TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE TUXTLA GUTIÉRREZ

DEPARTAMENTO DE ELECTRICA Y ELECTRONICA

INGENIERIA ELECTRONICA

RESIDENCIA PROFESIONAL

NOMBRE DEL PROYECTO:

SISTEMA DE CONTROL DE VELOCIDAD BASADO EN LÓGICA DIFUSA DE UN VEHÍCULO RECOLECTOR DE PET

Trabajo:
INFORME TÉCNICO

Méndez Hernández Alejandro Isidoro 132709876

Carrera: Ingeniería Electrónica.

Asesor: Ing. Álvaro Hernández Sol

Tuxtla Gutiérrez Chiapas; enero del 2018

-

INDICE

CAPITULO I

1.1 INTRODUCCIÓN

La lógica difusa intenta emular el comportamiento humano mediante el uso de conjuntos difusos o no bien definidos. Ha cobrado mucho auge en las últimas décadas debido a la simplicidad de su aplicación, ya que se puede implementar sin necesidad de conocer el modelo matemático del sistema (que de hecho realizar un sistema de control difuso en un sistema que se puede modelar y resolver mediante un modelo matemático produce peores resultados) más sin embargo existe la necesidad del conocimiento del comportamiento del sistema o lo que es lo mismo el conocimiento de un experto que ha trabajado con el sistema.

La lógica difusa se basa en el hecho de que una afirmación no tiene que ser estrictamente falsa o verdadera, si no que puede tomar valores intermedios los cuales se evalúan por medio de un grado de membresía. Por ejemplo, si usando lógica convencional definimos un conjunto para temperatura caliente con un rango de valores de 30º a 50º C y un conjunto para temperatura fría con un rango de valores menores a 30° hasta 0° al evaluarlos nos daríamos cuenta que para una temperatura igual a 29.9° seguiría perteneciendo al conjunto de temperatura fría, lo cual en la realidad este valor es más caliente que frio. Usando conjuntos difusos con los mismos valores el resultado sería de una pertenencia mayor al 95% al conjunto caliente y menos del 5% al conjunto frio. Con el ejemplo anterior podemos darnos cuenta que la lógica difusa presenta una mejor representación de la realidad.

En este trabajo se presentan las simulaciones y pruebas que se realizaron para llevar a cabo este proyecto, el cual consiste en aplicar la lógica difusa para realizar el control de velocidad de un vehículo recolector de PET. El vehículo fue armado con piezas de VEX ROBOTICS y controlado con un microcontrolador Arduino.

1.2 JUSTIFICACIÓN

La lógica difusa ofrece la ventaja de aplicar el control sin necesidad de conocer el modelo matemático del sistema. Lo cual es adecuado en el caso de este proyecto ya que no tenemos el modelo matemático del vehículo.

Sin duda se han realizado muchos prototipos de vehículos recolectores de PET, más sin embargo son pocos o al menos en el Instituto Tecnológico Nacional de Tuxtla Gutiérrez, no se ha realizado un prototipo implementándole un control de tipo difuso.

Además como ya se dijo antes el uso de la lógica difusa en sistemas de control ha cobrado mucho auge desde que se implementó el primer control de este tipo por lo cual se hace imperativo sumarnos a esta línea de investigación y realizar proyectos para usar esta herramienta a nuestro beneficio.

1.3 OBJETIVOS

1.3.1 OBJETIVO GENERAL

- Realizar un sistema para el control de velocidad basado en lógica difusa, de un vehículo autónomo recolector de PET.

1.3.2 OBJETIVOS ESPECIFICOS

- Construir el vehículo.
- Obtener la pendiente máxima a la cual el vehículo podrá mantener la velocidad constante.
- Diseñar el controlador difuso con Matlab.
- Implementar los conjuntos difusos simulados en Matlab en Arduino.

1.4 CARACTERIZACION DEL ÁREA EN QUE SE PARTICIPÓ

1.4.1 ANTECDENTES DE LA EMPRESA

El Instituto Tecnológico de Tuxtla Gutiérrez, ubicado en carretera Panamericana km.1080, como se observa en la figura 2.1, brinda servicios educativos de calidad certificada, con la misión de formar de manera integral a profesionistas de excelencia en el campo de la ciencia y la tecnología con actitud emprendedora, respeto al ambiente y apego a los valores éticos.

Además, es una institución de excelencia en la educación superior tecnológica del sureste, comprometida con el desarrollo sustentable de la región, el cual es uno de los ejes transversales de su oferta educativa, tanto a nivel licenciatura como posgrado.

Así mismo, trabaja con sistemas de gestión ambiental certificados, en donde los estudiantes aplican sus conocimientos en función del cuidado del ambiente: aire, agua, suelo, flora, fauna y seres humanos.

Entre las principales líneas estratégicas de educar con responsabilidad ambiental se encuentran los Sistemas de Gestión de Calidad, Sistema de Gestión Ambiental, Programas Académicos Acreditados por COPAES y por CONACYT; se trata de transformar conciencias, evolucionar culturas, sensibilizar a la población y mostrar responsabilidad ante las acciones que realiza en pro de la conservación de la biodiversidad del planeta.

Como parte de la oferta educativa de esta institución, se encuentra el departamento de Ingeniería Eléctrica y Electrónica el cual se encarga de coordinar distintas actividades académicas que permiten a los alumnos poner en práctica los conocimientos adquiridos dentro de las aulas, además de ayudarlos a desarrollar habilidades y aptitudes que les permitan resolver problemas de manera eficiente.

Algunas actividades que dicho departamento realiza son:

- EXPROYECA: exposición en la cual los alumnos presentan los proyectos realizados en distintas materias.
- Torneo Estatal Interuniversitario de Robótica: es un concurso que reúne a distintas universidades de la ciudad para competir en distintas categorías, tales como zumo, mini zumo, seguidor de línea, recolector de PET y laberinto.

- Certificaciones: como parte de su compromiso con la calidad educativa de los alumnos se prepara y brinda la posibilidad de certificar a los alumnos en el software SOLIDWORKS.
- Conferencias: a lo largo de su formación académica los alumnos reciben conferencias que abordan temas relacionados con la carrera.

De igual manera, es importante mencionar que este departamento cuenta con el apoyo de distintos laboratorios dentro de la institución, uno de ellos es el Laboratorio de Ingeniería Electrónica en el cual los alumnos refuerzan los conocimientos teóricos adquiridos en clases mediante prácticas que les permiten desarrollar capacidades y habilidades para su desenvolvimiento futuro como profesionistas.

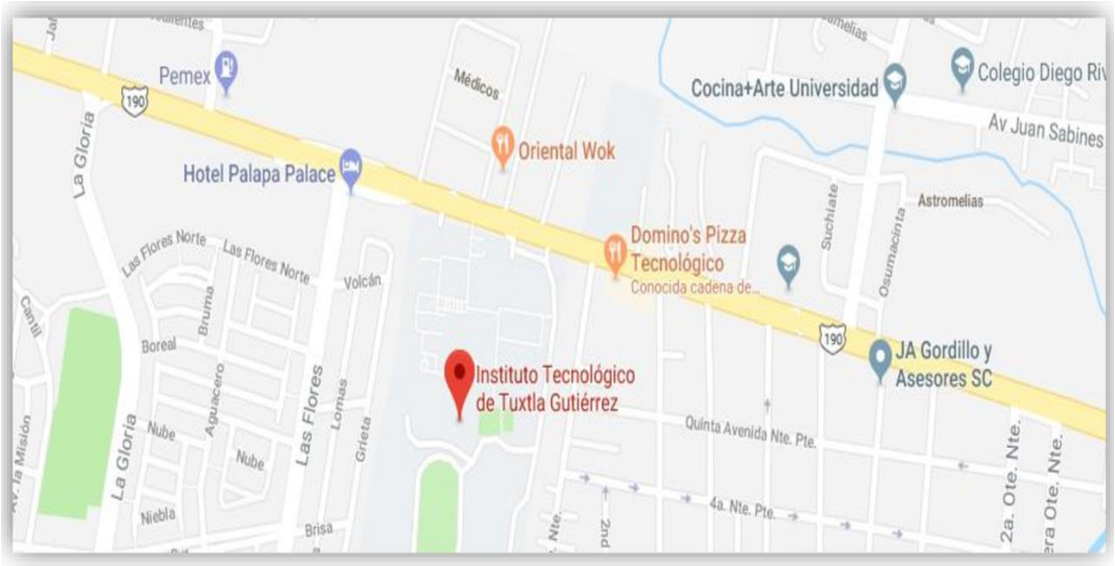


Figura 1.1 Mapa de la ubicación del Instituto Tecnológico de Tuxtla Gutiérrez

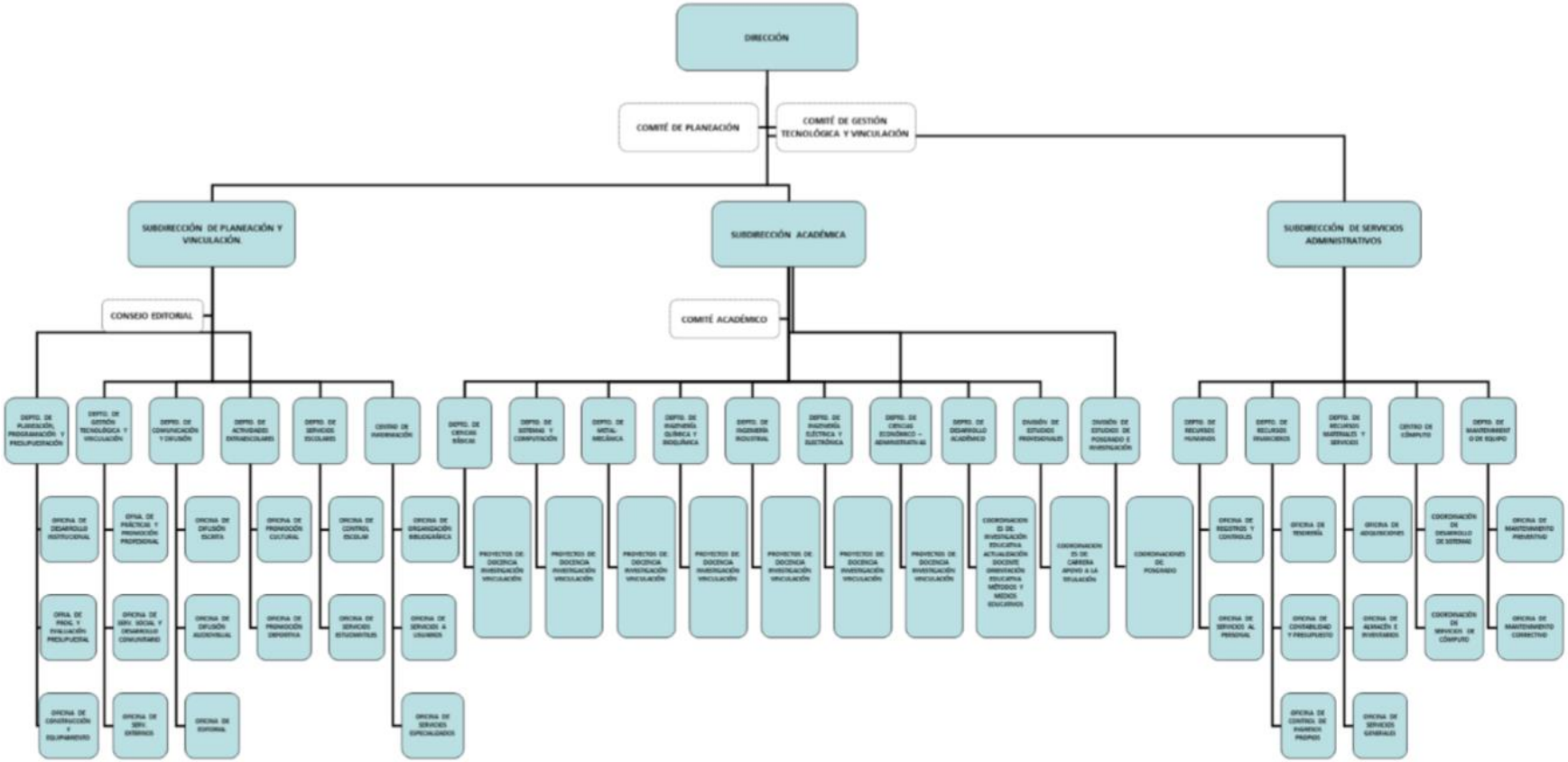


Figura 1.2 Organigrama de la empresa

1.4.2 MISIÓN, VISIÓN Y VALORES

- **Misión**

Formar de manera integral profesionistas de excelencia en el campo de la ciencia y la tecnología con actitud emprendedora, respeto al medio ambiente y apego a los valores éticos.

- **Visión**

Ser una Institución de Excelencia en la Educación Superior Tecnológica del Sureste, comprometida con el desarrollo socioeconómico sustentable de la región.

- **Valores**

El ser humano, el espíritu de servicio, el liderazgo, el trabajo en equipo, la calidad, el alto desempeño y respeto al medio ambiente.

2.1.4 DESCRIPCIÓN DEL ÁREA DÓNDE SE REALIZÓ EL PROYECTO

En el Instituto Tecnológico de Tuxtla Gutiérrez, como parte del apoyo académico que se brinda, el Laboratorio de Ingeniería Electrónica cuenta con el Área de Investigación y Desarrollo Tecnológico, la cual se encarga de gestionar diversas actividades que tienen como objetivo principal ayudar a la formación académica de los alumnos; dicha área se encuentra bajo la coordinación del Ing. Álvaro Hernández Sol.

Entre las actividades más relevantes que esta área coordina se encuentran:

- Curso-Taller de programación de la tarjeta de desarrollo ARDUINO: es un curso teórico-práctico de los aspectos básicos de dicha tarjeta y se imparte, generalmente, a los alumnos de primer ingreso. Es importante recalcar que dicho curso es impartido por alumnos de la carrera con mayor avance curricular.
- Torneo Estatal de Robótica VEX-REEDUCA-ITTG: concurso organizado por la empresa REEDUCA en conjunto con el Instituto Tecnológico de Tuxtla Gutiérrez, en el cual alumnos de instituciones de nivel medio superior y superior del estado de Chiapas compiten en diversas categorías.

De igual manera, en esta área se desarrollan distintos proyectos en conjunto con los alumnos, algunos de estos proyectos son:

- Prototipo de rostro humano: el objetivo de este proyecto consiste en crear una cara robótica que sea capaz de realizar gestos humanos.

- Guantes traductores: este proyecto tiene como beneficiarios a los discapacitados del habla, ya que traduce el lenguaje de señas a audio.
- Pantalla traductora: este proyecto beneficia a los discapacitados auditivos ya que por medio de una pantalla muestra en lenguaje de señas el audio recibido.

1.5 PROBLEMAS A RESOLVER

El controlador difuso de velocidad le brindará al vehículo recolector de PET autonomía al mantener la velocidad constante ya que como sabemos en una aplicación real los terrenos por donde se desplaza un vehículo no son uniformes y tienen subidas y bajadas.

1.6 ALCANCES Y LIMITACIONES

1.6.1 ALCANCES

- Se obtendrá la velocidad óptima para la cual el control podrá brindar una respuesta aceptable.

- Se obtendrán las pendientes a las cuales el vehículo podrá mantener la velocidad constante.

- El control difuso entregará a la salida un valor de potencia modulado por ancho de pulso (PWM), el cual servirá para aplicarle mayor potencia a los motores en caso de pendientes positivas y menor potencia en caso de pendientes negativas.

1.6.2 LIMITACIONES

- La velocidad constante será de 2.6 m/s y no habrán más opciones.
- El rango de pendientes a la cual el control brindara una respuesta aceptable estará limitado y dependerá de las características de los motores. La pendiente máxima fijada fue a los 10°.
- El vehículo no cuenta con manejo por control remoto o sistema de navegación por lo que solo avanza en línea recta.

CAPITULO II LÓGICA DIFUSA Y SISTEMAS DE CONTROL

2.1 Conjuntos Difusos

La teoría sobre conjuntos difusos, partió de observaciones realizadas por Zadeh para el modelado de sistemas y la limitante que en repetidas ocasiones se tenía para describir ciertos fenómenos mediante expresiones matemáticas. Por tal motivo, plantea su principio de la incompatibilidad el cual predica que: “Conforme la complejidad de un sistema aumenta, nuestra capacidad para ser precisos y construir instrucciones sobre su comportamiento disminuye hasta el umbral más allá del cual, la precisión y el significado son características excluyentes” [1].

En 1985 Takagi y Sugeno desarrollaron un nuevo método para la aplicación del control difuso llamado Takagi-Sugeno-Kang (TSK), como una alternativa del método Mamdani.

Un conjunto difuso es una clasificación de objetos con un grado de membresía continuo. Como conjunto, es caracterizado por una función de membresía (característica), la cual asigna a cada elemento un grado de membresía dentro del rango de cero a uno (0,1) [1]**.**

Sea $\mathrm{X} = \{\mathrm{x}\}$ un conjunto clásico cuyos elementos son representados por$x$. Luego un conjunto difuso A en X representa un conjunto de pares ordenados:

$$A = \big(x, \mu_A(x)\big), \qquad x \in X$$

Por ejemplo, supóngase que se desea clasificar a los miembros de un equipo de fútbol según su estatura en tres conjuntos, bajos, medianos y altos. Podría plantearse que se es bajo si se tiene una estatura inferior a, por ejemplo, 160 cm, que se es mediano si la estatura es superior o igual a 160 cm e inferior a 180 cm. y se es alto si la estatura es superior o igual a 180 cm, con lo que se lograría una clasificación en conjuntos concretos [2].

Sin embargo. ¿Qué tan grande es la diferencia que existe entre dos jugadores del equipo, uno con estatura de 179,9 cm y otro de 180,0 cm? Ese milímetro de diferencia quizás no represente en la práctica algo significativo, y sin embargo los dos jugadores han quedado rotulados con etiquetas distintas: uno es mediano y el otro es alto. Si se optase por efectuar la misma clasificación con conjuntos difusos, estos cambios abruptos se evitarían. Debido a que las fronteras entre los conjuntos permitirían cambios graduales en la clasificación.

