PROYECTO DE INVESTIGACIÓN:

“IMPLEMENTACIÓN DE ALGORITMOS DE CONTROL DE VELOCIDAD Y TORQUE DE MOTORES DE CORRIENTE CONTINUA APLICADOS A UN MODELO DE AGV MULTIPROPÓSITO DE POTENCIA MEDIA UTILIZANDO RASPBERRY PI.”

ESTUDIANTES:

Sebastián Herrera Aristizábal

Cod. 1093219941

Julio Alejandro Hincapié Correa

Cod. 1088007505

Trabajo presentado como requisito para optar por el título de Ingeniero Electricista

UNIVERSIDAD TECNOLÓGICA DE PEREIRA

FACULTAD DE INGENIERÍA ELÉCTRICA

FEBRERO, 2016

TABLA DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

Los motores de corriente continua y sus unidades de control han sido ampliamente utilizados en diversos procesos industriales y electrodomésticos, tales como sillas de ruedas eléctricas, robótica, máquinas laminadoras, herramientas, entre otros. Muchas aplicaciones requieren un control de velocidad muy precisa. Sin embargo, los motores de corriente continua son inestables en su funcionamiento debido a que los parámetros del sistema pueden ser variantes en el tiempo. Estas variaciones son probablemente debido a imprecisiones en la detección de corriente, aumento de la temperatura y cambios en las condiciones de operación, así como a los errores de algún sensor. En los últimos años, muchos investigadores han estado estudiando nuevas y diferentes técnicas de control con el fin de mejorar la regulación en el rendimiento de la velocidad del motor DC, en [2] Los autores aplican un sintonizador de ganancia difusa en conjunto con una técnica de control tradicional PI (proporcional Integral) para el control del motor, una técnica como PID es utilizada en [3], donde los autores se valen además de una mejora en el método, también de elementos y software como Xilinx y la FPGA con el fin de hacer control de un motor DC, en [5] se implementa un control por Modulación de ancho de pulso (PWM) utilizando microcontroladores e implementando todo el circuito de control de realimentación en lazo cerrado.

En el presente proyecto de investigación se abordará el tema de control de velocidad y torque de un motor DC utilizando un sistema de bajo costo y a través de una técnica de control eficiente, la cual se determinará mediante la comparación de acciones de control clásicas y modernas [2] [3] [4] [5].

Raspberry Pi es un sistema de computación embebido de bajo costo, pequeño y potente que permite a personas de todas las edades explorar en el mundo de la computación y robótica, además, aprender lenguajes de programación como Scratch y Python. Cuenta con la habilidad de interactuar con el mundo físico, lo que permite que sea usado en bastantes proyectos digitales desde reproductores de música y detectores de personas, hasta estaciones de detección de clima, casas cantantes para pájaros con cámaras infrarrojas y muchos otros proyectos más alrededor del mundo [1].

2. METODOLOGÍA

La secuencia de actividades realizadas en este proyecto fue:

- Recopilación de información, la cual permitió tener una base teórica para validar el proyecto a realizar. En esta parte se consultó la bibliografía especializada que permitió plantear el estado del arte del proyecto.

- Plantear soluciones al problema enunciado según la teoría recopilada, proponiendo esquemas que involucraron algunas de las herramientas propuestas en el marco teórico (según el estudio realizado para diferentes circunstancias en la variación de la carga y de la velocidad, se seleccionó la técnica respectiva). Dando soluciones teóricas que permitieron agilizar el proceso de creación de los algoritmos para el fin buscado. La idea fue observar que técnicas se podían emplear de tal forma que se diera una solución óptima al problema propuesto.

- Desarrollo de la estrategia de prueba utilizando las herramientas de control seleccionadas. En esta sección se implementaron y probaron las técnicas propuestas, realizando las respectivas simulaciones utilizando software especializado SIMULINK/MATLAB.

