FuzzySet to this FuzzyRuleConsequent

    FuzzyRule *fuzzyRule15 = new FuzzyRule(15, ifeZAnddePG, thenuPP15); // Instantiating a FuzzyRule objects

    fuzzy->addFuzzyRule(fuzzyRule15); // Including the FuzzyRule into Fuzzy
```

```
//-------------------------------------------------------------REGLA 16-20----------------------

// Building FuzzyRule "IF error = small negative and derror = big negative THEN control = big negative"

    FuzzyRuleAntecedent *ifePPAnddeNG = new FuzzyRuleAntecedent(); // Instantiating a FuzzyRuleAntecedent objects

    ifePPAnddeNG->joinWithAND(ePP,deNG); // Creating a FuzzyRuleAntecedent with just a single FuzzySet

    FuzzyRuleConsequent *thenuNP16= new FuzzyRuleConsequent(); // Instantiating a FuzzyRuleConsequent objects

    thenuNP16->addOutput(uNP); // Including a FuzzySet to this FuzzyRuleConsequent

    FuzzyRule *fuzzyRule16 = new FuzzyRule(16, ifePPAnddeNG, thenuNP16); // Instantiating a FuzzyRule objects

    fuzzy->addFuzzyRule(fuzzyRule16); // Including the FuzzyRule into Fuzzy


    // Building FuzzyRule "IF error = small negative and derror = big negative THEN control = big negative"

    FuzzyRuleAntecedent *ifePPAnddeNP = new FuzzyRuleAntecedent(); // Instantiating a FuzzyRuleAntecedent objects

    ifePPAnddeNP->joinWithAND(ePP,deNP); // Creating a FuzzyRuleAntecedent with just a single FuzzySet

    FuzzyRuleConsequent *thenuZ17= new FuzzyRuleConsequent(); // Instantiating a FuzzyRuleConsequent objects

    thenuZ17->addOutput(uZ); // Including a FuzzySet to this FuzzyRuleConsequent

    FuzzyRule *fuzzyRule17 = new FuzzyRule(17, ifePPAnddeNP, thenuZ17); // Instantiating a FuzzyRule objects

    fuzzy->addFuzzyRule(fuzzyRule17); // Including the FuzzyRule into Fuzzy


    // Building FuzzyRule "IF error = small negative and derror = big negative THEN control = big negative"

    FuzzyRuleAntecedent *ifePPAnddeZ = new FuzzyRuleAntecedent(); // Instantiating a FuzzyRuleAntecedent objects

    ifePPAnddeZ->joinWithAND(ePP,deZ); // Creating a FuzzyRuleAntecedent with just a single FuzzySet

    FuzzyRuleConsequent *thenuPP18= new FuzzyRuleConsequent(); // Instantiating a FuzzyRuleConsequent objects
```

```
    thenuPP18->addOutput(uPP); // Including a FuzzySet to this FuzzyRuleConsequent

    FuzzyRule *fuzzyRule18 = new FuzzyRule(18, ifePPAnddeZ, thenuPP18); // Instantiating a FuzzyRule objects

    fuzzy->addFuzzyRule(fuzzyRule18); // Including the FuzzyRule into Fuzzy


    // Building FuzzyRule "IF error = small negative and derror = big negative THEN control = big negative"

    FuzzyRuleAntecedent *ifePPAnddePP = new FuzzyRuleAntecedent(); // Instantiating a FuzzyRuleAntecedent objects

    ifePPAnddePP->joinWithAND(ePP,dePP); // Creating a FuzzyRuleAntecedent with just a single FuzzySet

    FuzzyRuleConsequent *thenuPP19= new FuzzyRuleConsequent(); // Instantiating a FuzzyRuleConsequent objects

    thenuPP19->addOutput(uPP); // Including a FuzzySet to this FuzzyRuleConsequent

    FuzzyRule *fuzzyRule19 = new FuzzyRule(19, ifePPAnddePP, thenuPP19); // Instantiating a FuzzyRule objects

    fuzzy->addFuzzyRule(fuzzyRule19); // Including the FuzzyRule into Fuzzy


    // Building FuzzyRule "IF error = small negative and derror = big negative THEN control = big negative"

