UNIVERSIDAD POLITÉCNICA SALESIANA
SEDE GUAYAQUIL

FACULTAD DE INGENIERÍAS

CARRERA: INGENIERÍA ELECTRÓNICA

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE:

INGENIERO ELECTRÓNICO

TEMA:

"DISEÑO E IMPLEMENTACIÓN DEL ALGORITMO DE CONTROL PARA UN ROBOT BALANCE, USANDO FUZZY LOGIC EN LA PLATAFORMA DE NATIONAL INSTRUMENTS"

AUTORES:
GONZÁLEZ MACÍAS BRYAN XAVIER
BAUTISTA ESPAÑA ALEX DAVID

DIRECTOR:
ING. BYRON XAVIER LIMA CEDILLO, MSc.

GUAYAQUIL – ECUADOR
2018

CERTIFICADO DE RESPONSABILIDAD Y AUTORÍA

Nosotros, González Macías Bryan Xavier y Bautista España Alex David, estudiantes de Ingeniería Electrónica de la Universidad Politécnica Salesiana, certificamos que los conceptos desarrollados, análisis realizados y las conclusiones del presente trabajo son de exclusiva responsabilidad de los autores.

Guayaquil, Julio del 2018

González Macías Bryan Xavier
C.I.: 0802323261

Bautista España Alex David
C.I.: 0803488402

CERTIFICADO DE CESIÓN DE DERECHOS

Nosotros, González Macías Bryan Xavier con cedula de identidad N° 0802323261 y Bautista España Alex David con cedula de identidad N° 0803488402, a través del presente certificado, cedemos los derechos de propiedad intelectual correspondiente al trabajo de titulación "**DISEÑO E IMPLEMENTACIÓN DEL ALGORITMO DE CONTROL PARA UN ROBOT BALANCE, USANDO FUZZY LOGIC EN LA PLATAFORMA DE NATIONAL INSTRUMENTS"** a la Universidad Politécnica Salesiana, mismo que ha sido desarrollado para optar por el título de Ingeniero Electrónico según lo establecido por la ley de propiedad intelectual y por su normatividad institucional vigente.

Guayaquil, Julio del 2018

González Macías Bryan Xavier
C.I.: 0802323261

Bautista España Alex David
C.I.: 0803488402

CERTIFICADO DE DIRECCIÓN DEL TRABAJO DE TITULACIÓN

Por medio de la presente doy constancia que los Sres. González Macías Bryan Xavier y Bautista España Alex David han 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 Cedillo MSc.
DIRECTOR DE PROYECTO DE TITULACIÓN

DEDICATORIA

Dedico este proyecto técnico de titulación, primeramente a Dios que a lo largo de mi vida siempre me ha demostrado que jamás estoy solo, a mis padres Marco y Maribel que han sido el pilar fundamental para la culminación de esta ingeniería por ustedes he podido llegar tan lejos con una mente madura y el corazón intacto de un niño, a mis hermanos Francisco y Hannah que siempre me recuerdan por qué la familia es primero y cuán grande llega a ser el corazón al querer a alguien, a mi hija Lia que cambio mi mundo desde el primer día que la vi, a mi compañera y amiga de toda la vida Claudia por disfrutar del viaje de la vida conmigo a mis maestros, amigos y conocidos que a lo largo de la vida contribuyeron con sus conocimientos compartidos.

Bryan Xavier González Macías

DEDICATORIA

Dedico este Proyecto técnico a Dios por haberme permitido llegar hasta este momento tan importante de mi vida.

A mis padres por ser el pilar fundamental y brindarme todo el apoyo requerido a lo largo de este periodo universitario y enseñarme que en la vida se puede alcanzar lo que uno se proponga a pesar de las dificultades que se presenten a lo largo del camino…

A mis hermanos, comparto con ustedes la alegría de alcanzar esta meta…

A mis abuelos Oscar y Celinda que, a pesar de nuestra distancia física, siento que estuvieron conmigo y que este momento hubiera sido tan especial para ellos como para mí...

A mis tíos y demás familiares por ese apoyo y palabras de aliento para no desmayar en las metas planteadas a lo largo de esta meta…

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

Alex David Bautista España

AGRADECIMIENTO

Agradezco principalmente a Dios que en cada paso de mi vida está siempre presente.

A mi Mamá Maribel el motor fundamental de mi vida, a su amor incondicional que me ha brindado a través de los años, su paciencia, y su constante esfuerzo, este logro es para ti desde el primer día en que permitiste que un pelado pilas pudiera soñar en grande.

A mi padre Marco la bendición más grande de todas, por darme tantas cosas buenas e inculcarme valores más allá de lo convencional, por nunca dejar sola a mi madre y ser el engranaje de nuestras vidas, por su entrega y paciencia más allá del deber de ser padre, está dicha la comparto contigo pues no hubiera llegado tan lejos sin ti, me enseñaste la cosa más importante de todas como ser un buen ser humano, hijo y profesional.

A mis hermanos que aunque son adoptados los amo sin medida.

A mi abuelo Jorge que aunque no esté presente espero que pueda verme y aunque nunca pude decirlo gracias por dejarme los más grandes recuerdos de mi infancia.

A mi incondicional amigo Jaime Solís por ser un hermano con el cual contar en momentos de necesidad y su gran calidad de persona.

A mis amigos Bruno, Fabricio, Ángel, Jorge, Malik, Mishelle y José tantos momentos vividos a su lado deja el mejor recuerdo de mi vida en la Universidad.

A mi gran amigo, maestro y tutor Byron Lima por ser un padre más en el cual contar, y su práctica manera de impulsar a siempre superarse, ya su pupilo será un colega más.

Gracias a todos.

Bryan Xavier González Macías

AGRADECIMIENTO

Gracias a Dios por todas las bendiciones y protecciones brindadas durante todo mi camino y darme fuerzas para superar todos los obstáculos y dificultades que se presentaron a lo largo de mi vida.

A mi madre por ese apoyo incondicional y por haberme enseñado a no desfallecer y bajar los brazos fortaleciéndome con sus sabios consejos.

A mi familia por su apoyo incondicional y acompañarme en todo este arduo camino compartiendo alegrías y tristezas.

Al Ing. Marcos Holguín y a toda su familia por sus consejos, apoyo y confianza brindada a lo largo de todo este tiempo.

Al Dr. Marco Costa y su familia por el apoyo incondicional y demostrarme la gran fe y confianza que tienen puesta en mí.

Al Ing. Byron Lima, director de proyecto de Titulación, por su valiosa guía y asesoramiento brindando disponibilidad para siempre aclarar dudas en la realización de nuestro proyecto.

A la Lcda. Mariela Quispe por toda la ayuda brindada en todo el periodo académico.

A mi amigo Martin Robalino y Junior Mendoza por todo el apoyo y la ayuda brindada facilitando todos los implementos para el desarrollo de nuestro proyecto.

A mis amigos y a cada una de las personas que he tenido la oportunidad de conocer y ayudaron directa e indirectamente en esta etapa de mi vida y me han dejado sus enseñanzas en el ámbito personal y profesional.

A Counter Strike 1.6 por ayudarme a quitar tanta tensión y estrés a lo largo de este periodo y luego enfocarme en mis actividades y Proyecto de Titulación. 1000 gracias Counter…

Alex David Bautista España

RESUMEN

AÑO	ALUMNOS	TUTOR DEL PROYECTO	TEMA DEL PROYECTO
2018	GONZÁLEZ MACÍAS BRYAN XAVIER BAUTISTA ESPAÑA ALEX DAVID	ING. BYRON LIMA CEDILLO MSc.	"DISEÑO E IMPLEMENTACIÓN DEL ALGORITMO DE CONTROL PARA UN ROBOT BALANCE, USANDO FUZZY LOGIC EN LA PLATAFORMA DE NATIONAL INSTRUMENTS"

El presente proyecto técnico de titulación, "**DISEÑO E IMPLEMENTACIÓN DEL ALGORITMO DE CONTROL PARA UN ROBOT BALANCE, USANDO FUZZY LOGIC EN LA PLATAFORMA DE NATIONAL INSTRUMENTS"**, se lo realiza para el laboratorio de Fabricación Flexible de la Universidad Politécnica Salesiana sede Guayaquil para los estudiantes de ciclos superiores especialmente en las materias de informática industrial y electiva I , para poder realizar entrenamientos basados en la programación de robot balance utilizando lógica difusa.

La propuesta del proyecto de titulación se basa en el diseño e implementación de un algoritmo de control para un robot balance, usando fuzzy logic en la plataforma de National Instruments.

El desarrollo del proyecto consiste en una primera parte, ensamblar adecuadamente el robot DIY Balanduino Kit y verificar el respectivo funcionamiento.

Se realiza el diseño del robot balance en SolidWorks y luego se implementa un algoritmo de control basado en la plataforma de national instruments (Labview 2013) con control lógico difuso que será capaz de balancearse y mantener su equilibrio sobre sus ruedas.

Esto se consigue usando un acelerómetro, giroscopio y tarjeta Arduino Mega que nos ayuda en el proceso en tiempo real sobre los ángulos que forma el robot respecto al suelo y así poder obtener una señal de control que será transmitida a los motores de las ruedas para su accionamiento.

Además, su monitoreo se realiza desde una PC para poder observar el comportamiento del mismo.

PALABRAS CLAVES: Fuzzy Logic, PID, LabVIEW, SolidWorks, Matlab, modelo matemático, acelerómetro, giroscopio, Arduino Mega.

ABSTRACT

YEAR	STUDENTS	PROJECT TUTOR	PROJECT THEME
2018	GONZÁLEZ MACÍAS BRYAN XAVIER BAUTISTA ESPAÑA ALEX DAVID	ING. BYRON LIMA CEDILLO MSc.	"DESIGN AND IMPLEMENTATION OF CONTROL ALGORITHM FOR A ROBOT BALANCE, USING FUZZY LOGIC ON THE PLATFORM OF NATIONAL INSTRUMENTS"

The present technical Project of titling "DESIGN AND IMPLEMENTATION OF CONTROL ALGORITHM FOR A ROBOT BALANCE, USING FUZZY LOGIC ON THE PLATFORM OF NATIONAL INSTRUMENTS", was realized for the Laboratory of Flexible Manufacturing of the Polytechnic University Salesian headquarters Guayaquil for the students of Higher cycles especially in the areas of industrial and elective computing.

The proposal of the titling project is based on the design and execution of a control algorithm for a robot balance, using fuzzy logic in the National Instruments platform.

The development of the project consists of a first part, properly assembling the DIY Balanduino Kit robot and verify the respective operation.

The design of the balance robot in SolidWorks will be carried out and then a control algorithm based on the national instruments platform (Labview 2013) will be implemented with fuzzy logic control that will be able to balance and maintain its balance on its wheels.

This will be achieved with an accelerometer, gyroscope and Arduino Blend card that helps us in the process in real time on the angles that form the robot with respect to the ground and thus we can obtain a signal of the control that is transmitted to the motors of the wheels for the its Drive

In addition, its monitoring is done from a PC to be able to observe the behavior of the same.

KEYWORDS: Fuzzy Logic, PID, Labview, SolidWorks, Matlab, mathematical model, accelerometer, gyroscope, Arduino Mega.

ÍNDICE GENERAL

CERTIFICADO DE RESPONSABILIDAD Y AUTORÍA……II
CERTIFICADO DE CESIÓN DE DERECHOS III
CERTIFICADO DE DIRECCIÓN DEL TRABAJO DE TITULACIÓN..... IV
DEDICATORIA V
DEDICATORIA..... VI
AGRADECIMIENTO VII
AGRADECIMIENTO..... VIII
RESUMEN IX
ABSTRACT.....XI
ÍNDICE GENERAL……XIII
ÍNDICE DE FIGURAS XV
ÍNDICE DE TABLAS XX
INTRODUCCIÓN 1
1. EL PROBLEMA 2
1.1 Descripción del problema..... 2
1.2 Antecedentes 2
1.3 Importancia y alcance..... 2
1.4 Delimitación del problema 3
1.4.1 Delimitación temporal..... 3
1.4.2 Delimitación espacial 3
1.4.3 Delimitación académica 3
1.5 Objetivos 4
1.5.1 Objetivo general 4
1.5.2 Objetivos específicos..... 4
2. FUNDAMENTO TEÓRICO 5
2.1 Técnicas de control en lazo abierto 5
2.2 Sistemas de control lazo cerrado 5
2.3 Modelos matemáticos..... 5

2.4 Software SolidWorks 6

2.5 Software Labview 6

2.5.1 Panel frontal 7

2.5.2 Diagrama de bloques 7

2.6 Software Matlab 2013 8

2.7 Controladores Difusos 8

2.7.1 Ventajas de Controladores Difusos 9

2.7.2 Desventajas de Controladores Difusos 9

2.8 Micro controlador (Arduino mega 2560) 10

2.9 Acelerómetro + Giroscopio (MPU 6050) 10

2.10 Sistema de Potencia 11

2.11 Batería 12

2.12 Motores 12

2.13 Encoder 13

2.14 Módulo Bluetooth 14

2.15 Módulo LC STUDIO 14

3. MARCO METODOLÓGICO 15

3.1 Diseño del Robot balance 15

3.2 Cuerpo del robot balance 16

3.3 Soporte de prototipo 17

3.4 Varilla de Bronce roscada 17

3.5 Construcción del robot balance 17

3.6 Diseño para el sistema de control 18

3.7 Modelo matemático del Robot 18

3.8 Modelado del circuito eléctrico de motor DC 18

3.9 Parámetros experimentales 28

3.10 Constante de la fuerza contra electromotriz ($\boldsymbol{Ke}$) 29

3.11 Momento de Inercia, Ip 30

3.12 Resumen de parámetros 31

3.13 Validación del modelo matemático del sistema 32

3.15 Diseño del controlador difuso. 37

4. MANUAL DE PRÁCTICAS 41

4.1 Práctica # 1. 41

4.2 Práctica # 2 48

4.3 Práctica # 3. 56

4.4 Práctica # 4. 62

4.5 Práctica # 5. 68

5. RESULTADOS 75

5.1 Prueba y evaluación del Controlador PID 75

5.2 Análisis de resultado 77

CONCLUSIONES Y RECOMENDACIONES 79

BIBLIOGRAFÍA 80

ANEXOS 82

ÍNDICE DE FIGURAS

Figura 1. Ubicación de la Universidad Politécnica Salesiana 3
Figura 2. Solidworks 2017 6
Figura 3. Labview 2013 6
Figura 4. Panel frontal 7
Figura 5. Proceso para adquisición de datos. 7
Figura 6. Matlab 2013b 8
Figura 7. Tarjeta micro controlador Arduino mega 2560 10
Figura 8. MPU 6050 11
Figura 9. Tarjeta VNH5019 sistema de potencia marca Pololu 11
Figura 10. Batería LiPo 12
Figura 11. Motor y Encoder 12
Figura 12. Salida de Encoder de dos canales 13
Figura 13. Modulo Bluetooth HC-05 14
Figura 14. Módulo LC STUDIO 14
Figura 15. Modelo propuesto del Robot Balance 15
Figura 16. Vista frontal 16
Figura 17. Vista lateral 16
Figura 18. Soporte de Acrílico 17
Figura 19. Varilla de bronce roscada 17
Figura 20 Diagrama de construcción del Robot 18
Figura 21. Diagrama DC de un motor 19
Figura 22. Diagrama de cuerpo libre de una de las ruedas 21
Figura 23. Diagrama de cuerpo libre del péndulo 23
Figura 24. Respuesta del sistema en lazo abierto con impulso unitario 28
Figura 25. Datos de frecuencia de los motores. 29
Figura 26. Diseño del péndulo. 30
Figura 27. Parámetros del péndulo. 31
Figura 28. Simulación del modelo analítico del sistema 32
Figura 29. Validación del sistema en lazo abierto ángulo vs tiempo 33
Figura 30. Circuito de potencia como ganancia. 34
Figura 31. Diagrama para la simulación del controlador PID 34
Figura 32. Arquitectura de control para el equilibrio. 35

Figura 33. Lugar geométrico de las raíces del controlador 35
Figura 34. Compensador para el controlador PID..... 36
Figura 35. Respuesta de la simulación del controlador PID. 36
Figura 36. Designación de entradas y salida. 37
Figura 37. Función de pertenencia para la entrada Error. 38
Figura 38. Función de membrecía para la entrada dError. 38
Figura 39. Función de pertenencia de la salida PWM..... 38
Figura 40. Superficie del controlador Difuso. 39
Figura 41. Diseño del bloque de simulación del controlador Difuso. 39
Figura 42. Respuesta de la simulación del controlador Difuso. 40
Figura 43. Interconexión ARDUINO MEGA, MPU, LC STUDIO..... 42
Figura 44. Código de librerías Arduino. 43
Figura 45. Código de creación de archivo..... 43
Figura 46. Variables del acelerómetro y Giroscopio..... 43
Figura 47. Inicialización y pin de conexión. 44
Figura 48. Inicialización modulo MPU y serial. 44
Figura 49. Fórmula para tangente del acelerómetro. 45
Figura 50. Lectura del giroscopio y cálculo del ángulo. 45
Figura 51. Inicio a la escritura y monitoreo del tiempo. 46
Figura 52. Línea de error en el archivo. 46
Figura 53. Datos del ángulo de inclinación 0°. 46
Figura 54. Robot balance con ángulo de inclinación igual a 0°. 47
Figura 55. Datos del ángulo de inclinación a 15°. 47
Figura 56. Robot balance con ángulo de inclinación a 15°. 47
Figura 57. Panel frontal del interfaz. 49
Figura 58. Diagrama de bloques. 49
Figura 59. Interconexión de la tarjeta Arduino mega y Bluetooth. 50
Figura 60. Creación de librerías y variables para transmisión de datos. 51
Figura 61. Asignación de pines para interconexión de motores. 51
Figura 62. Variables de velocidad de motores y del controlador. 52
Figura 63. Asignación de pines salida para los motores. 52
Figura 64. Sentencias para llamar a variables del controlador. 53
Figura 65. Función serial para enviar datos por Bluetooth. 54
Figura 66. Creación de sentencia para el PWM en los motores. 55

Figura 67. Panel frontal con adquisición de datos 1 55
Figura 68. Panel frontal con adquisición de datos 2 55
Figura 69. Código para incluir librerías. 56
Figura 70. Definición de parámetros iniciales para Motores. 57
Figura 71. Definición de pines para motores. 57
Figura 72. Comunicación con el módulo MPU 6050..... 58
Figura 73. Variables para velocidad de motores. 58
Figura 74. Variables del PID..... 58
Figura 75. Definición de velocidad y MPU 6050. 59
Figura 76. Sentencia para controlador, PWM y MPU 6050. 59
Figura 77. Transformación de radianes a grados. 60
Figura 78. Valores del PID..... 60
Figura 80. Comportamiento del ángulo con el controlador PID. 61
Figura 81. Respuesta del Controlador PID..... 62
Figura 79. Sentencia de Motores..... 61
Figura 82. Código de librerías para el controlador difuso..... 63
Figura 83. Asignar de pines para motores..... 63
Figura 84. Creación de variables para MPU y cálculo de ángulo. 64
Figura 85. Variables del controlador difuso..... 64
Figura 86. Configuración de pines de salida e inicialización del serial. 65
Figura 87. Sentencias para el controlador difuso. 66
Figura 88. Transformación de radianes a grados. 66
Figura 89. Cálculo de las entradas del sistema..... 66
Figura 90. Sentencia para el PWM para los motores. 67
Figura 91. Comportamiento del ángulo con el controlador difuso. 68
Figura 92. Respuesta del controlador difuso..... 68
Figura 93. Diagrama de interconexión de tarjeta Arduino y Módulo Bluetooth. 69
Figura 94. Asignación de librerías y variables utilizadas..... 71
Figura 95. Asignación de variables para inicialización del driver VNH5019. 71
Figura 96. Sentencia para llamar al controlador PID. 72
Figura 97. Asignación de valores PID. 72
Figura 98. Sentencia para Motores..... 73
Figura 99. Asignación de condiciones del Bluetooth..... 74
Figura 102. Movimiento del Robot usando un dispositivo móvil..... 74

Figura 103. Obtención de datos con perturbaciones. 75
Figura 104. Respuesta del Ángulo del controlador PID con perturbación cada 10 seg.75
Figura 105. Respuesta del controlador PID con perturbación cada 10 seg. 76
Figura 106. Respuesta del ángulo respecto al tiempo del controlador Difuso con perturbación cada 10 seg. 76
Figura 107. Respuesta de la señal de control Difuso con perturbación cada 10 seg.. 77
Figura 108. Comparación de Controladores. 77

ÍNDICE DE TABLAS

Tabla 1. Componentes del motor DC 13
Tabla 2. Valores experimentales de Ra 28
Tabla 3. Constante de fuerza contra electromotriz del motor DC 30
Tabla 4. Parámetros del robot construido. 32
Tabla 5. Variables lingüísticas. 37
Tabla 6. Reglas para el control Difuso. 39
Tabla 7. Conexiones de tarjeta Arduino y módulos. 42
Tabla 8. Conexión de Arduino mega con Bluetooth. 50
Tabla 9. Conexión de la tarjeta micro controladora Arduino con módulo Bluetooth. 69

INTRODUCCIÓN

En la actualidad la lógica difusa es un nuevo modelo de controlador que nos ayuda en la programación y control de equipos electrónicos.

Con el pasar de los años será cada vez más común debido a su demanda en áreas de seguridad y establecimientos de recreación (centros comerciales, hoteles etc.) en los cuales su uso se limita a poder recorrer grandes distancias sin necesidad de caminar, conforme avance la tecnología y los costos de construcción disminuyan se podrá adquirir a un costo más bajo y su uso se destina no solo a aplicaciones de investigación sino a uso personal a una escala mucho mayor.

Para el Trabajo técnico de titulación, se implementaran los conocimientos aprendidos en la Universidad y llevarlos a una manera más real utilizando nuevos equipos y tecnología más reciente de controladores y aplicándola a prácticas realizadas desde otra perspectiva tomando en cuenta materias como informática industrial, electiva I, teniendo como ayuda el software national instruments (Labview 2013).

El trabajo técnico de titulación está orientado para los estudiantes de la Universidad Politécnica Salesiana de la carrera de Ingeniería Electrónica, para lograr que estén actualizados con el manejo de los controladores PID y Difusos, por consiguiente nuestro trabajo técnico manifiesta cinco prácticas didácticas resueltas y detallas en su totalidad.

Para la implementación de los controladores PID y Difuso se realizaran pruebas de una manera experimental y un modelo matemático con ayuda del software de Solidworks, con la finalidad de mostrar su auto balaceo, y el objetivo principal de mantener el equilibrio en forma vertical.

1. EL PROBLEMA

1.1 Descripción del problema

Actualmente la Universidad Politécnica Salesiana sede Guayaquil cuenta con módulos didácticos para implementar modelos difusos, a pesar de esto, los modelos planteados y prácticas realizadas por los estudiantes carece de una profundización más compleja de los tipos de controladores que se desarrollan en los laboratorios, se busca reforzar y repotenciar esta área con la creación de un robot balance y poder verificar el funcionamiento de controladores difusos de una manera más práctica y dinámica.

1.2 Antecedentes

Actualmente en la carrera de Ingeniería Electrónica de la Universidad Politécnica Salesiana sede Guayaquil se cuenta con módulos didácticos con relación a la lógica difusa, pero que en sus prácticas carecen de una profundización más compleja y didáctica del tema para la comprensión de los nuevos controladores.

La propuesta del proyecto está basado en la creación de un robot Balance el cual mostraría en base a prácticas como actúan los controladores difusos en sus diferentes etapas de control, implementado para el laboratorio de fabricación flexible de la Universidad Politécnica Salesiana sede Guayaquil con el propósito de que los estudiantes puedan implementar y verificar con la práctica los conocimientos logrados en la materia de informática industrial I y Electiva I.

1.3 Importancia y alcance

El presente trabajo técnico de titulación tiene como principal punto el dar a conocer el uso de los controladores difusos que actualmente está dirigido a una rama de la Ingeniería Electrónica de manera investigativa, el cual de una manera sencilla y eficaz plantea una nueva estrategia de enseñanza para los estudiantes generando grandes posibilidades en el alumnado para plantear posibilidades en donde enfocar su ingeniería.

Los principales beneficiarios de esta implementación de control con lógica difusa son los estudiantes de la Universidad Politécnica Salesiana Sede Guayaquil de la Carrera de Ingeniería Electrónica, especialmente los que estén cursando la materia de informática industrial y electiva I para fortalecer el aprendizaje mediante la simulación y aplicación de los controladores que se utilizan actualmente.

1.4 Delimitación del problema

1.4.1 Delimitación temporal

El proyecto técnico tuvo un tiempo de duración de 1 año 2 meses a partir de su aprobación por la dirección de carrera.

1.4.2 Delimitación espacial

El proyecto técnico se usará en el laboratorio de Fabricación Flexible de la Universidad Politécnica Salesiana Sede Guayaquil, carrera de Ingeniería Electrónica.

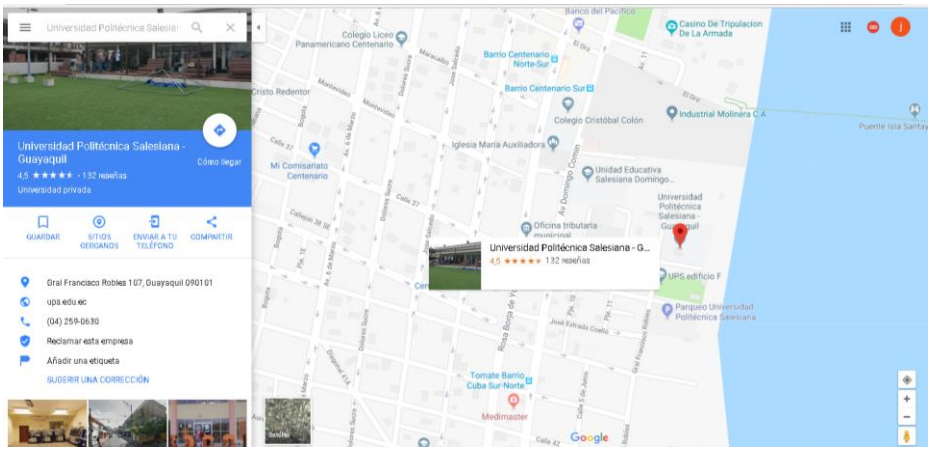


Figura 1. Ubicación de la Universidad Politécnica Salesiana

1.4.3 Delimitación académica

- Se realizarán 5 pruebas para elaborar un manual de prácticas didácticas resueltas.
- El Robot tiene alcance para materias de la carrera de Ingeniería Electrónica como, Informática Industrial I y Electiva I.

1.5 Objetivos

1.5.1 Objetivo general

Diseñar e implementar un algoritmo de control para un robot balance, usando fuzzy logic en la plataforma de National Instruments.

1.5.2 Objetivos específicos

- Elaborar un manual de 5 prácticas didácticas resueltas.
- Medir parámetros del robot balancín usando técnicas experimentales.
- Diseñar el algoritmo de control difuso para el sistema de robot balancín.
- Evaluar el algoritmo de control propuesto a través del análisis temporal y seguimiento de referencia.
- Comprobar su eficiencia comparándolo con un controlador PID básico.

2. FUNDAMENTO TEÓRICO

2.1 Técnicas de control en lazo abierto

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

Cada entrada cuenta con una referencia para una condición, donde su precisión va a depender de la calibración que se realiza en la operación. Cuando el sistema detecta la concurrencia de perturbaciones no realiza las tareas deseadas, por lo tanto las técnicas de control en lazo abierto solo se usa cuando se conoce la conexión que existe entre la entrada y la salida sin perturbaciones internas y externas. [1]

2.2 Sistemas de control lazo cerrado

El sistema de control lazo cerrado o también denominado sistema realimentado se usa indistintamente, se le proporciona al controlador una señal de error para que realice la acción, dando como resultado la desigualdad entre una señal de entrada y una señal de realimentación (considerada una señal de salida propia o funciones de salida con sus derivadas y/o integrales). El termino de controladores en lazo cerrado implica acciones de control realimentado con el propósito de bajar el error y obtener el valor que se desea en la salida del sistema. [1]

2.3 Modelos matemáticos

Los modelos matemáticos son utilizados de formas distintas dependiendo el sistema al que se lo relacione y las circunstancia específicas, existen modelos matemáticos considerados más convenientes que otros. Para los análisis de respuesta transitoria o de respuesta en frecuencia de sistemas lineales y no lineales con entradas y salidas invariantes con el tiempo.