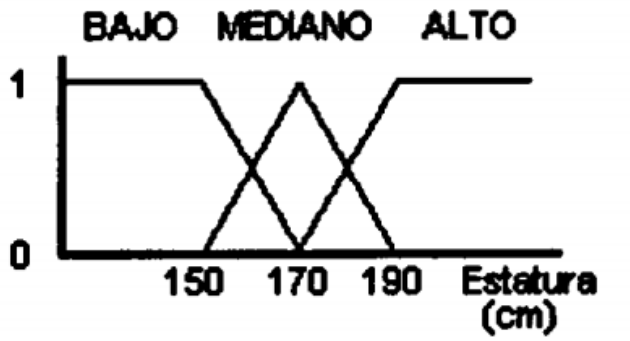


Figura 2.1. Funciones de pertenencia del ejemplo

La figura 2.1 muestra cómo podría hacerse tal clasificación: el universo de discurso sería el conjunto continuo de todas las posibles estaturas (el intervalo [130 cm, 210 cm] por ejemplo). Las funciones de pertenencia de cada uno de los tres conjuntos bajo, mediano y alto se han graficado. La forma de estas funciones de pertenencia no debe ser necesariamente la de la figura 2.1, pues depende de lo que se entienda por bajo, mediano y alto.

2.1.2 FUNCIONES DE MEMBRESIA

Las funciones de membresía constituyen un método utilizado para representar los conjuntos difusos, de modo que a cada punto en el espacio se le asigna un valor o grado de pertenencia a dicho conjunto, definiendo así al conjunto mismo. De este modo, $\mu_A(x)$ es el grado de membresía de x en A y es limitado a valores entre 0 y 1, siendo 0 el más bajo y 1 el más alto. Nótese el contraste con la teoría de conjuntos clásica o discreta, en el cual $\mu_A(x)$ solo puede valer 1 indicando que x pertenece a A, o 0 en caso contrario [1].

Las funciones de membresía son clasificadas según su forma geométrica, donde las más utilizadas son la triangular, trapezoidal, gaussiana y sigmoidal, las cuales se muestran en la figura 2.3.

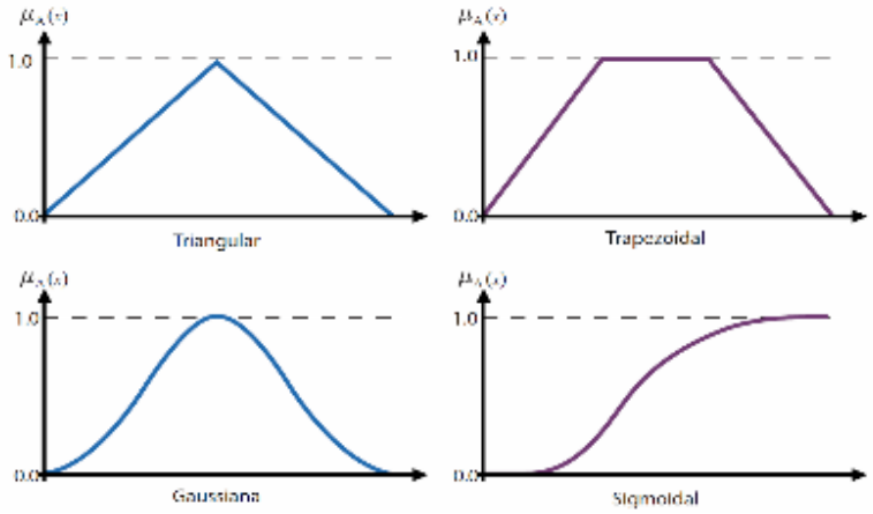


Figura 2.2. Funciones de membresía comúnmente utilizadas

- Función Triangular

Definida mediante el límite inferior a, el superior b y el valor modal m, tal que $a < m < b$. La función no tiene porqué ser simétrica.

$$\mu_A(x)\begin{cases} 0, & si\ x \le a \\ \dfrac{x-a}{m-a}, & si\ a < x \le m \\ \dfrac{b-x}{m-b}, & si\ m < x < b \\ 0, & si\ x \ge b \end{cases}$$

- Función Trapezoidal

Definida por sus límites inferior a, superior d, y los límites de soporte inferior b y superior c, tal que $a < b < c < d$.

$$\mu_A(x)\begin{cases} 0, & si\ (x < a)\ o\ (x > d) \\ \dfrac{x-a}{b-a}, & si\ a \le x \le b \\ 1, & si\ b \le x \le c \\ \dfrac{d-x}{d-c}, & si\ c \le x \le d \end{cases}$$

En este caso, si los valores de b y c son iguales, se obtiene una función triangular.

Existen otros tipos de funciones de membresía, pero basta con mencionar las funciones triangular y trapezoidal ya que son las únicas que se utilizaron en este proyecto.

2.1.1 OPERACIONES DIFUSAS

A los subconjuntos se les puede aplicar determinados operadores o bien se puede realizar operaciones entre ellos. Al aplicar un operador sobre un solo conjunto se obtendrá otro conjunto, lo mismo sucede cuando se realiza una operación entre conjuntos.

Las operaciones lógicas se utilizan en controladores y modelos difusos, son necesarias en la evaluación del antecedente de reglas.

Se definen a continuación 3 operaciones básicas a realizar sobre conjuntos, estas operaciones son complemento, unión e intersección. Sean las etiquetas A y B las que identifican a dos conjuntos borrosos asociados a una variable lingüística x, las operaciones se definen como:

- Complemento $\mu_{\bar{A}}(x) = 1 - \mu_A(x)$
- Unión. Operador lógico OR $\mu_{A\cup B}(x) = \max[\mu_A(x), \mu_B(x)]$
- Intersección. Operador lógico AND $\mu_{A\cap B}(x) = \min[\mu_A(x), \mu_B(x)]$

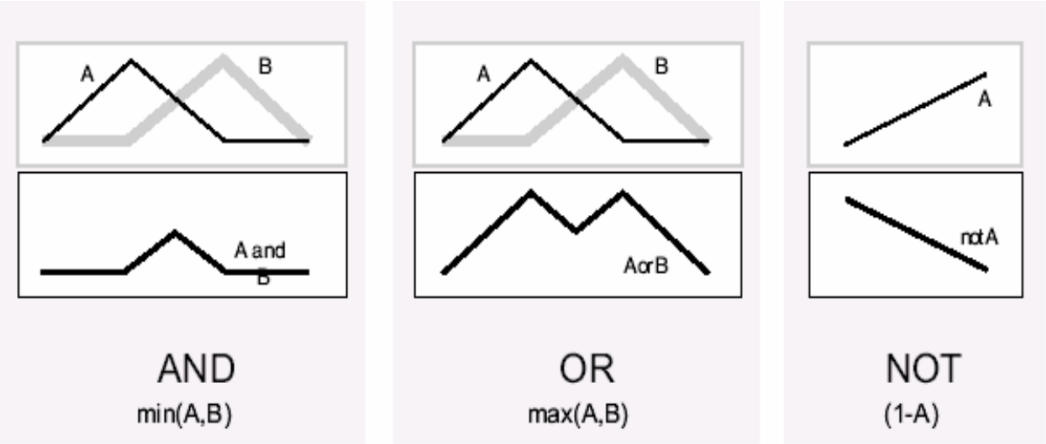


Figura 2.3. Representación de las operaciones básicas entre conjuntos difusos

2.2 CONTROL DIFUSO

Un sistema de control difuso es un dispositivo capaz de interpretar señales de campo, recibidas a través de los sensores dispuestos en el proceso y tomar una acción de control conforme a la base de reglas definidas en su motor de inferencia, estas acciones son enviadas a los actuadores, los cuales permiten modificar el estado del proceso [3].

El control difuso, puede ser expresado mejor como un control a través de palabras que interpretan el sentido común, en lugar de números, o bien sentencias en lugar de ecuaciones [1].

En la siguiente figura se muestra el proceso y las transformaciones que sufren las variables.

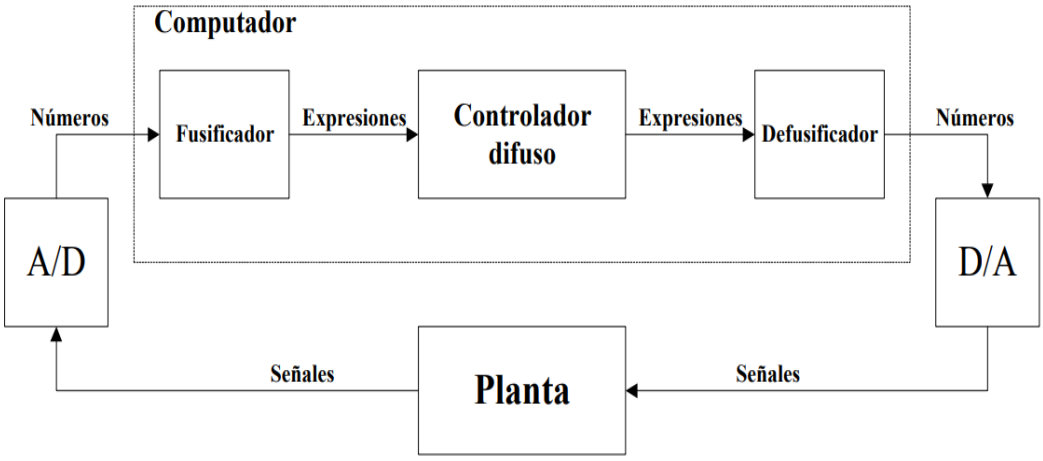


Figura 2.4. Lazo de control difuso

2.2.1 FUSIFICACION

A la traducción de los valores del mundo real a lógica difusa, a través de funciones de membresía, se le llama fuzzyficación [4].

Durante este proceso se podría obtener mayor resolución al representar el dominio con mayor cantidad de conjuntos difusos, lo cual resulta inconveniente puesto que se produce un incremento en la cantidad de reglas causando un aumento de la complejidad de cálculo y por consiguiente mayores gastos computacionales. La cantidad de conjuntos que aquí se definen debe ser impar (usualmente entre 3 y 9), con el fin de que esté disponible un punto central que tiende a inhibir oscilaciones numéricas entre conjuntos adyacentes [5].

2.2.2 BASE DE CONOCIMIENTO

Los controladores difusos usan reglas, estas combinan uno o más conjuntos borrosos de entrada llamados antecedentes o premisas y le asocian un conjunto borroso de salida llamado consecuente o consecuencia. Involucran a conjuntos difusos, lógica difusa e inferencia difusa. A estas reglas se les llama reglas borrosas o difusas o fuzzy rules. Son afirmaciones del tipo SI-ENTONCES. Los conjuntos borrosos del antecedente se asocian mediante operaciones lógicas borrosas AND, OR, etc. [6].

Las reglas borrosas son proposiciones que permiten expresar el conocimiento que se dispone sobre la relación entre antecedentes y consecuentes. Para expresar este conocimiento de manera completa normalmente se precisan varias reglas, que se agrupan formando lo que se conoce como base de reglas, es decir, la edición de esta base determina cual será el comportamiento del controlador difuso y es aquí donde se emula el conocimiento o experiencia del operario y la correspondiente estrategia de control.

Existe una gran variedad de tipos de reglas, dos grandes grupos son los que en general se emplean, las reglas difusas de Mamdani y las reglas difusas de TakagiSugeno.

- Reglas difusas de Mandami

Está definida por las siguientes condiciones:

$$IF\ x_1\ es\ A\ and\ x_2\ es\ B\ and\ x_3\ es\ C\ then\ u_1\ es\ D, u_2\ es\ E$$

Donde x_1, x_2 y x_3 son las variables de entrada (por ejemplo, error, derivada del error y derivada segunda del error), A, B y C son funciones de membresía de entrada (p.ej., alto, medio, bajo), u_1 y u_2 son las acciones de control (p.ej.,

apertura de válvulas) en sentido genérico son todavía variables lingüísticas (todavía no toman valores numéricos), D y E son las funciones de membresía de la salida, en general se emplean singleton por su facilidad computacional, y and es un operador lógico difuso, podría ser otro. La primera parte de la sentencia "$IF\ x_1\ es\ A\ and\ x_2\ es\ B\ and\ x_3\ es\ C$" es el antecedente y la restante es el consecuente [6].

- Reglas difusas de Takagi-Sugeno

Está definida por la siguiente expresión:

$$IF\ x_1\ es\ A\ and\ x_2\ es\ B\ and\ x_3\ es\ C\ then\ u_1 = f(x_1, x_2, x_3), u_2 = g(x_1, x_2, x_3)$$

En principio es posible emplear $f(x)$ y $g(x)$ como funciones no lineales, pero la elección de tal función puede ser muy compleja, por lo tanto en general se emplean funciones lineales [6].

2.2.3 DEFUSIFICACION

El resultado producido de la evaluación de reglas es por supuesto difuso (salidas difusas), el cual es una expresión lingüística que si la consideramos la salida de nuestro control, tendríamos el problema de que nuestro sistema a controlar no pueda interpretar tal orden lingüístico, lo que hace necesario convertir estas salidas difusas en un valor real, a esto se le llama Generación de Valores Reales. El cuál combina todas las salidas difusas en un resultado específico para cada variable de salida, para esto se emplean diferentes métodos [7].

Los métodos que más frecuentemente se usan en la Generación de valores reales son los siguientes:

• Centro de Gravedad

• Centro de Gravedad por Singleton

• Medio de Máximo (M o M)

• Izquierdo de Máximo (L o M)

• Derecho de máximo (R o M)

A continuación se describe únicamente el método de centro de gravedad que es el utilizado en este proyecto.

La defusificación (defuzzyfication) es un proceso matemático usado para convertir un conjunto difuso en un número real. El sistema de inferencia difusa obtiene una

conclusión a partir de la información de la entrada, pero es en términos difusos. Esta conclusión o salida difusa es obtenida por la etapa de inferencia borrosa, esta genera un conjunto borroso pero el dato de salida del sistema debe ser un número real y debe ser representativo de todo el conjunto obtenido en la etapa de agregado, es por eso que existen diferentes métodos de defusificación y arrojan resultados distintos, el "más común y ampliamente usado" es el centroide [8].

Con el método de defusificación del centroide (ecuación 2.1) se transforma la salida difusa en un número real el cual es la coordenada equis (x) del centro de gravedad de tal conjunto difuso de salida.

$$y_d = \frac{\int_S y\mu_Y(y)dy}{\int_s \mu_Y(y)dy}$$

Ecuación 2.1 Defusificación por centro de gravedad

Donde μ_Y es la función de pertenencia del conjunto de salida Y, cuya variable de salida es y, S es el dominio o rango de integración.

CAPITULO III MATLAB Y ARDUINO

3.1 FUZZYLOGIC

Fuzzy Logic Toolbox ™ proporciona funciones, aplicaciones y un bloque Simulink ® para analizar, diseñar y simular sistemas basados en lógica difusa.

La caja de herramientas permite modelar comportamientos complejos del sistema usando reglas lógicas simples, y luego implementar estas reglas en un sistema de inferencia difusa.

Para acceder al toolbox fuzzy se debe digitar la palabra fuzzy en la línea de comandos y luego oprimir enter.

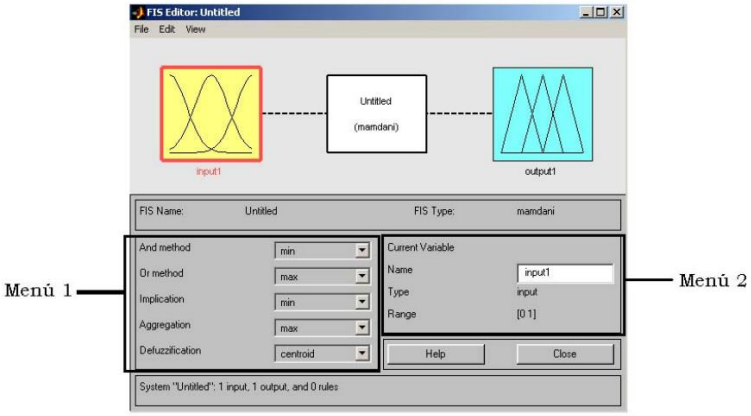


Figura 3.1. Menú principal del Fuzzy Toolbox

En el Menú 1, se podrá modificar los métodos de los operadores lógicos and y or, los métodos de implicación, de agregación y de defuzificación.

En el Menú 2, se podrá cambiar el nombre de la variable que se encuentre seleccionada, por ejemplo, modificar el nombre “input1” por “Pendiente”.

Para elegir el tipo de modelo a usar, Sugeno o Mamdani, se debe acceder al menú File -> New FIS… -> Mamdani (Sugeno).

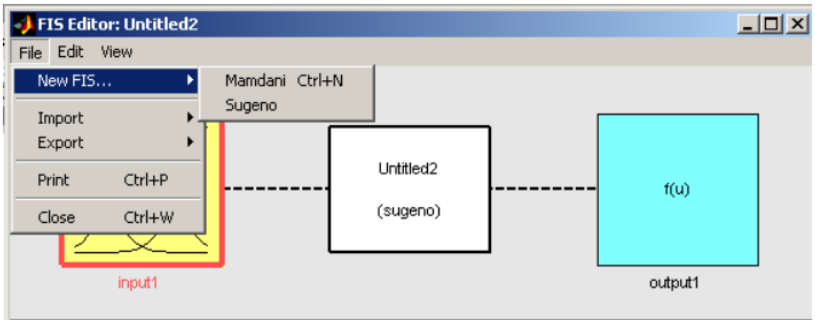


Figura 3.2 Elección de Modelo

Para agregar una variable, ya sea de entrada o de salida, se debe seleccionar el menú Edit -> Add Variable -> Input (Output).