- Experimentación con la planta escogida (motor DC de potencia media), aplicando las técnicas que resultaron ser más viables para el control del torque y la velocidad del motor. Se hizo la programación de las técnicas de control en lenguaje Python en el sistema computacional embebido Raspberry Pi.

- Presentación y normalización de resultados.

3. MANEJO Y CONFIGURACIÓN DE RASPBERRY PI

3.1 Estudio de hardware de Raspberry Pi

Raspberry Pi es un sistema de computación embebido del tamaño de una tarjeta de crédito, que funciona con un monitor corriente o TV, ratón y teclados USB convencionales; es capaz de realizar cualquier actividad esperada por un computador de escritorio común, desde navegación por internet y reproducir videos de alta definición, realizar hojas de cálculo, procesamiento de texto hasta jugar vídeo juegos [1].

Raspberry Pi cuenta con el siguiente Hadware para su funcionamiento: Un SoC (system on chip) es un integrado que incorpora todos los componentes del sistema. En el caso de la Raspberry Pi, lleva un Broadcom BCM2835 que incluye: el procesador (ARM1176JZF-S), la tarjeta gráfica con aceleración gráfica 3D y de video en alta definición, 512 Mb de RAM, tarjeta de sonido estéreo y bus USB. Una Tarjeta de red, Conector RJ-45 conectado a un integrado lan9512 -jzx de SMSC que proporciona conectividad a 10/100 Mbps. Puertos de Salida, dos buses USB, puerto de entrada de memoria SD, salida analógica de audio estéreo por jack de 3.5 mm, salida digital de video + audio HDMI, salida analógica de video RCA, pines de entrada y salida de propósito general [6].

3.1.1 GPIO (Pines multipropósito)

El SoC Broadcom BCM2835 dispone de numerosos pines de entrada/salida de propósito general (GPIO) que proveen de un puerto SPI (puerto serie), un puerto I^2C, salidas PWM y salidas/entradas digitales [6].

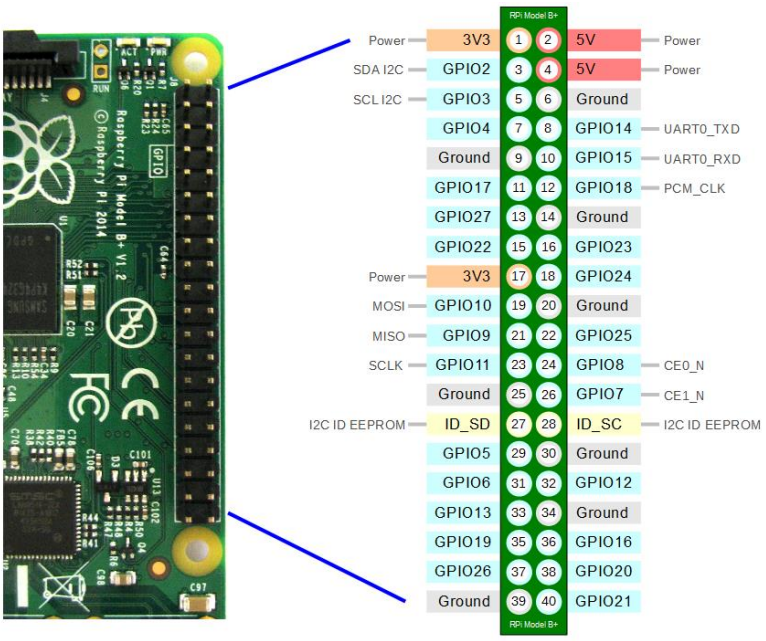

Figura 3.1 Diagrama de Pines GPIO RASPBERRY PI B+ [7]

Estos pines permitirán comunicación con otros dispositivos TTL de forma directa como puede ser Arduino y microcontroladores varios sin contar que sobre ellos se pueden conectar dispositivos directamente Como LEDs, transistores, relés, etc [6].