Una vez adquirido un modelo matemático, se usan recursos analíticos para el estudio y sintetizarlo con la ayuda de una computadora. [1]

2.4 Software SolidWorks

Es un software que dispone de herramientas y aplicaciones necesarias para el diseño y modelado mecánico de piezas asistido por computadoras con un sistema eficaz como Microsoft Windows. El programa permite crear conjuntos de piezas para el elaboración de planos técnicos. El proceso consiste en traspasar la idea del dibujante al sistema CAD contribuyendo virtualmente, los diseños en 3D ayudan con el desarrollo de mejores productos y mayor rapidez. Cuando existen buenas ideas para un producto, dispondrá de las herramientas que precisa para diseñarlo en menos tiempo y a un menor coste. [10]

Figura 2. Solidworks 2017 [10]

2.5 Software Labview

Es una plataforma de National Instruments que brinda los instrumentos necesarias para la elaboración de instrumentación y automatización de procesos, sobresaliendo en diseños y técnicas de control de alto nivel en graficas con interconexión de bloques desarrollada en dos ventanas las cuales son. [2]

Figura 3. Labview 2013 [2]

2.5.1 Panel frontal

Es la ventana principal que contiene el diseño de toda la instrumentación como controles, numeradores, motores, tubería, luces botones etc., con la cual va a interactuar el usuario. En la figura 4 podemos apreciar el panel frontal donde se realiza un ejemplo para mostrar una señal en tiempo real. [7]

Figura 4. Panel frontal

2.5.2 Diagrama de bloques

Es la ventana que comprende el código de programación, donde se representa toda la instrumentación con los elementos del panel frontal la cual está representada por cuadros interconectados para el flujo de datos, representada por cuadros de contorno gris, que determinaran las series que se ejecutaran por cada etapa del proceso como se aprecia en la figura 5.[7]

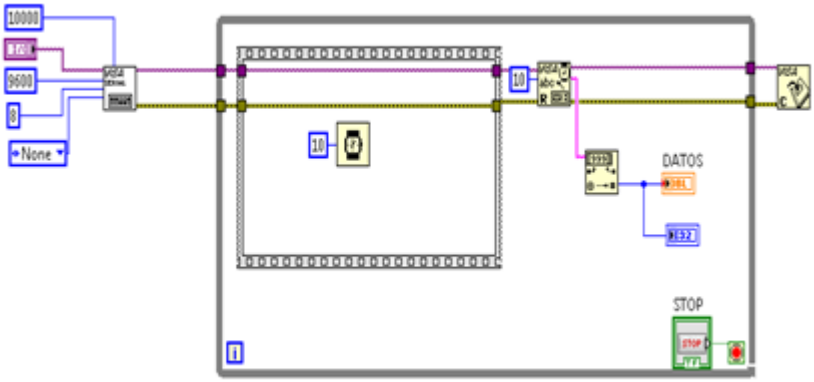


Figura 5. Proceso para adquisición de datos.

2.6 Software Matlab 2013

Es una herramienta matemática que propone entornos para desarrollar de manera integrada con un dialecto de programación propio, cuenta con instrucciones que permite resolver problemas científicos. Entre sus aplicaciones a la computación y matemática se puede mencionar el desarrollo de algoritmos, el modelado, la simulación, la exploración, la visualización, análisis de datos y la creación de graficas científicas. Entre los requerimientos del proyecto se tiene Matlab (R2013b) y/o posteriormente Simulink. [3]

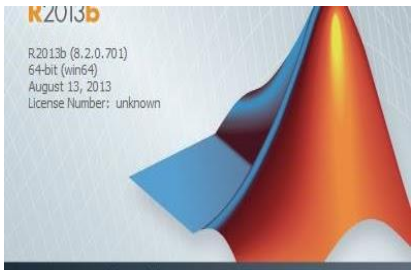


Figura 6. Matlab 2013b [3]

2.7 Controladores Difusos

Los controladores difusos son una rama de la Inteligencia Artificial y existen muchas estructuras donde se pueden formar dos grupos, el uno de control directo y el otro será un control adaptable, la estructura y el plan de control está dada por el proceso que deseamos controlar, recolectando los datos que se tenga para el controlador. El objetivo es realizar un estudio y elaborar un control difuso con mayor aplicación y realizar una comparación entre un controlador difusos y un controlador PID convencional.

Las variables del proceso de controladores difusos se las representa con letras minúsculas y con letras mayúsculas se indicarán las variables de entradas y salidas que están normalizadas en el sistema de inferencia difusa, en los bloques de pre- y pos-proceso las variables se convertirán de minúsculas a mayúsculas.

En Ingeniería se realizan implementaciones para mejorar los sistemas expertos en tiempo real, para poder expresar fácilmente parámetros de una forma más sencilla en comparación de los sistemas PID. [11]

2.7.1 Ventajas de Controladores Difusos

Los controladores difusos se han consolidando en el transcurso de los años permitiendo utilizar términos lingüísticos para poder realizar la evaluación de cantidades mayores de variables donde se procede a la comparación de entradas y salidas, permitiendo verificar que un sistema pueda ser más fiable y estable que un controlador común.

Este proceso es aconsejable para controladores que tengan algo de complejidad, o cuando no existe un modelo matemático o en procesos no lineales.

Los Controladores difusos se asemejan con la intuición del diseñador, resultando ser más accesibles, económicos y manejables a comparación de los controladores comunes teniendo una buena aceptación y no permiten tener problema con la complejidad y una mala definición con sistemas de grandes escalas. [11]

2.7.2 Desventajas de Controladores Difusos

Para diseñar controladores difusos se deben tener en cuenta criterios y compromisos para lograr que el sistema tenga la precisión que se requiere, la velocidad de respuesta y mucha flexibilidad.

Para ser más precisos se requiere de cuantiosas reglas y conjuntos de variables que determinen que el tiempo de solución tenga un incremento y necesita ser acoplado y puesto en operación.

Se debe tener en cuenta que los factores de la planta para que el sistema tenga flexibilidad y se adapte a los cambios que se realicen y se pueda aprender de los errores cometidos donde serán necesarios más cálculos que también aumentaran los número de simulaciones para una mejor respuesta del tiempo en el sistema. [11]

2.8 Micro controlador (Arduino mega 2560)

Los micros controladores son dispositivos electrónicos que permiten ejecutar funciones programadas y guardadas en su memoria, incluyendo unidades de procesamiento y periféricos de entradas y salidas tales como analógicas y digitales.

La tarjeta Arduino es una tarjeta micro controladora que brinda los instrumentos necesarias para ejecutar algoritmos de control utilizando el software de Arduino mega con su versión 2560, brinda 54 entradas/ salidas digitales (donde 14 son utilizadas como PWM), 16 entradas analógicas y 4 puertos especiales. Cuenta con una programación basada en el lenguaje C, con una capacidad de utilizar programación dirigida a objetos con una velocidad de reloj de 16 MHZ. [4]

Figura 7. Tarjeta micro controlador Arduino mega 2560 [4]

2.9 Acelerómetro + Giroscopio (MPU 6050)

Es un dispositivo electrónico muy preciso con la capacidad de determinar la velocidad angular (°/s) considerando los ejes cartesianos (x, y, z). Este integrado tiene un acelerómetro y un giroscopio de 3 ejes cada uno, para nuestro proyecto utilizaremos un MPU 6050 ya que contiene un hardware de cambio de analógico a digital de 16 bits para cada canal. Para interactuar con la tarjeta micro controladora Arduino accede sus datos a través de comunicación I2C. [6]

Figura 8. MPU 6050 [6]

2.10 Sistema de Potencia

Es un circuito que maneja la potencia del voltaje y corriente de niveles significativos que podrían dañar los dispositivos electrónicos que se encuentran conectados y trabajan a niveles bajos de voltaje y corriente.

Es necesario utilizar circuitos que permitan alimentar y proteger los motores para evitar posibles daños en el sistema, la tarjeta que se aprecia en la figura 11. cuenta con un diseño practico para ser montada en las tarjetas Arduino permitiendo controlar los motores de manera bidireccional con un rango de voltaje de 5.5 a 24V soportando una corriente de 12 A continua y 30 A pico por motor. Utiliza señales de modulación por ancho pulso (PWM) como entradas de control para los motores.

Esta tarjeta evalúa la corriente utilizada por los motores por medio de las salidas analógicas M1CS y M2CS, además permite ver si los motores se encuentran conectados, esto es por sus puertos de salida digital M1EN y M2EN. [4]

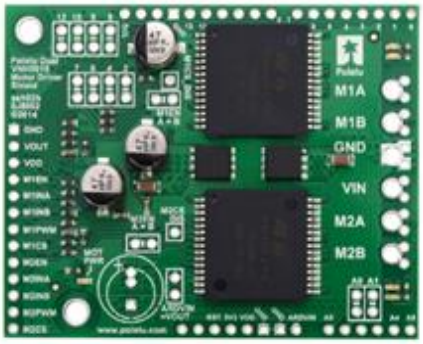


Figura 9. Tarjeta VNH5019 sistema de potencia marca Pololu [4]

2.11 Batería

Es un elemento recargable compuesto por células secundarias en paralelo para incrementar la capacidad de corriente de carga, con la capacidad de dotar de autonomía a un vehículo. Para realizar una selección apropiada se consideró el consumo del circuito electrónico, como el de los motores. La velocidad que se requiere deber ser instantánea. [4]

Figura 10. Batería LiPo [4]

2.12 Motores

Es una máquina que se encarga de generar un control en el sistema de una manera automatizada, ayudando a sostener el equilibrio después de un par aplicado logrando que el robot realice los movimientos necesarios para su equilibrio, utilizaremos dos motores de 12 VCD con una caja de engranaje de 50:1 con una velocidad de salida de 200 rpm. [4]

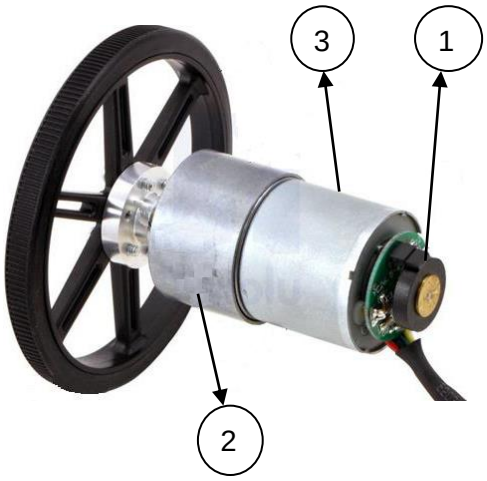


Figura 11. Motor y Encoder [4]

Elemento	Descripción
1	Encoder incremental
2	Caja de engranes 50:1
3	Motor 12 V @ 12 kg-cm

Tabla 1. Componentes del motor DC. [4]

2.13 Encoder

Es un instrumento electrónico en forma de disco magnético que encuentra instalado en el eje de los motores, ayudando a medir de la velocidad angular de las ruedas, genera la cantidad determinada de pulsos por revoluciones que necesita un motor para poder conocer la velocidad y la posición angular de un eje. Con el sentido en el gira se puede determinar la forma de las señal donde se enviaran dos señales que se encuentran desfasadas 90° , si se recibe un dato del canal 1 en alta (uno lógico)seguida de un dato del canal 2 también alta se puede considerar que el sentido de rotación es horario o anti horario teniendo en cuenta la programación, pero si la señal del canal 2 esta baja (cero lógico) se puede considerar que el giro se realizara en sentido opuesto. Como se aprecia en la figura. [4]

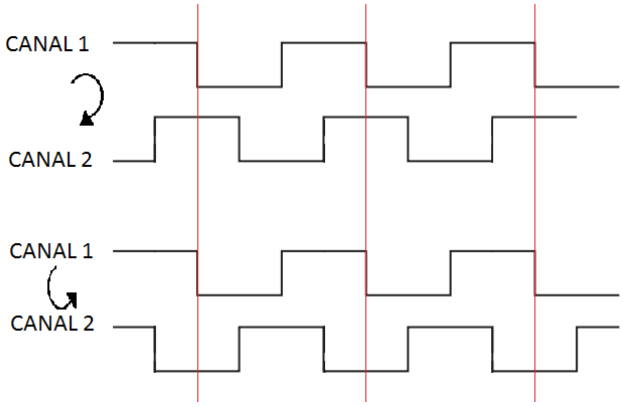


Figura 12. Salida de Encoder de dos canales [4]

Utilizaremos un Encoder de marca POLOLU de efecto hall de dos canales donde se podrá obtener una resolución de 64 pulsos por revolución.

2.14 Módulo Bluetooth

El módulo Bluetooth HC-05 es un dispositivo que permite la comunicación de datos por medio de un enlace de radiofrecuencia entre dispositivos facilitando la comunicación entre Arduino y un Smartphone y creando pequeñas redes sin uso de conexiones físicas. [4]

Figura 13. Modulo Bluetooth HC-05 [4]

2.15 Módulo LC STUDIO

El módulo LC STUDIO es un dispositivo que se utiliza para leer y guardar datos de una tarjeta SD. Se pueden almacenar datos leídos por sensores para obtener los datos de la posición en la cual se encuentra el robot y posteriormente procesarlos o bien leer desde la tarjeta SD. [7]

Figura 14. Módulo LC STUDIO [7]

3. MARCO METODOLÓGICO

3.1 Diseño del Robot balance

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

El robot fue diseñado en SolidWorks donde se tomaron en cuenta las medidas de cada una de las partes y componentes electrónicos que forman parte de su funcionamiento tales como motores, ruedas, batería, sensores y microcontrolador.

En la figura 15. se muestra el robot con los componentes que lo conforman de diversas vistas del propio. [4]

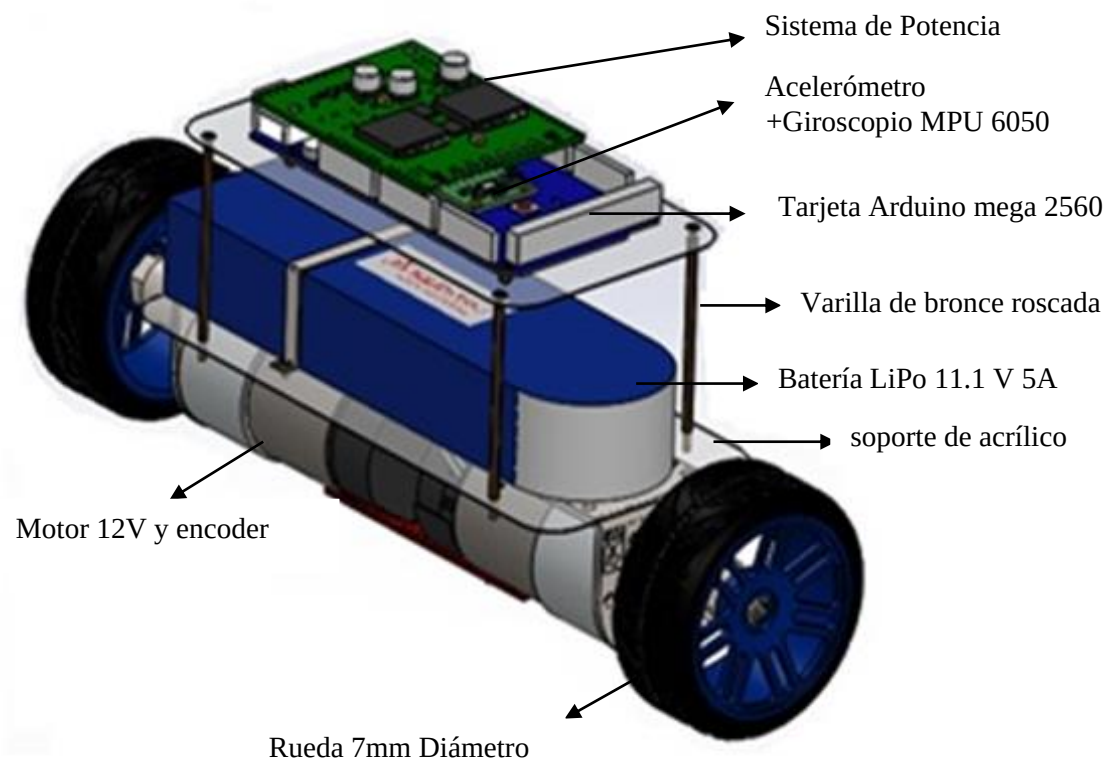


Figura 15. Modelo propuesto del Robot Balance

3.2 Cuerpo del robot balance

La estructura del robot balance consta de acrílico y esqueletos de bronce para poder tener un fácil control y visibilidad de los elementos y poder realizar maniobras con sus conexiones. [4]

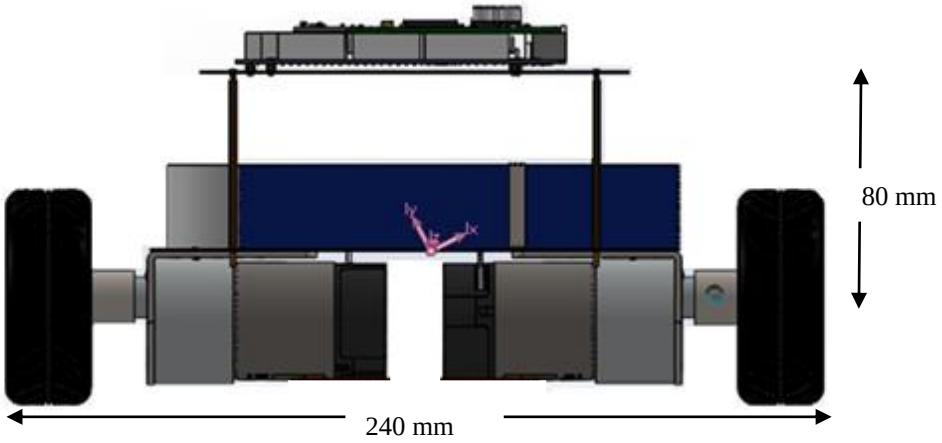


Figura 16. Vista frontal

Figura 17. Vista lateral

3.3 Soporte de prototipo

El soporte del prototipo esta hecho de material acrílico muy fácil de maquinar, esta pieza sirve para ubicar los elementos tales como tarjetas, sujetador de motores y batería. [4]

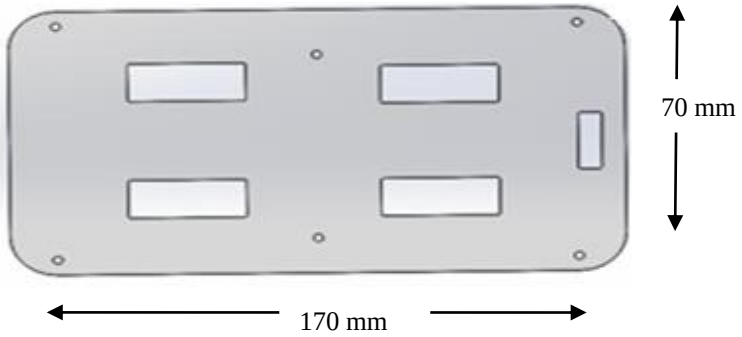


Figura 18. Soporte de Acrílico

3.4 Varilla de Bronce roscada

Esta varilla sirve para realizar el acople de toda la estructura del robot para tener en cuerpo más rígido, la figura 21. Muestra la varilla que utilizaremos en el proyecto. [4]

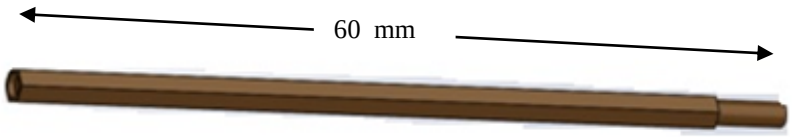


Figura 19. Varilla de bronce roscada

3.5 Construcción del robot balance

Por otro lado, se muestra la estructura de los principales elementos del Robot Balance [4]

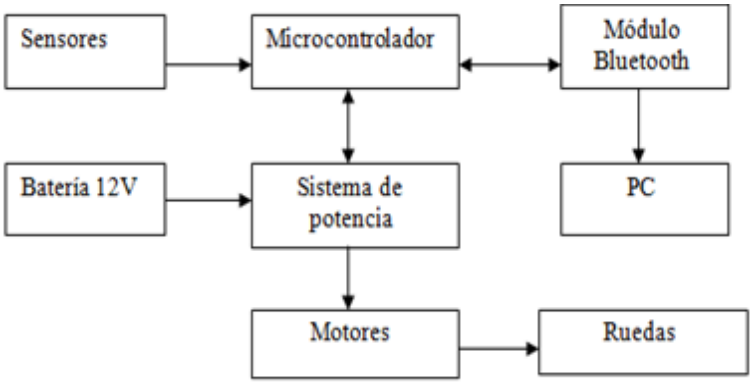


Figura 20 Diagrama de construcción del Robot [4]

3.6 Diseño para el sistema de control

Primeramente, se presenta el bosquejo del control para el equilibrio del sistema carro péndulo, después se plantean las diferentes técnicas para el control que será utilizado en el controlador del robot, se va a implementar un controlador PID convencional y un controlador difuso. Finalmente, se establece una edificación de control que se acople como estructura para diseñar el control del sistema completo. [4]

3.7 Modelo matemático del Robot

En este sección se realiza el modelo matemático y la simulación aplicando técnicas experimentales del sistema carro-péndulo y del vehículo diferencial. Las simulaciones de estos corresponden al sistema en lazo abierto. Primeramente, se procede a establecer el modelo del carro-péndulo considerando el enfoque newtoniano analizando el plano (x, y), después se realiza un análisis del vehículo diferencial sobre el plano (xo, yo) para determinar su cinemática. [4]

3.8 Modelado del circuito eléctrico de motor DC

Para esquematizar un controlador para el sistema se solicita un modelo matemático adecuado, con la finalidad de detallar la dinámica del sistema. En esta parte se realiza el modelo en espacio de estados en el cual se incorporara la dinámica tanto del cuerpo del robot como de las ruedas y los motores de DC.

Posteriormente, se observa el esquema DC de un motor de con una carga aplicada, en este ocasión es el torque preciso para sostener el péndulo en equilibrio. Con la objetivo de relacionar el voltaje que ingresa para controlar el péndulo. [4]

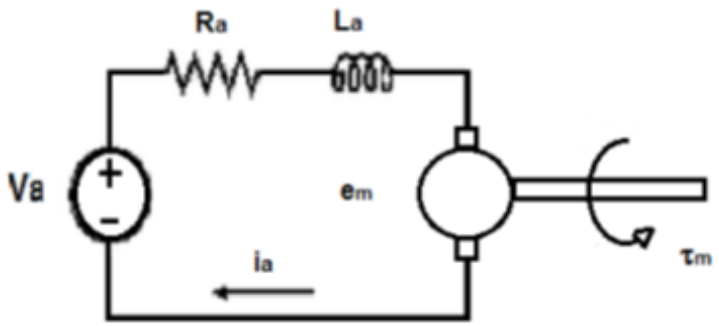


Figura 21. Diagrama DC de un motor [4]

Considerando la ley de voltaje Kirchoff se tiene la siguiente ecuación: [4]

$$\boldsymbol{V_a} = \boldsymbol{L}\frac{\boldsymbol{di}}{\boldsymbol{dt}} + \boldsymbol{R_a i} + \boldsymbol{e_m} \qquad (3)$$

Donde $\boldsymbol{V_a}$ es el voltaje que se está aplicado, i está considerada como la corriente de armadura, L la inductancia, $\boldsymbol{R_a}$ es la resistencia del motor y la fuerza contra electromotriz está definida por: [4]

$$\boldsymbol{e_m} = K_e\, \omega_m \qquad (4)$$

Donde $\boldsymbol{K_e}$ representa la constante contra electromotriz y $\boldsymbol{\omega_m}$ representa la velocidad angular del motor. En nuestro caso se procede a desprecia la inductancia, [4]

$$i = \frac{V_a - e_m}{R_a} \qquad (5)$$

Por consiguiente, se considera la ecuación, [4]

$$\tau_m = K_m i \qquad (6)$$

Donde τ_m está representando el par aplicado por el motor y la constante de torque estará representada por K_m. [4]

Aplicando la segunda ley de Newton sobre el eje del motor, donde también se desprecia la fricción quedando definida por: [4]

$$\sum M = \tau_m - t_a = I_m \dot{\omega}_m \qquad (7)$$

Considerando que t_a será el torque de fuerza que tendrá que vencer para sostener el péndulo en equilibrio, I_m representara el momento de inercia del eje del motor. [4]

Por otro lado sustituyendo las ecuaciones (4) y (6) en (5) y (7), [4]

$$i = \frac{V_a - K_e \omega_m}{R_a} \qquad (8)$$

$$\dot{\omega}_m = \frac{K_m i - \tau_a}{I_m} \qquad (9)$$

Por consiguiente (8) en (9), [4]

$$\dot{\omega}_m = \frac{K_m V_a}{I_m R_a} - \frac{K_m K_e \omega_m}{I_m R_a} - \frac{\tau_a}{I_m} \qquad (10)$$

La ecuación (10) describe la dinámica del motor completa. [4]

En la figura 24. Se procede a conseguir el modelo dinámico de las ruedas con su respectivo diagrama de cuerpo libre. Considerando oportuna a la simetría del robot se va a suponer que las fuerzas que se efectúan sobre ambas ruedas son las mismas. [4]

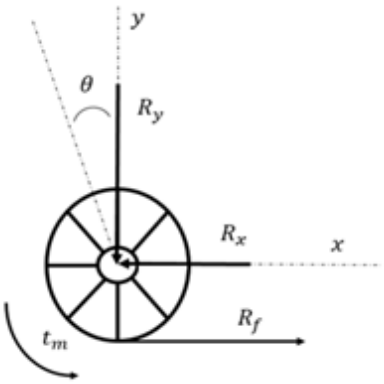


Figura 22. Diagrama de cuerpo libre de una de las ruedas [4]

Aplicando la segunda ley de Newton sobre las ruedas se tiene: [4]

$$M_R\ddot{x} = R_f - R_x \qquad (11)$$

Consideraremos que la masa de la rueda está representada por M_R, $\ddot{x}$ la aceleración lineal, R_f esta dada por la fricción contra el suelo y R_x la reacción de la rueda sobre el péndulo, ahora se procede con el estudio de las fuerzas de rotación: [4]

$$I_R\ddot{\theta}_R = \tau_m - R_f r \qquad (12)$$

Donde I_R significa el momento de inercia de la rueda, $\ddot{\theta}_R$ la aceleración angular de la rueda, τ_m esta dada por el par del motor, la reacción de fricción está representada por R_f y r es el radio de la rueda. [4]

Retomando las ecuaciones del modelo del motor y remplazando en (12), tenemos como resultado que, [4]

$$I_R\ddot{\theta}_R = -\frac{K_m K_e}{R_a}\dot{\theta}_R + \frac{K_m V_a}{R_a} - R_f r \qquad (13)$$

Considerando que θ_R representa el ángulo de la rueda, y procedemos a despejar Rf de la ecuación (13), [4]

$$R_f = -\frac{K_m K_e}{R_a r}\dot{\theta}_R + \frac{K_m}{R_a r}V_a - \frac{I_R}{r}\ddot{\theta}_R \qquad (14)$$

Reemplazando la ecuación (14) en la (11) resulta, [4]

$$M_R\ddot{x} = -\frac{K_m K_e}{R_a r}\dot{\theta}_R + \frac{K_m}{R_a r}V_a - \frac{I_R}{r}\ddot{\theta}_R - R_x \qquad (15)$$

Para el desplazamiento angular lineal se utilizan las ecuaciones que se visualiza a continuación: [4]

$$\theta = \frac{x}{r} \qquad (16)$$

$$\dot{\theta}_R = \frac{\dot{x}}{r} \qquad (17)$$

$$\ddot{\theta}_R = \frac{\ddot{x}}{r} \qquad (18)$$

Por otro lado, sustituyendo las ecuaciones (17) y (18) en (15) se obtiene: [4]

$$M_R\ddot{x} = -\frac{K_m K_e}{R_a r^2}\dot{x} + \frac{K_m}{R_a r}V_a - \frac{I_R}{r^2}\ddot{x} - R_x \qquad (19)$$

Realizando la agrupación de términos tenemos: [4]

$$\left(M_R + \frac{I_R}{r^2}\right)\ddot{x} = -\frac{K_m K_e}{R_a r^2}\dot{x} + \frac{K_m}{R_a r}V_a - R_x \qquad (20)$$

considerando que ambas ruedas son iguales y trabajan con el mismo consumo de carga y los actuadores cuenten con las mismas características, se procede a multiplicar la ecuación (20) por un factor de dos, [4]

$$2\left(M_R + \frac{I_R}{r^2}\right)\ddot{x} = -2\,\frac{K_m K_e}{R_a r^2}\dot{x} + 2\frac{K_m}{R_a r}V_a - 2R_x \qquad (21)$$

La ecuación (21) representara la dinámica del motor.