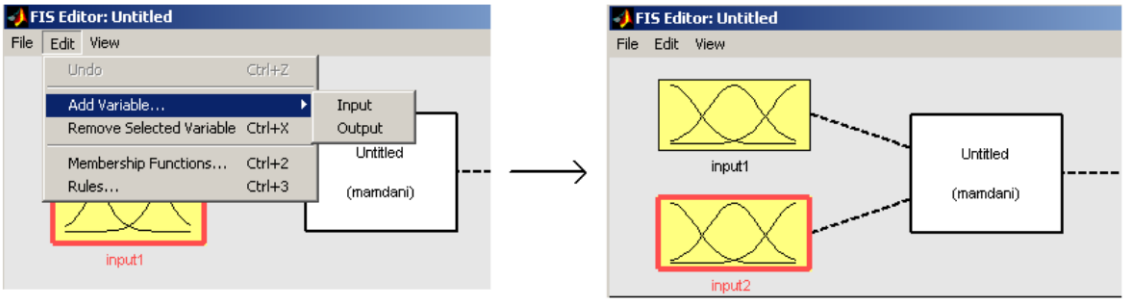


Figura 3.3 Agregar una variable

Las funciones de pertenencia, tanto para las variables de entrada como para las de salida, se modifican en un menú especial Membership Function Editor que aparece al hacer doble click en la variable de interés.

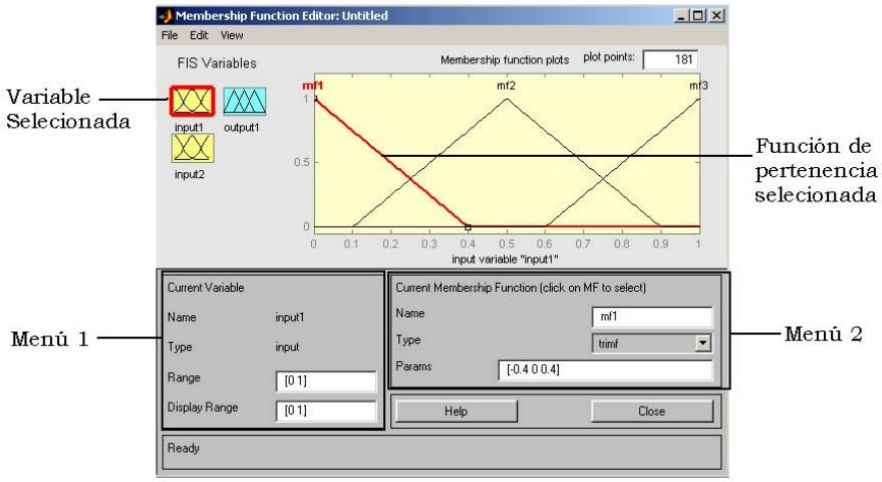


Figura 3.4 Editor de Funciones de Pertenencia, Membership Editor

En el Menú 1, se puede modificar el rango de la función de pertenencia, en el cual la estará definida.

En el Menú 2, es posible modificar el nombre de la función de pertenencia, los parámetros de la función de pertenencia y también su forma, la cual está seleccionada triangular en este caso, siendo ésta la más común.

Para poder modificar las reglas del modelo se debe acceder al Rule Editor, haciendo doble click sobre el modelo.

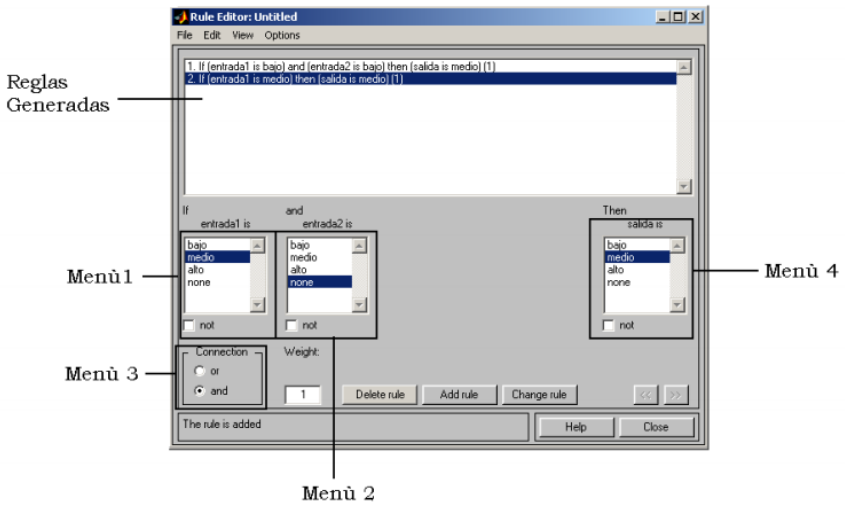


Figura 3.5 Editor de Reglas, Rule Editor

Según el número de variables de entrada y salida que existan y sus funciones de pertenencia será el número de reglas que es posible generar. En el Menú 1 se selecciona el valor que toma la primera variable de entrada, en el Menú 2, el valor que toma la segunda variable de entrada (si es necesario es posible negarla marcando not). En el Menú 3, se selecciona el tipo de conexión lógica entre ambos valores seleccionados (and, or), finalmente, en el Menú 4, se selecciona la salida que deberá entregar el controlador para los valores de entrada ya indicados. Luego, se presiona el botón Add rule, y la regla es agregada.

Para Eliminar una regla basta seleccionarla y apretar el botón Delete rule. Para modificarla se debe hacer click en el botón Change rule.

3.2 ARDUINO

Arduino Uno es una placa electrónica basada en el microcontrolador ATmega328. Cuenta con 14 entradas/salidas digitales, de las cuales 6 se pueden utilizar como salidas PWM (Modulación por ancho de pulsos) y otras 6 son entradas analógicas. Además, incluye un resonador cerámico de 16 MHz, un conector USB, un conector de alimentación, una cabecera ICSP y un botón de reseteado. La placa incluye todo lo necesario para que el microcontrolador haga su trabajo, basta conectarla a un ordenador con un cable USB o a la corriente eléctrica a través de un transformador.

3.2.1 CARACTERÍSTICAS TÉCNICAS DE ARDUINO UNO R3

- Microcontrolador: ATmega328
- Voltaje: 5V
- Voltaje entrada (recomendado): 7-12V
- Voltaje entrada (limites): 6-20V
- Digital I/O Pin: 14 (de los cuales 6 son salida PWM)
- Entradas Analógicas: 6
- DC Corriente para I/O Pin: 40 mA
- DC Corriente para 3.3V Pin: 50 mA
- Flash Memory: 32 KB (ATmega328) de los cuales 0.5 KB son utilizados para el arranque
- SRAM: 2 KB (ATmega328)
- EEPROM: 1 KB (ATmega328)
- Clock Speed: 16 MHz

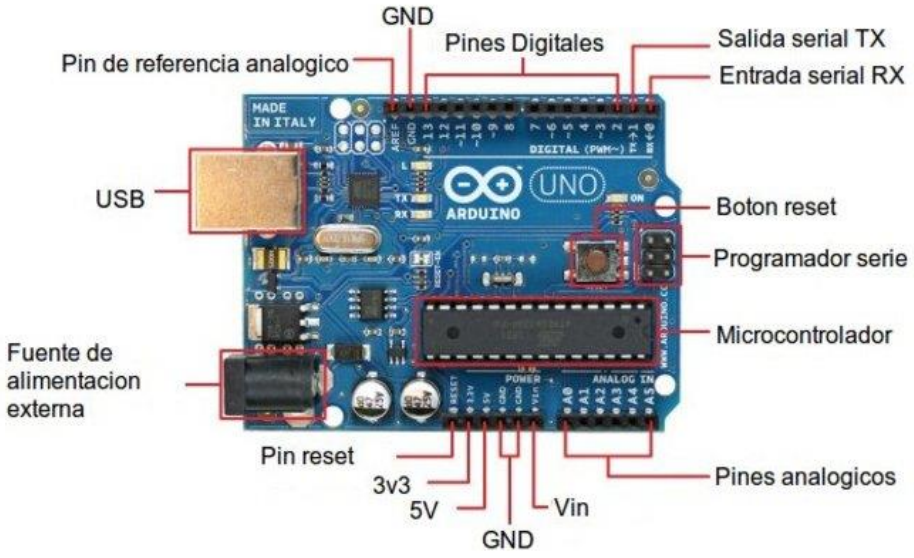


Figura 3.6 Diagrama de pines del Arduino uno

La placa de Arduino cuenta con 6 pines de entradas analógicas, que van desde el pin A0 al A5, de los cuales proporcionan 10bits, llamados bits de resolución. La tensión que miden va de 0 a 5v, aunque es posible cambiar su rango usando una función con el pin AREF.

Las entradas y salidas digitales son 14 y van desde el pin 0 al 13 y ofrecen una tensión de 5v.

3.2.2 PROGRAMACION

El entorno de programación más utilizado es Arduino IDE, descargable desde la página oficial de Arduino. Lo único que debemos hacer es seleccionar el sistema operativo desde donde vamos a ejecutar nuestro entorno Arduino IDE y listo.

Cuando abrimos el entorno de programación "Arduino IDE" veremos la siguiente ventana donde podremos empezar a programar:

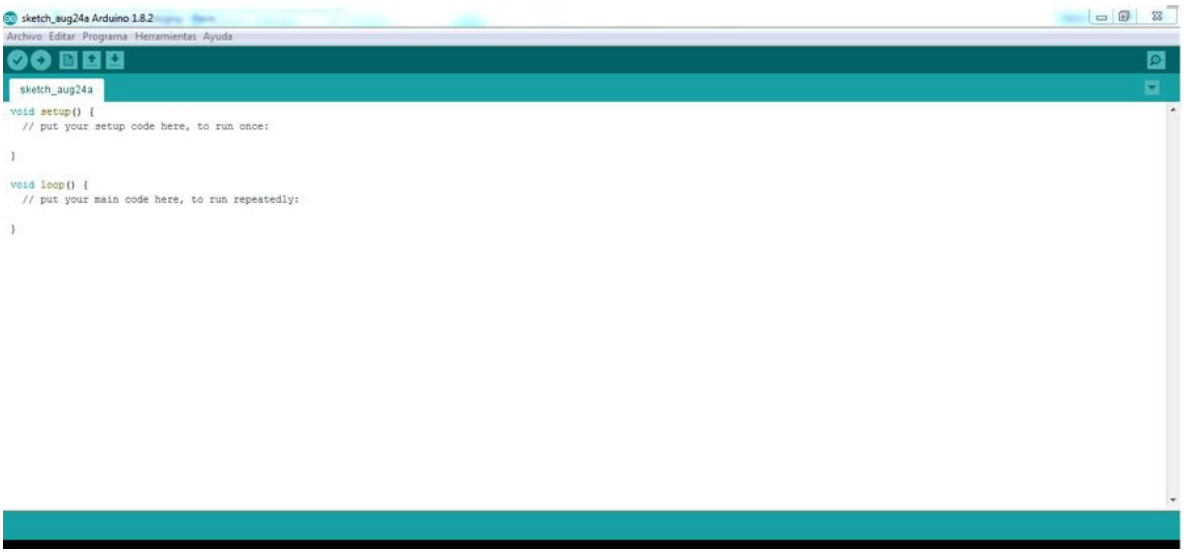


Figura 3.7 Entorno de programación de Arduino

En el menú horizontal superior podemos ver las pestañas: Archivo, editar, programas, herramientas y ayuda.

Una vez conectada nuestra placa Arduino Uno al ordenador a través del puerto USB deberemos seleccionarla desde la pestaña herramientas, dentro de

herramientas seleccionaremos "Placa" y dentro de placa seleccionamos "Arduino/Genuino Uno".

Por último tendremos que seleccionar el puerto que el ordenador te asigna para comunicarte con el entorno de programación, esto lo haremos desde la pestaña "herramientas", dentro de "herramientas" seleccionamos "puerto", y dentro de "puerto" tendremos que ver el puerto asignado para la comunicación vía USB con la placa Arduino.

3.2.3 eFLL (LIBRERÍA PARA TRABAJAR CON LÓGICA DIFUSA EN ARDUINO)

eFLL es una librería para trabajar con lógica difusa en Arduino. Fue Desarrollado por robótico Grupo de Investigación (RRG) en la Universidad del Estado de Piauí (UESPI - Teresina).

No tiene limitaciones explícitas en la cantidad de Fuzzy, Fuzzy Rules, Inputs u Outputs, la potencia de procesamiento y almacenamiento están limitados a cada microcontrolador. La biblioteca utiliza el proceso: (max- min y (minimum mamdani) para la inferencia y composición y (center of area) a la defuzzificación en un universo continuo.

OBJETOS DE LA LIBRERÍA:

- **Fuzzy objeto** - Este objeto incluye toda la Fuzzy sistema, a través de ella, se puede manipular los conjuntos borrosos, lingüísticas Reglas, entradas y salidas.

- **Objeto Fuzzy Input**: Este objeto agrupa todas las entradas de conjuntos difusos que pertenecen al mismo dominio.

- **Objeto Fuzzy Output**: Este objeto es similar a Fuzzy Input, se usa para agrupar todos las salidas que pertenece al mismo dominio.

- **Objeto FuzzySet**: Este es uno de los objetos principales de Fuzzy Library, con cada conjunto es posible modelar el sistema en cuestión. Actualmente la biblioteca admite funciones de membresía triangulares, trapezoidal e individual, que se ensamblan en base a los puntos A, B, C y D, se pasan por parámetro: ***FuzzySet (float a, float b, float c, float d)*.**

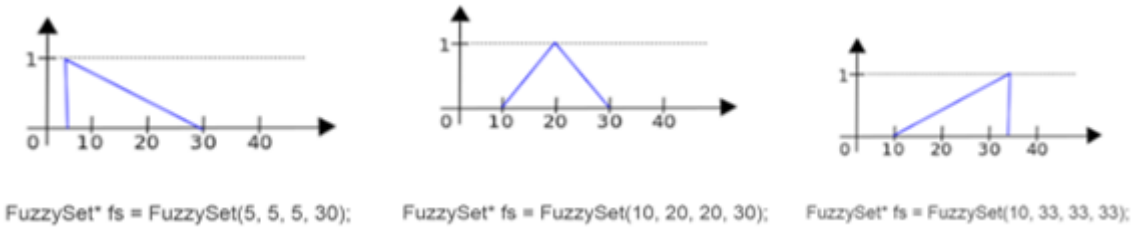


Figura 3.8 Ejemplos de una función de pertenencia triangular con la librería eFLL.

Las figuras deben estar centradas

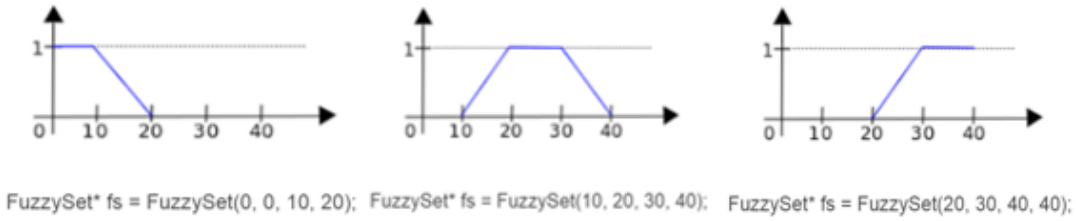


Figura 3.9 Ejemplos de una función de pertenencia trapezoidal con la librería eFLL.

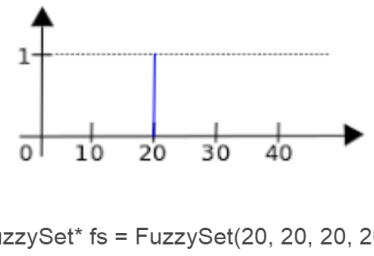


Figura 3.10 Ejemplos de una función singleton con la librería eFLL.

- **Objeto FuzzyRule:** este objeto se usa para montar la regla base del objeto Fuzzy, que contiene uno o más de este objeto. Por ejemplo: FuzzyRule fr = new FuzzyRule (ID, antecedente, consecuente).
- **Objeto FuzzyRuleAntecedent:** este objeto se utiliza para componer el objeto FuzzyRule, responsable de ensamblar el antecedente de la expresión condicional de una FuzzyRule, ejemplos:

$$if\ Distancia = Chica\ then\ Velocidad = Baja$$

$$FuzzyRuleAntecedent^{*} ifDistanciaChica = newFuzzyRuleAntecedent(\ \ );$$
$$ifDistanciaChica \rightarrow joinSingle(Chica);$$

- **Objeto FuzzyRuleConsequente**: Este objeto se utiliza para representar el objeto FuzzyRule, responsable de ensamblar la expresión de salida de un FuzzyRule, ejemplos:

$$if\ Distancia = Chica\ then\ Velocidad = Baja$$
$$FuzzyRuleConsequent^{*} thenVelocidadBaja = newFuzzyRuleConsequent();$$
$$thenVelocidadBaja \rightarrow addOutput(Baja);$$

Lo que resultaría en un objeto FuzzyRule como:

$$FuzzyRule^{*} fuzzyRule = newFuzzyRule(2, ifDistanciaChica, thenVelocidadBaja);$$

Estos son todos los objetos de la biblioteca eFLL que se utilizan en el proceso. El siguiente paso, generalmente interactivo, se maneja con tres métodos de la clase Fuzzy primero:

- **bool setInput (int ID, float value):** Se utiliza para pasar el valor de entrada Crisp al sistema, tenga en cuenta que el primer parámetro es el ID del objeto FuzzyInput.
- **bool fuzzify ():** Se usa para iniciar el proceso de fuzzificación, composición e inferencia.
- **float defuzzify (int ID):** Se utiliza para finalizar el proceso de fuzzification, tenga en cuenta que el parámetro ID pertenece al objeto FuzzyOutput del que desea obtener el valor de defuzzification.

A veces es necesario conocer la pertinencia con la que se activaron algunos o cada conjunto difuso. Para hacer esto, use el método **float getPertinence ();** de clase FuzzySet, por ejemplo:

$$FuzzySet^{*} Caliente = newFuzzySet(30{,}50{,}50{,}70);$$

$$\ldots$$
$$\ldots\ Después\ de\ la\ fuzzificacion -> \ fuzzyfy();$$
$$\ldots$$
$$float\ pertinenceOfCaliente\ =\ Caliente -> getPertinence();$$

O bien, si se disparó una regla en particular, use el método **bool isFiredRule (int ruleId).** Por ejemplo:

$$FuzzyRule^{*} fuzzyRule = newFuzzyRule(2, ifDistanciaChica, thenvelocidadBaja);$$
$$\ldots$$
$$\ldots \ Después\ de\ la\ fuzzificacion -> \ fuzzyfy();$$
$$\ldots$$
$$bool\ was\ TheRuleFired = fuzzy \rightarrow isFiredRule(2);$$

CAPITULO IV PROCEDIMIENTO Y DESCRIPCION DE LAS ACTIVIDADES REALIZADAS

En este proyecto se pretende realizar un sistema de control de velocidad basado en lógica difusa para un vehículo recolector de PET, el cual mantendrá la velocidad constante cuando la pendiente donde se desplace varíe.