3.2 Configuración e instalación de sistema operativo

Para el uso de Raspberry Pi es necesario descargar uno de los sistemas operativos funcionales, en este caso se hará uso de Raspbian, el cual se encuentra en [1] para su descarga directa y gratuita; una vez descargado, se extrae su contenido (un archivo de extensión img) y se copia la imagen en una memoria SD, la cual será introducida en Raspberry pi [8].

El siguiente paso consiste en conectar los periféricos necesarios para el funcionamiento del sistema embebido, teclado, mouse, monitor y fuente de alimentación; una vez se realizan las conexiones requeridas se enciende, y para iniciar sesión se deben ingresar el usuario "*pi*" y la contraseña "*Raspberry*" [8].

El primer menú que se muestra contiene 9 diferentes opciones las cuales se deberán configurar como se explica en [8], una vez realizada la configuración se finaliza y se accede al entorno gráfico de Raspbian.

3.2.1 Configuración Cliente SSH

Como se menciona en [8], el cliente SSH (secure shell) permite una conexión al Raspberry pi por medio de la red, lo cual, evitará el uso de elementos periféricos como monitor, mouse y teclado; para configurarlo se deben seguir los siguientes pasos:

En Raspberry:

1. Abrir “LXTerminal” y digitar los comandos: sudo apt-get update y luego: sudo apt-get upgrade, para actualizar el sistema digital Raspberry pi correctamente.

2. Digitar el comando: sudo nano /etc/network/interfaces, y en el código. se deben hacer cambios de manera que sea igual a la siguiente imagen:

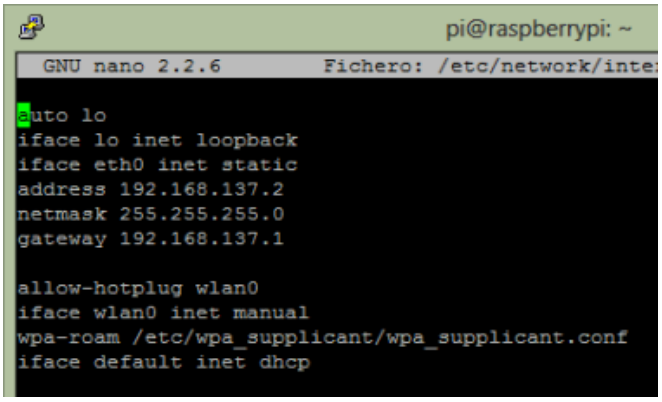

Figura 3.2 Modificación de interfaces

3. Digitar el comando: sudo nano /etc/resolv.conf y modificar las filas predeterminadas por: nameserver 8.8.8.8 y nameserver 8.8.4.4

En Windows:

1. Una vez conectada la Raspberry vía Ethernet al PC se debe abrir centro de redes y recursos compartidos, se selecciona la conexión vía Ethernet – Propiedades-Protocolo de internet versión 4 y asignar la dirección IP fija 192.168.137.1

2. Descargar e instalar el programa Putty como se explica en [9] y se tendrá acceso a Raspberry pi remotamente.

4. MOTOR DE CORRIENTE CONTINUA Y ACCIONES DE CONTROL

4.1 Motor DC

4.1.1 Definición

Los motores DC convierten potencia eléctrica en potencia mecánica. Estos fueron usados ampliamente en el pasado para todo tipo de aplicaciones, y hoy en día se continúan usando en aplicaciones de control de velocidad y posición [10].

La estructura física de la máquina consta de dos partes: el *estator* o parte estacionaria de la máquina y el *rotor* o parte giratoria de la máquina. La parte estacionaria de la máquina consta de una estructura que proporciona el soporte físico y las *piezas polares*, las cuales se proyectan hacia dentro y proveen el camino para el flujo magnético en la máquina. Los extremos de las piezas polares cercanos al rotor se extienden hacia fuera, sobre la superficie del rotor, para distribuir el flujo uniformemente sobre la superficie del rotor. Estos extremos son llamados *zapatas polares*. La superficie expuesta de una zapata polar se llama *cara polar* y la distancia entre la cara polar y el rotor se llama *entrehierro* [10].