En la figura 25. se precia el diagrama de cuerpo libre del péndulo para realizar la descripción de su dinámica: [4]

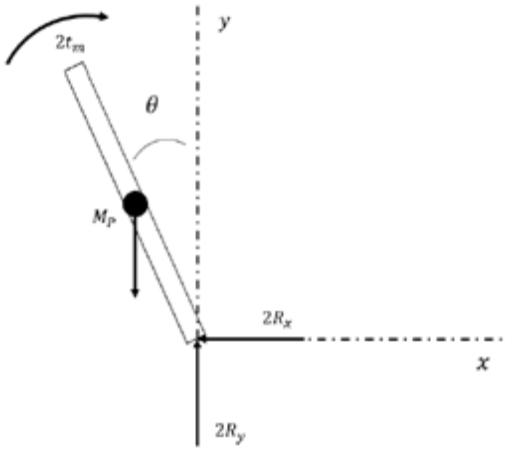


Figura 23. Diagrama de cuerpo libre del péndulo [4]

Considerando la segunda Ley de Newton se va a obtener la ecuación de las fuerzas en (x) para el péndulo, [4]

$$\sum F_x = M_p\ddot{x} \qquad (22)$$

Se considera a M_p como la masa del péndulo y $\ddot{x}$ será la aceleración lineal del péndulo en su centro de gravedad sobre el eje x. [4]

$$M_p\ddot{x} = 2R_x - M_p l\,\ddot{\theta}_p \cos\theta_p + M_p l\dot{\theta}_p^2 \operatorname{sen}\theta_p \qquad (23)$$

Si despejamos $2R_x$ de la ecuación (23) tenemos, [4]

$$2R_x = M_p\ddot{x} + M_p l\,\ddot{\theta}_p \cos\theta_p - M_p l\dot{\theta}_p^2 \operatorname{sen}\theta_p \qquad (24)$$

Por otro lado, se realiza el cálculo de las fuerzas en el eje y, [4]

$$\sum F_y = M_p\ddot{x}\cos\theta_p \qquad (25)$$

$$M_p\ddot{x}\cos\theta_p = 2R_x\cos\theta_p + 2R_y \operatorname{sen}\theta_p - M_p g \operatorname{sen}\theta_p - M_p l\ddot{\theta}_p \qquad (26)$$

Considerando la sumatoria de momentos alrededor del centro de masa del péndulo es: [4]

$$\sum M_o = I_p\ddot{\theta}_p \qquad (27)$$

$$-2R_x l\cos\theta_p - 2R_y l \operatorname{sen}\theta_p - 2\tau_m = I_p\ddot{\theta}_p \qquad (28)$$

Si remplazamos la ecuación (5) en (6) se tiene la siguiente expresión, [4]

$$t_m = -\frac{K_m K_e}{R} + \frac{K_m V_a}{R} \tag{29}$$

Reemplazando τ_m de la ecuación (29) en la (28) se obtiene: [4]

$$-2R_x l \cos\theta_p - 2R_y l \operatorname{sen}\theta_p + 2\frac{K_m K_e}{R_a r}\dot{x} - 2\frac{K_m}{R_a}V_a = I_p\ddot{\theta}_p \tag{30}$$

Multiplicando la ecuación (26) por $-l$ se tiene: [4]

$$-M_p\ddot{x}\cos\theta_p = -2R_x l\cos\theta_p - 2R_y l \operatorname{sen}\theta_p + M_p g l \operatorname{sen}\theta_p + M_p l^2\ddot{\theta}_p$$
(31)

y reemplazando la ecuación (31) en (30), [4]

$$I_p\ddot{\theta}_p - 2\frac{K_m K_e}{R_a r}\dot{x} + 2\frac{K_m}{R_a}V_a + M_p l g \operatorname{sen}\theta_p + M_p l^2\ddot{\theta}_p = -M_p\ddot{x}\cos\theta_p \tag{32}$$

si se reemplaza (23) en (21), queda la siguiente expresión: [4]

$$2\left(M_R + \frac{I_R}{r^2}\right)\ddot{x} = -2\frac{K_m K_e}{R_a r^2}\dot{x} + 2\frac{K_m}{R_a r}V_a - M_p\ddot{x} - M_p l\ddot{\theta}_p\cos\theta_p + M_p l\dot{\theta}_p^2 \operatorname{sen}\theta_p$$
(33)

Las ecuaciones (32) y (33) demuestran la dinámica del sistema carro péndulo. Se puede concluir que son ecuaciones no lineales, se realiza una linealización alrededor de una zona de operación, el cual está considerado como la localización de

equilibrio del robot, esto con la meta de emplear un control lineal que pueda posicionar el péndulo en forma vertical hacia arriba. [4]

Asumiremos que $\theta_p = \pi - \emptyset$, donde $\emptyset$ es un ángulo muy pequeño.

Por lo tanto, tenemos que:

$$\cos\theta_p \approx -1 \;\; y \;\; \operatorname{sen}\theta_p \approx -\emptyset \;\; y \;\; \left(\frac{d\dot{\theta}_p}{dt}\right)^2 = 0$$

Las ecuaciones linealizadas resultan de reescribir (32) y (33) en forma de espacio de estados, [4]

$$\ddot{\emptyset} = \frac{M_p l}{I_p + M_p l^2}\ddot{x} + 2\frac{K_m K_e}{R_a r\left(I_p + M_p l^2\right)}\dot{x} - 2\frac{K_m}{R_a\left(I_p + M_p l^2\right)}V_a + \frac{M_p g l}{I_p + M_p l^2}\emptyset$$

(34)

$$\ddot{x} = \frac{2K_m}{R_a r\left(2M_R + 2\frac{I_R}{r^2} + M_P\right)}V_a - \frac{2K_m K_e}{R_a r^2\left(2M_R + 2\frac{I_R}{r^2} + M_P\right)}\dot{x} + \frac{M_p l}{\left(2M_R + 2\frac{I_R}{r^2} + M_P\right)}\ddot{\emptyset}$$

(35)

Luego se obtiene, [4]

$$\beta = \left(2M_R + 2\frac{I_R}{r^2} + M_p\right)$$

$$\alpha = \left[I_p\,\beta + 2M_p l^2\left(M_R + \frac{I_R}{r^2}\right)\right]$$

Finalmente se obtiene una matriz en forma de espacio de estados. [4]

$$\dot{\boldsymbol{Z}} = \boldsymbol{AZ} + \boldsymbol{BU}$$

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\emptyset} \\ \ddot{\emptyset} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & \frac{2K_m K_e(M_p lr - I_p - M_p l^2)}{Rr^2\alpha} & \frac{M_p^2 g l^2}{\alpha} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{2K_m K_e(r\beta - M_p l)}{Rr^2\alpha} & \frac{M_p g l\beta}{\alpha} & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \emptyset \\ \dot{\emptyset} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{2K_m(I_p + M_p l^2 - M_p lr)}{Rr\alpha} \\ 0 \\ \frac{2K_m(M_p l - r\beta)}{Rr\alpha} \end{bmatrix} V_a$$

(36)

Dando como resulta,

$$z = \begin{bmatrix} x & \dot{x} & \emptyset & \dot{\emptyset} \end{bmatrix} \quad y \quad U = V_a$$

Luego de haber realizado el modelado del sistema se realiza la simulación en computadora con el software de Matlab 2013.

La simulación se implementó en lazo abierto, sin ningún tipo de control, con el objetivo de verificar que es un sistema inestable para lo cual será necesario realizar modificaciones en el sistema dinámico a través del establecimiento de una entrada de control para poder lograr la actuación deseado del sistema. [4]

Posteriormente, se aplica un impulso unitario para obtener una respuesta del sistema: [4]

(1)=1

Se asignan las siguientes condiciones iniciales para ver el comportamiento del sistema. [4]

$$z(0) = \begin{bmatrix} x(0) & \dot{x}(0) & \emptyset(0) & \dot{\emptyset}(0) \end{bmatrix} \quad = \quad [0 \;\; 0 \;\; 0 \;\; 0]^{\tau}$$

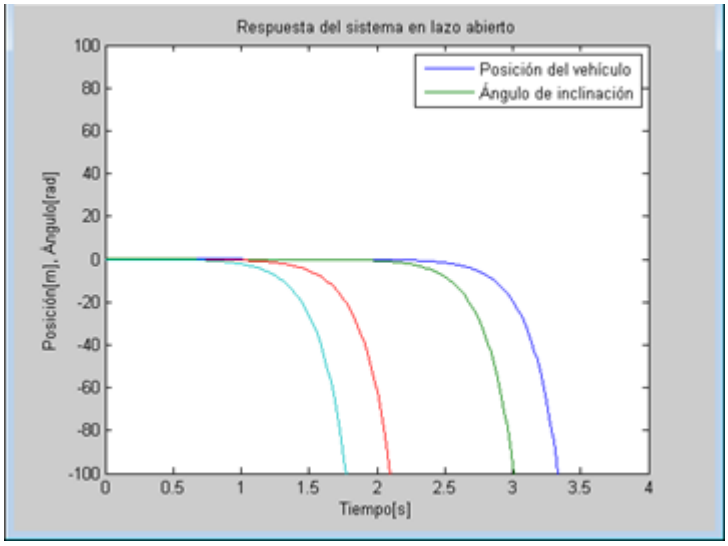


Figura 24. Respuesta del sistema en lazo abierto con impulso unitario

En la figura 26. Se aprecia que el ángulo tiende al infinito, demostrando que es un sistema inestable. Por consiguiente se implementa un sistema de control para conseguir establecer el equilibrio con el ángulo deseado. [4]

3.9 Parámetros experimentales

Considerando las características con la cual operan los motores: la resistencia de armadura obtenida con un multímetro midiendo los dos terminales del motor, obteniendo como resultado R_a = 4,5 Ω, la constante de fuerza contra electromotriz (K_e) y la constante par (K_m) [4]

Muestra	Ra (Ohms)
1	8,400
2	7,600
3	3,3
4	3,4
5	3,3
6	2,9
7	2,900
Promedio	4,542857

Tabla 2. Valores experimentales de Ra

3.10 Constante de la fuerza contra electromotriz (K_e)

Para definir la constante de fuerza se efectúa la medición de la velocidad angular (ω) del motor sin carga con diferentes valores de voltaje.

Para obtener el valore de medición angular se utiliza una fuente de voltaje regulable y de forma adicional un osciloscopio Peak Teach.

El Encoder rotativo convierte una oscilación angular en una secuencia de pulso digitales, lo cual indica que hay una relación entre la frecuencia medida en sus terminales de velocidad angular del eje:

$$\omega = 2\pi f$$

Donde ω es la velocidad angular del eje representada en rad/seg. Y f representa la frecuencia en Hertz. [1] [4]

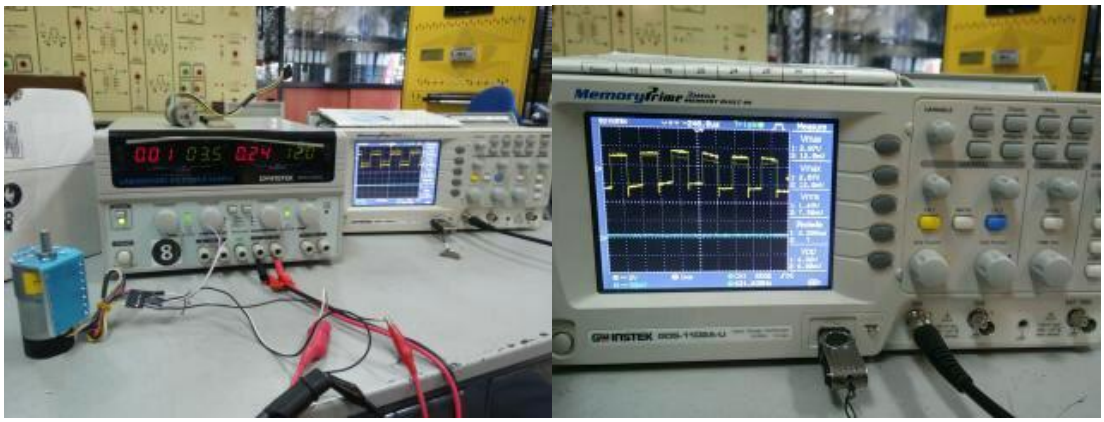


Figura 25. Datos de frecuencia de los motores.

Analizando que no se pudo obtener la constante de fuerza contra electromotriz del creador se procedió a obtener de manera experimental, valorando con la siguiente expresión.

$$K_e = \frac{V_a - R_a i}{\omega}$$

Para adquirir un valor más preciso de (ke) se procede a medir con diferentes valores de voltajes dando como resultado un promedio como se indica en la Tabla 3: [1] [4]

VA (Volt)	RA (Ohm)	IA (Amp)	Frec (Hz)	W(rad/s)	K_e
1,5	4,5428571	0,13	402	39,47	0,01959
2	4,5428571	0,14	622	61,06	0,022337
3	4,5428571	0,16	913	89,63	0,03
4	4,5428571	0,18	1138	111,72	0,03
5	4,5428571	0,19	1431	140,49	0,03
6	4,5428571	0,21	1723	169,16	0,03
7	4,5428571	0,22	2039	200,18	0,03
8	4,5428571	0,23	2244	220,3	0,03
9	4,5428571	0,24	2823	277,15	0,03
10	4,5428571	0,25	2864	281,17	0,03
11	4,5428571	0,26	2966	291,19	0,03
12	4,5428571	0,27	3338	327,71	0,03
				PROMEDIO:	0,03

Tabla 3. Constante de fuerza contra electromotriz del motor DC

3.11 Momento de Inercia, I_p

Para obtener los parámetros del momento de inercia del péndulo, se procede al diseño del mismo, considerando densidades y parámetros de los materiales por el cual está conformado en el software SolidWorks 2017. [1] [4]

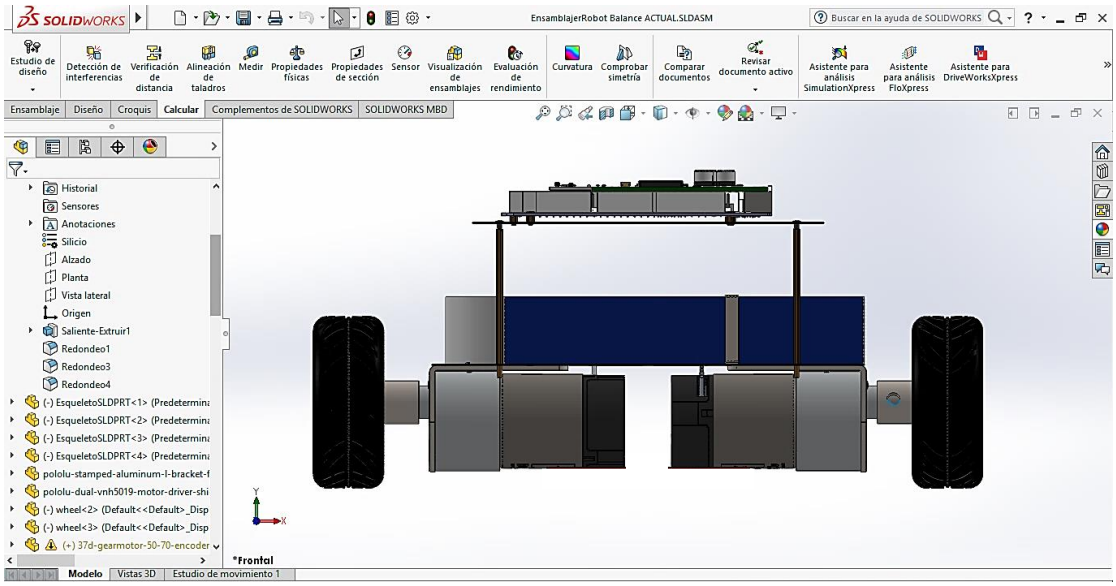


Figura 26. Diseño del péndulo.

Figura 27. Parámetros del péndulo.

Se procede a realizar la medición para verificar la aproximación de las partes que conforman al péndulo dando como resultado que son muy próximas a los valores mostrados por el CAD $(M_p = 1037.02\ gr)$ de la figura 29.

Podemos mencionar que los resultados obtenidos del centro de masa en el CAD muestran los datos de momento de inercia del péndulo, donde el eje de giro es paralelo al eje del Encoder. [1]

$$I_p = 0.00669218454\ kg * m^2$$

3.12 Resumen de parámetros

En la siguiente tabla podemos apreciar los parámetros para los modelos a utilizar en el bosquejo de los controladores. Considerando que se realizaron ciertos ajustes en las cifras para la validación del modelo para obtener una mejor respuesta. [1]

Descripción	Símbolo	Valores
Masa del péndulo	M_p	$1.037\ kg$
Masa de la rueda	M_R	$0.047\ kg$
Inercia del péndulo	I_p	$0.0066921845\ kg * m^2$
Inercia de la rueda	I_R	$0.003348934\ kg * m^2$
Radio de la rueda	r	$0.033\ m$
Constante de torque	K_m	$0.03\ Nm/A$
Gravedad	g	$9.81\ m/s^2$
Longitud al centro de masa	l	$0.01813\ m$
Constante contra electromotriz	K_e	$0.03\ Vs/rad$
Resistencia de devanado de motor	R_a	$4.54\ \Omega$

Tabla 4. Parámetros del robot construido.

3.13 Validación del modelo matemático del sistema

Para la validación del modelo se realiza varias pruebas con el propósito de observar la cercanía que tiene el modelo Real con lo obtenido. Se procede a generar una indicación de prueba para observar que tanto se aproxima con la respuesta Real; cabe mencionar que esta prueba fue realizada con los motores desconectados del sistema. [1]

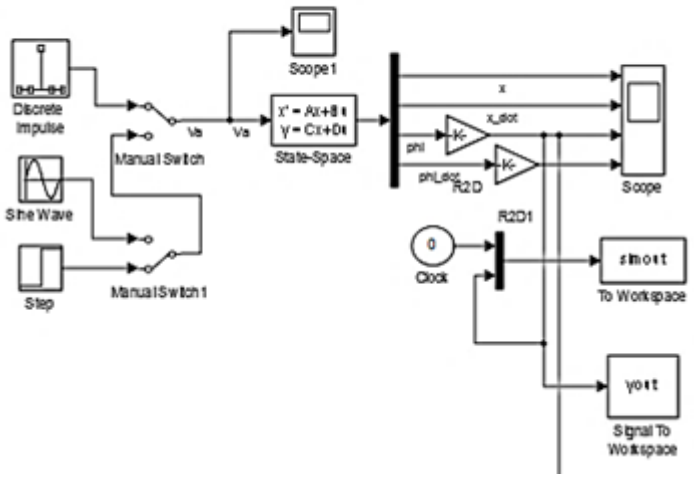


Figura 28. Simulación del modelo analítico del sistema

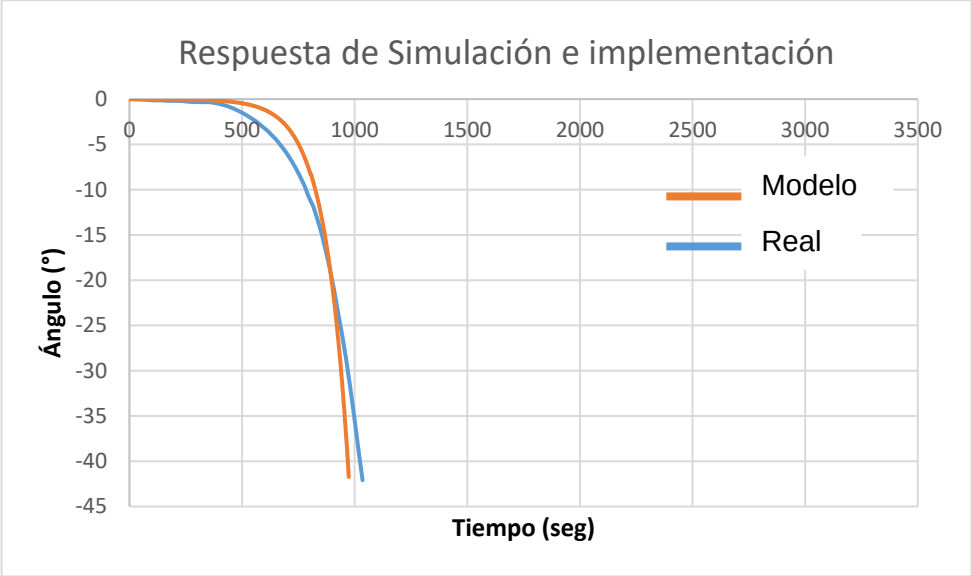


Figura 29. Validación del sistema en lazo abierto ángulo vs tiempo

3.14 Diseño para sistema de control PID

Partiendo del modelo matemático establecido anteriormente se procedió a efectuar un control de estados por retroalimentación para alcanzar el equilibrio del péndulo. El controlador que se propuso es un controlador PID convencional. Partiendo de la función de transferencia obtenida en espacio de estado. [1] [4]

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\emptyset} \\ \ddot{\emptyset} \end{bmatrix} \begin{bmatrix} 0 & 0 & 0.0516 & 0 \\ 0 & 0.0516 & -0 & -1.3394 \\ 0 & 0 & -1.7621 & -0 \\ 0 & -1.7621 & -0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \emptyset \\ \dot{\emptyset} \end{bmatrix} + \begin{bmatrix} 1 \\ 0.0469 \\ -24.0298 \\ -1.2176 \end{bmatrix} V_a \tag{37}$$

$$G(s) = \frac{-1.7621s^2 - s}{s^4 + 0.0469s^3 - 24.0298s^2 - 1.2176s}$$

Se realiza la simulación para el controlador PID considerando que el sistema es inestable, implementando la herramienta SISOTOOL, tomando en cuenta el driver o circuito de potencia se plantea como una ganancia de -12 como se estima en la figura 32. [1]

```
%% Análisis en lazo cerrado del controlador PID
Gain = -12;
%Ganancia aproximada en la tarjeta de
%potencia
sisotool(Gain*G1);
```

Figura 30. Circuito de potencia como ganancia.

El controlador PID presenta un distintivo rechazo a la perturbación con el propósito de mantener el ángulo en (θ°). [1]

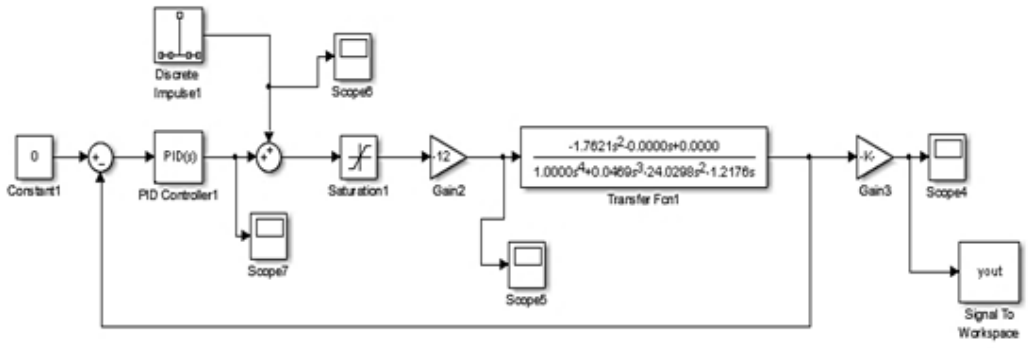


Figura 31. Diagrama para la simulación del controlador PID

Para la construcción del controlador PID se considera requisitos de diseño tales como, tiempo de asentamiento 4 segundos y un porcentaje de sobrepaso de 5% usando el método del lugar geométrico de las raíces se ajusta el controlador para hallar los respectivos valores de Kp, Ki, Kd comenzando con los polos y ceros obtenidos como se muestran en las figuras. [1]

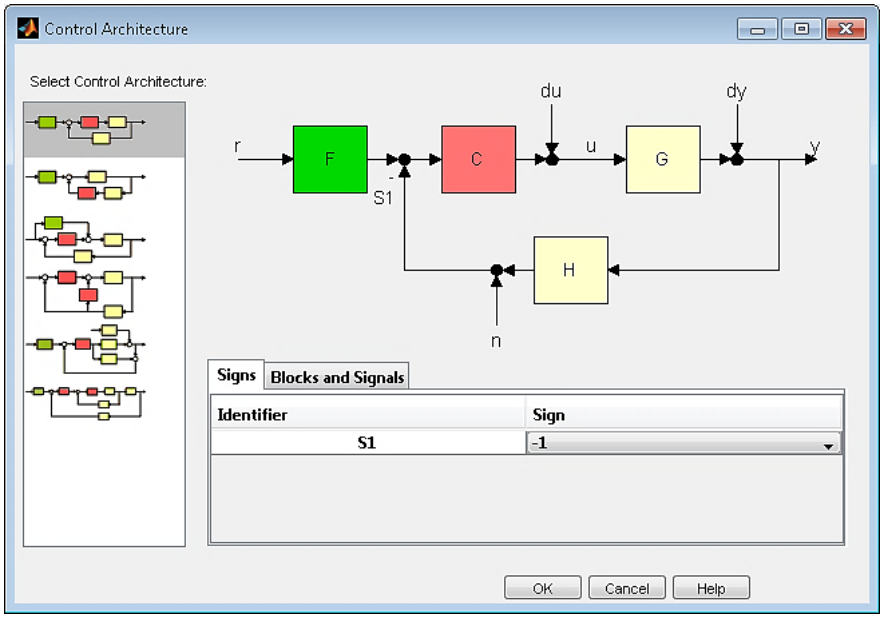


Figura 32. Arquitectura de control para el equilibrio.

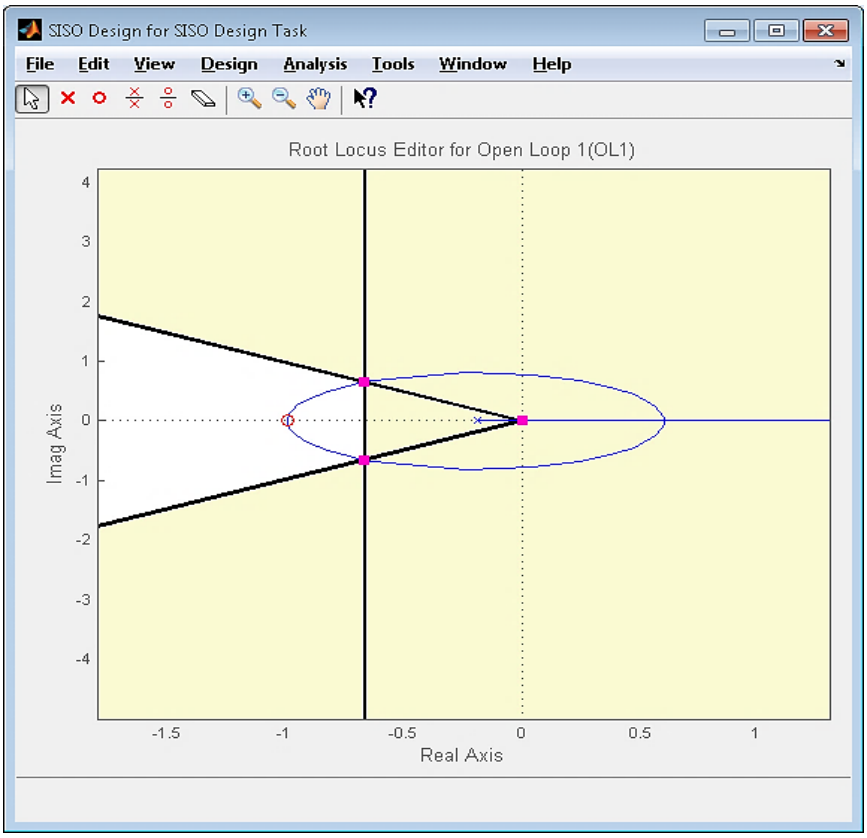


Figura 33. Lugar geométrico de las raíces del controlador

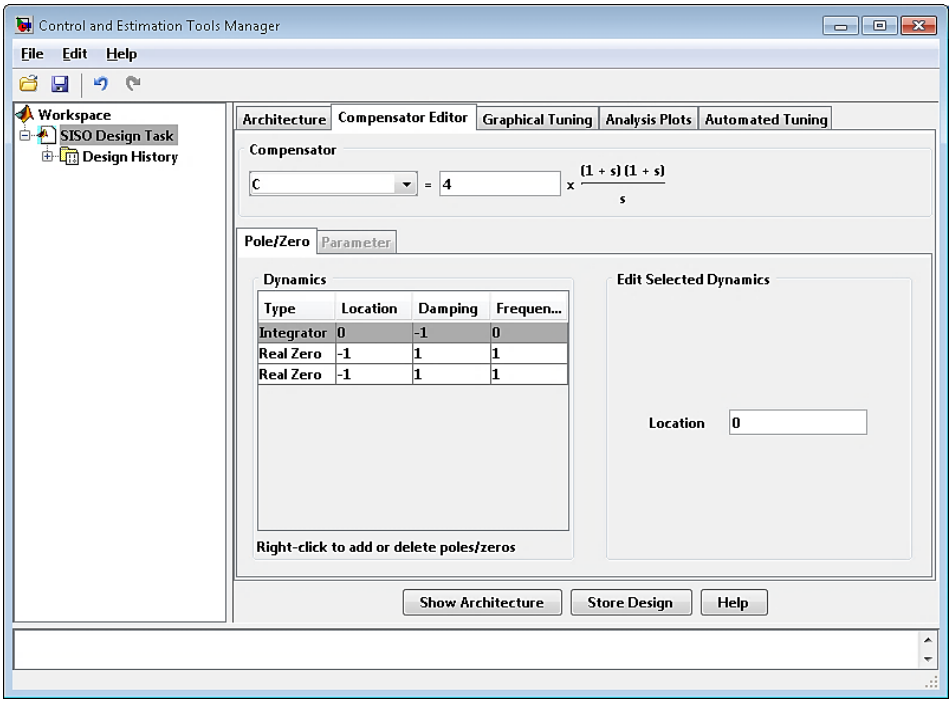


Figura 34. Compensador para el controlador PID.