    FuzzyRuleAntecedent *ifePPAnddePG = new FuzzyRuleAntecedent(); // Instantiating a FuzzyRuleAntecedent objects

    ifePPAnddePG->joinWithAND(ePP,dePG); // Creating a FuzzyRuleAntecedent with just a single FuzzySet

    FuzzyRuleConsequent *thenuPG20= new FuzzyRuleConsequent(); // Instantiating a FuzzyRuleConsequent objects

    thenuPG20->addOutput(uPG); // Including a FuzzySet to this FuzzyRuleConsequent

    FuzzyRule *fuzzyRule20 = new FuzzyRule(20, ifePPAnddePG, thenuPG20); // Instantiating a FuzzyRule objects

    fuzzy->addFuzzyRule(fuzzyRule20); // Including the FuzzyRule into Fuzzy

//-----------------------------------------------------REGLA 20-25-----------------------------

    // Building FuzzyRule "IF error = small negative and derror = big negative THEN control = big negative"

    FuzzyRuleAntecedent *ifePGAnddeNG = new FuzzyRuleAntecedent(); // Instantiating a FuzzyRuleAntecedent objects
```

```
ifePGAnddeNG->joinWithAND(ePG,deNG); // Creating a FuzzyRuleAntecedent with just a single FuzzySet

FuzzyRuleConsequent *thenuZ21= new FuzzyRuleConsequent(); // Instantiating a FuzzyRuleConsequent objects

thenuZ21->addOutput(uZ); // Including a FuzzySet to this FuzzyRuleConsequent

FuzzyRule *fuzzyRule21 = new FuzzyRule(21, ifePGAnddeNG, thenuZ21); // Instantiating a FuzzyRule objects

fuzzy->addFuzzyRule(fuzzyRule21); // Including the FuzzyRule into Fuzzy


FuzzyRuleAntecedent *ifePGAnddeNP = new FuzzyRuleAntecedent(); // Instantiating a FuzzyRuleAntecedent objects

ifePGAnddeNP->joinWithAND(ePG,deNP); // Creating a FuzzyRuleAntecedent with just a single FuzzySet

FuzzyRuleConsequent *thenuPP22= new FuzzyRuleConsequent(); // Instantiating a FuzzyRuleConsequent objects

thenuPP22->addOutput(uPP); // Including a FuzzySet to this FuzzyRuleConsequent

FuzzyRule *fuzzyRule22 = new FuzzyRule(22, ifePGAnddeNG, thenuPP22); // Instantiating a FuzzyRule objects

fuzzy->addFuzzyRule(fuzzyRule22); // Including the FuzzyRule into Fuzzy


FuzzyRuleAntecedent *ifePGAnddeZ = new FuzzyRuleAntecedent(); // Instantiating a FuzzyRuleAntecedent objects

ifePGAnddeZ->joinWithAND(ePG,deZ); // Creating a FuzzyRuleAntecedent with just a single FuzzySet

FuzzyRuleConsequent *thenuPP23= new FuzzyRuleConsequent(); // Instantiating a FuzzyRuleConsequent objects

thenuPP23->addOutput(uPP); // Including a FuzzySet to this FuzzyRuleConsequent

FuzzyRule *fuzzyRule23 = new FuzzyRule(23, ifePGAnddeZ, thenuPP23); // Instantiating a FuzzyRule objects

fuzzy->addFuzzyRule(fuzzyRule23); // Including the FuzzyRule into Fuzzy


FuzzyRuleAntecedent *ifePGAnddePP = new FuzzyRuleAntecedent(); // Instantiating a FuzzyRuleAntecedent objects

ifePGAnddePP->joinWithAND(ePG,dePP); // Creating a FuzzyRuleAntecedent with just a single FuzzySet
```