En una máquina DC hay dos devanados principales: Los devanados del inducido (armadura) y los devanados de campo. Los *devanados del inducido* están definidos como aquellos en los cuales es inducido el voltaje, y los *devanados de campo* están definidos como aquellos que producen el flujo magnético principal en la máquina. En una máquina de corriente continua, los devanados del inducido están localizados en el rotor y los devanados de campo están localizados en el estator. Debido a que los devanados del inducido están localizados en el rotor, el rotor de una máquina DC es llamado en ocasiones *armadura* [10].

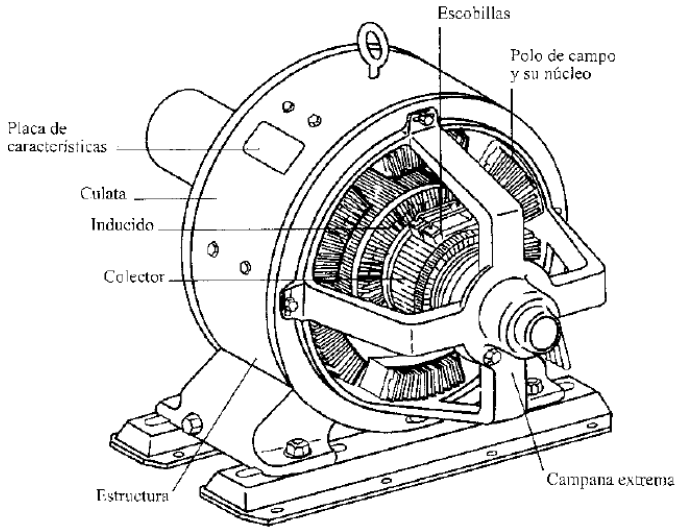

Figura 4.1 Componentes principales máquina DC [10]

4.1.2 Circuito equivalente motor DC

El circuito del inducido está representado por una fuente ideal de voltaje $\mathbf{E_A}$ y una resistencia $\mathbf{R_A}$**.** Está representación es el equivalente Thevenin de la estructura total del rotor, incluidas las bobinas del rotor, los interpolos y los devanados de compensación, si los hay. La caída de voltaje en la escobilla está representada por una pequeña batería $\mathbf{V_{esc}}$ opuesta en dirección al flujo de corriente de la máquina. Las bobinas de campo que producen el flujo magnético en el generador están representadas por la inductancia L_F y la resistencia R_F. La resistencia separada $\mathbf{R_{adj}}$ representa una resistencia exterior variable, utilizada para controlar la cantidad de corriente en el circuito de campo [11].

Existen algunas variantes y simplificaciones de este circuito equivalente básico. Con frecuencia, el voltaje de caída en la escobilla es sólo una pequeña fracción del voltaje generado en una máquina. En casos en los cuales no es demasiado crítico, el voltaje de caída en la escobilla puede despreciarse o incluirse aproximadamente en el valor de $\mathbf{R_A}$. A veces, la resistencia interna de las bobinas de campo también se agrupa con la resistencia variable y a este total se le llama $\mathbf{R_F}$ [11]. Ver ***Figura 4.2***.

El voltaje interno generado en esta máquina está dado por la **Ecuación 4.1**, así:

$$E_A = K\varphi\omega \quad (4.1)$$

Y el par inducido desarrollado por la máquina está dado por la **Ecuación 4.2**, así:

$$\tau_{ind} = K\varphi I_A \quad (4.2)$$

Estas dos ecuaciones, la correspondiente a la ley de voltajes de Kirchhoff del circuito del inducido y la curva de magnetización de la máquina, son las herramientas necesarias para analizar el funcionamiento de un motor DC [11].