Ya que estamos partiendo de cero es necesario realizar varias pruebas con el vehículo para obtener la base de conocimiento del sistema que nos permitirá definir los conjuntos difusos.

4.2 CONSTRUCCIÓN DEL VEHÍCULO

Para armar el chasis del vehículo se utilizaron piezas de Vex, las cuales constan de unas barras de aluminio con agujeros uniformemente espaciados:

Figura 4.1 Piezas que se usaron para construir el chasis del vehículo

El chasis del vehículo es un rectángulo con medidas 44.5x32, con 4 motores.

Las llantas utilizadas miden 5 pulgadas de diámetro, este es un dato importante que se utilizó para poder medir la distancia que recorría el vehículo.

Figura 4.2 Llantas usadas en el vehículo.

Se usaron 4 llantas en el vehículo las cuales están colocadas una en cada "esquina del vehículo".

Los motores utilizados son el modelo de 2 cables 269 de Vex. Las especificaciones del proveedor a 7.2 volts son:

- Torque a rotor bloqueado 8.7 in-lb [0.97 N-m]
- Velocidad a rotor libre 100 RPM
- Corriente a rotor bloqueado 2.6 Amps
- Corriente a rotor libre 0.18 Amps

Figura 4.3 Motor usado.

Para poder llevar a cabo el conteo de las vueltas que daban las llantas y así poder calcular la distancia se utilizaron encoders ópticos Vex. Estos trabajan incidiendo la luz en el borde de un disco equipado con las ranuras uniformemente espaciadas

alrededor de la circunferencia. A medida que el disco gira, la luz pasa a través de las ranuras y es bloqueada por los espacios opacos entre las ranuras. El codificador detecta entonces cuántas hendiduras han tenido luz.

Figura 4.4 Encoders utilizados en el vehículo.

Para poder medir el tiempo que le tomaba al vehículo recorrer una distancia determinada se utilizó un arreglo que consta de un Arduino, una pantalla LCD de 16x2 líneas y dos sensores, uno al inicio y el otro al final del recorrido, en la figura 4.7 se puede apreciar este arreglo.

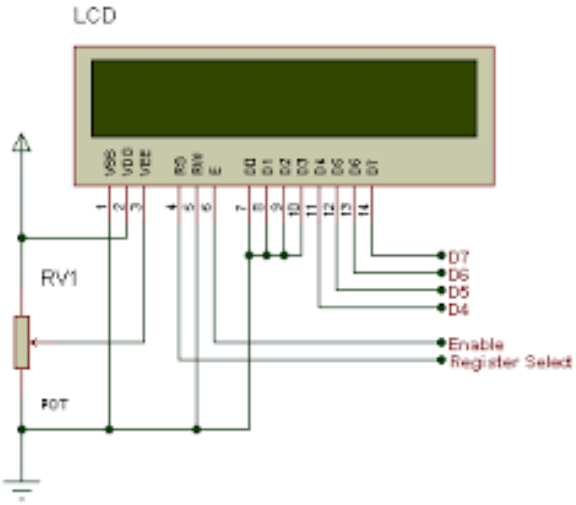


Figura 4.5 LCD de 16x2 líneas y diagrama de conexiones.

Después de tener el chasis armado lo siguiente fue montar los motores y llantas, los motores fueron sujetados al chasis con tornillos y las llantas fueron sujetadas a los motores, los encoders se colocaron de una manera tal que quedaran entre las llantas y los motores pero sujetos al chasis del vehículo, finalmente encima del

chasis se colocó una pieza rectangular de acrílico para que sirviera de base para colocar los Arduinos y la fuente que suministra la corriente a todo el sistema. En la figura 4.6 se puede apreciar el diagrama de conexiones realizados entre los distintos Arduinos, puentes H, sensores y el display LCD y las figuras 4.8 y 4.9 son vistas lateral y frontal respectivamente del vehículo armado.

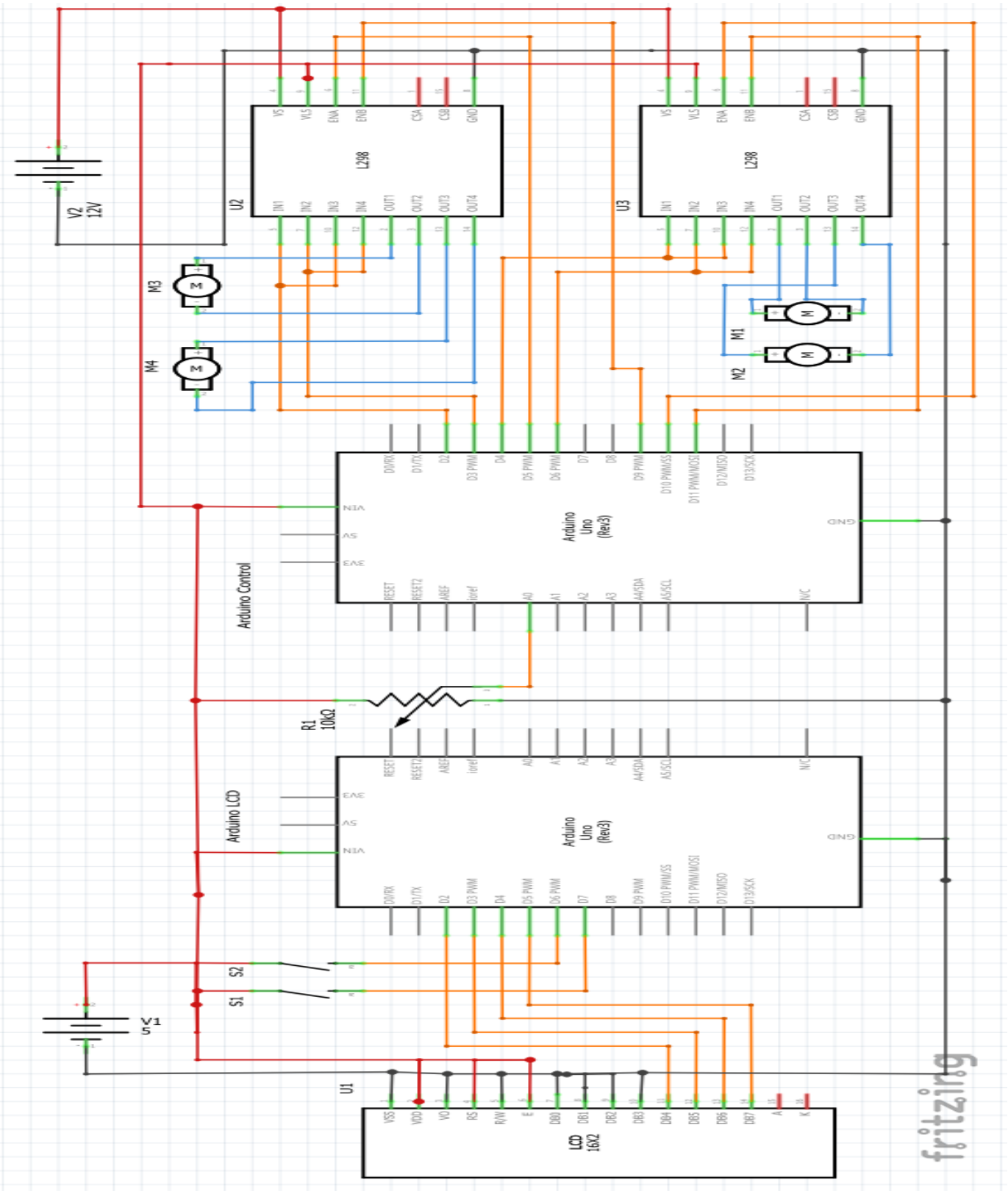


Figura 4.6 Diagrama esquemático de conexiones.

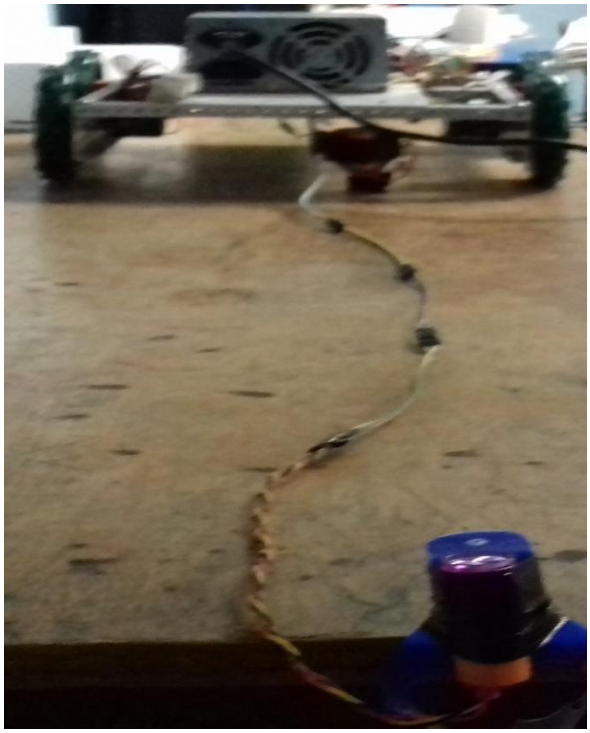


Figura 4.7 Arreglo de sensores ópticos

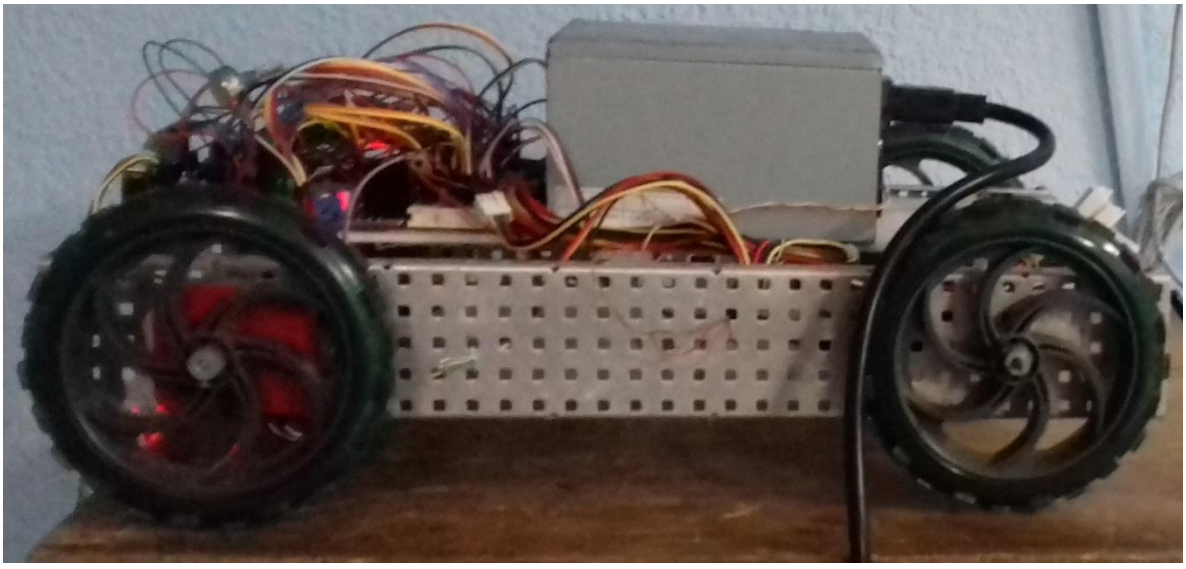


Figura 4.8 Imagen lateral del vehículo armado

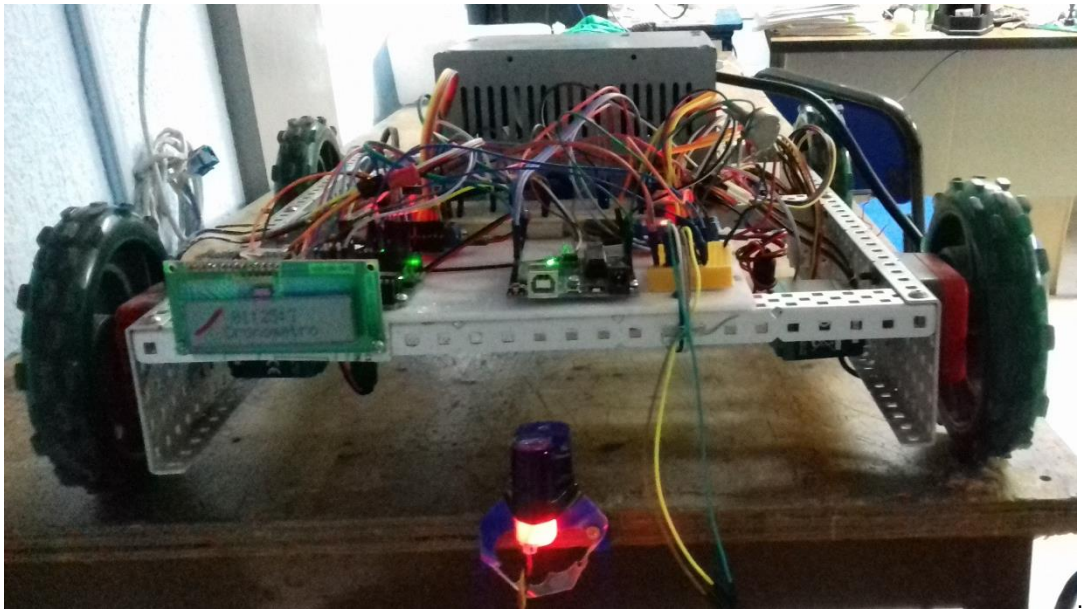


Figura 4.9 Imagen frontal del vehículo armado.

Para alimentar de corriente a los motores así como para poder variar la potencia hacia estos, se utilizaron dos driver puente H L298N. Este módulo posee dos puentes H que permiten controlar 2 motores DC.

El módulo permite controlar el sentido de giro y velocidad mediante señales TTL que se pueden obtener de microcontroladores y tarjetas de desarrollo como Arduino, Raspberry Pi o Launchpads de Texas Instruments.

Tiene integrado un regulador de voltaje de 5V encargado de alimentar la parte lógica del L298N, el uso de este regulador se hace a través de un Jumper y se puede usar para alimentar la etapa de control.

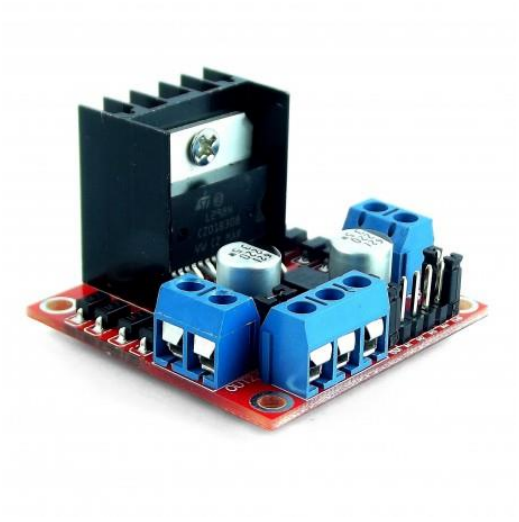
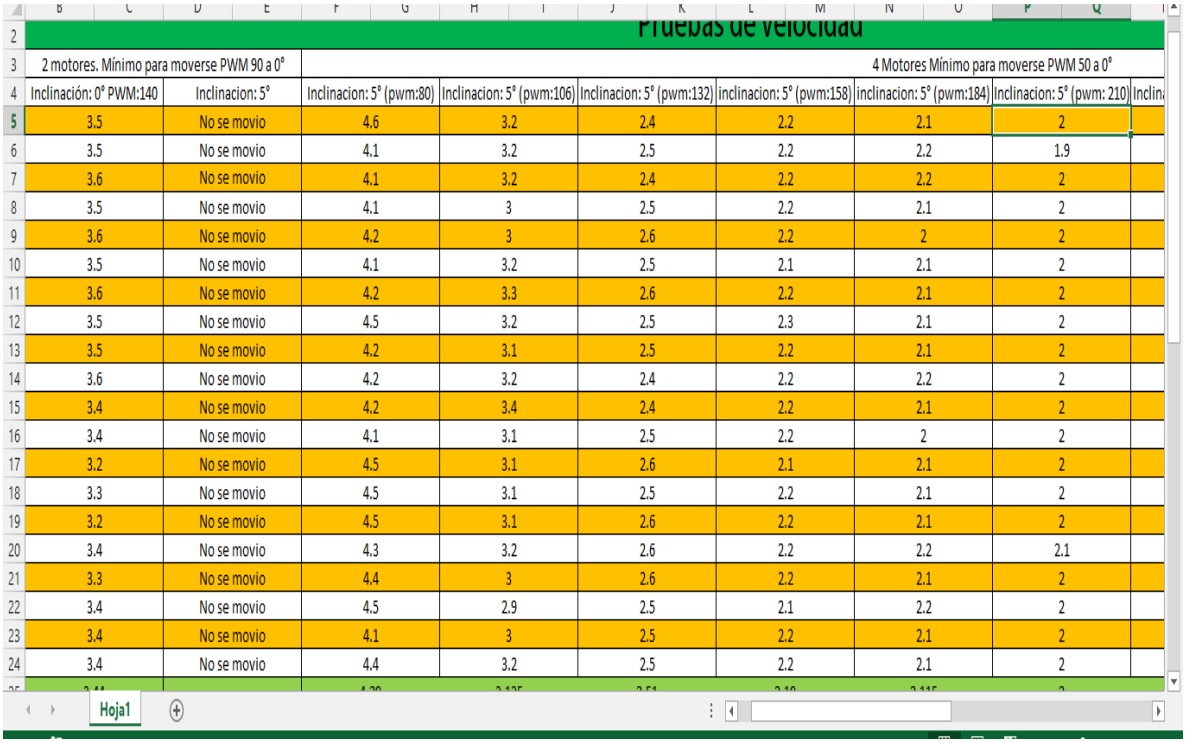


Figura 4.10 Driver puente H L298N.