```
    FuzzyRuleConsequent *thenuPG24= new FuzzyRuleConsequent(); // Instantiating a FuzzyRuleConsequent objects

    thenuPG24->addOutput(uPG); // Including a FuzzySet to this FuzzyRuleConsequent

    FuzzyRule *fuzzyRule24 = new FuzzyRule(24, ifePGAnddePP, thenuPG24); // Instantiating a FuzzyRule objects

    fuzzy->addFuzzyRule(fuzzyRule24); // Including the FuzzyRule into Fuzzy


    FuzzyRuleAntecedent *ifePGAnddePG = new FuzzyRuleAntecedent(); // Instantiating a FuzzyRuleAntecedent objects

    ifePGAnddePG->joinWithAND(ePG,dePG); // Creating a FuzzyRuleAntecedent with just a single FuzzySet

    FuzzyRuleConsequent *thenuPG25= new FuzzyRuleConsequent(); // Instantiating a FuzzyRuleConsequent objects

    thenuPG25->addOutput(uPG); // Including a FuzzySet to this FuzzyRuleConsequent

    FuzzyRule *fuzzyRule25 = new FuzzyRule(25, ifePGAnddePG, thenuPG25); // Instantiating a FuzzyRule objects

    fuzzy->addFuzzyRule(fuzzyRule25); // Including the FuzzyRule into Fuzzy

//-----------------------------------------------------REGLA 25-30-------------------------------------------

    // Building FuzzyRule "IF error = big negative THEN control2 = big negative"

    FuzzyRuleAntecedent *ifeNG = new FuzzyRuleAntecedent(); // Instantiating a FuzzyRuleAntecedent objects

    FuzzyRuleConsequent *thenu2NG26 = new FuzzyRuleConsequent(); // Instantiating a FuzzyRuleConsequent objects

    thenu2NG26->addOutput(u2NG); // Including a FuzzySet to this FuzzyRuleConsequent

    FuzzyRule *fuzzyRule26 = new FuzzyRule(26, ifeNG, thenu2NG26); // Instantiating a FuzzyRule objects

    fuzzy->addFuzzyRule(fuzzyRule26); // Including the FuzzyRule into Fuzzy


    // Building FuzzyRule "IF error = small negative THEN control2 = small negative"

    FuzzyRuleAntecedent *ifeNP = new FuzzyRuleAntecedent(); // Instantiating a FuzzyRuleAntecedent objects

    FuzzyRuleConsequent *thenu2NP27 = new FuzzyRuleConsequent(); // Instantiating a FuzzyRuleConsequent objects
```

```
    thenu2NP27->addOutput(u2NP); // Including a FuzzySet to this FuzzyRuleConsequent

    FuzzyRule *fuzzyRule27 = new FuzzyRule(27, ifeNP, thenu2NP27); // Instantiating a FuzzyRule objects

    fuzzy->addFuzzyRule(fuzzyRule27); // Including the FuzzyRule into Fuzzy


// Building FuzzyRule "IF error = zero THEN control2 = zero"

    FuzzyRuleAntecedent *ifeZ = new FuzzyRuleAntecedent(); // Instantiating a FuzzyRuleAntecedent objects

    FuzzyRuleConsequent *thenu2Z28 = new FuzzyRuleConsequent(); // Instantiating a FuzzyRuleConsequent objects

    thenu2Z28->addOutput(u2Z); // Including a FuzzySet to this FuzzyRuleConsequent

    FuzzyRule *fuzzyRule28 = new FuzzyRule(28, ifeZ, thenu2Z28); // Instantiating a FuzzyRule objects

    fuzzy->addFuzzyRule(fuzzyRule28); // Including the FuzzyRule into Fuzzy


// Building FuzzyRule "IF error = small positivo THEN control2 = small positivo"

    FuzzyRuleAntecedent *ifePP = new FuzzyRuleAntecedent(); // Instantiating a FuzzyRuleAntecedent objects