Utilizando el software de Matlab 2013 se realiza la simulación para observar la respuesta como se aprecia en la figura 37:

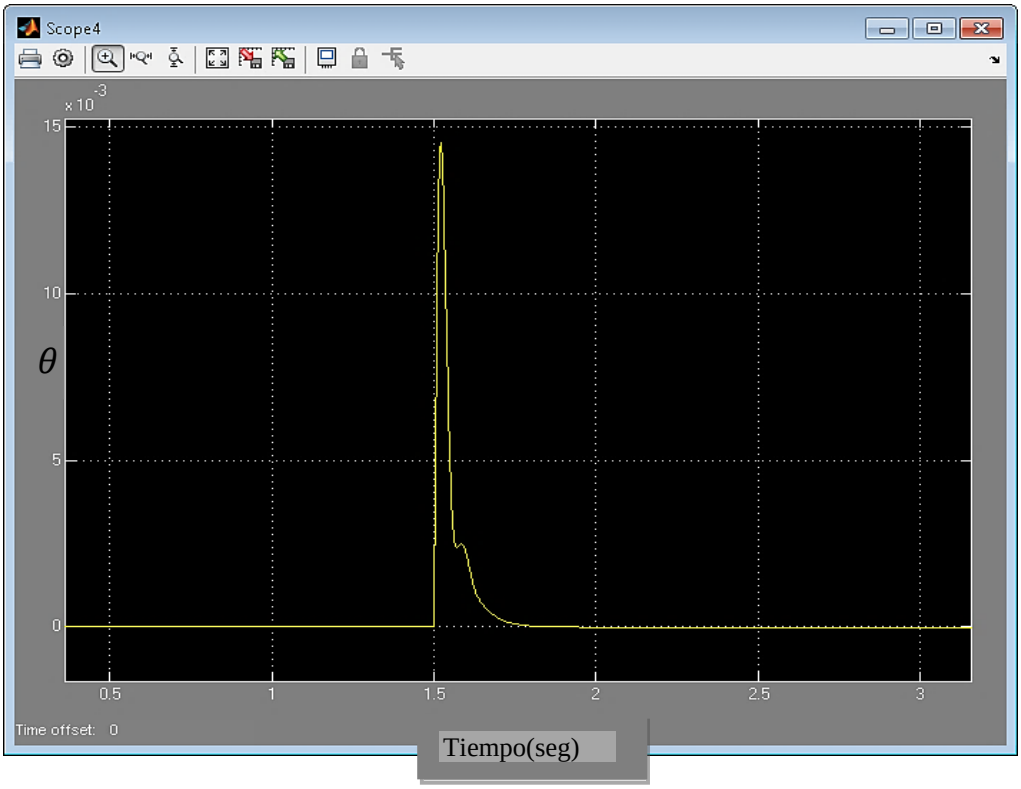


Figura 35. Respuesta de la simulación del controlador PID.

3.15 Diseño del controlador difuso.

Para ejecutar el diseño del controlador difuso, se necesita identificar cuantas entradas y salidas tiene el sistema para poder dimensionar correctamente el controlador que utilizaremos.

Se considera un controlador difuso donde se ubican 7 conjuntos para las variables que se asignaran a las entradas. Cada conjunto posee una abreviatura representada con letras NB, NM, NS, ZE, PS, PM, PB cuyo significado se detalla a continuación: [8]

NB	Negativo Grande
NM	Negativo Mediano
NS	Negativo Pequeño
ZE	Cero
PS	Positivo Pequeño
PM	Positivo Mediano
PB	Positivo Grande

Tabla 5. Variables lingüísticas.

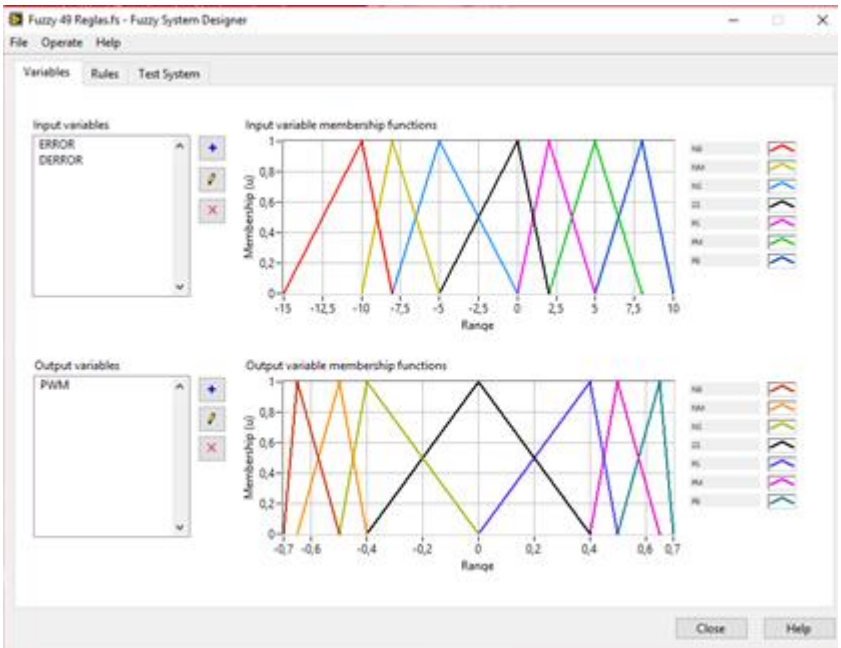


Figura 36. Designación de entradas y salida.

En el controlador se plantean 49 reglas donde se designan funciones de pertenencias triangulares usando la herramienta fuzzy System Designer de Labview 2013.

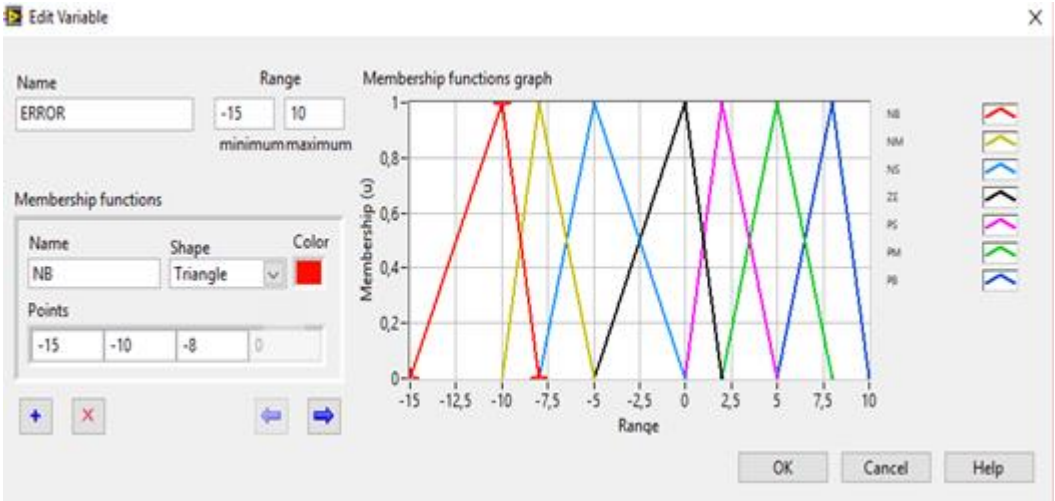


Figura 37. Función de pertenencia para la entrada Error.

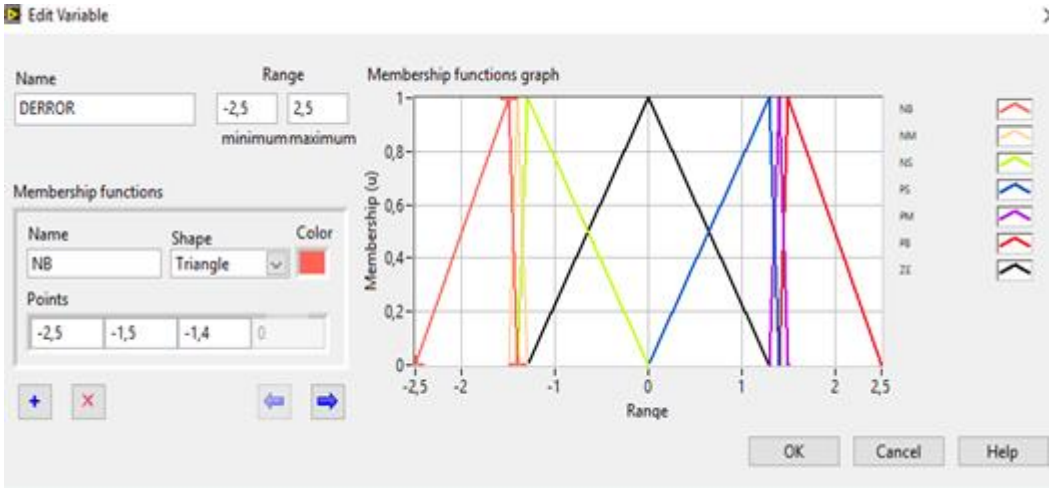


Figura 38. Función de membrecía para la entrada dError.

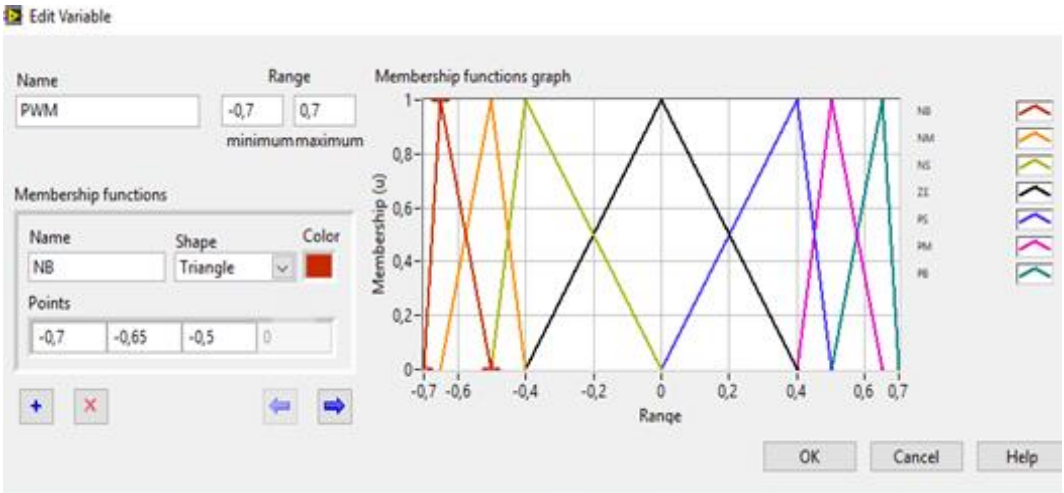


Figura 39. Función de pertenencia de la salida PWM.

La elaboración de reglas se utiliza en base a un controlador difuso convencional

		dError						
		NB	**NM**	**NS**	**ZE**	**PS**	**PM**	**PB**
	NB	NB	NB	NB	NB	NM	NS	ZE
	NM	NB	NB	NB	NM	NS	ZE	PS
	NS	NB	NB	NM	NS	ZE	PS	PM
Error	**ZE**	NB	NM	NS	ZE	PS	PM	PB
	PS	NM	NS	ZE	PS	PM	PB	PB
	PM	NS	ZE	PS	PM	PB	PB	PB
	PB	ZE	PS	PM	PB	PB	PB	PB

Tabla 6. Reglas para el control Difuso.

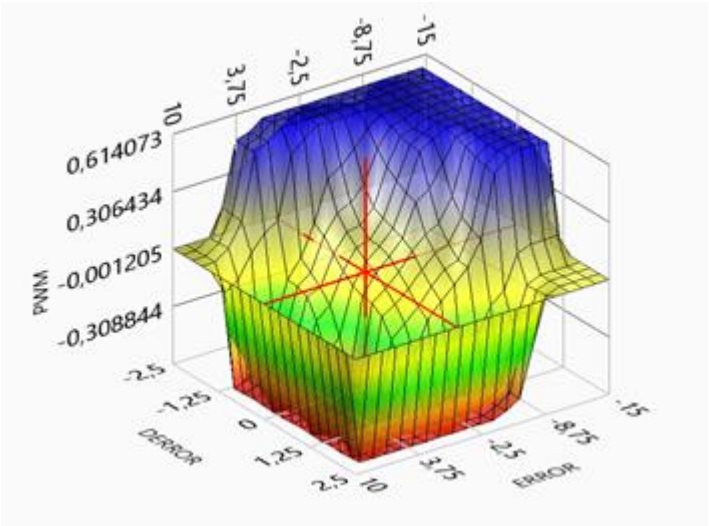


Figura 40. Superficie del controlador Difuso.

Para la demostración del controlador difuso se utiliza el software de Matlab y la herramienta Simulink para obtener la respuesta para la simulación del sistema.

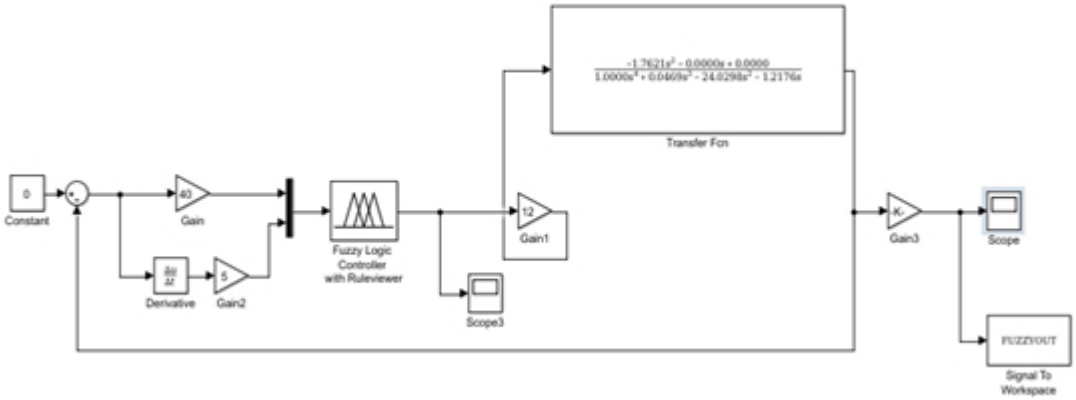


Figura 41. Diseño del bloque de simulación del controlador Difuso.

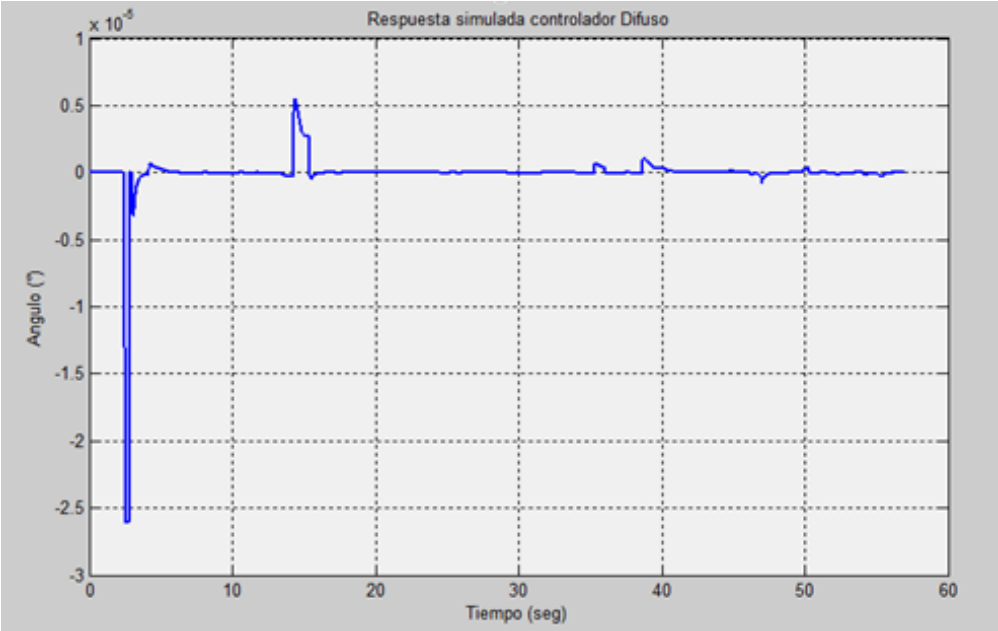


Figura 42. Respuesta de la simulación del controlador Difuso.

4. MANUAL DE PRÁCTICAS

4.1 Práctica # 1.

UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR		**GUÍA DE PRÁCTICA**
CARRERA: Ingeniería Electrónica		**ASIGNATURA:** Electiva I
N° DE PRÁCTICA	**1**	**TÍTULO DE LA PRÁCTICA:** "Interacción de sensores y actuadores."

Objetivos:

- Conocer los componentes del robot balance.
- Implementar un código de programación para realizar la interacción de sensores y actuadores del robot balance.

INSTRUCCIONES:	**1. Energizar el robot balance con la batería LiPo 11.1 VDC.**
	2. Verificar la conexión del sensor MPU 6050 y el módulo LC STUDIO SD a la tarjeta micro controladora Arduino mega 2560.
	3. Configurar el código en el software de Arduino.
	4. Compilar el código en Arduino mega 2560.

ACTIVIDADES POR DESARROLLAR

En esta práctica se va a establecer la interacción de sensores y actuadores por los cuales está compuesto el robot balance.

Para realizar la práctica necesitaremos que estén interconectados los sensores y actuadores del robot balance.

- Sensor MPU 6050.
- Módulo LC STUDIO SD.

Descripción de la práctica.

Se realiza la conexión del sensor MPU6050 y el módulo LC Studio SD con la tarjeta micro controladora Arduino mega 2560, se configura el módulo LC STUDIO para que haya comunicación con el módulo MPU 6050 , por medio del software de Arduino.

Paso 1.

Realizar las conexiones de los módulos con la tarjeta micro controladora Arduino mega como se aprecia en la Figura 45:

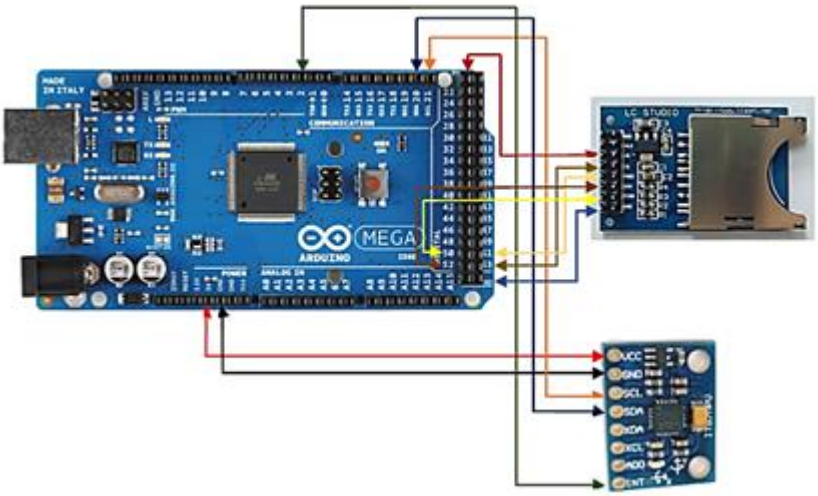


Figura 43. Interconexión ARDUINO MEGA, MPU, LC STUDIO.

Tabla de Conexiones		
Arduino	**Color**	**MPU**
5V		Vcc
GND		GND
SCL 21		SCL
SDA 20		SDA
2		INT

Arduino	**Color**	**LC STUDIO**
5V		5V
53		CS
51		MOSI
52		SCK
50		MISO
GND		GND

Tabla 7. Conexiones de tarjeta Arduino y módulos.

Paso 2.

Crear un Archivo en el Software Arduino para leer datos e incluyendo las librerías a utilizar:

```
prueba_con_lectura_serial_mpu6050
#include <TimerOne.h>
#include <Wire.h>
#include <SD.h>
#include <SPI.h>
//Direccion I2C de la IMU
#define MPU 0x68

#define A_R 16384.0*
#define G_R 131.0
```

Figura 44. Código de librerías Arduino.

Paso 3.

Crear un archivo para almacenar datos y realizar la transformación de radianes a grados:

```
//Conversion de radianes a grados 180/PI
#define RAD_A_DEG = 57.2957790.

File myFile; // Creación del archivo myFile para almacenar los datos
```

Figura 45. Código de creación de archivo.

Paso 4.

Declaración de variables para la leer datos del acelerómetro y giroscopio:

```
//Declaración de las variables a usar
 int16_t AcX, AcY, AcZ, GyX, GyY, GyZ;

  volatile float Acc[2];
  volatile float Gy[2];
  volatile float Angle[2];
```

Figura 46. Variables del acelerómetro y Giroscopio.

Paso 5.

Asignación del pin para la inicialización de la lectura de datos con mensajes de confirmación exitosa o de error:

```
void setup()
{
  Serial.begin(115200); //Velocidad del serial e inicialización
  Serial.print("Iniciando SD ..."); //Mensaje de inicio de lectura del Módulo Lc Studio SD
  if (!SD.begin(4)) {               //Asignación del pin el cual inicializa el módulo
    Serial.println("No se pudo inicializar"); //Mensaje de error al haber un problema de inicio con el Módulo Lc Studic
    return;
  }
  Serial.println("inicializacion exitosa"); //Mensaje de inicio exitoso del Módulo Lc Studio SD

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
 if(!SD.exists("datalog.csv"))
  {
      myFile = SD.open("datalog.csv", FILE_WRITE);    //
      if (myFile) {
        Serial.println("Archivo nuevo, Escribiendo encabezado(fila 1)");//Mensaje de las variables que se van a adquiri
        myFile.println("Tiempo(ms),Angle[0]");
        myFile.close(); // cierra el archivo
      } else {

        Serial.println("Error creando el archivo datalog.csv");   //Mensaje de error
      }
  }
```

Figura 47. Inicialización y pin de conexión.

Paso 6.

Inicialización del módulo MPU 6050 y del serial con la función Wire para leer los datos del acelerómetro:

```
////////////////////////////////////////////////////////////////////////////////
//Inicialización del MPU6050 usando la función Wire
Wire.begin();
Wire.beginTransmission(MPU);
Wire.write(0x6B);
Wire.write(0);
Wire.endTransmission(true);
Serial.begin(115200); // Inicio del serial
}


////////////////////////////////////////////////////////////////////////////////
void loop()
{

////////////////////////////////////////////////////////////////////////////////
//Leer los valores del Acelerometro de la IMU
   Wire.beginTransmission(MPU);
   Wire.write(0x3B); //Pedir el registro 0x3B - corresponde al AcX
   Wire.endTransmission(false);
   Wire.requestFrom(MPU,6,true); //A partir del 0x3B, se piden 6 registros
   AcX=Wire.read()<<8|Wire.read(); //Cada valor ocupa 2 registros
   AcY=Wire.read()<<8|Wire.read();
   AcZ=Wire.read()<<8|Wire.read();
```

Figura 48. Inicialización modulo MPU y serial.

Paso 7.

Una vez obtenidos los datos del acelerómetro se realiza el cálculo matemático para la tangente llevando la lectura del ángulo de radianes a grados:

```
//A partir de los valores del acelerómetro, se calculan los angulos Y, X
//respectivamente, con la formula de la tangente.
 Acc[1] = atan(-1*(AcX/A_R)/sqrt(pow((AcY/A_R),2) + pow((AcZ/A_R),2)))*RAD_TO_DEG;
 Acc[0] = atan((AcY/A_R)/sqrt(pow((AcX/A_R),2) + pow((AcZ/A_R),2)))*RAD_TO_DEG;
```

Figura 49. Fórmula para tangente del acelerómetro.

Paso 8.

Lectura de datos del giroscopio para realizar el cálculo del ángulo y aplicación del filtro complementario para la rechazo de ruidos:

```
//Leer los valores del Giroscopio
Wire.beginTransmission(MPU);
Wire.write(0x43)
Wire.endTransmission(false);
Wire.requestFrom(MPU,4,true); //A diferencia del Acelerometro, solo se piden 4 registros
GyX=Wire.read()<<8|Wire.read();
GyY=Wire.read()<<8|Wire.read();

//Calculo del angulo del Giroscopio
Gy[0] = GyX/G_R;
Gy[1] = GyY/G_R;

//Aplicar el Filtro Complementario
Angle[0] = 0.98 *(Angle[0]+Gy[0]*0.010) + 0.02*Acc[0];
Angle[1] = 0.98 *(Angle[1]+Gy[1]*0.010) + 0.02*Acc[1];
```

Figura 50. Lectura del giroscopio y cálculo del ángulo.

Paso 9.

Se abre el archivo de datalog.csv para el inicio de la escritura de datos, junto a la función millis para el monitoreo del tiempo donde se obtienen y se guardan datos del ángulo y tiempo con el respectivo cierre del archivo:

```
myFile = SD.open("datalog.csv", FILE_WRITE);//Abrimos  el archivo datalog.csv

if (myFile) {

    Serial.print("Escribiendo SD:"); //Mensaje de inicio de escritura de datos en el archivo

    myFile.print(millis());  //Uso de la función millis para monitorear el tiempo en que se adquiere cada dat
    myFile.print(",");
    // myFile.print(", sensor1=");
    myFile.println(Angle[0]);  // Adquisición de la variable Angle[0]

    myFile.close(); //cerramos el archivo
```

Figura 51. Inicio a la escritura y monitoreo del tiempo.

Paso 10.

Para dar seguimiento a la adquisición de datos y mostrar si hay complicaciones en la configuración se agrega una línea mostrando un error al abrir el archivo y posterior a eso revisar detenidamente el código para hacer las correcciones respectivas:

```
    }

    else {
      Serial.println("Error al abrir el archivo"); //Mensaje de error
    }

  }
```

Figura 52. Línea de error en el archivo.

Resultados obtenidos

Para visualizar los resultados obtenido en la práctica desarrollada nos dirigimos al monitor serial del programa de Arduino para verificar los datos del robot con los ángulos de inclinación en la cual se encuentra.

```
Tiempo(ms)= 101662  Angulo X=0.94
Tiempo(ms)= 101698  Angulo X=0.86
Tiempo(ms)= 101735  Angulo X=0.75
Tiempo(ms)= 101771  Angulo X=0.68
Tiempo(ms)= 101808  Angulo X=0.81
Tiempo(ms)= 101843  Angulo X=0.93
Tiempo(ms)= 101880  Angulo X=0.98
Tiempo(ms)= 101916  Angulo X=0.95
Tiempo(ms)= 101953  Angulo X=0.92
Tiempo(ms)= 101989  Angulo X=0.89
Tiempo(ms)= 102026  Angulo X=0.84
Tiempo(ms)= 102063  Angulo X=0.74
Tiempo(ms)= 102098  Angulo X=0.70
Tiempo(ms)= 102135  Angulo X=0.62
```

Figura 53. Datos del ángulo de inclinación 0°.

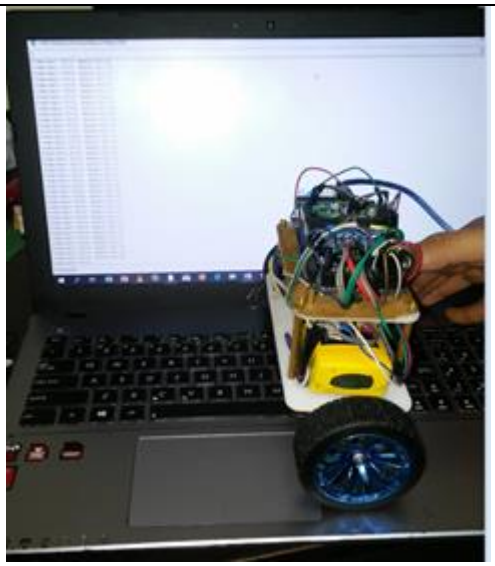


Figura 54. Robot balance con ángulo de inclinación igual a 0°.

```
Tiempo(ms)= 106966  Angulo X=13.26
Tiempo(ms)= 107003  Angulo X=13.41
Tiempo(ms)= 107040  Angulo X=13.57
Tiempo(ms)= 107078  Angulo X=13.73
Tiempo(ms)= 107115  Angulo X=13.90
Tiempo(ms)= 107153  Angulo X=14.03
Tiempo(ms)= 107191  Angulo X=14.17
Tiempo(ms)= 107228  Angulo X=14.30
Tiempo(ms)= 107266  Angulo X=14.43
Tiempo(ms)= 107302  Angulo X=14.54
Tiempo(ms)= 107340  Angulo X=14.67
Tiempo(ms)= 107377  Angulo X=14.80
Tiempo(ms)= 107415  Angulo X=14.93
Tiempo(ms)= 107453  Angulo X=15.03
Tiempo(ms)= 107490  Angulo X=15.10
Tiempo(ms)= 107528  Angulo X=15.18
```

Figura 55. Datos del ángulo de inclinación a 15°.