La corriente para alimentar todos los circuitos se tomó de una fuente de PC, la cual entrega diferentes voltajes, entre ellos 5V que son los necesarios para alimentar al Arduino, los sensores y la pantalla LCD.

Figura 4.11 Fuente de poder.

Para poder realizar un control difuso se debe tener una base de datos acerca del comportamiento del sistema el cual se obtiene en base a la experiencia de una persona que ha trabajado con este. En nuestro caso la base de datos se obtuvo a través de pruebas. Para ello fue necesario poner en marcha el vehículo con lo cual se determinó la mínima potencia necesaria para que el carrito se moviera sobre una pendiente de 0 grados, también se determinó la pendiente máxima a la que el vehículo podría mantener la velocidad.



2	Pruebas de velocidad								
3	2 motores. Mínimo para moverse PWM 90 a 0°		4 Motores Mínimo para moverse PWM 50 a 0°						
4	Inclinación: 0° PWM:140	Inclinacion: 5°	Inclinacion: 5° (pwm:80)	Inclinacion: 5° (pwm:106)	Inclinacion: 5° (pwm:132)	inclinacion: 5° (pwm:158)	inclinacion: 5° (pwm:184)	Inclinacion: 5° (pwm: 210)	Inclina
5	3.5	No se movio	4.6	3.2	2.4	2.2	2.1	2	
6	3.5	No se movio	4.1	3.2	2.5	2.2	2.2	1.9	
7	3.6	No se movio	4.1	3.2	2.4	2.2	2.2	2	
8	3.5	No se movio	4.1	3	2.5	2.2	2.1	2	
9	3.6	No se movio	4.2	3	2.6	2.2	2	2	
10	3.5	No se movio	4.1	3.2	2.5	2.1	2.1	2	
11	3.6	No se movio	4.2	3.3	2.6	2.2	2.1	2	
12	3.5	No se movio	4.5	3.2	2.5	2.3	2.1	2	
13	3.5	No se movio	4.2	3.1	2.5	2.2	2.1	2	
14	3.6	No se movio	4.2	3.2	2.4	2.2	2.2	2	
15	3.4	No se movio	4.2	3.4	2.4	2.2	2.1	2	
16	3.4	No se movio	4.1	3.1	2.5	2.2	2	2	
17	3.2	No se movio	4.5	3.1	2.6	2.1	2.1	2	
18	3.3	No se movio	4.5	3.1	2.5	2.2	2.1	2	
19	3.2	No se movio	4.5	3.1	2.6	2.2	2.1	2	
20	3.4	No se movio	4.3	3.2	2.6	2.2	2.2	2.1	
21	3.3	No se movio	4.4	3	2.6	2.2	2.1	2	
22	3.4	No se movio	4.5	2.9	2.5	2.1	2.2	2	
23	3.4	No se movio	4.1	3	2.5	2.2	2.1	2	
24	3.4	No se movio	4.4	3.2	2.5	2.2	2.1	2	

Hoja1

Figura 4.12 Fragmento de datos obtenidos con las primeras pruebas del vehículo.

En esta fase de experimentación se determinó la mínima potencia necesaria para que el vehículo pudiera moverse con una pendiente de 0 grados. En la imagen se puede observar que para que el vehículo pudiera moverse utilizando solo 2 motores se necesitaba mínimo un PWM = 90 y utilizando los 4 motores se necesitaba al menos un PWM = 50.

Al realizar las pruebas se descubrió que para valores de PWM mayores a 210 los motores ya no podían ir más rápido, esto sirvió para determinar la pendiente máxima a la que el vehículo podría mantener la velocidad, realizando corridas con este valor de PWM se determinó que la pendiente máxima a la que se podría garantizar una respuesta eficiente del control era a los 10°.

Con los datos anteriores fue posible empezar a construir el control difuso. Para ello se realizaron primero simulaciones en Matlab utilizando únicamente 3 conjuntos difusos de entrada y 3 de salida.

Se realizaron muchas pruebas para lograr poder definir los conjuntos de entrada y salida que dieran la optima respuesta que se esperaba del vehículo. De entre todas destacaron unicamente dos versiones las cuales tenian los mismos conjuntos de entrada pero diferentes conjuntos de salida.

En la figura 4.13 se muestran las funciones de membresia de entrada para ambas opciones del control.

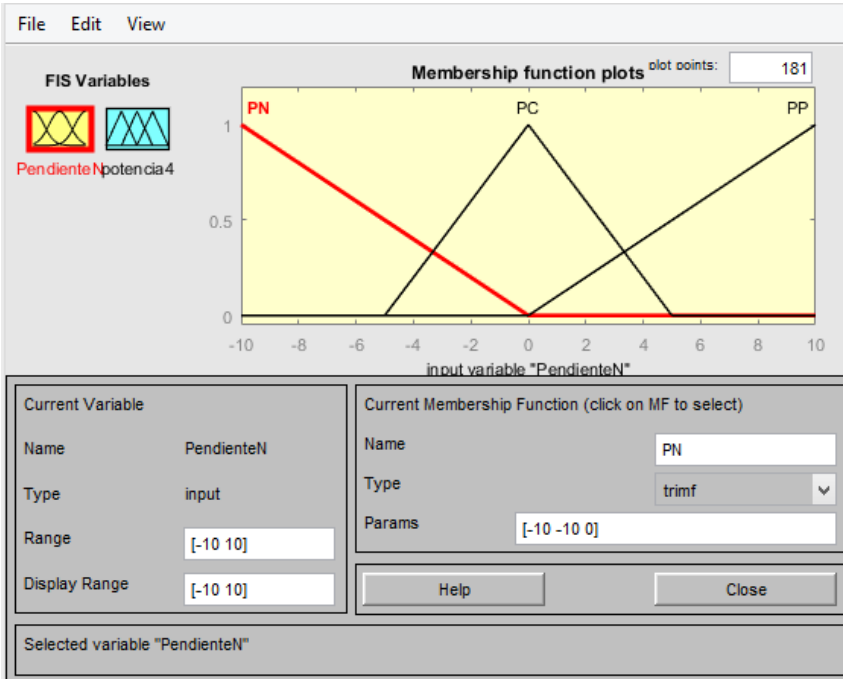


Figura 4.13 Funciones de membresía de entrada para control difuso.

En la figura 4.14 se muestran las funciones de membresia de salida para la primera opción y en la figura 4.16 las funciones de membresia de salida para la segunda opción. En las figuras 4.15 y 4.17 se muestran las graficas que describen el comportamiento de los valores de salida, es decir, los valores defuzificados o reales, para la primera y segunda opción respectivamente.

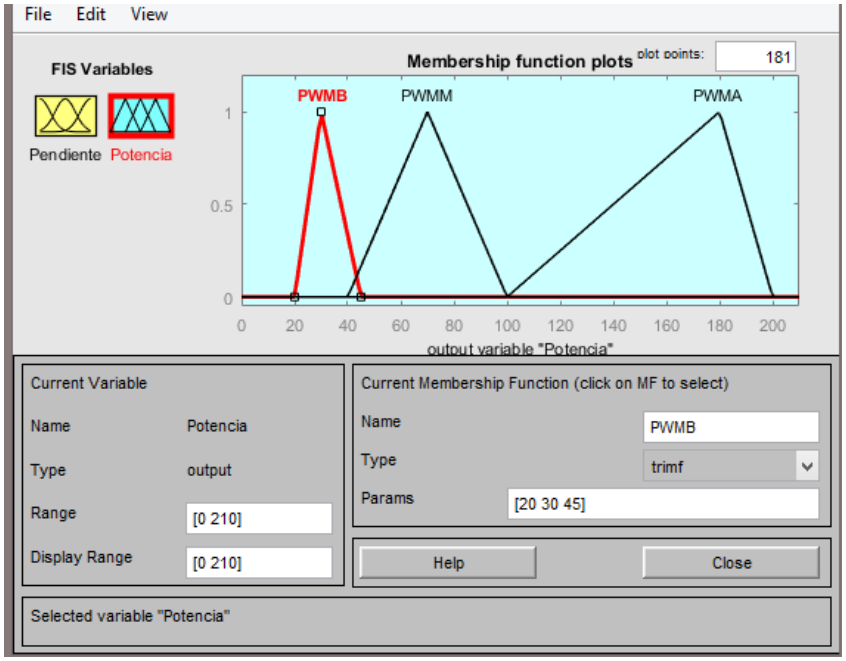


Figura 4.14 Funciones de membresía de salida para el control difuso.

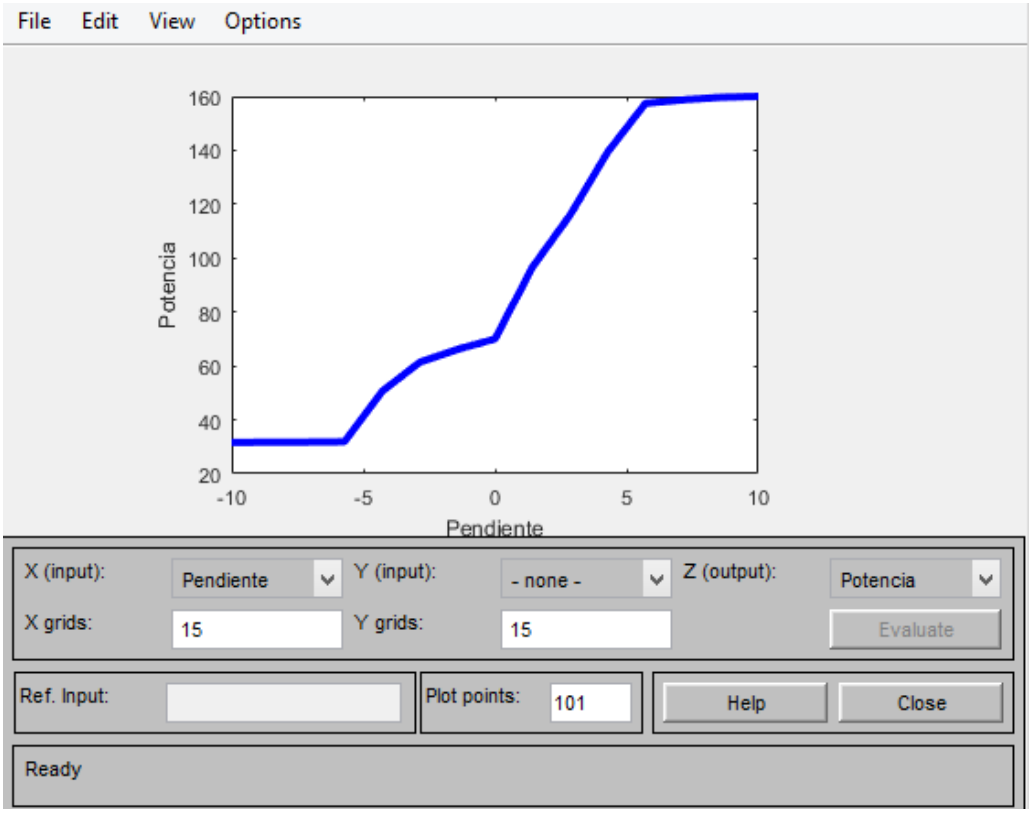


Figura 4.15 Gráfica de salida para los conjuntos difusos de la figura 4.14

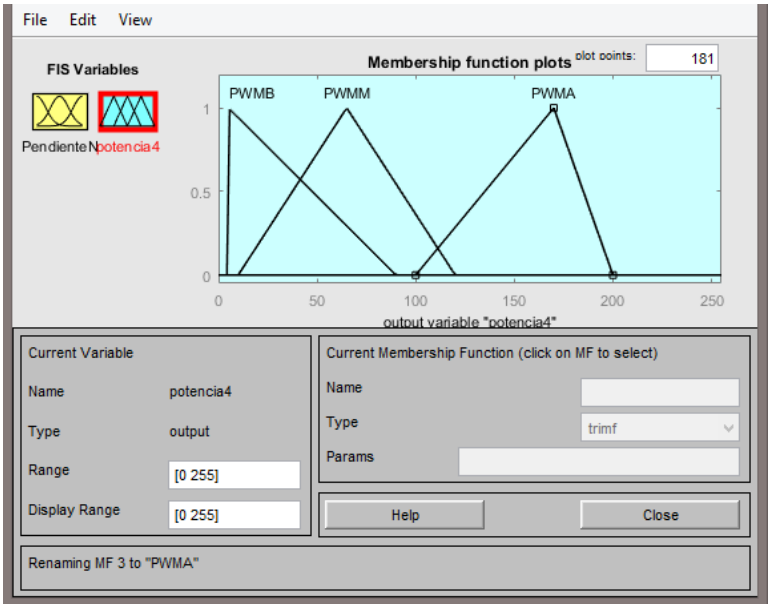


Figura 4.16 Funciones de membresía para el control difuso.

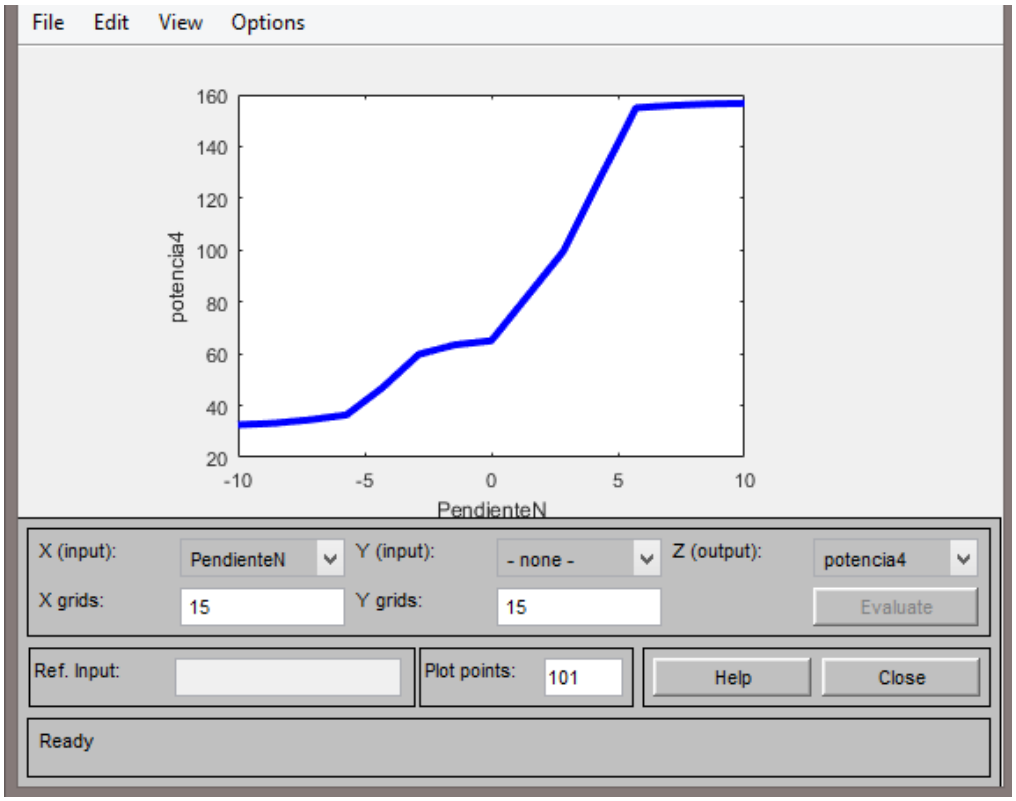


Figura 4.17 Gráfica de salida para los conjuntos difusos de la figura 4.16

Sin embargo analizando las gráficas de las figuras 4.15 y 4.17 los resultados obtenidos con únicamente tres conjuntos no son suficiente para tener un buen rendimiento del vehículo. En primer lugar porque para valores de pendientes cercanos a 5° la salida se dispara de una forma nada conveniente y para valores mayores a 5° la salida se comporta de una forma casi constante, es decir que para valores cercanos a los 5° el vehículo iría más rápido que la velocidad establecida y para valores de pendiente mayores a 5° los motores no recibirían la suficiente potencia y el vehículo no podría mantener la velocidad establecida sino que iría más lento y eventualmente se detendría. Por lo cual se decidió realizar el control difuso con cinco conjuntos de entrada y cinco de salida.

Al igual que se hizo anteriormente, se realizaron varias simulaciones probando distintos valores para las funciones de membresia tanto de entrada como de salida al final se obtuvieron dos opciones que fueron las que mejores resultados mostraron.

En la figura 4.18 y 4.19 se muestran las funciones de membresia de entrada para la primera y segunda opcion del control difuso respectivamente. En ellas se pueden apreciar las diferencias entre los conjuntos de cada opción.

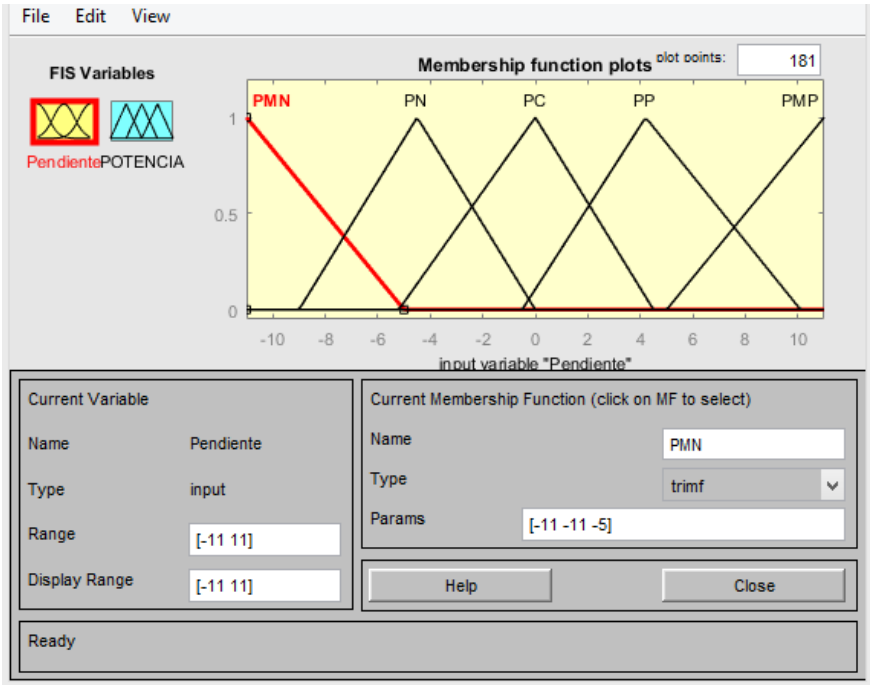


Figura 4.18 Funciones de membresía de entrada para la primera opción del control difuso.