    FuzzyRuleConsequent *thenu2PP29 = new FuzzyRuleConsequent(); // Instantiating a FuzzyRuleConsequent objects

    thenu2PP29->addOutput(u2PP); // Including a FuzzySet to this FuzzyRuleConsequent

    FuzzyRule *fuzzyRule29 = new FuzzyRule(29, ifePP, thenu2PP29); // Instantiating a FuzzyRule objects

    fuzzy->addFuzzyRule(fuzzyRule29); // Including the FuzzyRule into Fuzzy


// Building FuzzyRule "IF error = big positivo THEN control2 = big positivo"

    FuzzyRuleAntecedent *ifePG = new FuzzyRuleAntecedent(); // Instantiating a FuzzyRuleAntecedent objects

    FuzzyRuleConsequent *thenu2PG30 = new FuzzyRuleConsequent(); // Instantiating a FuzzyRuleConsequent objects

    thenu2PG30->addOutput(u2PG); // Including a FuzzySet to this FuzzyRuleConsequent

    FuzzyRule *fuzzyRule30 = new FuzzyRule(30, ifePG, thenu2PG30); // Instantiating a FuzzyRule objects
```

```
    fuzzy->addFuzzyRule(fuzzyRule30); // Including the FuzzyRule into Fuzzy
//-------------------------------------------------------------------------------------------
}
```

Paso 4

Realizaremos la programacion dentro del main que nos permitira ejecutar las funciones creadas

```
int main(){
    FLC();
    BLUETOOTH.baud(9600);
    BLUETOOTH.format(8,Serial::None,1);
    BLUETOOTH.printf("\nCONTROL HIBRIDO\n");
    wait(5);
    led_1=1;
    BLUETOOTH.printf("\nPULSAR S1\n");
    while(!S1);
    fondos(iteracion);
    BLUETOOTH.printf("\nPULSAR S1\n");
    //printf("PULSAR S1\n");
    while(!S1);
    lineas(iteracion);
    BLUETOOTH.printf("\nEL PROMEDIO ES\n");
    promedio();
    BLUETOOTH.printf("\nPULSAR S2");
    while(!S2);
    while(!S1){
        fuzzy->setInput(1,errorActual);
        fuzzy->setInput(2,errorDerivativo);
        fuzzy->fuzzify();
        KD = fuzzy->defuzzify(1);
        KD = fmap(KD,-1.0,1.0,0.0,11);
        KP = fuzzy->defuzzify(2);
```

```
        KP = fmap(KP,-1.0,1.0,0.0,0.5);
        position = lectura();
        pid();
        BLUETOOTH.printf("%1.0f;%1.3f\n", position,u);
      }
    wait(1);
    while(1) {
        motors(0,0);
    }
}
```

Anexo 9: PRÁCTICA #9

Paso 1

Se debe realizar los pasos de la práctica #1 hasta la práctica #4

Paso 2

Declarar las siguientes variables a utilizar

```
float errorActual = 0;
float errorIntegral = 0;
float errorDerivativo = 0;
float errorPasado = 0;
float Vmax = 0.3;
float Sp =0;
float Kp =0;
float Ki =0;
float Kd =0;
```

Paso 4

Creamos las nuevas funciones para el control PID de nuestro robot Line Follower

```
void PIDvar(float kp, float kd, float ki) {
   KP = kp;
   KD = kd;
   KI = ki;
}
float PID(int POS, int setpoint, float lim) {
   errorActual = ((int)POS) - setpoint;
   errorIntegral += KI*errorActual;
   errorDerivativo = errorActual - errorPasado;
   errorPasado = errorActual;
   float U = (KP*errorActual) + errorIntegral + (KD*errorDerivativo);
   if (U > lim)
      U = lim;
   else if (U < -lim)
      U = -lim; }
```

Paso 5

Creamos las funciones para receptar y enviar los datos hacia nuestra app