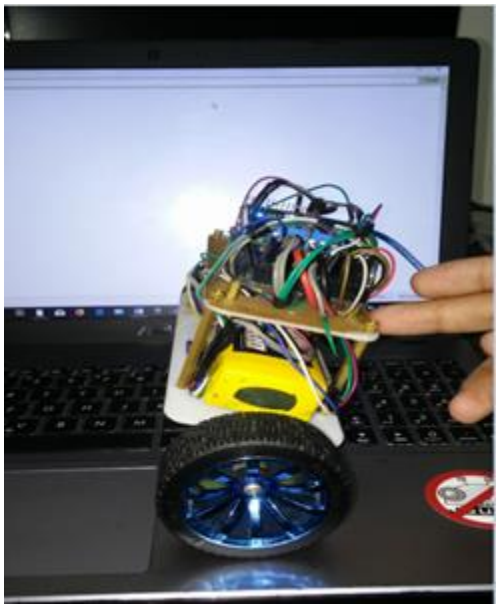


Figura 56. Robot balance con ángulo de inclinación a 15°.

4.2 Práctica # 2.

UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR			**GUÍA DE PRÁCTICA**
CARRERA: Ingeniería Electrónica			**ASIGNATURA:** Electiva I
N° DE PRÁCTICA :	**2**	**TÍTULO DE LA PRÁCTICA:** "Adquisición de datos en tiempo real "	

Objetivos:

- Elaborar un código de programación que permita adquirir datos en tiempo real.
- Diseñar una interfaz en Labview que nos permita monitorear la gráfica del robot balance.

INSTRUCCIONES:	**1. Elaborar una interfaz de Labview para monitorear la gráfica del robot.**
	2. Realizar las conexiones del módulo bluetooth con la tarjeta micro controladora Arduino mega.
	3. Energizar el robot con la batería de 11.1 VDC.
	4. Elaborar un código de programación en el software de Arduino.
	5. Compilar y cargar el programa a la tarjeta Arduino mega.

ACTIVIDADES POR DESARROLLAR

En esta práctica se va a diseñar una interfaz en Labview que nos permita el monitoreo y adquisición de datos del robot balance.

Realizar la conexión del módulo bluetooth con la tarjeta micro controladora Arduino mega para después energizar el robot.

Elaborar un código de programación en el software de Arduino que nos permita comunicar al bluetooth con el interfaz de Labview.

Visualizar en la interfaz de Labview la respuesta obtenida tanto por el puerto serial y por bluetooth.

Observar la velocidad de transmisión y adquisión de datos por los dos métodos.

Paso 1.

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

Figura 57. Panel frontal del interfaz.

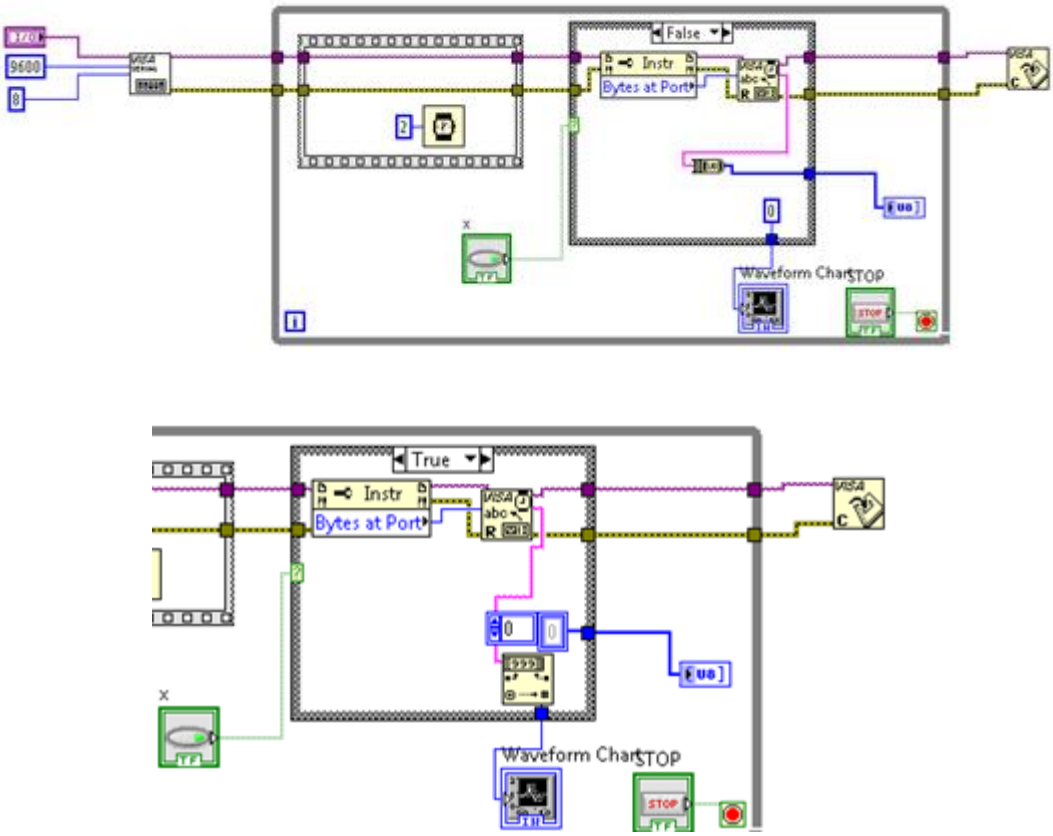


Figura 58. Diagrama de bloques.

Paso 2.

Realizar la conexión del módulo bluetooth con la tarjeta micro controladora Arduino mega como se aprecia en la figura 57:

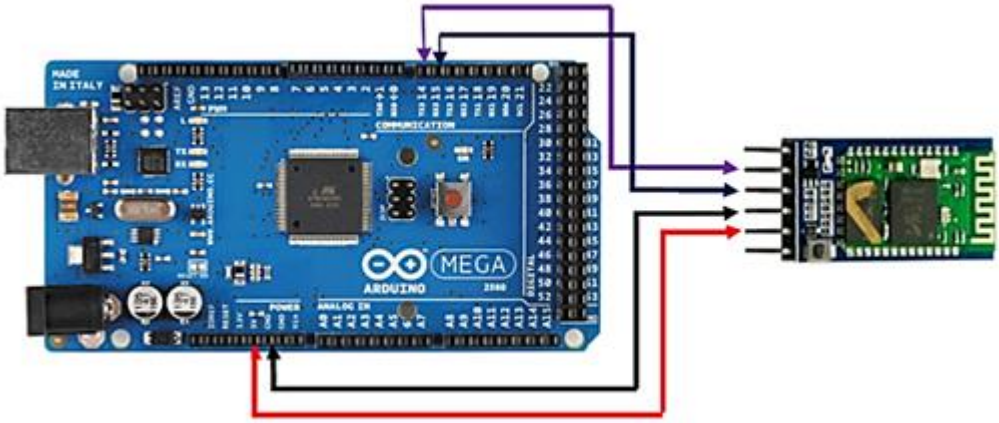


Figura 59. Interconexión de la tarjeta Arduino mega y Bluetooth.

Tabla de Conexiones		
Arduino	**Color**	**Bluetooth**
5V		5V
GND		GND
TX		RX
RX		TX

Tabla 8. Conexión de Arduino mega con Bluetooth.

Paso 3.

Elaborar e implementar un código de programación en Arduino para enviar datos en tiempo real a la interfaz de Labview.

Agregar las librerías partiendo de condiciones para las variables a utilizar incluyendo el filtro Kalman para la eliminación de ruidos e interferencias del robot como se aprecia a continuación:

```
////// Llamado de las librerias a usar
#define ENCODER_OPTIMIZE_INTERRUPTS
#define RESTRICT_PITCH
#include <Wire.h>
#include <Kalman.h>
#include <I2Cdev.h>
#include <MPU6050.h>
#include <DualVNH5019MotorShield.h>

////// Asignación de un valor inicial a una variable creada

#define Gry_offset 0  //offset del giroscopio
#define Gyr_Gain 131   //ganancia usada en el giroscopio
#define Angle_offset 0  // offset del ángulo
#define RMotor_offset 0  // offset del motor lado derecho
#define LMotor_offset 0  // offset del motor lado izquierdo
#define pi 3.14159       // valor dePI
#define DMotor_offset 0   //offset PWM derecho
#define IMotor_offset 0   //offset PWM izquierdo
#define targetAngle 0    // setpoint
#define sampleTime  0.001 // tiempo de muestreo

Kalman kalmanX; // FILTRO KALMAN
Kalman kalmanY;
Kalman kalmanZ;
```

Figura 60. Creación de librerías y variables para transmisión de datos.

Paso 4.

Asignación de pines para la interconexión de los motores y crear las variables para usar el sensor MPU.

```
//Asignación de pines para conexión del motor
int MotorDer1=7;   //// PINES DEL MOTOR 2 EN EL ROBOT (MOTOR DERECHO)
int MotorDer2=8;
int MotorIzq1=2;   //// PINES DEL MOTOR 1 EN EL ROBOT (MOTOR IZQUIERDO)
int MotorIzq2=4;
int PWM_Derecho=10;   //PIN PMW DERECHO
int PWM_Izquierdo=9;  //PIN PWM IZQUIERDO

int valor = 0;   // Variable en la que se guarda el valor leido del ADC 0 que pertenece al LM35
int select = 0; // Variable en la cual se guarda el caracter enviado del LabView y así leer los sensores

/// CREACIÓN DE VARIABLES A USAR DEL SENSOR MPU 6050
MPU6050 accelgyro;
double accX, accY, accZ;
double gyroX, gyroY, gyroZ;
int ax, ay, az;
int gx, gy, gz;

//CREACIÓN DE VARIABLES
int total;
float speedDifference;
```

Figura 61. Asignación de pines para interconexión de motores.

Paso 5.

Crear las variables para asignar la velocidad de los motores y las del controlador PID con la inicialización del Driver:

```
////CREACIÓN DE VARIABLES A USAR
int velocidad=150;//Declaramos una variable para guardar la velocidad
float Sum_derecha,Sum_izquierda,Speed;
double gyroXangle, gyroYangle;
double compAngleX, compAngleY;
double kalAngleX, kalAngleY;
float Sum_Right, Sum_derecha_Temp, Sum_Left, Sum_izquierda_Temp, Distancia, Distance_Right, Distance_Left;
float Turn_Speed = 0, Run_Speed = 0;
float IOutput, DOutput, Input, Output;
uint32_t timer;

// CREACIÓN DE VARIABLES DEL PID
unsigned long preTime, lastTime;
float errSum, dErr, error, lastErr;
int timeChange;

float Angle_Delta, Angle_Recursive, Angle_Confidence;
float Angle_Raw, Angle_Filtered, omega, dt;
volatile float prevAngle=0;
int Angle_Filtered1;


DualVNH5019MotorShield motorDriver; // INICIALIZACIÓN DEL DRIVER VNH5019
```

Figura 62. Variables de velocidad de motores y del controlador.

Paso 6.

Asignar los pines de salida para los motores y la velocidad del serial para realizar la comunicación del Bluetooth e inicializar al sensor MPU:

```
void setup()
{
   Serial.begin(9600); //ASIGNACIÓN DE VELOCIDAD E INICIO DEL SERIAL
   Wire.begin();

 //ASIGNACIÓN DE PINES DE SALIDA
  pinMode(MotorDer1, OUTPUT);
  pinMode(MotorDer2, OUTPUT);
  pinMode(MotorIzq1, OUTPUT);
  pinMode(MotorIzq2, OUTPUT);
  pinMode(PWM_Derecho, OUTPUT);
  pinMode(PWM_Izquierdo, OUTPUT);

 accelgyro.initialize(); //INICIALIZACIÓN DEL MPU6050
 }
```

Figura 63. Asignación de pines salida para los motores.

Paso 7.

Realizar la declaración para el uso de las sentencias para llamar a las variables del controlador, MPU y PWM asignando la transformación de radianes a grados con la función arco tangente como se aprecia a continuación:

```
/// USO DE UNA SENTENCIA WHILE PARA LLAMAR AL CONTROLADOR PID
 while(1)
 {

   filter();  //LLAMADA DE LA LECTURA DEL MPU 6050
   myPID();   // LLAMADA DEL CONTROLADOR PID
   PWMControl(); // LLAMADA PARA EL CONTROL DEL PWM

    if (abs(Angle_Filtered) > 30)  /// CONDICIÓN QUE DESACTIVA Y APAGA LOS MOTORES AL LLEGAR EL ÁNGULO A VALORES MAYORES A 30 GRADOS
    {
      motorDriver.setSpeeds(0, 0); // SETEO DE VELOCIDAD EN 0
      }

     }
       }

/// CONVERSIÓN POR EL MÉTODO ARCO TANGENTE DE LOS VALORES DEL MPU 6050 DE RAD. A GRADOS
void filter () {

  accelgyro.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
  Angle_Raw = (atan2(ay, az) * 180 / pi + Angle_offset);
  omega = gx / Gyr_Gain + Gry_offset;

  unsigned long now = micros();
  timeChange = now - preTime;
  preTime = now;


  dt = timeChange * 0.000001;
  Angle_Delta = (Angle_Raw - Angle_Filtered) * 0.64;
  Angle_Recursive = Angle_Delta * dt + Angle_Recursive;
  Angle_Confidence = Angle_Recursive + (Angle_Raw - Angle_Filtered) * 1.6 + omega;
  Angle_Filtered = Angle_Confidence * dt + Angle_Filtered;
  Angle_Filtered1 = Angle_Filtered-1; // OFFSET UBICADO EN EL ÁNGULO DE -1 PARA LOGRAR EL SETPOINT
```

Figura 64. Sentencias para llamar a variables del controlador.

Paso 8.

Usar la función serial print para concadenar la variable del ángulo filtrado y pueda ser enviada por medio del Bluetooth a la interfaz de Labview.

Asignar los valores del controlador PID tales como Kp, Ki, Kd para realizar el cálculo del error y dError para que puedan ser mostrados en la salida del controlador:

```
    Serial.println(Angle_Filtered1); //VISUALIZACIÓN EN EL SERIAL DE LA VARIABLE Angle_Filtered1
  //Serial2.println(speedDifference); //VISUALIZACIÓN EN EL SERIAL DE LA VARIABLE speedDifference

  }

  //CREACIÓN DEL CONTROLADOR PID

  void myPID() {

     //VALORES DE KP,KI,KD
     int kp = 80;
     int ki = 1;
     int kd = 4;

    double error = Angle_Filtered1 - targetAngle; //VARIABLE ERROR
    errSum += errSum + error;
    errSum = constrain(errSum, -100, 100);
    double dErr = error - lastErr;    // VARIABLE DERIVADA DEL ERROR
    speedDifference = kp * (error) + ki * (errSum)*sampleTime + kd *omega; //CONTROLADOR PID
    prevAngle = Angle_Filtered1;
    lastErr = error;

     // SALIDA DEL CONTROLADOR PID
     DOutput = -speedDifference;
     IOutput = speedDifference;
   }
```

Figura 65. Función serial para enviar datos por Bluetooth.

Paso 9.

Crear las sentencias para el uso del PWM en los motores:

```
//CREACIÓN DE LA SENTENCIA PARA LOS MOTORES USANDO PWM

void PWMControl()
{
  if (IOutput > 0)
  {
    digitalWrite(MotorIzq1, HIGH);
    digitalWrite(MotorIzq2, LOW);

  }
  else if (IOutput < 0)
  {
    digitalWrite(MotorIzq1, LOW);
    digitalWrite(MotorIzq2, HIGH);
  }
  else
  {
   digitalWrite(MotorIzq1, HIGH);
   digitalWrite(MotorIzq2, HIGH);
  }
  if (DOutput > 0)
  {
    digitalWrite(MotorDer1, HIGH);
    digitalWrite(MotorDer2, LOW);
  }

  else if (DOutput < 0)
  {
    digitalWrite(MotorDer1, LOW);
    digitalWrite(MotorDer2, HIGH);
  }
  else
  {
    digitalWrite(MotorDer1, HIGH);
    digitalWrite(MotorDer2, HIGH);
  }

    analogWrite(PWM_Derecho,min(255,abs(IOutput*0.3) + IMotor_offset)); // PWM DERECHO
    analogWrite(PWM_Izquierdo,min(255,abs(DOutput*0.3) + DMotor_offset)); // PWM IZQUIERDO
}
```

Figura 66. Creación de sentencia para el PWM en los motores.

Resultados Obtenidos:

Luego de haber realizado el diseño de la interfaz procedemos a realizar la conexión con el robot asignando el puerto serial que se va a utilizar para la transmisión de datos en tiempo real del ángulo con respecto al tiempo del controlador PID.

Figura 67. Panel frontal con adquisición de datos 1

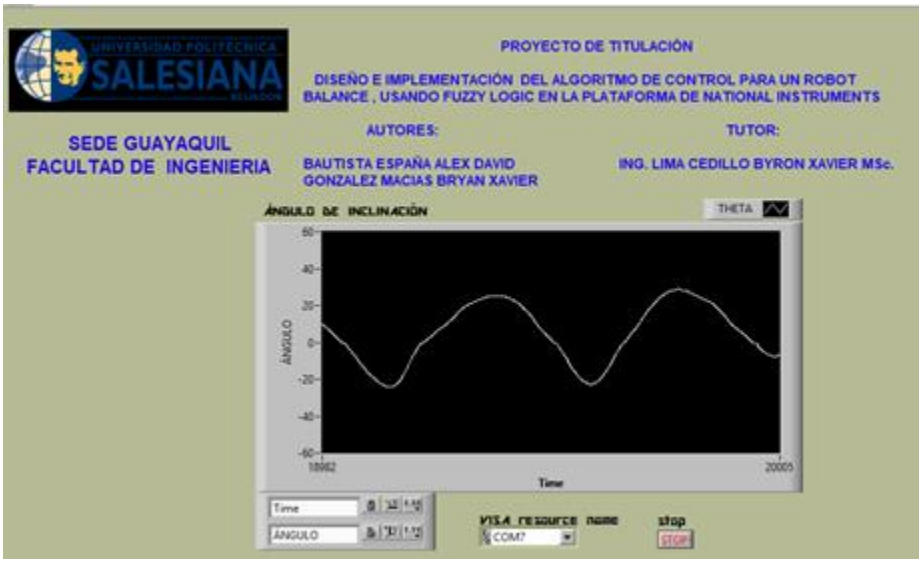


Figura 68. Panel frontal con adquisición de datos 2

4.3 Práctica # 3.

UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR		**GUÍA DE PRÁCTICA**
CARRERA: Ingeniería Electrónica		**ASIGNATURA:** Informática Industrial
N° DE PRÁCTICA	**3**	**TÍTULO DE LA PRÁCTICA:** "Diseño y prueba de controlador PID básico."

Objetivos:

- Diseñar un código para establecer un controlador PID.
- Presentar los resultados obtenidos del controlador.

INSTRUCCIONES:	**1. Energizar el robot balance con la batería de 11.1 VDC.**
	2. Elaborar el código en el software de Arduino.
	3. Compilar el código y subir a la tarjeta de Arduino mega 2560.
	4. Presentar los resultados que se obtienen en el desarrollo del programa.

ACTIVIDADES POR DESARROLLAR

La práctica se fundamenta en diseñar y posteriormente implementar de un controlador PID utilizando el software de Arduino.

Realizar la conexión de la tarjeta micro controladora Arduino con los demás componentes y proceder a energizar para poder así obtener las gráficas del controlador en tiempo real.

Paso 1.

Se abre el software de Arduino para diseñar el código y configurar las librerías a utilizar:

```
////// Llamado de las librerias a usar
#define ENCODER_OPTIMIZE_INTERRUPTS
#define RESTRICT_PITCH
#include <Wire.h>
#include <Kalman.h>
#include <I2Cdev.h>
#include <MPU6050.h>
#include <DualVNH5019MotorShield.h>
```

Figura 69. Código para incluir librerías.

Paso 2.

Definir parámetros iniciales para el giroscopio, motores, ángulos, ganancia y el filtro kalman para eliminar o interferencias :

```
////// Asignación de un valor inicial a una variable creada

#define Gry_offset 0  //offset del giroscopio
#define Gyr_Gain 131   //ganancia usada en el giroscopio
#define Angle_offset 0  // offset del ángulo
#define RMotor_offset 0  // offset del motor lado derecho
#define LMotor_offset 0  // offset del motor lado izquierdo
#define pi 3.14159       // valor dePI
#define DMotor_offset 0   //offset PWM derecho
#define IMotor_offset 0   //offset PWM izquierdo
#define targetAngle 0    // setpoint
#define sampleTime  0.001 // tiempo de muestreo

Kalman kalmanX; // FILTRO KALMAN
Kalman kalmanY;
Kalman kalmanZ;
```

Figura 70. Definición de parámetros iniciales para Motores.

Paso 3.

Delación de pines para la comunicación con los motores:

```
int MotorDer1=7;   //// PINES DEL MOTOR 2 EN EL ROBOT (MOTOR DERECHO)
int MotorDer2=8;
int MotorIzq1=2;   //// PINES DEL MOTOR 1 EN EL ROBOT (MOTOR IZQUIERDO)
int MotorIzq2=4;
int PWM_Derecho=10;   //PIN PMW DERECHO
int PWM_Izquierdo=9;  //PIN PWM IZQUIERDO
```

Figura 71. Definición de pines para motores.

Paso 4.

Definición de variables para comunicación con el módulo MPU 6050:

```
/// CREACIÓN DE VARIABLES A USAR DEL SENSOR MPU 6050
MPU6050 accelgyro;
double accX, accY, accZ;
double gyroX, gyroY, gyroZ;
int ax, ay, az;
int gx, gy, gz;

//CREACIÓN DE VARIABLES
int total;
float speedDifference;
```

Figura 72. Comunicación con el módulo MPU 6050.

Paso 5.

Creación de variables para controlar la velocidad y movimientos de los motores:

```
////CREACIÓN DE VARIABLES A USAR
int velocidad=150;//Declaramos una variable para guardar la velocidad
float Sum_derecha,Sum_izquierda,Speed;
double gyroXangle, gyroYangle;
double compAngleX, compAngleY;
double kalAngleX, kalAngleY;
float Sum_Right, Sum_derecha_Temp, Sum_Left, Sum_izquierda_Temp, Distancia, Distance_Right, Distance_Left;
float Turn_Speed = 0, Run_Speed = 0;
float IOutput, DOutput, Input, Output;
uint32_t timer;
```

Figura 73. Variables para velocidad de motores.

Paso 6.

Definición de variables para el controlador PID:

```
// CREACIÓN DE VARIABLES DEL PID
unsigned long preTime, lastTime;
float errSum, dErr, error, lastErr;
int timeChange;

float Angle_Delta, Angle_Recursive, Angle_Confidence;
float Angle_Raw, Angle_Filtered, omega, dt,Angle_Filteredl;
volatile float prevAngle=0;

DualVNH5019MotorShield motorDriver; // INICIALIZACIÓN DEL DRIVER VNH5019
```

Figura 74. Variables del PID.

Paso 7.

Definir la velocidad para el inicio del serial y el módulo MPU 6050:

```
void setup()
{
   Serial.begin(115200); //ASIGNACIÓN DE VELOCIDAD E INICIO DEL SERIAL
   Wire.begin();

 //ASIGNACIÓN DE PINES DE SALIDA
  pinMode(MotorDer1, OUTPUT);
  pinMode(MotorDer2, OUTPUT);
  pinMode(MotorIzq1, OUTPUT);
  pinMode(MotorIzq2, OUTPUT);
  pinMode(PWM_Derecho, OUTPUT);
  pinMode(PWM_Izquierdo, OUTPUT);

 accelgyro.initialize(); //INICIALIZACIÓN DEL MPU6050
 }
```

Figura 75. Definición de velocidad y MPU 6050.

Paso 8.

Definir la sentencia para llamar al controlador, PWM y lectura del MPU incluyendo condiciones de apagado cuando el ángulo tengo valores mayores a 30°:

```
void loop()
{

/// USO DE UNA SENTENCIA WHILE PARA LLAMAR AL CONTROLADOR PID
 while(1)
 {

   filter();  //LLAMADA DE LA LECTURA DEL MPU 6050
   myPID();   // LLAMADA DEL CONTROLADOR PID
   PWMControl(); // LLAMADA PARA EL CONTROL DEL PWM

    if (abs(Angle_Filtered) > 30)  /// CONDICIÓN QUE DESACTIVA Y APAGA LOS MOTORES AL LLEGAR EL ÁNGULO A VALORES MAYORES A 30 GRA
    {
      motorDriver.setSpeeds(0, 0); // SETEO DE VELOCIDAD EN 0
      }

     }
       }
```

Figura 76. Sentencia para controlador, PWM y MPU 6050.

Paso 9.

Realizar la transformación de radianes a grados utilizando métodos matemáticos como arco tangente para obtener datos en grados:

```
/// CONVERSIÓN POR EL MÉTODO ARCO TANGENTE DE LOS VALORES DEL MPU 6050 DE RAD. A GRADOS
void filter () {

  accelgyro.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
  Angle_Raw = (atan2(ay, az) * 180 / pi + Angle_offset);
  omega = gx / Gyr_Gain + Gry_offset;

  unsigned long now = micros();
  timeChange = now - preTime;
  preTime = now;
  dt = timeChange * 0.000001;
  Angle_Delta = (Angle_Raw - Angle_Filtered) * 0.64;
  Angle_Recursive = Angle_Delta * dt + Angle_Recursive;
  Angle_Confidence = Angle_Recursive + (Angle_Raw - Angle_Filtered) * 1.6 + omega;
  Angle_Filtered = Angle_Confidence * dt + Angle_Filtered;
  Angle_Filtered1 = Angle_Filtered-1; // OFFSET UBICADO EN EL ÁNGULO DE -1 PARA LOGRAR EL SETPOINT



  Serial.println(Angle_Filtered1); //VISUALIZACIÓN EN EL SERIAL DE LA VARIABLE Angle_Filtered1
  Serial2.println(speedDifference); //VISUALIZACIÓN EN EL SERIAL DE LA VARIABLE speedDifference

}
```

Figura 77. Transformación de radianes a grados.

Paso 10.

Asignación de valores para el PID y elaboración del mismo junto a su salida:

```
//CREACIÓN DEL CONTROLADOR PID

void myPID() {

   //VALORES DE KP,KI,KD
   int kp = 80;
   int ki = 1;
   int kd = 4;

  double error = Angle_Filtered1 - targetAngle; //VARIABLE ERROR
  errSum += errSum + error;
  errSum = constrain(errSum, -100, 100);
  double dErr = error - lastErr;    // VARIABLE DERIVADA DEL ERROR
  speedDifference = kp * (error) + ki * (errSum)*sampleTime + kd *omega; //CONTROLADOR PID
  prevAngle = Angle_Filtered1;
  lastErr = error;

   // SALIDA DEL CONTROLADOR PID
   DOutput = -speedDifference;
   IOutput = speedDifference;
```

Figura 78. Valores del PID.

Paso 11.

Creación de la sentencia para el accionar de los motores usando el PWM e identificando derecho e izquierdo:

```
  if (DOutput > 0)
  {
    digitalWrite(MotorDer1, HIGH);
    digitalWrite(MotorDer2, LOW);
  }
  else if (DOutput < 0)
  {
    digitalWrite(MotorDer1, LOW);
    digitalWrite(MotorDer2, HIGH);
  }
  else
  {
    digitalWrite(MotorDer1, HIGH);
    digitalWrite(MotorDer2, HIGH);
  }
    analogWrite(PWM_Derecho,min(255,abs(IOutput*0.3) + IMotor_offset)); // PWM DERECHO
    analogWrite(PWM_Izquierdo,min(255,abs(DOutput*0.3) + DMotor_offset)); // PWM IZQUIERDO
}
//CREACIÓN DE LA SENTENCIA PARA LOS MOTORES USANDO PWM

void PWMControl()
{
  if (IOutput > 0)
  {
    digitalWrite(MotorIzq1, HIGH);
    digitalWrite(MotorIzq2, LOW);

  }
  else if (IOutput < 0)
  {
    digitalWrite(MotorIzq1, LOW);
    digitalWrite(MotorIzq2, HIGH);
  }
  else
  {
   digitalWrite(MotorIzq1, HIGH);
   digitalWrite(MotorIzq2, HIGH);
  }
```

Figura 79. Sentencia de Motores.