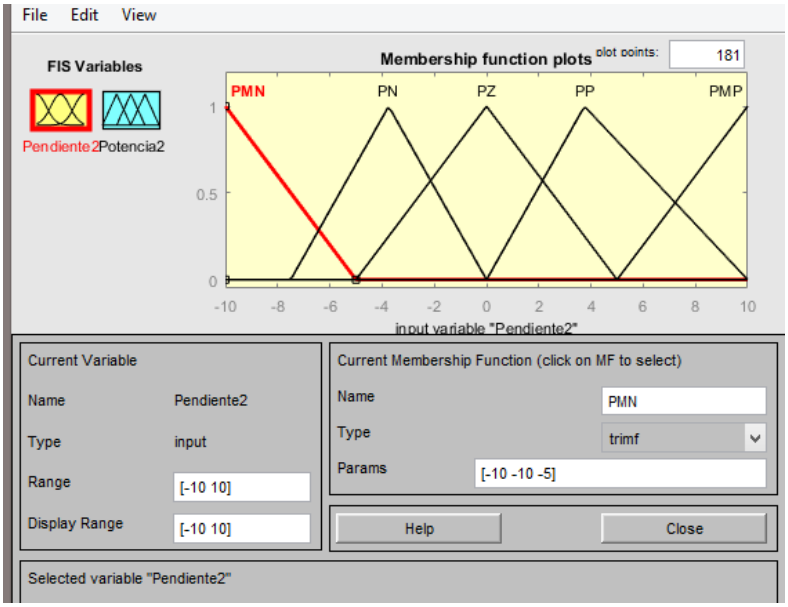


Figura 4.19 Funciones de membresía de entrada para la segunda opción del control difuso.

En la figura 4.20 y 4.21 se muestran las funciones de membresia de salida para la primera y segunda opcion del control difuso respectivamente. En ellas se pueden apreciar las diferencias entre los conjuntos de cada opción.

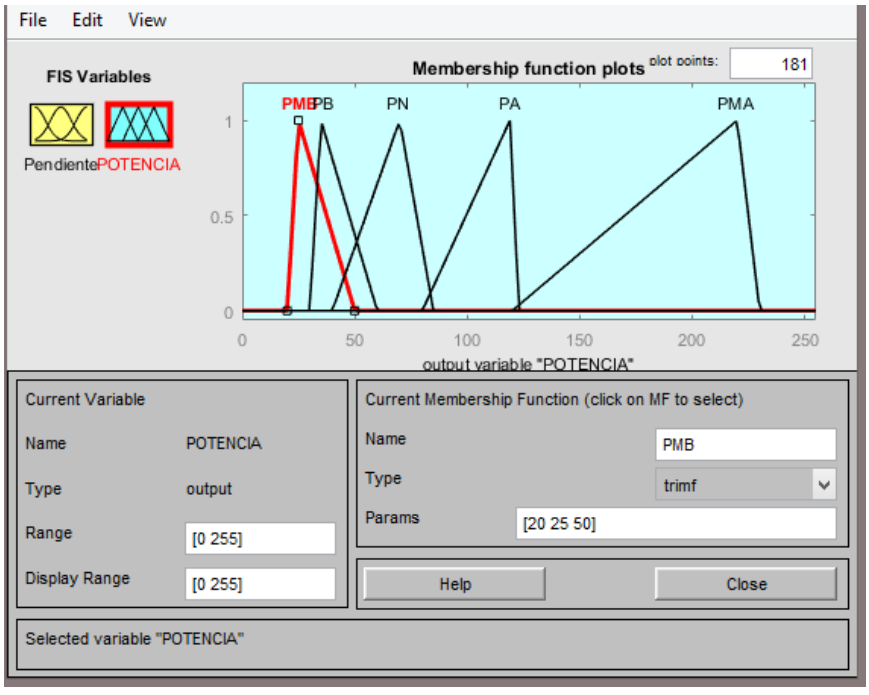


Figura 4.20 Funciones de membresía de salida para la primera opción del control difuso.

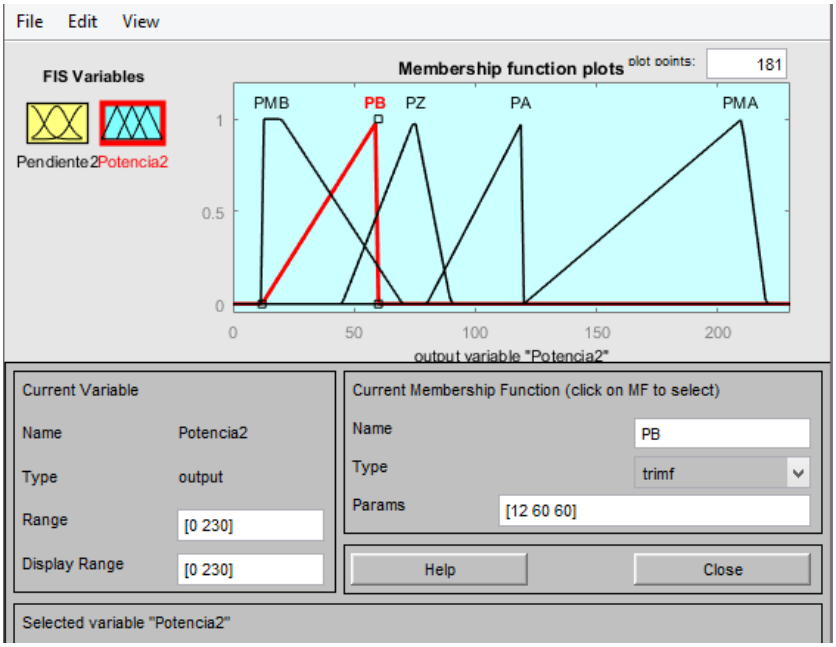


Figura 4.21 Funciones de membresía de salida para la segunda opción del control difuso.

En las figuras 4.22 y 4.23 se muestran las graficas que describen el comportamiento de los valores de salida, es decir, los valores defuzificados o reales, para la primera y segunda opción respectivamente.

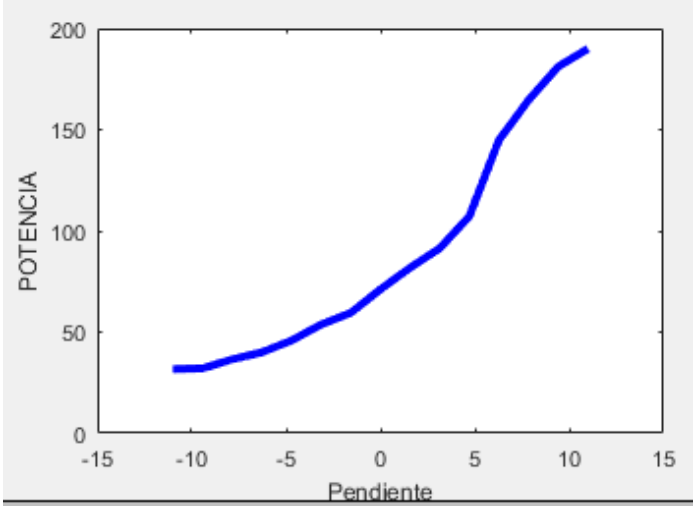


Figura 4.22 Grafica de salida para la primera opción el control difuso.

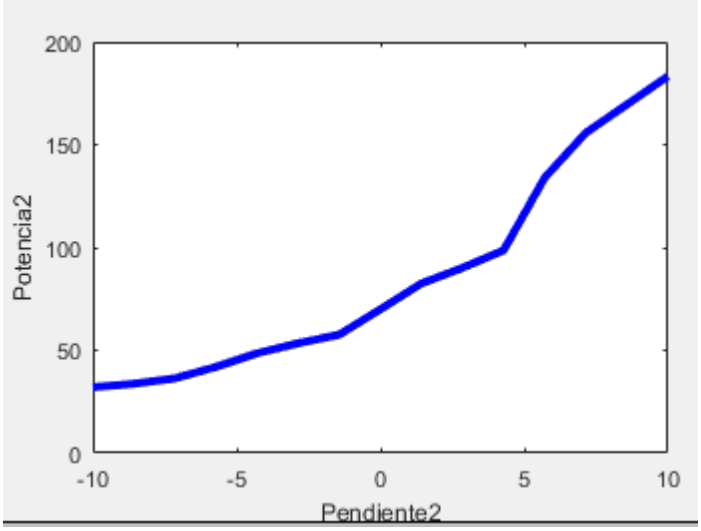


Figura 4.23 Grafica de salida para la segunda opción el control difuso.

Al analizar las gráficas de las figuras 4.22 y 4.23 notamos que para pendientes menores que 0 los valores en la salida del control van aumentando de una forma no tan acelerada y esto es bueno ya que para pendientes negativas no necesitamos de mucha potencia ya que además de la fuerza de los motores contamos con la gravedad a favor. Por otro lado cuando los valores de pendientes son mayores que 0, los valores en la salida del control aumentan más rápidamente, estos resultados favorecen al control ya que para pendientes positivas contamos con la gravedad en nuestra contra lo cual exige mayor potencia a los motores.

Una vez realizadas las simulaciones en Matlab se procedió a pasar el programa a Arduino utilizando la librería para Lógica difusa descrita en el capítulo anterior.

CAPITULO V RESULTADOS OBTENIDOS

5.1 PROGRAMA EN ARDUINO

El diagrama de flujo del programa se muestra en la figura 5.1.

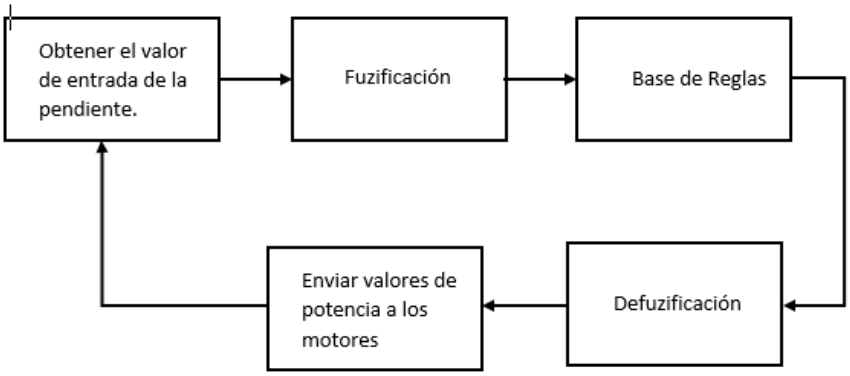


Figura 5.1 Diagrama de flujo del programa en Arduino.

Como se puede apreciar en el diagrama de flujo el programa es un ciclo que se ejecuta siempre a menos que sea interrumpido desde fuera. En primer lugar se debe obtener el valor de la pendiente el cual se emula con un potenciómetro y programación. Con el potenciómetro conectado a un puerto analógico del Arduino, se realiza un mapeo de la lectura de esta entrada. Los valores posibles están comprendidos entre -10° y 10°.

Los pasos siguientes, fuzificación, la evaluación de la base de reglas y la defuzificación se realizan mediante el uso de la librería eFLL que fue descrita en el capítulo III.

Finalmente los valores defuzificados, son enviados en forma de ancho de pulsos al driver puente H que suministra la potencia a los motores de acuerdo al valor de PWM obtenido.

5.2 RESULTADOS

Pendiente	Arduino	Matlab
13°	184.67	192
12°	184.67	192
11°	184.67	192
10°	184.67	192
9°	176.9	180
8°	167.97	170
7°	156.98	158
6°	140.45	142
5°	107.14	107
4°	96.29	99.4
3°	89.19	90.4
2°	83.96	84.1
1°	78.68	78.5
0°	71.46	71.1
-1°	61.01	61.6
-2°	57.81	57.9
-3°	54.29	54.2
-4°	50	50.1
-5°	43.89	43.7
-6°	40.34	40.4
-7°	38.82	38.9
-8°	36.63	36
-9°	32.22	32.2
-10°	31.83	31.9
-11°	--------	-------

Tabla 5.1 Tabla de los valores obtenidos en Matlab y Arduino para la primera opción.

Pendiente	Arduino	Matlab
13°	183.33	183
12°	183.33	183
11°	183.33	183
10°	183.33	183
9°	173.45	174
8°	164.26	164
7°	154.11	154
6°	139.84	140
5°	106.22	107
4°	93.57	96.5
3°	89.38	90.9
2°	85.26	85.8
1°	79.59	79.7
0°	70	70
-1°	59.51	59.9
-2°	55.64	55.6
-3°	52.9	53.1
-4°	48.71	50
-5°	42.67	43.2
-6°	38.82	40.8
-7°	36.64	36.7
-8°	34.37	34.7
-9°	32.75	33.1
-10°	31.66	32.1
-11°	---------	-------

Tabla 5.2 Tabla de los valores obtenidos en Matlab y Arduino para la segunda opción.

En las tablas 5.1 y 5.2 se muestran los valores obtenidos a la salida del control difuso, tanto en la simulación en Matlab como los del control implementado en Arduino.

En las figuras 5.2 y 5.3 se muestran los datos de las tablas anteriores en forma de gráficas, en ellas se confrontan los valores de PWM obtenidos a la salida del control difuso contra los valores de la pendiente de entrada. Se puede apreciar

que las diferencias entre los valores obtenidos en Matlab y Arduino son muy pequeñas, las gráficas son prácticamente las mismas.

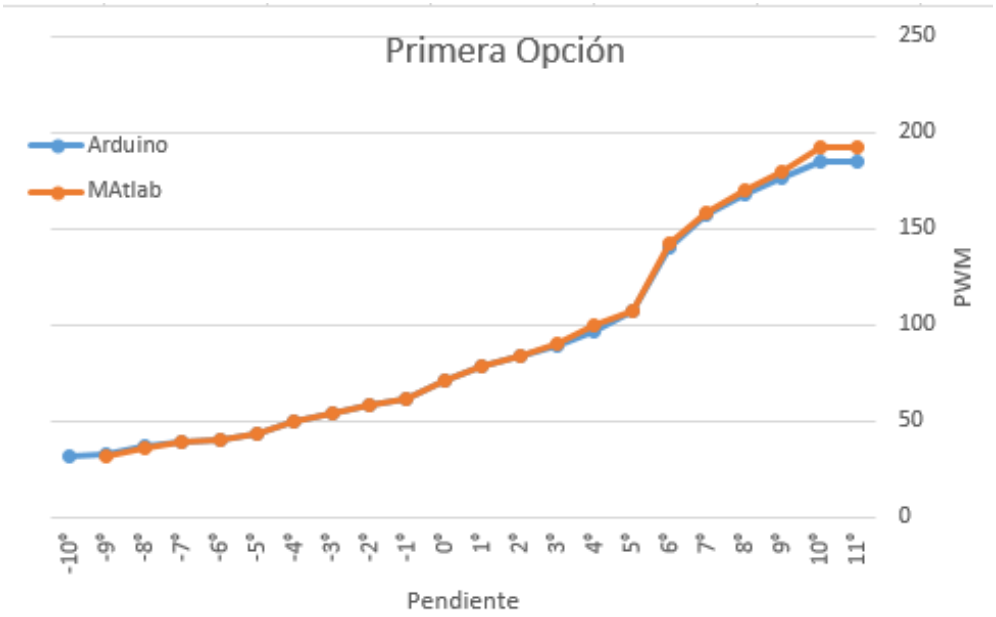


Figura 5.2 Comparación de los valores de salidas obtenidas en Matlab y Arduino.

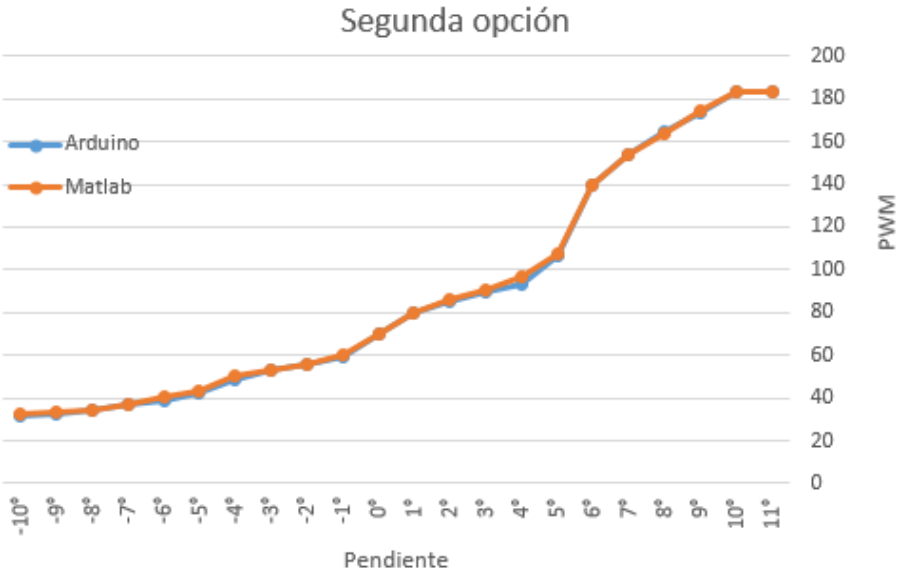


Figura 5.4 Comparación de los valores de salidas obtenidas en Matlab y Arduino.

Por último al realizar las pruebas finales del control de velocidad, se realizaron varias corridas del vehículo, con lo cual se obtuvo un promedio del tiempo que le tomaba al vehículo recorrer la distancia de un metro por cada valor de pendiente. Los datos obtenidos se graficaron, las graficas se divieron en los resultados

obtenidos para pendientes positivas y negativas y al final se hace una comparación de las dos opciones.