```
int readBuffer(char *buffer,int count){
   int i=0;
   t.start();  // start timer
   while(1) {
      while (BLUETOOTH.readable()) {
         char c = BLUETOOTH.getc();
         if (c == '\r' || c == '\n') c = '\0';
         buffer[i++] = c;
         if(i > count)break;
      }
      if(i > count)break;
      if(t.read() > 1) {
         t.stop();
         t.reset();
         break;
      }
   }
   wait(0.5);
   while(BLUETOOTH.readable()) {
      char c = BLUETOOTH.getc();
   }
   return 0;
}
void cleanBuffer(char *buffer, int count)
{
   for(int i=0; i < count; i++) {
      buffer[i] = '\0';
   }}
```

Paso 6

Realizaremos la programacion dentro del main que nos permitira ejecutar las funciones creadas

```
int main(){
   BLUETOOTH.baud(9600);
   BLUETOOTH.format(8,Serial::None,1);
   BLUETOOTH.printf("**Control PID**\r\n");
   wait(2);
   BLUETOOTH.printf("Esperando datos...\r\n");
   wait(2);
   while(!S2) {
      if (BLUETOOTH.readable()) {
         readBuffer(bufferllegada,100);
         if(NULL != ( ptr = strstr(bufferllegada,"SP"))) {
            BLUETOOTH.printf("Buffer original = [%s] \n\n", bufferllegada);
            int i;
            char **arr = NULL;
            int count =0;
            int c = dtmsplit(bufferllegada, ";", &arr, &count);
            BLUETOOTH.printf("Found %d tokens.\n", count);
            for (i = 0; i < count; i++) {
               BLUETOOTH.printf("string #%d: %s\n", i, arr[i]);
            }
            BLUETOOTH.printf("SP-string=%s\n",arr[1]);
            BLUETOOTH.printf("KP-string=%s\n",arr[3]);
            BLUETOOTH.printf("KI-string=%s\n",arr[5]);
            BLUETOOTH.printf("KD-string=%s\n",arr[7]);
            BLUETOOTH.printf("\n");
            Sp=atof(arr[1]);         //atoi ->  Convert string to integer C++
            Kp=atof(arr[3]);
            Ki=atof(arr[5]);
```

```
            Kd=atof(arr[7]);
            led_2=1;
            BLUETOOTH.printf("\nPULSAR S2\n");
            free(arr);
        }
    }
}
BLUETOOTH.printf("\nPULSAR S1\n");
while(!S1) {
    led_1=1;
    led_2=0;
    while(!S1);
    for (int i = 0; i < 10; i++) {
        BLUETOOTH.printf("%1f\n", Sp);
        BLUETOOTH.printf("%1.6f\n", Kp);
        BLUETOOTH.printf("%1.6f\n", Ki);
        BLUETOOTH.printf("%1.6f\n", Kd);
        wait(1);
    }
    break;
}
BLUETOOTH.printf("\nPULSAR S1\n");
while(!S1);
fondos(iteracion);
BLUETOOTH.printf("\nPULSAR S1\n");
//printf("PULSAR S1\n");
while(!S1);
lineas(iteracion);
BLUETOOTH.printf("\nEL PROMEDIO ES\n");
promedio();
```

```
    BLUETOOTH.printf("\nPULSAR S2");
    while(!S2);
    while(!S1) {
        toggle_led_ticker.attach(&toggle_led, 0.2);
        position = lectura();
        PIDvar(Kp, Kd, Ki);
        float u = PID(position, Sp, 0.2);
        if (position==750 && u<=0.05f) {
            u=0;
        }
BLUETOOTH.printf("%1.0f;%1.0f\n", position,Sp);
        motors(u,-u);
    }
}
```

Anexo 10: PRÁCTICA #10

Paso 1

Diseñar una interfaz con el software Labview para adquisición de datos en tiempo real

Figura 77. Interfaz grafica

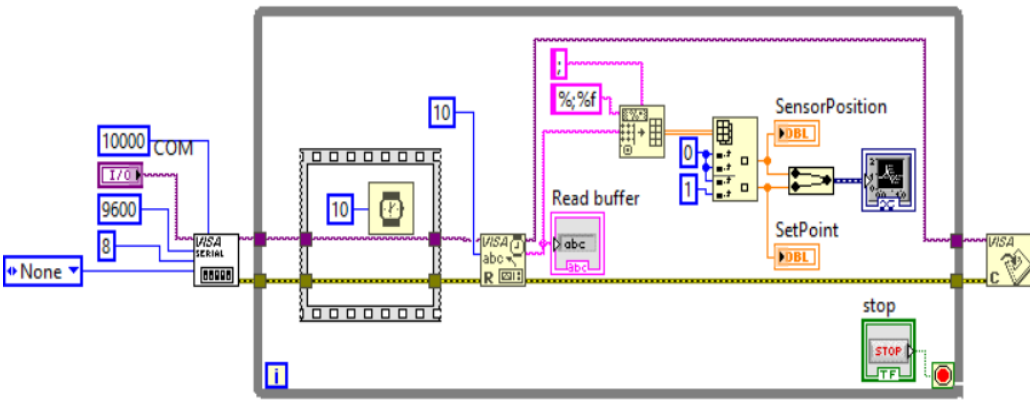


Figura 78. Diagrama de bloques labview.

Paso 2

Realizar la conexión del modulo bluetooth con la tarjeta stm32l432kc

Paso 3

Implementar un algoritmo en Stm32 para enviar datos en tiempo reala a la interfaz de Labview