Resultados obtenidos

Para visualizar los datos del ángulo del controlador PID se procede abrir el monitor serial teniendo en cuenta la línea de código del serial (Angle_filterd1) de la figura del paso 9:

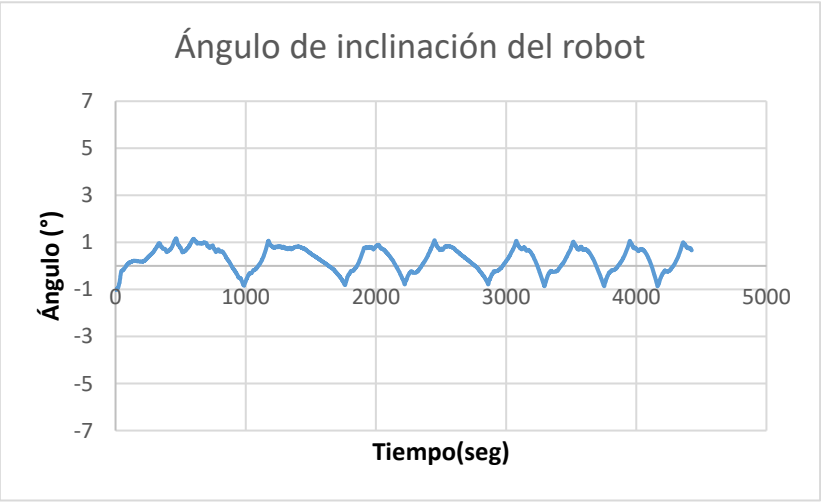


Figura 80. Comportamiento del ángulo con el controlador PID.

Para visualizar los datos del controlador PID se procede abrir el monitor serial teniendo en cuenta la línea de código del serial (speedDifference) de la figura 78:

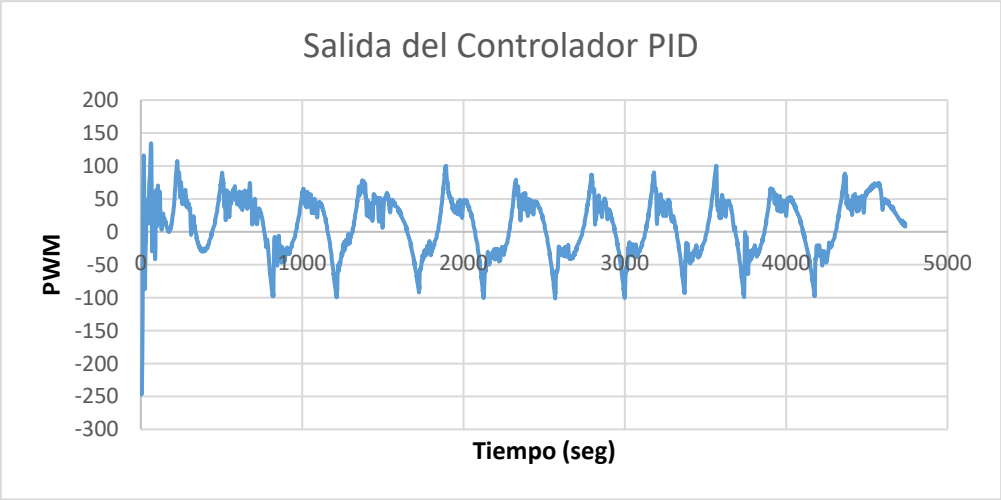


Figura 81. Respuesta del Controlador PID.

4.4 Práctica # 4.

UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR		**GUÍA DE PRÁCTICA**
CARRERA: Ingeniería Electrónica		**ASIGNATURA:** Informática Industrial
N° DE PRÁCTICA	**4**	**TÍTULO DE LA PRÁCTICA:** "Diseño y prueba del controlador difuso."
Objetivos: • Diseñar un código de programación para establecer un controlador difuso. • Presentar los resultados obtenidos del controlador.		
INSTRUCCIONES:		**1. Energizar el robot balance con la batería de 11.1 VDC.**
		2. Elaborar el código en el software de Arduino.
		3. Compilar el código y subir a la tarjeta de Arduino mega 2560.
		4. Presentar los resultados que se obtienen en el desarrollo del

	programa.

ACTIVIDADES POR DESARROLLAR

La práctica está basada en la elaboración un controlador difuso utilizando el software de Arduino.

Realizar la conexión de la tarjeta micro controladora Arduino con los demás componentes y proceder a energizar para poder así obtener las gráficas del controlador.

Paso 1.

Abrir el software de Arduino para implementar el código y agregar las librerías a utilizar en el controlador difuso como se aprecia en la figura:

```
//LLAMADO DE LIBRERIAS A USAR
//Llamado de las librerias a usar
#include <FuzzyRule.h>
#include <FuzzyComposition.h>
#include <Fuzzy.h>
#include <FuzzyRuleConsequent.h>
#include <FuzzyOutput.h>
#include <FuzzyInput.h>
#include <FuzzyIO.h>
#include <FuzzySet.h>
#include <FuzzyRuleAntecedent.h>
#define ENCODER_OPTIMIZE_INTERRUPTS
#include <Wire.h>
#include <Kalman.h>
#include <I2Cdev.h>
#include <MPU6050.h>
#include <DualVNH5019MotorShield.h>
```

Figura 82. Código de librerías para el controlador difuso.

Paso 2.

Agregar el filtro Kalman para eliminar ruidos e interferencias del controlador y asignar los pines para la interconexión de los motores:

```
Kalman kalmanX; // FILTRO KALMAN
Kalman kalmanY;
Kalman kalmanZ;


int MotorDer1=7;      //// PINES DEL MOTOR 2 EN EL ROBOT (MOTOR DERECHO)
int MotorDer2=8;
int MotorIzq1=2;      //// PINES DEL MOTOR 1 EN EL ROBOT (MOTOR IZQUIERDO
int MotorIzq2=4;
int PWM_Derecho=10;    //  PIN DEL PWM DERECHO
int PWM_Izquierdo=9;   //  PIN DEL PWM IZQUIERDO
```

Figura 83. Asignar de pines para motores.

Paso 3.

Agregar las variables para el módulo MPU 6050, velocidad de los motores y el

cálculo de ángulo usando el filtro complementario:

```
/// CREACIÓN DE  VARIABLES DEL ACELERÓMETRO Y GIROSCOPIO MPU 6050
MPU6050 accelgyro;
double accX, accY, accZ;
double gyroX, gyroY, gyroZ;
int ax, ay, az;
int gx, gy, gz;


int total;
volatile long count = 0;
int Control_fuzzy; // variable del controlador fuzzy

////VARIABLES CREADAS
int velocidad=150;            //  Declaramos una variable para guardar la velocidad
float Sum_derecha,Sum_izquierda,Speed;
double gyroXangle, gyroYangle; //  Ángulo calculado usando solo el giroscopio
double compAngleX, compAngleY; //  Cálculo del Ángulo usando filtro Complementario
double kalAngleX, kalAngleY;   //  Cálculo del Ángulo usando filtro kalman
float Sum_Right, Sum_derecha_Temp, Sum_Left, Sum_izquierda_Temp, Distancia, Distance_Right, Distance_Left;
float Turn_Speed = 0, Run_Speed = 0;
float IOutput, DOutput, Input, Output;
uint32_t timer;
```

Figura 84. Creación de variables para MPU y cálculo de ángulo.

Paso 4.

Asignar las variables del controlador difuso y declarar el llamado de la librería Fuzzy:

```
// VARIABLES DEL CONTROLADOR FUZZY
unsigned long preTime, lastTime;
float errSum, dErr, error, lastErr, errorl, derror;
int timeChange;

float Angle_Delta, Angle_Recursive, Angle_Confidence;
float Angle_Raw, Angle_Filtered, omega, dt,Angle_Filteredl;
volatile float prevAngle=0;

////////////////////////////////////////////////////////////////////////////
//Declarando la libreria fuzzy.
Fuzzy* fuzzy = new Fuzzy();
////////////////////////////////////////////////////////////////////////////
DualVNH5019MotorShield motorDriver;// INICIALIZACIÓN DEL DRIVER VNH5019
```

Figura 85. Variables del controlador difuso.

Paso 5.

Configurar los pines de salida para los motores y agregar la inicialización del serial para realizar el llamado a las reglas del controlador difuso:

```
void setup()
{
   Serial.begin(115200); //INICIALIZACIÓN DEL SERIAL
   Wire.begin();

  //CONFIGURACIÓN DE LOS PINES DE SALIDA
  pinMode(MotorDer1, OUTPUT);
  pinMode(MotorDer2, OUTPUT);
  pinMode(MotorIzq1, OUTPUT);
  pinMode(MotorIzq2, OUTPUT);
  pinMode(PWM_Derecho, OUTPUT);
  pinMode(PWM_Izquierdo, OUTPUT);

  FUZZYREGLAS(); // SE LLAMA AL SKETCH EN DONDE DECLARAMOS LAS REGLAS DEL CONTROLADOR FUZZY

 accelgyro.initialize(); //INICIALIZACIÓN DEL MPU 6050

 }
```

Figura 86. Configuración de pines de salida e inicialización del serial.

Paso 6.

Asignar las sentencias para realizar el llamado del sensor MPU, Controlador difuso y el control de PWM incluyendo condiciones para detener motores cuando el ángulo sea mayor a 40° para evitar daños si el robot balance no alcanza a estabilizar:

```
void loop()
{

 /// SENTENCIA WHILE PARA LLAMAR AL CONTROLADOR FUZZY
 while(1)
 {

   filter(); // LLAMADA DE SENSOR MPU6050
   myFuzzy(); // LLAMADA DEL CONTROLADOR FUZZY
   PWMControl(); // LLAMADA DEL CONTROL DE PWM

   if (abs(Angle_Filtered) > 40)    //CONDICIÓN PARA DETENER EL ROBOT CUANDO EL ANGULO SEA MAYOR A 40 GRADOS
    {
      motorDriver.setSpeeds(0, 0);  // SETEA EN 0 LA VELOCIDAD DE LOS MOTORES
      }
     }
       }
```

Figura 87. Sentencias para el controlador difuso.

Paso 7.

Asignar la configuración para realizar la transformación de radianes a grados para los datos obtenidos del sensor MPU:

```
//CONVERSIÓN POR EL MÉTODO ARCO TANGENTE PARA EL VALOR ANGULO DEL MPU 6050
void filter () {

  accelgyro.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
  Angle_Raw = (atan2(ay, az) * 180 / pi + Angle_offset);
  omega = gx / Gyr_Gain + Gry_offset;
  // Filter datas to get the real gesture
  unsigned long now = micros();
  timeChange = now - preTime;
  preTime = now;
  dt = timeChange * 0.000001;
  Angle_Delta = (Angle_Raw - Angle_Filtered) * 0.64;
  Angle_Recursive = Angle_Delta * dt + Angle_Recursive;
  Angle_Confidence = Angle_Recursive + (Angle_Raw - Angle_Filtered) * 1.6 + omega;
  Angle_Filtered = Angle_Confidence * dt + Angle_Filtered;
  Angle_Filtered1 = Angle_Filtered - 1;  // Variable en Grados obtenida por el MPU6050 con un offset de -1


   Serial.println(Angle_Filtered1); //


}
```

Figura 88. Transformación de radianes a grados.

Paso 8.

Asignar la configuración para realizar la fuzificación del controlador y el cálculo de las variables de entrada del sistema error y dError:

```
//////CREACIÓN DEL CONTROLADOR FUZZY

void myFuzzy() {


  // CONTROLADOR FUZZY
  double error = Angle_Filtered1-targetAngle; // CÁLCULO DEL ERROR
  double dErr = error - lastErr; // CÁLCULO DEL DERROR
  fuzzy->setInput(1,error);
  fuzzy->setInput(2,dErr);
  fuzzy->fuzzify();   //Se fuzifica el sistema
  float output1 = fuzzy->defuzzify(1);
  lastErr = error;

  //// SALIDA DEL CONTROLADOR FUZZY
  DOutput =  output1;
  IOutput =  -output1;

 }
```

Figura 89. Cálculo de las entradas del sistema.

Paso 9.

Asignar la sentencias para el uso del PWM en los motores:

```
/// SENTENCIA DE LOS MOTORES PARA EL USO DEL PWM
void PWMControl()
{
  if (IOutput > 0)
  {
    digitalWrite(MotorIzq1, HIGH);
    digitalWrite(MotorIzq2, LOW);

  }
  else if (IOutput < 0)
  {
    digitalWrite(MotorIzq1, LOW);
    digitalWrite(MotorIzq2, HIGH);
  }
  else
  {
   digitalWrite(MotorIzq1, HIGH);
   digitalWrite(MotorIzq2, HIGH);
  }
  if (DOutput > 0)
  {
    digitalWrite(MotorDer1, HIGH);
    digitalWrite(MotorDer2, LOW);
  }

  analogWrite(PWM_Derecho,min(255,abs(IOutput*255) + IMotor_offset)); // 11
  analogWrite(PWM_Izquierdo,min(255,abs(DOutput*255) + DMotor_offset)); // 12
}
```

Figura 90. Sentencia para el PWM para los motores.

Resultados obtenidos

Para visualizar los datos del ángulo del controlador difuso se procede abrir el monitor serial teniendo en cuenta la línea de código del serial (Angle_filterd1) de la Figura 91:

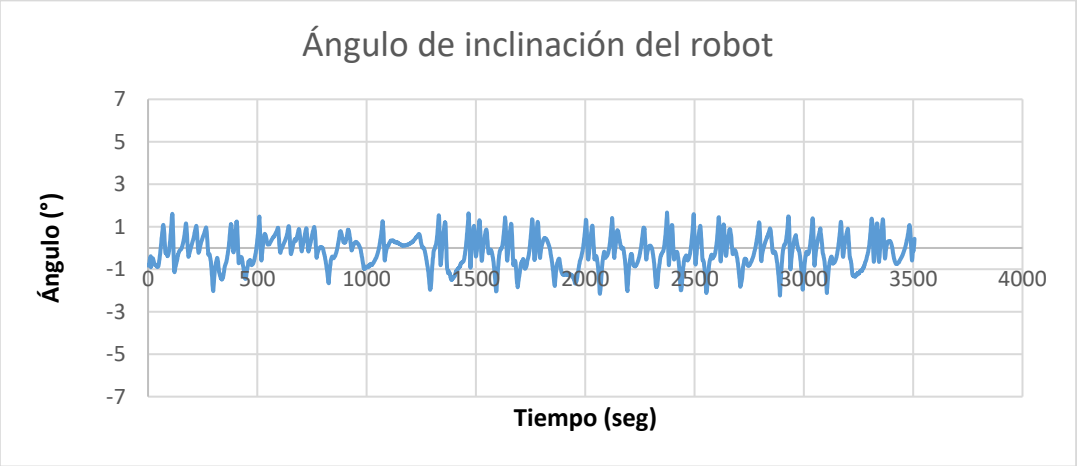


Figura 91. Comportamiento del ángulo con el controlador difuso.

Paso 11.

Para visualizar los datos del controlador difuso se procede abrir el monitor serial teniendo en cuenta la línea de código del serial (output1) de la figura del paso 8:

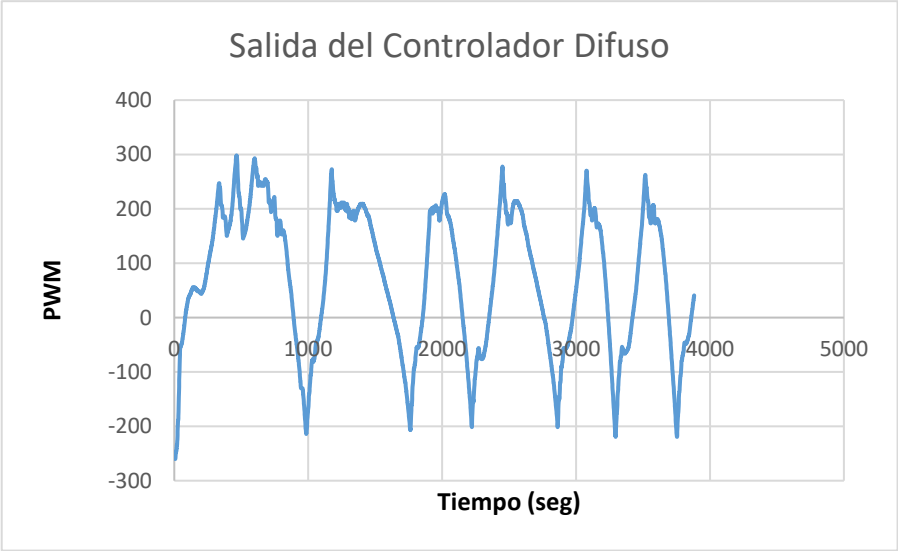


Figura 92. Respuesta del controlador difuso.

4.5 Práctica # 5.

UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR		**GUÍA DE PRÁCTICA**
CARRERA: Ingeniería Electrónica		**ASIGNATURA:** Informática Industrial
N° DE PRÁCTICA	**5**	**TÍTULO DE LA PRÁCTICA:** "Control remoto del robot auto Balaceado."
Objetivos: • Diseñar un código de programación para que el robot pueda ser controlado usando un dispositivo movil. • Mostar los resultados obtenido en el desarrollo de la práctica.		
		1. Realizar la conexión de la tarjeta micro controladora Arduino con el Bluetooth.
		2. energizar el robot con la batería de

INSTRUCCIONES:	11.1 VDC.
	3. codificar y subir el código de programación a la tarjeta Arduino mega.

ACTIVIDADES POR DESARROLLAR

En esta práctica se va a diseñar un código en el software de Arduino que permita que el robot balance sea controlado a través de un teléfono móvil mediante una comunicación por Bluetooth.

Tabla de Conexiones		
Arduino	**Color**	**Bluetooth**
5V		5V
GND		GND
TX		RX
RX		TX

Realizar las conexiones de la tarjeta micro controladora Arduino mega con el modulo Bluetooth HC-05 y proceder a energizar para subir el programa y poder verificar los resultados de la práctica

Paso 1.

Realizar la conexión de la tarjeta micro controladora Arduino con el módulo Bluetooth como se percibe en la figura 93:

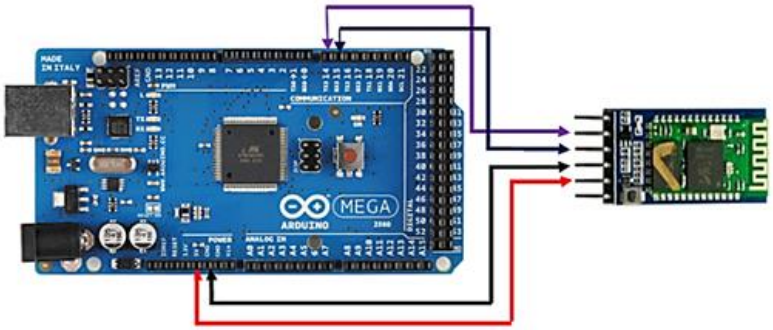


Figura 93. Diagrama de interconexión de tarjeta Arduino y Módulo Bluetooth.

Tabla 9. Conexión de la tarjeta micro controladora Arduino con Módulo Bluetooth.

Paso 2.

Abrir el software de Arduino para implementar el código y agregar las librerías a utilizar.

```
////// Llamado de las librerias a usar
#define ENCODER_OPTIMIZE_INTERRUPTS
#define RESTRICT_PITCH
#include <Wire.h>
#include <I2Cdev.h>
#include <MPU6050.h>
#include <DualVNH5019MotorShield.h>

////// Asignación de un valor inicial a una variable creada

#define Gry_offset 0  //offset del giroscopio
#define Gyr_Gain 131   //ganancia usada en el giroscopio
#define Angle_offset 0  // offset del ángulo
#define RMotor_offset 0  // offset del motor lado derecho
#define LMotor_offset 0  // offset del motor lado izquierdo
#define pi 3.14159       // valor dePI
#define DMotor_offset 0   //offset PWM derecho
#define IMotor_offset 0   //offset PWM izquierdo
float targetAngle=0;    // setpoint
#define sampleTime  0.001 // tiempo de muestreo


int MotorDer1=7;   //// PINES DEL MOTOR 2 EN EL ROBOT (MOTOR DERECHO)
int MotorDer2=8;
int MotorIzq1=2;   //// PINES DEL MOTOR 1 EN EL ROBOT (MOTOR IZQUIERDO)
int MotorIzq2=4;
int PWM_Derecho=10;   //PIN PMW DERECHO
int PWM_Izquierdo=9;  //PIN PWM IZQUIERDO

int vSpeed = 0;     // Default speed, from 0 to 255
int state;
float kIm = 0.3;
float kDm = 0.3;

/// CREACIÓN DE VARIABLES A USAR DEL SENSOR MPU 6050
MPU6050 accelgyro;
double accX, accY, accZ;
double gyroX, gyroY, gyroZ;
int ax, ay, az;
int gx, gy, gz;

//CREACIÓN DE VARIABLES
int total;
float speedDifference;

////CREACIÓN DE VARIABLES A USAR
int velocidad=150;//Declaramos una variable para guardar la velocidad
float Sum_derecha,Sum_izquierda,Speed;
double gyroXangle, gyroYangle;
double compAngleX, compAngleY;
double kalAngleX, kalAngleY;
float Sum_Right, Sum_derecha_Temp, Sum_Left, Sum_izquierda_Temp, Distancia, Distance_Right, Distance_Left;
float Turn_Speed = 0, Run_Speed = 0;
float IOutput, DOutput, Input, Output;
uint32_t timer;
```

```
// CREACIÓN DE VARIABLES DEL PID
unsigned long preTime, lastTime;
float errSum, dErr, error, lastErr;
int timeChange;

float Angle_Delta, Angle_Recursive, Angle_Confidence;
float Angle_Raw, Angle_Filtered, omega, dt,Angle_Filteredl;
volatile float prevAngle=0;
```

Figura 94. Asignación de librerías y variables utilizadas.

Paso 3.

Inicializar el driver VNH5019 y declarar las variables que se van a utilizar para dar inicio al sensor MPU 6050.

```
DualVNH5019MotorShield motorDriver; // INICIALIZACIÓN DEL DRIVER VNH5019


void setup()
{
   Serial.begin(9600); //ASIGNACIÓN DE VELOCIDAD E INICIO DEL SERIAL
   Serial2.begin(9600);
   Wire.begin();

 //ASIGNACIÓN DE PINES DE SALIDA
  pinMode(MotorDer1, OUTPUT);
  pinMode(MotorDer2, OUTPUT);
  pinMode(MotorIzq1, OUTPUT);
  pinMode(MotorIzq2, OUTPUT);
  pinMode(PWM_Derecho, OUTPUT);
  pinMode(PWM_Izquierdo, OUTPUT);

 accelgyro.initialize(); //INICIALIZACIÓN DEL MPU6050
 }
```

Figura 95. Asignación de variables para inicialización del driver VNH5019.

Paso 4.

En el lazo void loop se ubica la sentencia para hacer el llamado del controlador PID y realizar la conversión de radianes a grados usando el método de arco tangente.

```
void loop()
{


/// USO DE UNA SENTENCIA WHILE PARA LLAMAR AL CONTROLADOR PID
 while(1)
 {

   filter();  //LLAMADA DE LA LECTURA DEL MPU 6050
   myPID();   // LLAMADA DEL CONTROLADOR PID
   PWMControl(); // LLAMADA PARA EL CONTROL DEL PWM
   blue();
    if (abs(Angle_Filtered) > 30)  /// CONDICIÓN QUE DESACTIVA Y APAGA LOS MOTORES AL LLEGAR EL ÁNGULO A VALORES MAYORES A 30 GRADOS
    {
      motorDriver.setSpeeds(0, 0); // SETEO DE VELOCIDAD EN 0
      }

     }
       }

/// CONVERSIÓN POR EL MÉTODO ARCO TANGENTE DE LOS VALORES DEL MPU 6050 DE RAD. A GRADOS
void filter () {

  accelgyro.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
  Angle_Raw = (atan2(ay, az) * 180 / pi + Angle_offset);
  omega = gx / Gyr_Gain + Gry_offset;
```

Figura 96. Sentencia para llamar al controlador PID.

Paso 5.

Asignación de los valores del controlador PID y las variables de entrada y salida del sistema.

```
//CREACIÓN DEL CONTROLADOR PID

void myPID() {

   //VALORES DE KP,KI,KD
   int kp = 80;
   int ki = 1;
   int kd = 4;

  double error = Angle_Filtered1 - targetAngle; //VARIABLE ERROR
  errSum += errSum + error;
  errSum = constrain(errSum, -100, 100);
  double dErr = error - lastErr;    // VARIABLE DERIVADA DEL ERROR
  speedDifference = kp * (error) + ki * (errSum)*sampleTime + kd *omega; //CONTROLADOR PID
  prevAngle = Angle_Filtered1;
  lastErr = error;

   // SALIDA DEL CONTROLADOR PID
   DOutput = -speedDifference;
   IOutput = speedDifference;
 }
```

Figura 97. Asignación de valores PID.

Paso 6.

Crear las sentencias para el uso del PWM en los motores especificando si es derecho o izquierdo.

```
//CREACIÓN DE LA SENTENCIA PARA LOS MOTORES USANDO PWM

void PWMControl()
{
  if (IOutput > 0)
  {
    digitalWrite(MotorIzq1, HIGH);
    digitalWrite(MotorIzq2, LOW);

  }
  else if (IOutput < 0)
  {
    digitalWrite(MotorIzq1, LOW);
    digitalWrite(MotorIzq2, HIGH);
  }
  else
  {
   digitalWrite(MotorIzq1, HIGH);
   digitalWrite(MotorIzq2, HIGH);
  }
  if (DOutput > 0)
  {
    digitalWrite(MotorDer1, HIGH);
    digitalWrite(MotorDer2, LOW);
  }
  else if (DOutput < 0)
  {

    digitalWrite(MotorDer1, LOW);
    digitalWrite(MotorDer2, HIGH);
  }
  else
  {
    digitalWrite(MotorDer1, HIGH);
    digitalWrite(MotorDer2, HIGH);
  }
    analogWrite(PWM_Derecho,min(255,abs(IOutput*kDm) + IMotor_offset)); // PWM DERECHO
    analogWrite(PWM_Izquierdo,min(255,abs(DOutput*kIm) + DMotor_offset)); // PWM IZQUIERDO
}
```

Figura 98. Sentencia para Motores.

Paso 7.

Asignación de la variable state en donde se guardan los datos del serial y declaración de las condiciones de lectura del Bluetooth.

```
void blue() {

  //Apertura del puerto serial del Bluetooth y guardado de los datos en la variable state
    if(Serial2.available() > 0){
      state = Serial2.read();
    }

  /************************Adelante****************************/
  //If state is equal with letter 'F', car will go forward!
    if (state == 'F') {
      Serial.println(state);
      targetAngle = 2;
      kIm = 0.3 ;
      kDm = 0.3 ;
    }

  /************************Atrás***************************/
  //If state is equal with letter 'B', car will go backward
    else if (state == 'B') {

      targetAngle = -3;
      kIm = 0.3 ;
      kDm = 0.3 ;
    }
```

Figura 99. Asignación de condiciones del Bluetooth.

Resultados obtenidos

Luego de haber realizado la programación e interconexión de la tarjeta micro procesadora Arduino mega y el módulo Bluetooth HC-05, se procede a realizar las pruebas respectivas para verificar su funcionamiento.

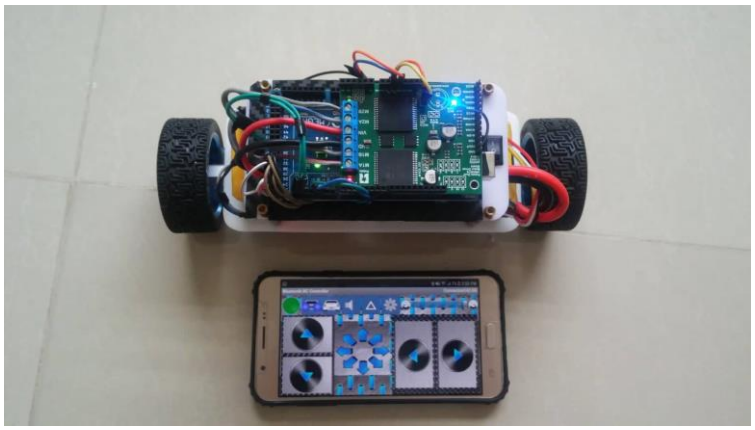


Figura 100. Movimiento del Robot usando un dispositivo móvil.