En la figura 5.5 y 5.6 se muestran las gráficas de los tiempos obtenidos para los conjuntos positivos de la primera y segunda opcion del controlador respectivamente. En ellas se puede apreciar que la velocidad es practicamente constante unicamente varia por decimas de segundo. Cabe mencionar que se realizaron pruebas con pendientes positivas de hasta 13° en las cuales la primera opción apenas logro terminar el recorrido con un tiempo promedio de 5 segundos y la segunda opción no dio la potencia suficiente por lo que el recorrido no se completo, los valores para la pendiente de 13° que aparecen en la gráfica de la segunda opción del control no son reales y unicamente se incluyeron para poder obtener la gráfica.

En la figura 5.7 se muestra la comparación entre las gráficas de la primera y segunda opción. En ella es posible apreciar que la opción que produce mejores resultados en este caso es la primera.

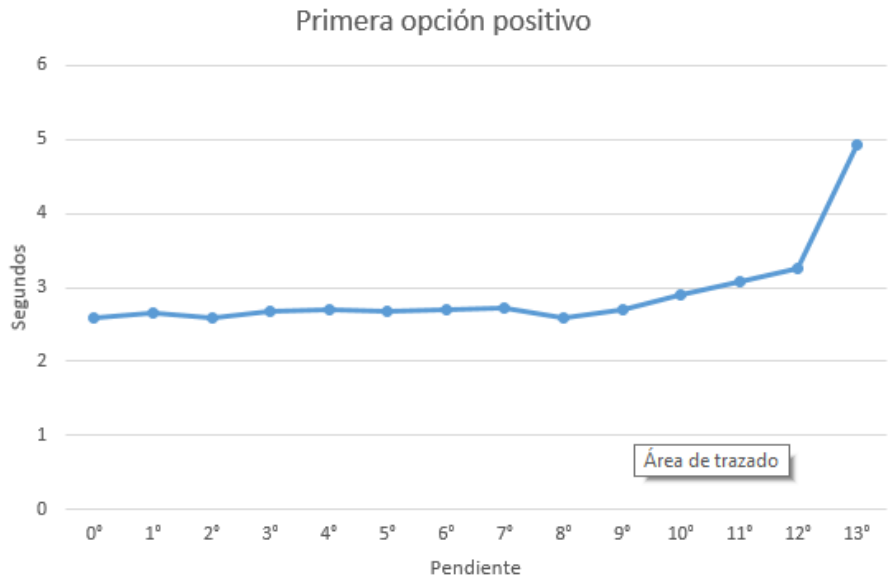


Figura 5.5 Gráfica del tiempo contra la pendiente para pendientes positivas.

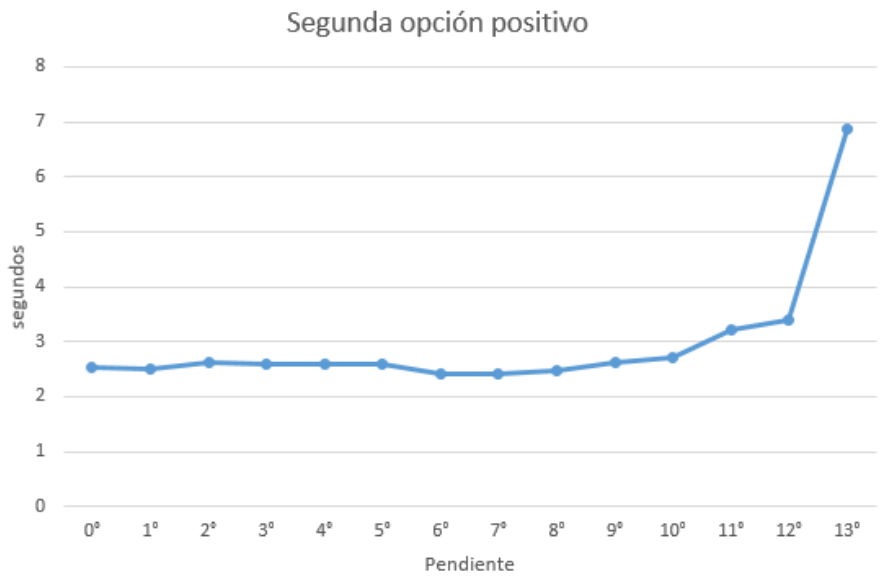


Figura 5.6 Gráfica del tiempo contra la pendiente para pendientes positivas.

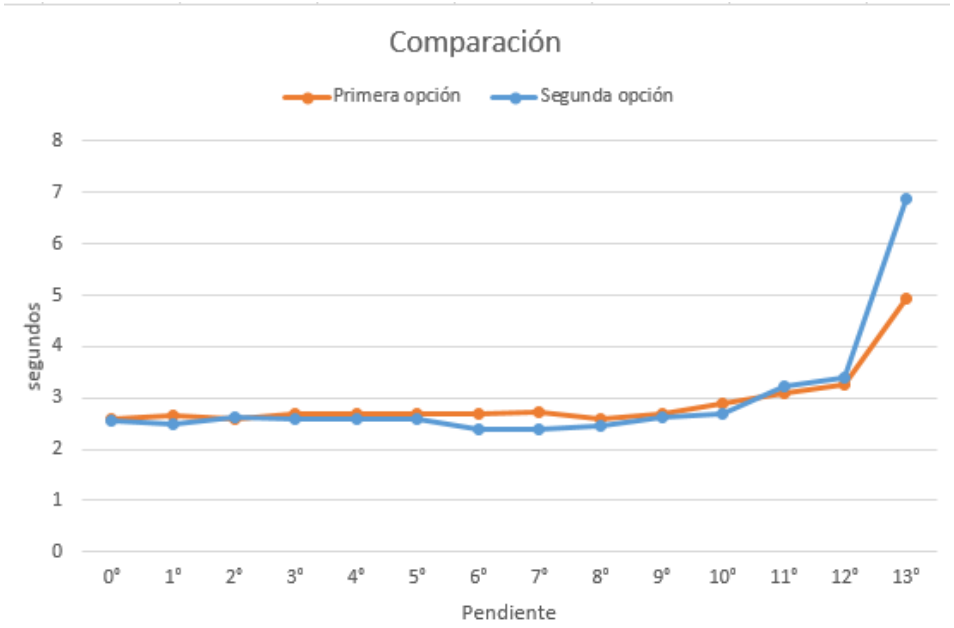


Figura 5.7 Comparación de las gráficas de la primera y segunda opción para conjuntos positivos.

En las figuras 5.8 y 5.9 se muestran las graficas de los tiempos obtenidos para los conjuntos negativos de la primera y segunda opción respectivamente. En ellas se aprecia que la velocidad se mantiene practicamente constante y varia unicamente por decimas de segundo. En este caso los valores de pendiente de -11° no son reales y se utilizaron unicamente para poder apreciar mejor las gráficas.

Por último en la figura 5.10 se muestra la comparación de las graficas para pendientes negativas. En la imagen se puede apreciar que, por diferencias que podrian descartarse, la opción que produce mejores resultados en el caso de pendientes negativas es la segunda

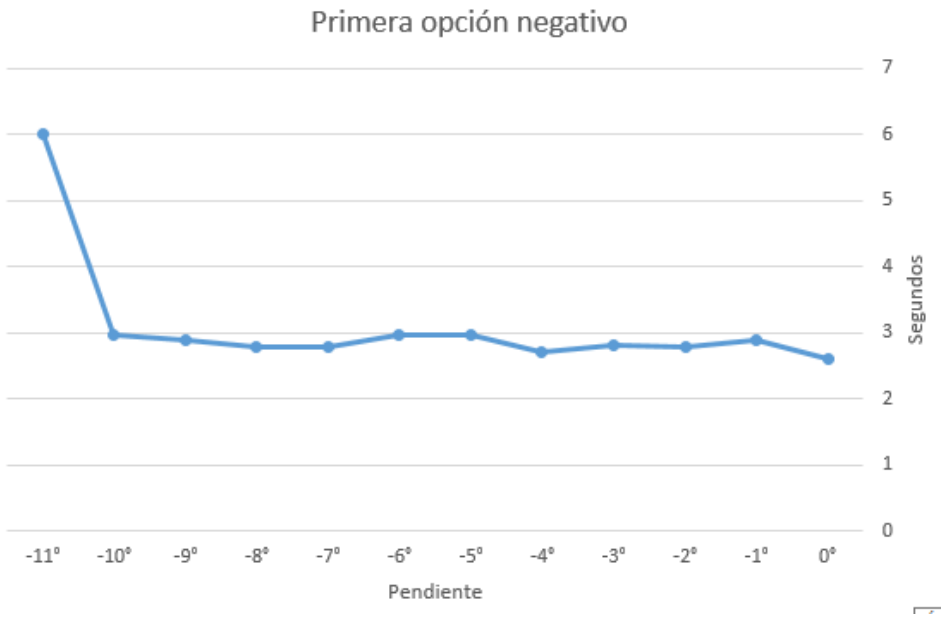


Figura 5.8 Gráfica del tiempo contra la pendiente, para pendientes negativas.

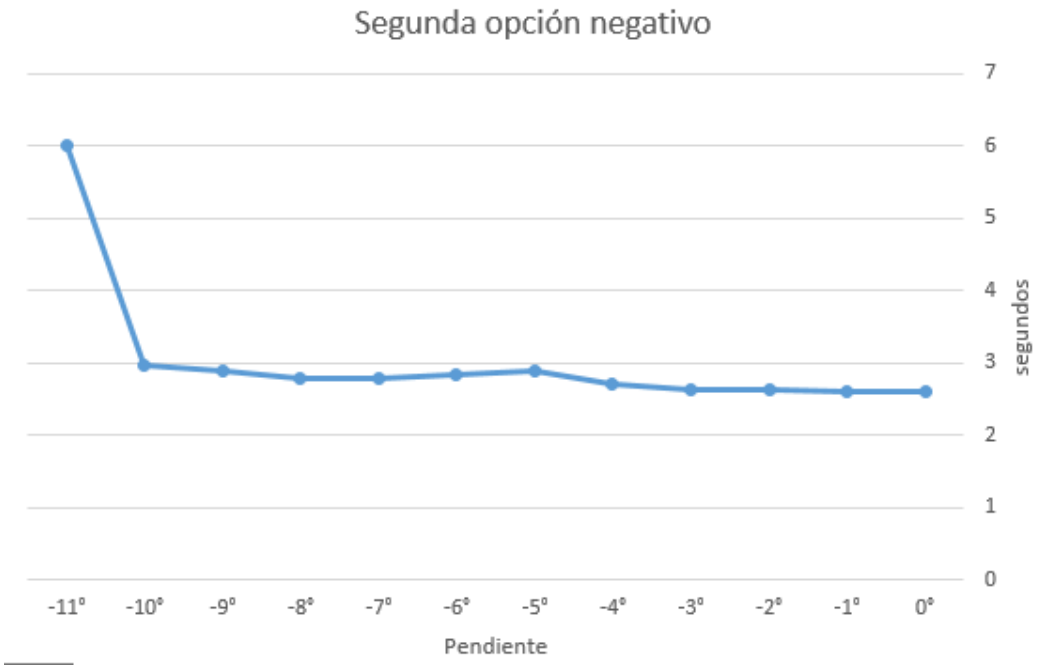


Figura 5.9 Gráfica del tiempo contra la pendiente para pendientes negativas.

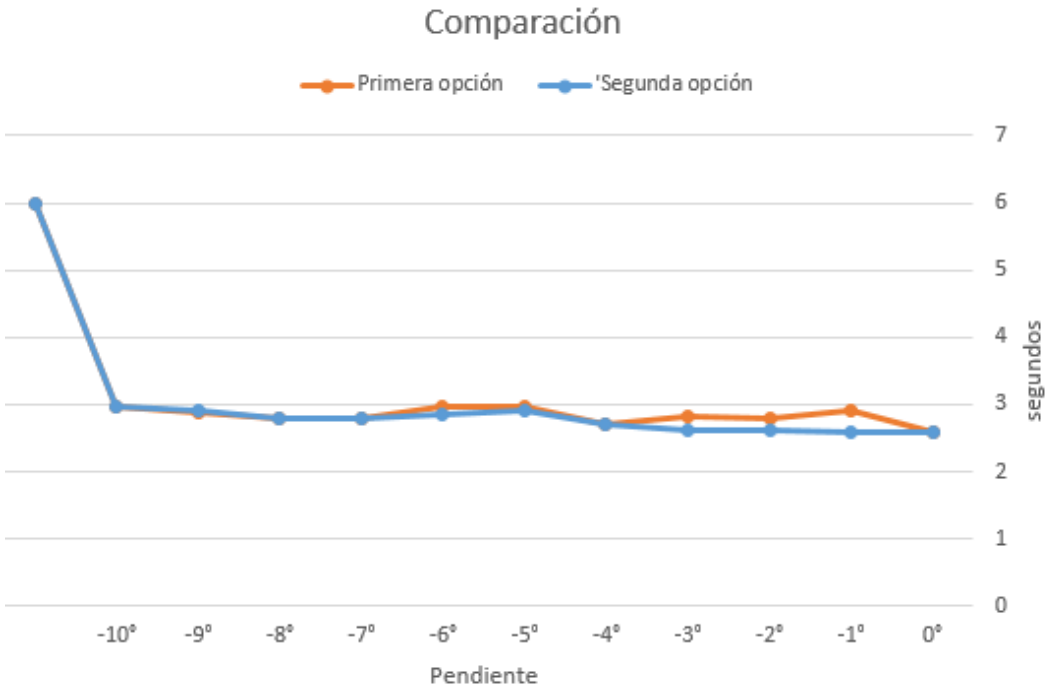


Figura 5.10 Comparación de las gráficas de la primera y segunda opción para pendientes negativas.

CONCLUSIONES

Las ventajas de los controladores basados en lógica difusa sobre los controladores convencionales son muchas cuando no se conoce el modelo matemático del sistema. Las complicaciones que se tuvieron en este proyecto se debieron a la completa inexperiencia en este campo.

Mas sin embargo los resultados obtenidos en este proyecto son sin duda satisfactorios ya que se logró el objetivo principal, el controlador implementado cumple con el objetivo de mantener la velocidad constante dentro de los límites establecidos.

Sin duda el sistema implementado se puede mejorar en trabajos futuros para hacerlo mucho más eficiente, por ejemplo puede sumársele al sistema un control de trayectorias para darle al vehículo una mayor autonomía

Referencias

[1] Valenzuela Hernández, J. G., Montoya Giraldo, O. D., & Giraldo Buitrago, D. (2013). Lógica Difusa Aplicada al Control Local del Péndulo Invertido con Rueda de Reacción. *Scientia Et Technica, vol. 18, núm. 4*, 623-632.

[2] *Oscar G.& Duarte V. (Abril 1999). Sistemas de lógica difusa. Fundamentos. Revista Ingeniería e Investigación, No. 42, 22-30.*

[3] William Gutiérrez Marroquín & . (Jesús Alfonso López Sotelo). Control difuso para un sistema de nivel implementado en un autómata programable. 22/01/2018, de Servicio Nacional de Aprendizaje Sitio web: http://repositorio.sena.edu.co/handle/11404/3245?mode=simple

[4] Guzmán, D., & Castaño, V. M. (2009). La lógica difusa en Ingeniería: principios, aplicaciones y futuro. Revista de Ciencia y Tecnología Vol. 24 Núm. 2 2009.

[5] Giraldo, D., & Tabares, I. (1999). Control de velocidad de un motor dc utilizando lógica difusa. Revista Scientia Et Technica, (9), 111-119.

[6] Beleño, B. A. G., & Robles, J. J. O. Diseño e implementación de un controlador lógico difuso, aplicado al control de la posición de un servomotor de cd en derivación utilizando FPGA.

[7] López Vite, Sinhué (2008). Simulación de reguladores difusos de velocidad para motores de cd con excitación separada (doctoral dissertation).

[8] Maldonado, Y., Trujillo, L., Duarte, M., García, E., & Reyes, D. Fuzzy logic based on Elvis-FPGA.