```
while(!S1){
```

```
    toggle_led_ticker.attach(&toggle_led, 0.2);
    position = lectura();
    PIDvar(Kp, Kd, Ki);
    float u = PID(position, Sp, Vmax);
    if (position==750 && u<=0.05f){u=0;}
    BLUETOOTH.printf("%1.0f %1.3f\n", position,u);
    motors(Vmax - Power, Vmax + Powe);
  }
```

Paso 4

Asignacion de pines para la interconexion de los motores y crear las variables para usar la regleta de sensores

```
//ENTRADA ANALOGICA
AnalogIn An(A0);
//ENTRADAS DE BOTONES
DigitalIn S2(D12);
DigitalIn S1(D13);
//SALIDAS PARA EL TB16FN12G
PwmOut     PWMB(D9);
DigitalOut BIN2(D11);
DigitalOut BIN1(D6);
DigitalOut AIN2(PA_1);
DigitalOut AIN1(PA_4);
PwmOut     PWMA(PA_3);
int iteracion=300;
float sensores[16];
float digital[16];
float lectura_fondo[16];
float lectura_linea[16];
float umbral[16];
```

Paso 5

Crear las variables para asignar la velocidad de los motores y las del controlador

```
float errorActual = 0;
float errorIntegral = 0;
float errorDerivativo = 0;
float errorPasado = 0;
float Vmax = 0.3;
float Sp =750;
float Kp =0.0003;
float Ki =0.000002; //0.000000002;
float Kd =0.01;
```

Paso 6

Asignar los pines de salida para la comunicación del bluetooth

```
//****************** CONFIGURACIÓN DE PUERTOS DE COMUNICACIÓN ******************
Serial BLUETOOTH (D1, D0); //puertos USART del FRDM para el módulo BLUETOOTH
```

Paso 7

Asignar los valores del controlador PID tales como KP, KI, KD para realizar el cálculo del error derror para q puedan se mostrados en la salida del controlador

```
float PID(int POS, int setpoint, float lim) {
   errorActual = ((int)POS) - setpoint;
   errorIntegral += KI*errorActual;
   errorDerivativo = errorActual - errorPasado;
   errorPasado = errorActual;
   float U = (KP*errorActual) + errorIntegral + (KD*errorDerivativo);
   if (U > lim)
      U = lim;
   else if (U < -lim)
      U = -lim;
   return U;
}
```

Paso 8

Crear las sentencias para el uso del PWM en los motores

```
void motors(double speedL, double speedR)
{
   if(speedL>0) {
      AIN1=1;
      AIN2=0;
   } else {
      AIN1=0;
      AIN2=1;
      speedL=speedL*-1;
   }
   if(speedR>0) {
      BIN1 = 1;
      BIN2 = 0;
   } else {
      BIN1 = 0;
      BIN2 = 1;
      speedR=speedR*-1;
   }
   PWMA.write(speedL);
   PWMB.write(speedR);
}
```