5. RESULTADOS

5.1 Prueba y evaluación del Controlador PID

Después de haber recopilado datos en la planta real se puede realizar la comparación de los controladores PID y Difuso para determinar las respuestas de los respectivos ángulos y señales de control con los cuales actúan los controladores.

En la siguiente figura podemos apreciar el comportamiento del controlador PID y su respectivo ángulo de inclinación, con una perturbación aplicada cada 10 seg.

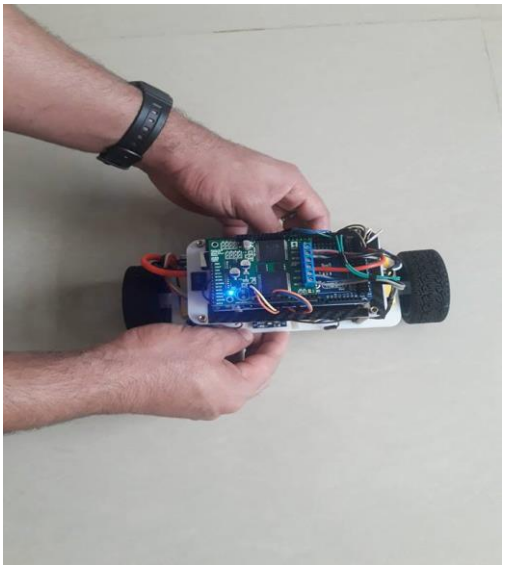


Figura 101. Obtención de datos con perturbaciones.

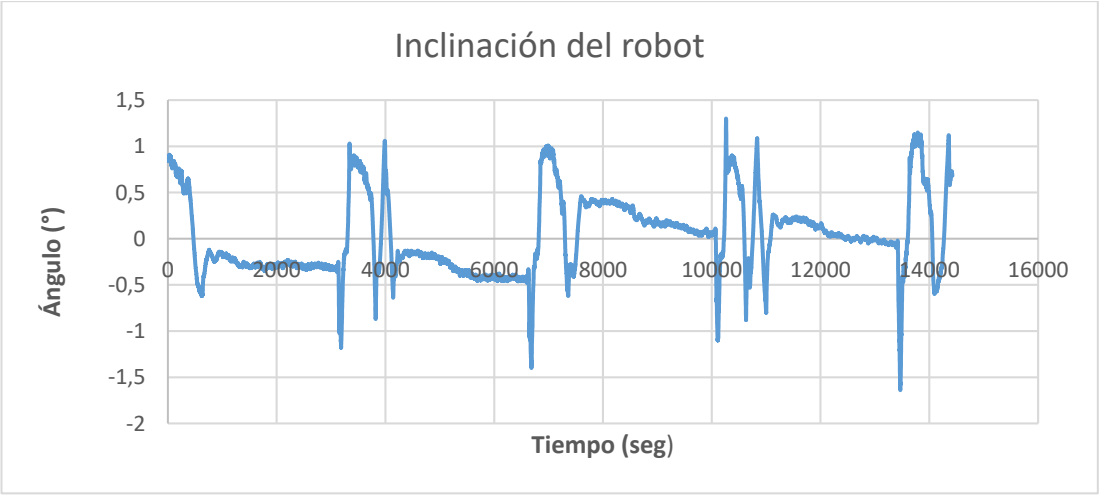


Figura 102. Respuesta del Ángulo del controlador PID con perturbación cada 10 seg.

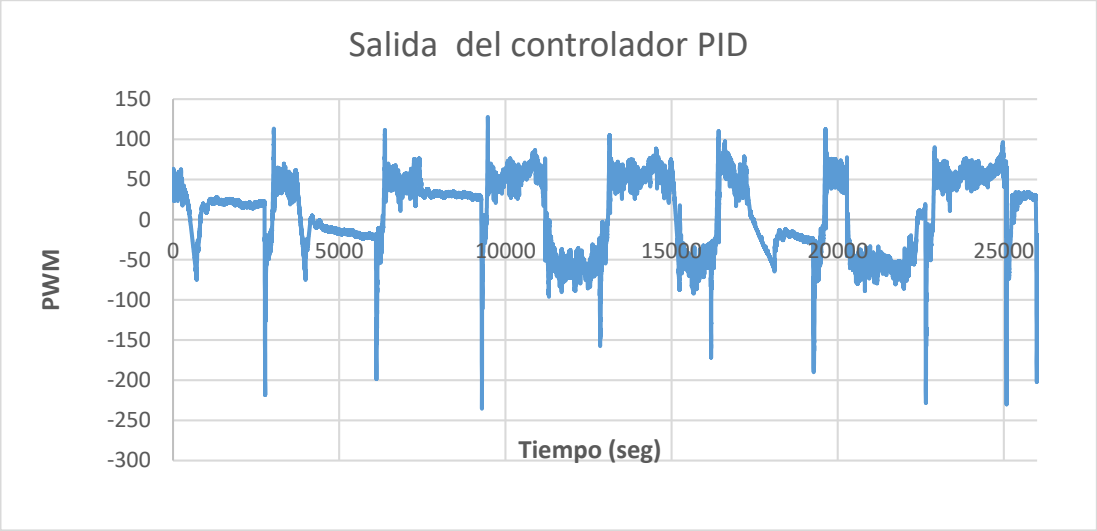


Figura 103. Respuesta del controlador PID con perturbación cada 10 seg.

En la siguiente figura podemos apreciar el comportamiento del controlador Difuso y su respectivo ángulo de inclinación.

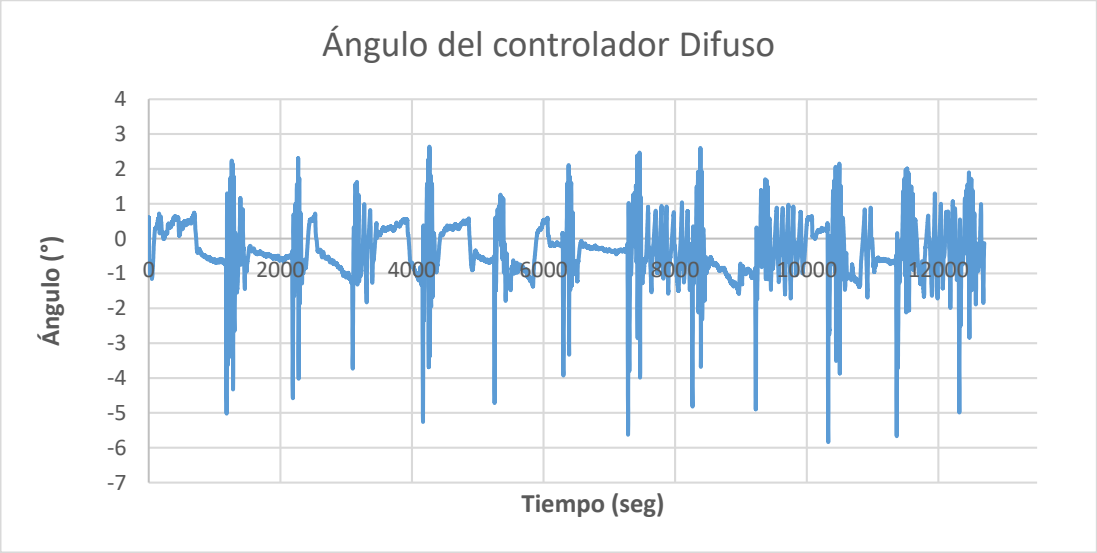


Figura 104. Respuesta del ángulo respecto al tiempo del controlador Difuso con perturbación cada 10 seg.

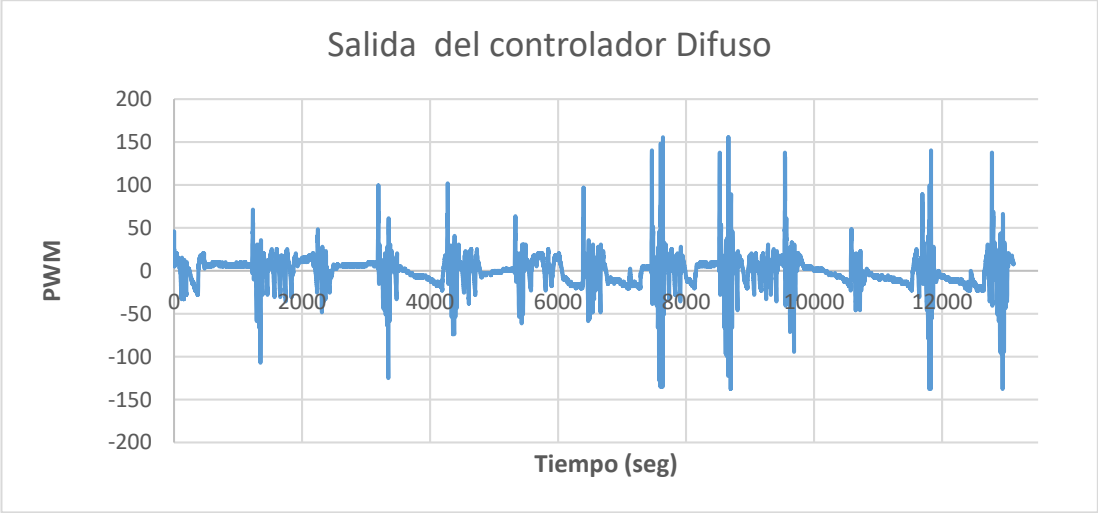


Figura 105. Respuesta de la señal de control Difuso con perturbación cada 10 seg.

5.2 Análisis de resultado

Para validar los modelos se toma las respuestas de los controladores y se las compara.

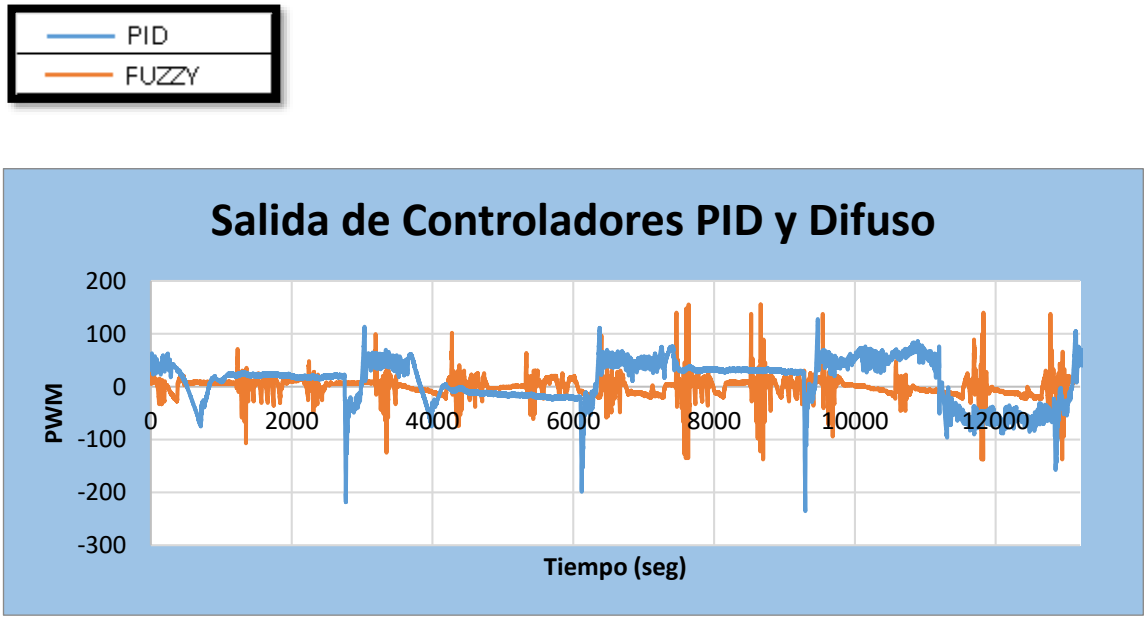


Figura 106. Comparación de Controladores.

El controlador PID presenta una ventajas y respuesta mucho más rápida que el controlador difuso para logra una estabilización.

Otra de las ventajas del PID ante el controlador difuso es la respuesta a perturbaciones y el tiempo de asentamiento al estabilizarse.

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

CONCLUSIONES Y RECOMENDACIONES

Conclusiones

1. El controlador PID posee mayor efectividad que el controlador difuso el cual muestra mayor tiempo de asentamiento y porcentaje de sobreimpulso.

2. Las reglas del controlador difuso fueron elaboradas a partir de pruebas en tiempo real donde se hallaron los rangos de trabajo.

3. Para la validación y diseño de los controladores planteados se toma varias técnicas experimentales y herramientas de software.

4. Al añadir perturbaciones el controlador PID presenta una mayor respuesta y rechazo a la misma en donde se presenta una salida sin gran cantidad de picos, por otro lado el controlador difuso en respuesta a la perturbación presenta una tardía estabilización y en su salida se muestra más oscilatorio con varios picos los cuales influyen en su tiempo de asentamiento.

5. El manejo del controlador PID lineal necesita de la afinación de los parámetros del controlador para que su labor sea alrededor de un punto de operación.

Recomendaciones

1. Para lograr un mayor equilibrio y respuesta del sistema en el robot balance se recomienda usar dos controladores PID para trabajos futuros.

2. Para un mejor uso del controlador difuso se recomienda realizar varias pruebas con los rangos hallados y verificar su respuesta.

3. Al trabajar con sensores ópticos para aplicaciones de seguimiento de línea es recomendable considerar la distancia que se tiene desde la superficie al sensor.

4. Al enviar datos en tiempo real es recomendable considerar el tiempo de transmisión del módulo a usar para evitar pérdida de datos en el lapso de las pruebas.

BIBLIOGRAFÍA

[1] R. C. V. H. W. A. Byron Lima, «Análisis de estabilidad de un péndulo invertido rotativo utilizando técnica de control no lineal,» *IOP ebooks,* p. 12, 2017.

[2] F. H. Manuel, «Instituto Politécnico Nacional,» 7 enero 2015. [En línea]. Available: http://tesis.ipn.mx/bitstream/handle/123456789/14336/Tesis%20LabVIEW-Final%20Mayo%202015.pdf?sequence=1.

[3] A. L. E. Herminio, «Universidad Nacional De Educación Enrique Guzman y Valle,» 2015. [En línea]. Available: http://repositorio.une.edu.pe/bitstream/handle/UNE/962/TM%20CE-Du%20A814%202015.pdf?sequence=1&isAllowed=y.

[4] A. A. Duarte, «Tecnologico Nacional de Mexico de la Paz,» Agosto 2016. [En línea]. Available: http://posgrado.itlp.edu.mx/uploads/archivos/57ab85cdaefba.pdf.

[5] T. T. L. Romo Caicedo Diego, «DSpace Escuela Politécnica Nacional,» Agosto 2016. [En línea]. Available: http://bibdigital.epn.edu.ec/bitstream/15000/16546/1/CD-7210.pdf.

[6] M. Electronico, «MPU 6050 Acelerometro y Giroscopio con arduino leonardo,» 16 Abril 2015. [En línea]. Available: https://www.makerelectronico.com/mpu6050-acelerometro-y-giroscopio-con-arduino-leonardo/.

[7] jjchop, «jjchop.wordpress.com,» 17 Diciembre 2014. [En línea]. Available: https://jjchop.wordpress.com/2014/12/17/adaptando-el-modulo-sd-de-lc-studio-para-arduino-uno/.

[8] W. Z. S. W. Junfeng Wu, "A Two-Wheeled Self-Balancing Robotwith the fuzzy PD Control Method," *Hindawi Publishing Corporation,* vol. 2012, no. 469491, p. 13, 2012.

[9] B. I. J. Diego, «Universidad Politecnica Salesiana sede Cuenca,» Noviembre 2014. [En línea]. Available: https://dspace.ups.edu.ec/bitstream/123456789/7508/1/UPS-CT004422.pdf.

[10] R. Cajo, «Evaluación de los algoritmos de control PID lineal y no lineal, aplicados al sistema mimo de doble rotor tipo helicóptero,» 24 9 2015. [En línea]. Available: http://www.dspace.espol.edu.ec/xmlui/handle/123456789/30111.

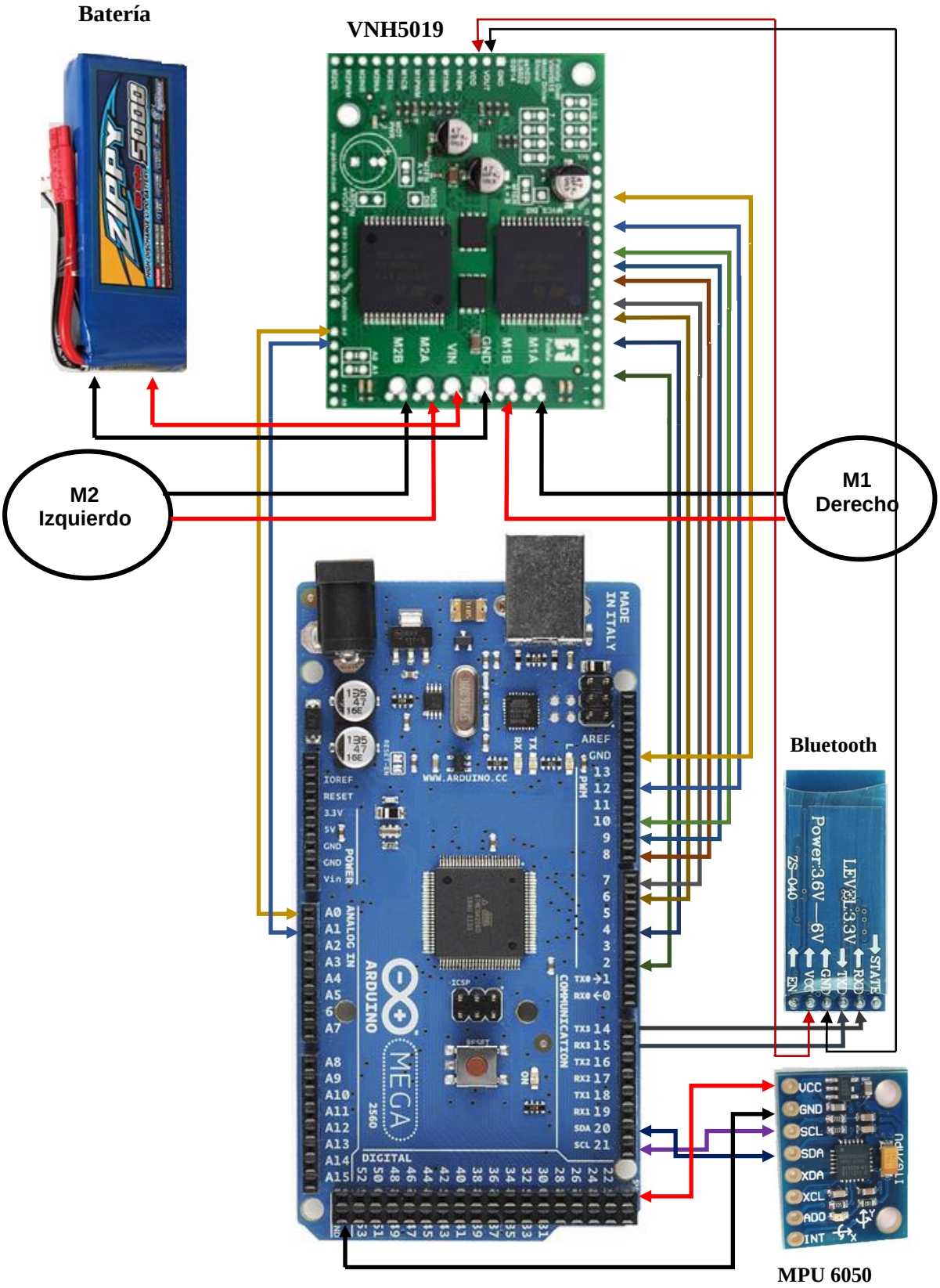
[11] E. Toapanta Cunalata, «Repositorio Institucional Universidad Técnica de Cotopaxi,» Diciembre 2015. [En línea]. Available:

http://repositorio.utc.edu.ec/bitstream/27000/3411/1/T-UTC-00688.pdf.

[12] P. Ponce Cruz, Inteligencia Artificial con aplicaciones a la ingeniería, México: Alfaomega, 2010.