ANEXO A PROGRAMACIÓN

Código en Arduino para la primera opción del controlador de velocidad basado en lógica difusa

```
#include <Fuzzy.h>
#include <FuzzyComposition.h>
#include <FuzzyInput.h>
#include <FuzzyIO.h>
#include <FuzzyOutput.h>
#include <FuzzyRule.h>
#include <FuzzyRuleAntecedent.h>
#include <FuzzyRuleConsequent.h>
#include <FuzzySet.h>

 volatile int encoder0Pos = 0, encoder1Pos = 0;
 int motorAPin0 = 4, motorBPin0 = 12;
 int motorAPin1 = 6, motorBPin1 = 13;
 int motorAPin_pwm = 5, motorBPin_pwm = 11;
 int motorCPin_pwm = 9, motorDPin_pwm = 10;
 float t = 0, s = 0;
 //int motor0PinB_pwm = 6, motor1PinB_pwm = 11;
 float perimetro = 3.1416*5.1*2.54, velocidad = 0;
 float Distancia0 = 0, Distancia1 = 0, PromDis = 0;
 float PosInicial = 0, Distancia = 0, DisAnt = 0;

 float Angulo = 0;

 Fuzzy* fuzzy = new Fuzzy();


 FuzzySet* PMN = new FuzzySet(-11,-11,-11,-5);
 FuzzySet* PN = new FuzzySet(-9,-4.5,-4.5,0);
 FuzzySet* PC = new FuzzySet(-5.2,0,0,4.5);
 FuzzySet* PP = new FuzzySet(-0.5,4.2,4.2,10.1);
 FuzzySet* PMP = new FuzzySet(5.1,11,11,11);


void setup()
  {
  pinMode(motorAPin0,OUTPUT);
  pinMode(motorAPin1,OUTPUT);
```

```
pinMode(motorAPin_pwm,OUTPUT);
pinMode(motorCPin_pwm,OUTPUT);
pinMode(motorBPin0,OUTPUT);
pinMode(motorBPin1,OUTPUT);
pinMode(motorBPin_pwm,OUTPUT);
pinMode(motorDPin_pwm,OUTPUT);
pinMode(2,INPUT_PULLUP);
pinMode(3,INPUT_PULLUP);
attachInterrupt( digitalPinToInterrupt(2), Encoder0, RISING);
attachInterrupt( digitalPinToInterrupt(3), Encoder1, RISING);
Serial.begin(9600);


FuzzyInput* Pendiente = new FuzzyInput(1);
Pendiente->addFuzzySet(PMN);
Pendiente->addFuzzySet(PN);
Pendiente->addFuzzySet(PC);
Pendiente->addFuzzySet(PP);
Pendiente->addFuzzySet(PMP);

fuzzy->addFuzzyInput(Pendiente);
FuzzyOutput* Potencia = new FuzzyOutput(1);
FuzzySet* PMB = new FuzzySet(20,25,25,50);
Potencia->addFuzzySet(PMB);
FuzzySet* PB = new FuzzySet(30,35,35,60);
Potencia->addFuzzySet(PB);
FuzzySet* PZ = new FuzzySet(40,70,70,85);
Potencia->addFuzzySet(PZ);
FuzzySet* PA = new FuzzySet(80,119,119,123);
Potencia->addFuzzySet(PA);
FuzzySet* PMA = new FuzzySet(120,220,230,235);
Potencia->addFuzzySet(PMA);

fuzzy->addFuzzyOutput(Potencia);

FuzzyRuleAntecedent* ifPendientePMN = new FuzzyRuleAntecedent();
ifPendientePMN->joinSingle(PMN);
FuzzyRuleConsequent* thenPotenciaPMB = new FuzzyRuleConsequent();
thenPotenciaPMB->addOutput(PMB);
FuzzyRule* fuzzyRule1 = new FuzzyRule(1, ifPendientePMN, thenPotenciaPMB);
fuzzy->addFuzzyRule(fuzzyRule1);
```

```
  FuzzyRuleAntecedent* ifPendientePN = new FuzzyRuleAntecedent();
  ifPendientePN->joinSingle(PN);
  FuzzyRuleConsequent* thenPotenciaPB = new FuzzyRuleConsequent();
  thenPotenciaPB->addOutput(PB);
  FuzzyRule* fuzzyRule2 = new FuzzyRule(2, ifPendientePN, thenPotenciaPB);
  fuzzy->addFuzzyRule(fuzzyRule2);


  FuzzyRuleAntecedent* ifPendientePC = new FuzzyRuleAntecedent();
  ifPendientePC->joinSingle(PC);
  FuzzyRuleConsequent* thenPotenciaPZ = new FuzzyRuleConsequent();
  thenPotenciaPZ->addOutput(PZ);
  FuzzyRule* fuzzyRule3 = new FuzzyRule(3, ifPendientePC, thenPotenciaPZ);
  fuzzy->addFuzzyRule(fuzzyRule3);


  FuzzyRuleAntecedent* ifPendientePP = new FuzzyRuleAntecedent();
  ifPendientePP->joinSingle(PP);
  FuzzyRuleConsequent* thenPotenciaPA = new FuzzyRuleConsequent();
  thenPotenciaPA->addOutput(PA);
  FuzzyRule* fuzzyRule4 = new FuzzyRule(4, ifPendientePP, thenPotenciaPA);
  fuzzy->addFuzzyRule(fuzzyRule4);


  FuzzyRuleAntecedent* ifPendientePMP = new FuzzyRuleAntecedent();
  ifPendientePMP->joinSingle(PMP);
  FuzzyRuleConsequent* thenPotenciaPMA = new FuzzyRuleConsequent();
  thenPotenciaPMA->addOutput(PMA);
 FuzzyRule* fuzzyRule5 = new FuzzyRule(5, ifPendientePMP, thenPotenciaPMA);
  fuzzy->addFuzzyRule(fuzzyRule5);
  }
void loop()
  {
  if(Distancia < 105){

  Angulo = map(analogRead(5),0,1023,-10,10);

  fuzzy->setInput(1, Angulo);
  fuzzy->fuzzify();

  Distancia0 = (perimetro/90)*encoder0Pos;
  Distancia1 = (perimetro/90)*encoder1Pos;
```

```
PromDis = (Distancia0 + Distancia1)/2;
Distancia = (PromDis - PosInicial) + DisAnt;

float Potencia = fuzzy->defuzzify(1);

  if( Angulo >= -10){
    if( Angulo <= 10 ){
   digitalWrite(motorAPin0,HIGH);
   digitalWrite(motorAPin1,LOW);
   digitalWrite(motorBPin0,HIGH);
   digitalWrite(motorBPin1,LOW);
   analogWrite(motorAPin_pwm,Potencia);
   analogWrite(motorCPin_pwm,Potencia);
   analogWrite(motorBPin_pwm,Potencia);
   analogWrite(motorDPin_pwm,Potencia);
   }
     }

   else {
   digitalWrite(motorAPin0,LOW);
   digitalWrite(motorAPin1,LOW);
   digitalWrite(motorBPin0,LOW);
   digitalWrite(motorBPin1,LOW);
   analogWrite(motorAPin_pwm,0);
   analogWrite(motorCPin_pwm,0);
   analogWrite(motorBPin_pwm,0);
   analogWrite(motorDPin_pwm,0);
     }

 t = millis();
 s = t/1000.0;
 velocidad = Distancia/s;
 Serial.print("Velocidad: "); Serial.println(velocidad);
 Serial.print("PWM: "); Serial.println(Potencia);
}


 else {
   digitalWrite(motorAPin0,LOW);
   digitalWrite(motorAPin1,LOW);
   digitalWrite(motorBPin0,LOW);
   digitalWrite(motorBPin1,LOW);
```

```
        analogWrite(motorAPin_pwm,0);
        analogWrite(motorCPin_pwm,0);
        analogWrite(motorBPin_pwm,0);
        analogWrite(motorDPin_pwm,0);
          }

        PosInicial = PromDis;
        DisAnt = Distancia;

    }
void Encoder0()
    {   encoder0Pos++;
    }

void Encoder1()
    {   encoder1Pos++;
    }
```

Código en Arduino para la segunda opción del controlador de velocidad basado en lógica difusa

```
#include <Fuzzy.h>
#include <FuzzyComposition.h>
#include <FuzzyInput.h>
#include <FuzzyIO.h>
#include <FuzzyOutput.h>
#include <FuzzyRule.h>
#include <FuzzyRuleAntecedent.h>
#include <FuzzyRuleConsequent.h>
#include <FuzzySet.h>

 volatile int encoder0Pos = 0, encoder1Pos = 0;
 int motorAPin0 = 4, motorBPin0 = 12;
 int motorAPin1 = 6, motorBPin1 = 13;
 int motorAPin_pwm = 5, motorBPin_pwm = 11;
 int motorCPin_pwm = 9, motorDPin_pwm = 10;
 float t = 0, s = 0;
 //int motor0PinB_pwm = 6, motor1PinB_pwm = 11;
 float perimetro = 3.1416*5.1*2.54, velocidad = 0;
 float Distancia0 = 0, Distancia1 = 0, PromDis = 0;
 float PosInicial = 0, Distancia = 0, DisAnt = 0;

 float Angulo = 0;

 Fuzzy* fuzzy = new Fuzzy();

 FuzzySet* PMN = new FuzzySet(-10,-10,-10,-5);
 FuzzySet* PN = new FuzzySet(-7.5,-3.75,-3.75,0);
 FuzzySet* PC = new FuzzySet(-5,0,0,5);
 FuzzySet* PP = new FuzzySet(0,3.75,3.75,10);
 FuzzySet* PMP = new FuzzySet(5,10,10,10);

void setup()
  {
  pinMode(motorAPin0,OUTPUT);
  pinMode(motorAPin1,OUTPUT);
  pinMode(motorAPin_pwm,OUTPUT);
  pinMode(motorCPin_pwm,OUTPUT);
  pinMode(motorBPin0,OUTPUT);
```

```
  pinMode(motorBPin1,OUTPUT);
  pinMode(motorBPin_pwm,OUTPUT);
  pinMode(motorDPin_pwm,OUTPUT);
  pinMode(2,INPUT_PULLUP);
  pinMode(3,INPUT_PULLUP);
  attachInterrupt( digitalPinToInterrupt(2), Encoder0, RISING);
  attachInterrupt( digitalPinToInterrupt(3), Encoder1, RISING);
  Serial.begin(9600);

  FuzzyInput* Pendiente = new FuzzyInput(1);
  Pendiente->addFuzzySet(PMN);
  Pendiente->addFuzzySet(PN);
  Pendiente->addFuzzySet(PC);
  Pendiente->addFuzzySet(PP);
  Pendiente->addFuzzySet(PMP);

  fuzzy->addFuzzyInput(Pendiente);

  FuzzyOutput* Potencia = new FuzzyOutput(1);

  FuzzySet* PMB = new FuzzySet(12,12,20,70);
  Potencia->addFuzzySet(PMB);
  FuzzySet* PB = new FuzzySet(12,60,60,60);
  Potencia->addFuzzySet(PB);
  FuzzySet* PZ = new FuzzySet(45,75,75,90);
  Potencia->addFuzzySet(PZ);
  FuzzySet* PA = new FuzzySet(80,120,120,120);
  Potencia->addFuzzySet(PA);
  FuzzySet* PMA = new FuzzySet(120,210,210,220);
  Potencia->addFuzzySet(PMA);

  fuzzy->addFuzzyOutput(Potencia);

  FuzzyRuleAntecedent* ifPendientePMN = new FuzzyRuleAntecedent();
  ifPendientePMN->joinSingle(PMN);
  FuzzyRuleConsequent* thenPotenciaPMB = new FuzzyRuleConsequent();
  thenPotenciaPMB->addOutput(PMB);
  FuzzyRule* fuzzyRule1 = new FuzzyRule(1, ifPendientePMN,
thenPotenciaPMB);

  fuzzy->addFuzzyRule(fuzzyRule1);
```

```
  FuzzyRuleAntecedent* ifPendientePN = new FuzzyRuleAntecedent();
  ifPendientePN->joinSingle(PN);
  FuzzyRuleConsequent* thenPotenciaPB = new FuzzyRuleConsequent();
  thenPotenciaPB->addOutput(PB);
  FuzzyRule* fuzzyRule2 = new FuzzyRule(2, ifPendientePN, thenPotenciaPB);

  fuzzy->addFuzzyRule(fuzzyRule2);

  FuzzyRuleAntecedent* ifPendientePC = new FuzzyRuleAntecedent();
  ifPendientePC->joinSingle(PC);
  FuzzyRuleConsequent* thenPotenciaPZ = new FuzzyRuleConsequent();
  thenPotenciaPZ->addOutput(PZ);
  FuzzyRule* fuzzyRule3 = new FuzzyRule(3, ifPendientePC, thenPotenciaPZ);

  fuzzy->addFuzzyRule(fuzzyRule3);

  FuzzyRuleAntecedent* ifPendientePP = new FuzzyRuleAntecedent();
  ifPendientePP->joinSingle(PP);
  FuzzyRuleConsequent* thenPotenciaPA = new FuzzyRuleConsequent();
  thenPotenciaPA->addOutput(PA);
  FuzzyRule* fuzzyRule4 = new FuzzyRule(4, ifPendientePP, thenPotenciaPA);

  fuzzy->addFuzzyRule(fuzzyRule4);

  FuzzyRuleAntecedent* ifPendientePMP = new FuzzyRuleAntecedent();
  ifPendientePMP->joinSingle(PMP);
  FuzzyRuleConsequent* thenPotenciaPMA = new FuzzyRuleConsequent();
  thenPotenciaPMA->addOutput(PMA);
  FuzzyRule* fuzzyRule5 = new FuzzyRule(5, ifPendientePMP,
thenPotenciaPMA);

  fuzzy->addFuzzyRule(fuzzyRule5);

  }

void loop()
  {

  if(Distancia < 105){
  Angulo = map(analogRead(5),0,1023,-10,10);
```

```
fuzzy->setInput(1, Angulo);
fuzzy->fuzzify();

Distancia0 = (perimetro/90)*encoder0Pos;
Distancia1 = (perimetro/90)*encoder1Pos;
PromDis = (Distancia0 + Distancia1)/2;
Distancia = (PromDis - PosInicial) + DisAnt;

float Potencia = fuzzy->defuzzify(1);

  if( Angulo >= -10){
    if( Angulo <= 10 ){
   digitalWrite(motorAPin0,HIGH);
   digitalWrite(motorAPin1,LOW);
   digitalWrite(motorBPin0,HIGH);
   digitalWrite(motorBPin1,LOW);
   analogWrite(motorAPin_pwm,Potencia);
   analogWrite(motorCPin_pwm,Potencia);
   analogWrite(motorBPin_pwm,Potencia);
   analogWrite(motorDPin_pwm,Potencia);
   }
      }

   else {
   digitalWrite(motorAPin0,LOW);
   digitalWrite(motorAPin1,LOW);
   digitalWrite(motorBPin0,LOW);
   digitalWrite(motorBPin1,LOW);
   analogWrite(motorAPin_pwm,0);
   analogWrite(motorCPin_pwm,0);
   analogWrite(motorBPin_pwm,0);
   analogWrite(motorDPin_pwm,0);
      }

  t = millis();
  s = t/1000.0;
  velocidad = Distancia/s;
  Serial.print("Velocidad: "); Serial.println(velocidad);
  Serial.print("PWM: "); Serial.println(Potencia);
  //Serial.print("PWM: "); Serial.println(Potencia2);
```

```
    Serial.print("Pendiente: "); Serial.println(Angulo);
    /*Serial.println(PN);
      analogWrite(motorAPin_pwm,0);
      analogWrite(motorBPin_pwm,0);
    Serial.println(PZ);*/
  }

   else {
      digitalWrite(motorAPin0,LOW);
      digitalWrite(motorAPin1,LOW);
      digitalWrite(motorBPin0,LOW);
      digitalWrite(motorBPin1,LOW);
      analogWrite(motorAPin_pwm,0);
      analogWrite(motorCPin_pwm,0);
      analogWrite(motorBPin_pwm,0);
      analogWrite(motorDPin_pwm,0);
        }

      PosInicial = PromDis;
      DisAnt = Distancia;

  }
void Encoder0()
  {   encoder0Pos++;
  }

void Encoder1()
  {   encoder1Pos++;
  }
```

Programa en Arduino para medir el tiempo que le toma al vehículo recorrer la distancia de un metro con ayuda del arreglo de sensores y la pantalla LCD.

```
#include<Timer.h>   //damos de alta la libreria de interrupciones del timer
#include <LiquidCrystal.h>  //damos de alta la libreria para la lcd

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);  //definimos los pines que vamso a utilizar para la lcd

int pinInicio = 6; //declaramos las variables que nos serviran a lo largo del programa
int pinFinal = 7, botonReset = 8;
int conta_ant;
int decimas, segundos, minutos;
bool control;
bool inicio, fin, reset;

Timer t;

int v = 0, v1 = 0, v2 = 0;
void setup() {

 Serial.begin(9600);
 lcd.begin(16,2);
 pinMode(pinInicio,INPUT); //definimos los pines que serviran para las entradas de los botones
 pinMode(pinFinal,INPUT);
 pinMode(botonReset,INPUT);
t.every(100,tiempo);  //definimos el objeto tipo timer que durara 100 milisegundos
}

void loop() {

  t.update(); //llamamos a la interrupcion
  lcd.setCursor(7,1);
  lcd.print("Cronometro");

  inicio = digitalRead(pinInicio); //con las variables p registramos los valores que tengamos en los pines donde estan conectados los botones esta sera para el boton de inicio
  fin = digitalRead(pinFinal);  //regstramos el valor del boton de paro
```

```
  reset = digitalRead(botonReset);  //y con esta ultima variable registramos el valor del boton de reinicio

  if(inicio == 0  && control == 0) //si el boton de inicio es pulsado
  {
    control = 1; //entonces igualamos la variable de control a 1 esto servira para poner en marcha el cronometro

    }


 if(fin == 0 && control == 1) //para el boton 2 sera basicamente lo mismo checar si esta pulsado y que la variable de identificacion sea igual a 0
  {
     v1 = 1;

    }
  if(v1 == 1)
  {
    control = 0;
  }

  lcd.setCursor(8,0);

  if(minutos < 10) //la variable conta2 servira para contar los minutos
  {
 lcd.print("0");  //si son menores a 10 entonces escribimos 0 en la parte de los segundos
}
 lcd.print(minutos); //e imprimimos la cantidad de decimas de minutos que hayan pasado
 lcd.print(":"); //imprimimos en la lcd un separador
 lcd.setCursor(11,0);  //indicamos en el lcd de donde se empezara a escribir nuevamente

 if(segundos < 10)  //mientras la variable conta1 que indicara los segundos
 {
   lcd.print("0");  //imprimiremos en el lcd un 0 para los segundos
 }
 lcd.print(segundos); //imprimimos el valor de conta1 en el lcd
 lcd.print(":");    //imprimimos el separador entre segundos y decimas de segundo
```

```
 lcd.setCursor(14,0); //colocamos el cursor de la lcd en la posicion 10 de la primera
linea
 lcd.print(decimas);    //y en esa posicion escribimos el valor de conta


if(reset==1&&v2==0&&control==0)  //programamos la condicion para el boton de
reinicio
  {
    decimas=0;    //declaramos todas las variables de decimas, segundos y minutos
a 0
    segundos=0;
    minutos=0;

    v2=1;     //e igualamos la variable v2 a 1

    }
  if(reset==0 && v2==1); //en cuanto el boton de reinicio ya no este pulsado y la
variable v2 sea 1
  {
    v2=0; //regresamos a v2 a 0

    }
}

void tiempo (){
 if(control == 1){ //en cuanto el boton de inicio sea pulsado el cronometro
empezara a correr
 decimas = decimas + 1;  //y vamos a incrementar el valor de la variable conta en 1
     if(decimas>9){decimas = 0;  //en cuanto la variable conta rebase el valor de 10
regresara a 0
      segundos = segundos + 1;}  //e incrementaremos el valor de la variable conta1
       if (segundos>59){   //en cuanto conta 1 sea mayor que 59 regresara a ser 0
      segundos = 0;
      minutos = minutos + 1; //y conta2 incrementara en 1
      }
    }
}
```