ANEXOS

Anexo 1. Diagrama de conexiones del Robot balance



ANEXO 2. Código empleado para llamar la función FUZZYREGLAS del controlador difuso.

```
//Llamado de las librerias a usar

#include <FuzzyRule.h>

#include <FuzzyComposition.h>

#include <Fuzzy.h>

#include <FuzzyRuleConsequent.h>

#include <FuzzyOutput.h>

#include <FuzzyInput.h>

#include <FuzzyIO.h>

#include <FuzzySet.h>

#include <FuzzyRuleAntecedent.h>



void FUZZYREGLAS() {


// CREACIÓN DE LA PRIMERA ENTRADA PARA EL CONTROLADOR FUZZY

 FuzzyInput*error = new FuzzyInput(1);


 // DECLARACIÓN DE RANGOS DE LA VARIABLE DE ENTRADA 1 ERROR

 FuzzySet*NB = new FuzzySet(-15, -10, -10, -8); // NEGATIVO GRANDE

 error->addFuzzySet(NB); //

 FuzzySet*NM = new FuzzySet(-10, -8, -8, -5); // NEGATIVO MEDIANO

 error->addFuzzySet(NM); //

 FuzzySet*NS = new FuzzySet(-8, -5, -5, 0); // NEGATIVO PEQUEÑO
```

```
error->addFuzzySet(NS); //

FuzzySet*ZE = new FuzzySet(-5, 0, 0,2 ); // CENTRO

error->addFuzzySet(ZE); //

FuzzySet*PS= new FuzzySet(0, 2, 2, 5); // POSITIVO PEQUEÑO

error->addFuzzySet(PS); //

FuzzySet*PM = new FuzzySet(2, 5, 5, 8); // POSITIVO MEDIANO

error->addFuzzySet(PM); //

FuzzySet*PB = new FuzzySet(5, 8, 8, 10); // POSITIVO GRANDE

error->addFuzzySet(PB); //


fuzzy->addFuzzyInput(error); // Agregando la entrada para error



// CREACIÓN DE LA SEGUNDA ENTRADA PARA EL CONTROLADOR FUZZY

FuzzyInput*derror = new FuzzyInput(2);


///DECLARACIÓN DE RANGOS DE LA VARIABLE DE ENTRADA 2 DERROR


FuzzySet*NBD = new FuzzySet(-2.5, -1.5, -1.5, -1.4); // NEGATIVO GRANDE

derror->addFuzzySet(NBD); //

FuzzySet*NMD = new FuzzySet(-1.5, -1.4, -1.4, -1.3); // NEGATIVO MEDIANO

derror->addFuzzySet(NMD); //

FuzzySet*NSD = new FuzzySet(-1.4, -1.3, -1.3, 0); // NEGATIVO PEQUEÑO

derror->addFuzzySet(NSD); //
```

```
FuzzySet*ZED = new FuzzySet(-1.3, 0, 0, 1.3); // CENTRO

derror->addFuzzySet(ZED); //

FuzzySet*PSD= new FuzzySet(0, 1.3, 1.3, 1.4); // POSITIVO PEQUEÑO

derror->addFuzzySet(PSD); //

FuzzySet*PMD = new FuzzySet(1.3, 1.4, 1.4, 1.5); // POSITIVO MEDIANO

derror->addFuzzySet(PMD); //

FuzzySet*PBD = new FuzzySet(1.4, 1.5, 1.5, 2.5); // POSITIVO GRANDE

derror->addFuzzySet(PBD); //


fuzzy->addFuzzyInput(derror); // Agregando la entrada para la derivada del error.


// CREACIÓN DE LA SALIDA PARA EL CONTROLADOR FUZZY

FuzzyOutput*velocidad = new FuzzyOutput(1);// With its ID in param


///DECLARACIÓN DE RANGOS DE LA VARIABLE DE SALIDA VELOCIDAD


FuzzySet*NBO = new FuzzySet(-0.7, -0.65, -0.65, -0.5); // NEGATIVO GRANDE

velocidad->addFuzzySet(NBO); //

FuzzySet*NMO = new FuzzySet(-0.65, -0.5, -0.5, -0.4); // NEGATIVO MEDIANO

velocidad->addFuzzySet(NMO); //

FuzzySet*NSO = new FuzzySet(-0.5, -0.4, -0.4, 0); // NEGATIVO PEQUEÑO

velocidad->addFuzzySet(NSO); //

FuzzySet*ZEO = new FuzzySet(-0.4, 0, 0, 0.4); // CENTRO

velocidad->addFuzzySet(ZEO); //
```

```
FuzzySet*PSO = new FuzzySet(0, 0.4, 0.4, 0.5); // POSITIVO PEQUEÑO

velocidad->addFuzzySet(PSO); //

FuzzySet*PMO = new FuzzySet(0.4, 0.5, 0.5, 0.65); // POSITIVO MEDIANO

velocidad->addFuzzySet(PMO); //

FuzzySet*PBO = new FuzzySet(0.5, 0.65, 0.65, 0.7); // POSITIVO GRANDE

velocidad->addFuzzySet(PBO); //


fuzzy->addFuzzyOutput(velocidad); // Agregando la salida del controlador difuso.



///////////////// GENERACIÓN DE LAS 49 REGLAS USADAS EN EL CONTROLADOR FUZZY


/*REGLA 1*/ FuzzyRuleAntecedent*iferrorNB_and_ifderrorNBD = new FuzzyRuleAntecedent(); iferrorNB_and_ifderrorNBD->joinWithAND(NB,NBD); // Agregando el antecedente

FuzzyRuleConsequent* thenvelocidadPB = new FuzzyRuleConsequent(); thenvelocidadPB->addOutput(PBO);// Agregando el consecuente a la salida

FuzzyRule*fuzzyRule1 = new FuzzyRule(1, iferrorNB_and_ifderrorNBD, thenvelocidadPB); fuzzy->addFuzzyRule(fuzzyRule1); // Adicionando la regla


/*REGLA 2*/FuzzyRuleAntecedent*iferrorNB_and_ifderrorNMD = new FuzzyRuleAntecedent(); iferrorNB_and_ifderrorNMD->joinWithAND(NB,NMD); // Agregando el antecedente

FuzzyRuleConsequent* thenvelocidadPB0 = new FuzzyRuleConsequent(); thenvelocidadPB0->addOutput(PBO);// Agregando el consecuente a la salida

FuzzyRule*fuzzyRule2 = new FuzzyRule(2, iferrorNB_and_ifderrorNMD,
```

```
thenvelocidadPB0); fuzzy->addFuzzyRule(fuzzyRule2); // Adicionando la regla


 /*REGLA 3*/FuzzyRuleAntecedent*iferrorNB_and_ifderrorNSD = new
FuzzyRuleAntecedent(); iferrorNB_and_ifderrorNSD->joinWithAND(NB,NSD); //
Agregando el antecedente

 FuzzyRuleConsequent* thenvelocidadPB1 = new FuzzyRuleConsequent();
thenvelocidadPB1->addOutput(PBO);// Agregando el consecuente a la salida

 FuzzyRule*fuzzyRule3 = new FuzzyRule(3, iferrorNB_and_ifderrorNSD,
thenvelocidadPB1); fuzzy->addFuzzyRule(fuzzyRule3); // Adicionando la regla


 /*REGLA 4*/FuzzyRuleAntecedent*iferrorNB_and_ifderrorZED = new
FuzzyRuleAntecedent(); iferrorNB_and_ifderrorZED->joinWithAND(NB,ZED); //
Agregando el antecedente

 FuzzyRuleConsequent* thenvelocidadPB2 = new FuzzyRuleConsequent();
thenvelocidadPB2->addOutput(PBO);// Agregando el consecuente a la salida

 FuzzyRule*fuzzyRule4 = new FuzzyRule(4, iferrorNB_and_ifderrorZED,
thenvelocidadPB2); fuzzy->addFuzzyRule(fuzzyRule4); // Adicionando la regla


 /*REGLA 5*/FuzzyRuleAntecedent*iferrorNB_and_ifderrorPSD = new
FuzzyRuleAntecedent(); iferrorNB_and_ifderrorPSD->joinWithAND(NB,PSD); //
Agregando el antecedente

 FuzzyRuleConsequent* thenvelocidadPB3 = new FuzzyRuleConsequent();
thenvelocidadPB3->addOutput(PBO);// Agregando el consecuente a la salida

 FuzzyRule*fuzzyRule5 = new FuzzyRule(5, iferrorNB_and_ifderrorPSD,
thenvelocidadPB3); fuzzy->addFuzzyRule(fuzzyRule5); // Adicionando la regla


 /*REGLA 6*/FuzzyRuleAntecedent*iferrorNB_and_ifderrorPMD = new
FuzzyRuleAntecedent(); iferrorNB_and_ifderrorPMD->joinWithAND(NB,PMD); //
Agregando el antecedente
```

```
 FuzzyRuleConsequent* thenvelocidadPM = new FuzzyRuleConsequent();
thenvelocidadPM->addOutput(PMO);// Agregando el consecuente a la salida

 FuzzyRule*fuzzyRule6 = new FuzzyRule(6, iferrorNB_and_ifderrorPMD,
thenvelocidadPM); fuzzy->addFuzzyRule(fuzzyRule6); // Adicionando la regla



 /*REGLA 7*/FuzzyRuleAntecedent*iferrorNB_and_ifderrorPBD = new
FuzzyRuleAntecedent(); iferrorNB_and_ifderrorPBD->joinWithAND(NB,PBD); //
Agregando el antecedente

 FuzzyRuleConsequent* thenvelocidadZE = new FuzzyRuleConsequent();
thenvelocidadZE->addOutput(ZEO);// Agregando el consecuente a la salida

 FuzzyRule*fuzzyRule7 = new FuzzyRule(7, iferrorNB_and_ifderrorPBD,
thenvelocidadZE); fuzzy->addFuzzyRule(fuzzyRule7); // Adicionando la regla



 /*REGLA 8*/FuzzyRuleAntecedent*iferrorNM_and_ifderrorNBD = new
FuzzyRuleAntecedent(); iferrorNM_and_ifderrorNBD->joinWithAND(NM,NBD); //
Agregando el antecedente

 FuzzyRuleConsequent* thenvelocidadPB4 = new FuzzyRuleConsequent();
thenvelocidadPB4->addOutput(PBO);// Agregando el consecuente a la salida

 FuzzyRule*fuzzyRule8 = new FuzzyRule(8, iferrorNM_and_ifderrorNBD,
thenvelocidadPB4); fuzzy->addFuzzyRule(fuzzyRule8); // Adicionando la regla



 /*REGLA 9*/FuzzyRuleAntecedent*iferrorNM_and_ifderrorNMD = new
FuzzyRuleAntecedent(); iferrorNM_and_ifderrorNMD->joinWithAND(NM,NMD); //
Agregando el antecedente

 FuzzyRuleConsequent* thenvelocidadPB5 = new FuzzyRuleConsequent();
thenvelocidadPB5->addOutput(PBO);// Agregando el consecuente a la salida

 FuzzyRule*fuzzyRule9 = new FuzzyRule(9, iferrorNM_and_ifderrorNMD,
thenvelocidadPB5); fuzzy->addFuzzyRule(fuzzyRule9); // Adicionando la regla
```

```
/*REGLA 10*/FuzzyRuleAntecedent*iferrorNM_and_ifderrorNSD = new
FuzzyRuleAntecedent(); iferrorNM_and_ifderrorNSD->joinWithAND(NM,NSD); //
Agregando el antecedente

FuzzyRuleConsequent* thenvelocidadPB6 = new FuzzyRuleConsequent();
thenvelocidadPB6->addOutput(PBO);// Agregando el consecuente a la salida

FuzzyRule*fuzzyRule10 = new FuzzyRule(10, iferrorNM_and_ifderrorNSD,
thenvelocidadPB6); fuzzy->addFuzzyRule(fuzzyRule10); // Adicionando la regla


/*REGLA 11*/FuzzyRuleAntecedent*iferrorNM_and_ifderrorZED = new
FuzzyRuleAntecedent(); iferrorNM_and_ifderrorZED->joinWithAND(NM,ZED); //
Agregando el antecedente

FuzzyRuleConsequent* thenvelocidadPB7 = new FuzzyRuleConsequent();
thenvelocidadPB7->addOutput(PBO);// Agregando el consecuente a la salida

FuzzyRule*fuzzyRule11 = new FuzzyRule(11, iferrorNM_and_ifderrorZED,
thenvelocidadPB7); fuzzy->addFuzzyRule(fuzzyRule11); // Adicionando la regla


/*REGLA 12*/FuzzyRuleAntecedent*iferrorNM_and_ifderrorPSD = new
FuzzyRuleAntecedent(); iferrorNM_and_ifderrorPSD->joinWithAND(NM,PSD); //
Agregando el antecedente

FuzzyRuleConsequent* thenvelocidadPM1 = new FuzzyRuleConsequent();
thenvelocidadPM1->addOutput(PMO);// Agregando el consecuente a la salida

FuzzyRule*fuzzyRule12 = new FuzzyRule(12, iferrorNM_and_ifderrorPSD,
thenvelocidadPM1); fuzzy->addFuzzyRule(fuzzyRule12); // Adicionando la regla


/*REGLA 13*/FuzzyRuleAntecedent*iferrorNM_and_ifderrorPMD = new
FuzzyRuleAntecedent(); iferrorNM_and_ifderrorPMD->joinWithAND(NM,PMD); //
Agregando el antecedente

FuzzyRuleConsequent* thenvelocidadZE1 = new FuzzyRuleConsequent();
thenvelocidadZE1->addOutput(ZEO);// Agregando el consecuente a la salida

FuzzyRule*fuzzyRule13 = new FuzzyRule(13, iferrorNM_and_ifderrorPMD,
```

```
thenvelocidadZE1); fuzzy->addFuzzyRule(fuzzyRule13); // Adicionando la regla


 /*REGLA 14*/FuzzyRuleAntecedent*iferrorNM_and_ifderrorPBD = new
FuzzyRuleAntecedent(); iferrorNM_and_ifderrorPBD->joinWithAND(NM,PBD); //
Agregando el antecedente

 FuzzyRuleConsequent* thenvelocidadNM = new FuzzyRuleConsequent();
thenvelocidadNM->addOutput(NMO);// Agregando el consecuente a la salida

 FuzzyRule*fuzzyRule14 = new FuzzyRule(14, iferrorNM_and_ifderrorPBD,
thenvelocidadNM); fuzzy->addFuzzyRule(fuzzyRule14); // Adicionando la regla


 /*REGLA 15*/FuzzyRuleAntecedent*iferrorNS_and_ifderrorNBD = new
FuzzyRuleAntecedent(); iferrorNS_and_ifderrorNBD->joinWithAND(NS,NBD); //
Agregando el antecedente

 FuzzyRuleConsequent* thenvelocidadPB8 = new FuzzyRuleConsequent();
thenvelocidadPB8->addOutput(PBO);// Agregando el consecuente a la salida

 FuzzyRule*fuzzyRule15 = new FuzzyRule(15, iferrorNS_and_ifderrorNBD,
thenvelocidadPB8); fuzzy->addFuzzyRule(fuzzyRule15); // Adicionando la regla


 /*REGLA 16*/FuzzyRuleAntecedent*iferrorNS_and_ifderrorNMD = new
FuzzyRuleAntecedent(); iferrorNS_and_ifderrorNMD->joinWithAND(NS,NMD); //
Agregando el antecedente

 FuzzyRuleConsequent* thenvelocidadPB9 = new FuzzyRuleConsequent();
thenvelocidadPB9->addOutput(PBO);// Agregando el consecuente a la salida

 FuzzyRule*fuzzyRule16 = new FuzzyRule(16, iferrorNS_and_ifderrorNMD,
thenvelocidadPB9); fuzzy->addFuzzyRule(fuzzyRule16); // Adicionando la regla


 /*REGLA 17*/FuzzyRuleAntecedent*iferrorNS_and_ifderrorNSD = new
FuzzyRuleAntecedent(); iferrorNS_and_ifderrorNSD->joinWithAND(NS,NSD); //
Agregando el antecedente
```

```
 FuzzyRuleConsequent* thenvelocidadPB10 = new FuzzyRuleConsequent();
thenvelocidadPB10->addOutput(PBO);// Agregando el consecuente a la salida

 FuzzyRule*fuzzyRule17 = new FuzzyRule(17, iferrorNS_and_ifderrorNSD,
thenvelocidadPB10); fuzzy->addFuzzyRule(fuzzyRule17); // Adicionando la regla



 /*REGLA 18*/FuzzyRuleAntecedent*iferrorNS_and_ifderrorZED = new
FuzzyRuleAntecedent(); iferrorNS_and_ifderrorZED->joinWithAND(NS,ZED); //
Agregando el antecedente

 FuzzyRuleConsequent* thenvelocidadPM2 = new FuzzyRuleConsequent();
thenvelocidadPM2->addOutput(PMO);// Agregando el consecuente a la salida

 FuzzyRule*fuzzyRule18 = new FuzzyRule(18, iferrorNS_and_ifderrorZED,
thenvelocidadPM2); fuzzy->addFuzzyRule(fuzzyRule18); // Adicionando la regla



 /*REGLA 19*/FuzzyRuleAntecedent*iferrorNS_and_ifderrorPSD = new
FuzzyRuleAntecedent(); iferrorNS_and_ifderrorPSD->joinWithAND(NS,PSD); //
Agregando el antecedente

 FuzzyRuleConsequent* thenvelocidadZE2 = new FuzzyRuleConsequent();
thenvelocidadZE2->addOutput(ZEO);// Agregando el consecuente a la salida

 FuzzyRule*fuzzyRule19 = new FuzzyRule(19, iferrorNS_and_ifderrorPSD,
thenvelocidadZE2); fuzzy->addFuzzyRule(fuzzyRule19); // Adicionando la regla



 /*REGLA 20*/FuzzyRuleAntecedent*iferrorNS_and_ifderrorPMD = new
FuzzyRuleAntecedent(); iferrorNS_and_ifderrorPMD->joinWithAND(NS,PMD); //
Agregando el antecedente

 FuzzyRuleConsequent* thenvelocidadNM1 = new FuzzyRuleConsequent();
thenvelocidadNM1->addOutput(NMO);// Agregando el consecuente a la salida

 FuzzyRule*fuzzyRule20 = new FuzzyRule(20, iferrorNS_and_ifderrorPMD,
thenvelocidadNM1); fuzzy->addFuzzyRule(fuzzyRule20); // Adicionando la regla
```

```
/*REGLA 21*/FuzzyRuleAntecedent*iferrorNS_and_ifderrorPBD = new FuzzyRuleAntecedent(); iferrorNS_and_ifderrorPBD->joinWithAND(NS,PBD); // Agregando el antecedente

FuzzyRuleConsequent* thenvelocidadNB = new FuzzyRuleConsequent(); thenvelocidadNB->addOutput(NBO);// Agregando el consecuente a la salida

FuzzyRule*fuzzyRule21 = new FuzzyRule(21, iferrorNS_and_ifderrorPBD, thenvelocidadNB); fuzzy->addFuzzyRule(fuzzyRule21); // Adicionando la regla


/*REGLA 22*/FuzzyRuleAntecedent*iferrorZE_and_ifderrorNBD = new FuzzyRuleAntecedent(); iferrorZE_and_ifderrorNBD->joinWithAND(ZE,NBD); // Agregando el antecedente

FuzzyRuleConsequent* thenvelocidadPB11 = new FuzzyRuleConsequent(); thenvelocidadPB11->addOutput(PBO);// Agregando el consecuente a la salida

FuzzyRule*fuzzyRule22 = new FuzzyRule(22, iferrorZE_and_ifderrorNBD, thenvelocidadPB11); fuzzy->addFuzzyRule(fuzzyRule22); // Adicionando la regla


/*REGLA 23*/FuzzyRuleAntecedent*iferrorZE_and_ifderrorNMD = new FuzzyRuleAntecedent(); iferrorZE_and_ifderrorNMD->joinWithAND(ZE,NMD); // Agregando el antecedente

FuzzyRuleConsequent* thenvelocidadPB12 = new FuzzyRuleConsequent(); thenvelocidadPB12->addOutput(PBO);// Agregando el consecuente a la salida

FuzzyRule*fuzzyRule23 = new FuzzyRule(23, iferrorZE_and_ifderrorNMD, thenvelocidadPB12); fuzzy->addFuzzyRule(fuzzyRule23); // Adicionando la regla


/*REGLA 24*/FuzzyRuleAntecedent*iferrorZE_and_ifderrorNSD = new FuzzyRuleAntecedent(); iferrorZE_and_ifderrorNSD->joinWithAND(ZE,NSD); // Agregando el antecedente

FuzzyRuleConsequent* thenvelocidadPM3 = new FuzzyRuleConsequent(); thenvelocidadPM3->addOutput(PMO);// Agregando el consecuente a la salida

FuzzyRule*fuzzyRule24 = new FuzzyRule(24, iferrorZE_and_ifderrorNSD,
```

```
thenvelocidadPM3); fuzzy->addFuzzyRule(fuzzyRule24); // Adicionando la regla


 /*REGLA 25*/FuzzyRuleAntecedent*iferrorZE_and_ifderrorZED = new
FuzzyRuleAntecedent(); iferrorZE_and_ifderrorZED->joinWithAND(ZE,ZED); //
Agregando el antecedente

 FuzzyRuleConsequent* thenvelocidadZE3 = new FuzzyRuleConsequent();
thenvelocidadZE3->addOutput(ZEO);// Agregando el consecuente a la salida

 FuzzyRule*fuzzyRule25 = new FuzzyRule(25, iferrorZE_and_ifderrorZED,
thenvelocidadZE3); fuzzy->addFuzzyRule(fuzzyRule25); // Adicionando la regla


 /*REGLA 26*/FuzzyRuleAntecedent*iferrorZE_and_ifderrorPSD = new
FuzzyRuleAntecedent(); iferrorZE_and_ifderrorPSD->joinWithAND(ZE,PSD); //
Agregando el antecedente

 FuzzyRuleConsequent* thenvelocidadNM2 = new FuzzyRuleConsequent();
thenvelocidadNM2->addOutput(NMO);// Agregando el consecuente a la salida

 FuzzyRule*fuzzyRule26 = new FuzzyRule(26, iferrorZE_and_ifderrorPSD,
thenvelocidadNM2); fuzzy->addFuzzyRule(fuzzyRule26); // Adicionando la regla


 /*REGLA 27*/FuzzyRuleAntecedent*iferrorZE_and_ifderrorPMD = new
FuzzyRuleAntecedent(); iferrorZE_and_ifderrorPMD->joinWithAND(ZE,PMD); //
Agregando el antecedente

 FuzzyRuleConsequent* thenvelocidadNB1 = new FuzzyRuleConsequent();
thenvelocidadNB1->addOutput(NBO);// Agregando el consecuente a la salida

 FuzzyRule*fuzzyRule27 = new FuzzyRule(27, iferrorZE_and_ifderrorPMD,
thenvelocidadNB1); fuzzy->addFuzzyRule(fuzzyRule27); // Adicionando la regla


 /*REGLA 28*/FuzzyRuleAntecedent*iferrorZE_and_ifderrorPBD  = new
FuzzyRuleAntecedent(); iferrorZE_and_ifderrorPBD->joinWithAND(ZE,PBD); //
Agregando el antecedente
```

```
 FuzzyRuleConsequent* thenvelocidadNB2 = new FuzzyRuleConsequent();
thenvelocidadNB2->addOutput(NBO);// Agregando el consecuente a la salida

 FuzzyRule*fuzzyRule28 = new FuzzyRule(28, iferrorZE_and_ifderrorPBD,
thenvelocidadNB2); fuzzy->addFuzzyRule(fuzzyRule28); // Adicionando la regla



 /*REGLA 29*/FuzzyRuleAntecedent*iferrorPS_and_ifderrorNBD = new
FuzzyRuleAntecedent(); iferrorPS_and_ifderrorNBD->joinWithAND(PS,NBD); //
Agregando el antecedente

 FuzzyRuleConsequent* thenvelocidadPB13 = new FuzzyRuleConsequent();
thenvelocidadPB13->addOutput(PBO);// Agregando el consecuente a la salida

 FuzzyRule*fuzzyRule29 = new FuzzyRule(29, iferrorPS_and_ifderrorNBD,
thenvelocidadPB13); fuzzy->addFuzzyRule(fuzzyRule29); // Adicionando la regla



 /*REGLA 30*/FuzzyRuleAntecedent*iferrorPS_and_ifderrorNMD = new
FuzzyRuleAntecedent(); iferrorPS_and_ifderrorNMD->joinWithAND(PS,NMD); //
Agregando el antecedente

 FuzzyRuleConsequent* thenvelocidadPM4 = new FuzzyRuleConsequent();
thenvelocidadPM4->addOutput(PMO);// Agregando el consecuente a la salida

 FuzzyRule*fuzzyRule30 = new FuzzyRule(30, iferrorPS_and_ifderrorNMD,
thenvelocidadPM4); fuzzy->addFuzzyRule(fuzzyRule30); // Adicionando la regla



 /*REGLA 31*/FuzzyRuleAntecedent*iferrorPS_and_ifderrorNSD = new
FuzzyRuleAntecedent(); iferrorPS_and_ifderrorNSD->joinWithAND(PS,NSD); //
Agregando el antecedente

 FuzzyRuleConsequent* thenvelocidadZE4 = new FuzzyRuleConsequent();
thenvelocidadZE4->addOutput(ZEO);// Agregando el consecuente a la salida

 FuzzyRule*fuzzyRule31 = new FuzzyRule(31, iferrorPS_and_ifderrorNSD,
thenvelocidadZE4); fuzzy->addFuzzyRule(fuzzyRule31); // Adicionando la regla
```

```
/*REGLA 32*/FuzzyRuleAntecedent*iferrorPS_and_ifderrorZED = new
FuzzyRuleAntecedent(); iferrorPS_and_ifderrorZED->joinWithAND(PS,ZED); //
Agregando el antecedente

FuzzyRuleConsequent* thenvelocidadNM3 = new FuzzyRuleConsequent();
thenvelocidadNM3->addOutput(NMO);// Agregando el consecuente a la salida

FuzzyRule*fuzzyRule32 = new FuzzyRule(32, iferrorPS_and_ifderrorZED,
thenvelocidadNM3); fuzzy->addFuzzyRule(fuzzyRule32); // Adicionando la regla


/*REGLA 33*/FuzzyRuleAntecedent*iferrorPS_and_ifderrorPSD = new
FuzzyRuleAntecedent(); iferrorPS_and_ifderrorPSD->joinWithAND(PS,PSD); //
Agregando el antecedente

FuzzyRuleConsequent* thenvelocidadNB3 = new FuzzyRuleConsequent();
thenvelocidadNB3->addOutput(NBO);// Agregando el consecuente a la salida

FuzzyRule*fuzzyRule33 = new FuzzyRule(33, iferrorPS_and_ifderrorPSD,
thenvelocidadNB3); fuzzy->addFuzzyRule(fuzzyRule33); // Adicionando la regla


/*REGLA 34*/FuzzyRuleAntecedent*iferrorPS_and_ifderrorPMD = new
FuzzyRuleAntecedent(); iferrorPS_and_ifderrorPMD->joinWithAND(PS,PMD); //
Agregando el antecedente

FuzzyRuleConsequent* thenvelocidadNB4 = new FuzzyRuleConsequent();
thenvelocidadNB4->addOutput(NBO);// Agregando el consecuente a la salida

FuzzyRule*fuzzyRule34 = new FuzzyRule(34, iferrorPS_and_ifderrorPMD,
thenvelocidadNB4); fuzzy->addFuzzyRule(fuzzyRule34); // Adicionando la regla


/*REGLA 35*/FuzzyRuleAntecedent*iferrorPS_and_ifderrorPBD = new
FuzzyRuleAntecedent(); iferrorPS_and_ifderrorPBD->joinWithAND(PS,PBD); //
Agregando el antecedente

FuzzyRuleConsequent* thenvelocidadNB5 = new FuzzyRuleConsequent();
thenvelocidadNB5->addOutput(NBO);// Agregando el consecuente a la salida

FuzzyRule*fuzzyRule35 = new FuzzyRule(35, iferrorPS_and_ifderrorPBD,
```

```
thenvelocidadNB5); fuzzy->addFuzzyRule(fuzzyRule35); // Adicionando la regla


 /*REGLA 36*/FuzzyRuleAntecedent*iferrorPM_and_ifderrorNBD = new
FuzzyRuleAntecedent(); iferrorPM_and_ifderrorNBD->joinWithAND(PM,NBD); //
Agregando el antecedente

 FuzzyRuleConsequent* thenvelocidadPM5 = new FuzzyRuleConsequent();
thenvelocidadPM5->addOutput(PMO);// Agregando el consecuente a la salida

 FuzzyRule*fuzzyRule36 = new FuzzyRule(36, iferrorPM_and_ifderrorNBD,
thenvelocidadPM5); fuzzy->addFuzzyRule(fuzzyRule36); // Adicionando la regla


 /*REGLA 37*/FuzzyRuleAntecedent*iferrorPM_and_ifderrorNMD = new
FuzzyRuleAntecedent(); iferrorPM_and_ifderrorNMD->joinWithAND(PM,NMD); //
Agregando el antecedente

 FuzzyRuleConsequent* thenvelocidadZE5 = new FuzzyRuleConsequent();
thenvelocidadZE5->addOutput(ZEO);// Agregando el consecuente a la salida

 FuzzyRule*fuzzyRule37 = new FuzzyRule(37, iferrorPM_and_ifderrorNMD,
thenvelocidadZE5); fuzzy->addFuzzyRule(fuzzyRule37); // Adicionando la regla


 /*REGLA 38*/FuzzyRuleAntecedent*iferrorPM_and_ifderrorNSD = new
FuzzyRuleAntecedent(); iferrorPM_and_ifderrorNSD->joinWithAND(PM,NSD); //
Agregando el antecedente

 FuzzyRuleConsequent* thenvelocidadNM4 = new FuzzyRuleConsequent();
thenvelocidadNM4->addOutput(NMO);// Agregando el consecuente a la salida

 FuzzyRule*fuzzyRule38 = new FuzzyRule(38, iferrorPM_and_ifderrorNSD,
thenvelocidadNM4); fuzzy->addFuzzyRule(fuzzyRule38); // Adicionando la regla


 /*REGLA 39*/FuzzyRuleAntecedent*iferrorPM_and_ifderrorZED = new
FuzzyRuleAntecedent(); iferrorPM_and_ifderrorZED->joinWithAND(PM,ZED); //
Agregando el antecedente
```

```
 FuzzyRuleConsequent* thenvelocidadNB6 = new FuzzyRuleConsequent();
thenvelocidadNB6->addOutput(NBO);// Agregando el consecuente a la salida

 FuzzyRule*fuzzyRule39 = new FuzzyRule(39, iferrorPM_and_ifderrorZED,
thenvelocidadNB6); fuzzy->addFuzzyRule(fuzzyRule39); // Adicionando la regla



 /*REGLA 40*/FuzzyRuleAntecedent*iferrorPM_and_ifderrorPSD = new
FuzzyRuleAntecedent(); iferrorPM_and_ifderrorPSD->joinWithAND(PM,PSD); //
Agregando el antecedente

 FuzzyRuleConsequent* thenvelocidadNB7 = new FuzzyRuleConsequent();
thenvelocidadNB7->addOutput(NBO);// Agregando el consecuente a la salida

 FuzzyRule*fuzzyRule40 = new FuzzyRule(40, iferrorPM_and_ifderrorPSD,
thenvelocidadNB7); fuzzy->addFuzzyRule(fuzzyRule40); // Adicionando la regla



 /*REGLA 41*/FuzzyRuleAntecedent*iferrorPM_and_ifderrorPMD = new
FuzzyRuleAntecedent(); iferrorPM_and_ifderrorPMD->joinWithAND(PM,PMD); //
Agregando el antecedente

 FuzzyRuleConsequent* thenvelocidadNB8 = new FuzzyRuleConsequent();
thenvelocidadNB8->addOutput(NBO);// Agregando el consecuente a la salida

 FuzzyRule*fuzzyRule41 = new FuzzyRule(41, iferrorPM_and_ifderrorPMD,
thenvelocidadNB8); fuzzy->addFuzzyRule(fuzzyRule41); // Adicionando la regla



 /*REGLA 42*/FuzzyRuleAntecedent*iferrorPM_and_ifderrorPBD = new
FuzzyRuleAntecedent(); iferrorPM_and_ifderrorPBD->joinWithAND(PM,PBD); //
Agregando el antecedente

 FuzzyRuleConsequent* thenvelocidadNB9 = new FuzzyRuleConsequent();
thenvelocidadNB9->addOutput(NBO);// Agregando el consecuente a la salida

 FuzzyRule*fuzzyRule42 = new FuzzyRule(42, iferrorPM_and_ifderrorPBD,
thenvelocidadNB9); fuzzy->addFuzzyRule(fuzzyRule42); // Adicionando la regla
```

```
/*REGLA 43*/FuzzyRuleAntecedent*iferrorPB_and_ifderrorNBD = new FuzzyRuleAntecedent(); iferrorPB_and_ifderrorNBD->joinWithAND(PB,NBD); // Agregando el antecedente

FuzzyRuleConsequent* thenvelocidadZE6 = new FuzzyRuleConsequent(); thenvelocidadZE6->addOutput(ZEO);// Agregando el consecuente a la salida

FuzzyRule*fuzzyRule43 = new FuzzyRule(43, iferrorPB_and_ifderrorNBD, thenvelocidadZE6); fuzzy->addFuzzyRule(fuzzyRule43); // Adicionando la regla


/*REGLA 44*/FuzzyRuleAntecedent*iferrorPB_and_ifderrorNMD = new FuzzyRuleAntecedent(); iferrorPB_and_ifderrorNMD->joinWithAND(PB,NMD); // Agregando el antecedente

FuzzyRuleConsequent* thenvelocidadNM5 = new FuzzyRuleConsequent(); thenvelocidadNM5->addOutput(NMO);// Agregando el consecuente a la salida

FuzzyRule*fuzzyRule44 = new FuzzyRule(44, iferrorPB_and_ifderrorNMD, thenvelocidadNM5); fuzzy->addFuzzyRule(fuzzyRule44); // Adicionando la regla


/*REGLA 45*/FuzzyRuleAntecedent*iferrorPB_and_ifderrorNSD = new FuzzyRuleAntecedent(); iferrorPB_and_ifderrorNSD->joinWithAND(PB,NSD); // Agregando el antecedente

FuzzyRuleConsequent* thenvelocidadNB10 = new FuzzyRuleConsequent(); thenvelocidadNB10->addOutput(NBO);// Agregando el consecuente a la salida

FuzzyRule*fuzzyRule45 = new FuzzyRule(45, iferrorPB_and_ifderrorNSD, thenvelocidadNB10); fuzzy->addFuzzyRule(fuzzyRule45); // Adicionando la regla


/*REGLA 46*/FuzzyRuleAntecedent*iferrorPB_and_ifderrorZED = new FuzzyRuleAntecedent(); iferrorPB_and_ifderrorZED->joinWithAND(PB,ZED); // Agregando el antecedente

FuzzyRuleConsequent* thenvelocidadNB11 = new FuzzyRuleConsequent(); thenvelocidadNB11->addOutput(NBO);// Agregando el consecuente a la salida

FuzzyRule*fuzzyRule46 = new FuzzyRule(46, iferrorPB_and_ifderrorZED,
```

```
thenvelocidadNB11); fuzzy->addFuzzyRule(fuzzyRule46); // Adicionando la regla


 /*REGLA 47*/FuzzyRuleAntecedent*iferrorPB_and_ifderrorPSD = new
FuzzyRuleAntecedent(); iferrorPB_and_ifderrorPSD->joinWithAND(PB,PSD); //
Agregando el antecedente

 FuzzyRuleConsequent* thenvelocidadNB12 = new FuzzyRuleConsequent();
thenvelocidadNB12->addOutput(NBO);// Agregando el consecuente a la salida

 FuzzyRule*fuzzyRule47 = new FuzzyRule(47, iferrorPB_and_ifderrorPSD,
thenvelocidadNB12); fuzzy->addFuzzyRule(fuzzyRule47); // Adicionando la regla


 /*REGLA 48*/FuzzyRuleAntecedent*iferrorPB_and_ifderrorPMD = new
FuzzyRuleAntecedent(); iferrorPB_and_ifderrorPMD->joinWithAND(PB,PMD); //
Agregando el antecedente

 FuzzyRuleConsequent* thenvelocidadNB13 = new FuzzyRuleConsequent();
thenvelocidadNB13->addOutput(NBO);// Agregando el consecuente a la salida

 FuzzyRule*fuzzyRule48 = new FuzzyRule(48, iferrorPB_and_ifderrorPMD,
thenvelocidadNB13); fuzzy->addFuzzyRule(fuzzyRule48); // Adicionando la regla


 /*REGLA 49*/FuzzyRuleAntecedent*iferrorPB_and_ifderrorPBD = new
FuzzyRuleAntecedent(); iferrorPB_and_ifderrorPBD->joinWithAND(PB,PBD); //
Agregando el antecedente

 FuzzyRuleConsequent* thenvelocidadNB14 = new FuzzyRuleConsequent();
thenvelocidadNB14->addOutput(NBO); // Agregando el consecuente a la salida

 FuzzyRule*fuzzyRule49 = new FuzzyRule(49, iferrorPB_and_ifderrorPBD,
thenvelocidadNB14); fuzzy->addFuzzyRule(fuzzyRule49); // Adicionando la regla

  }
```