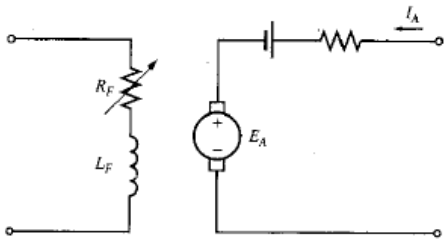

Figura 4.2 Circuito equivalente simplificado de un motor DC [11]

4.1.3 Modelo matemático del motor DC y función de transferencia de la planta

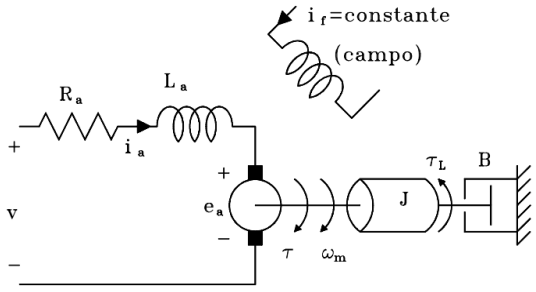

Figura 4.3 Motor DC controlado en el inducido [12]

$$K_e = K_a\phi \quad (4.3)$$

$$e_a = K_e w_m \quad (4.4)$$

$$K_t = K_b\phi \quad (4.5)$$

$$T = K_t i_a \quad (4.6)$$

w_m: $Velocidad\ angular\ motor$

ϕ: Flujo en el entrehierro

T: $Torque\ generado$

Segunda ley de Kirchhoff

$$v - e_a = R_a i_a + L_a \frac{di_a}{dt} \quad (4.7)$$

Segunda ley de Newton

$$T - T_{l=} J \frac{dw_m}{dt} + B w_m \quad (4.8)$$

Transformando a Laplace (4.7) y (4.8)

$$I_a(S) = \frac{V(S) - E_a(S)}{L_a S + R_a} \quad (4.9)$$

$$W_m\,(S) = \frac{T(S) - T_l(S)}{JS + B} \quad (4.10)$$

Remplazando (4.4) en (4.9)

$$I_a(S) = \frac{V(S) - K_e W_m(S)}{L_a S + R_a} \quad (4.11)$$

Partiendo de las ecuaciones (4.9), (4.10) y (4.11) se obtiene el diagrama de bloques que representa el motor DC controlado en el inducido [12].

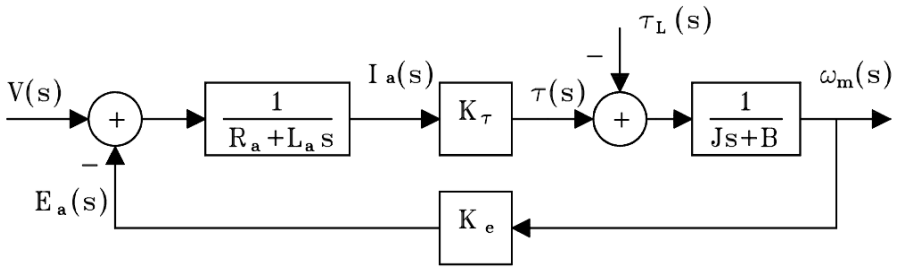

Figura 4.4 Diagrama de bloques de la planta [12]

$$G_s = \frac{\frac{K_t}{L_aJS^2 + (L_aB + R_aJ)S + R_aB}}{1 + \frac{K_tK_e}{L_aJS^2 + (L_aB + R_aJ)S + R_aB}} = \frac{K_t}{L_aJS^2 + (L_aB + R_aJ)S + R_aB + K_tK_e} \quad (4.12)$$

4.1.4 Determinación de parámetros del motor DC

Las siguientes pruebas se realizaron en el laboratorio de máquinas eléctricas de la facultad de ingenierías, en el módulo de labvolt, a un motor DC de 180W referencia MY1016.

Modelo	MY 1016
Standard	180W 24V/36V
Corriente en vacío/A	≤1.5/1.0
Velocidad en vacío /rpm	2000
Torque nominal/N·m	0.70
Velocidad nominal/rpm	1500
Corriente máxima/A	≤10.6/7.1
Eficiencia/%	≥76
Uso	Scooter Eléctrico

Tabla 4.1 Características motor DC

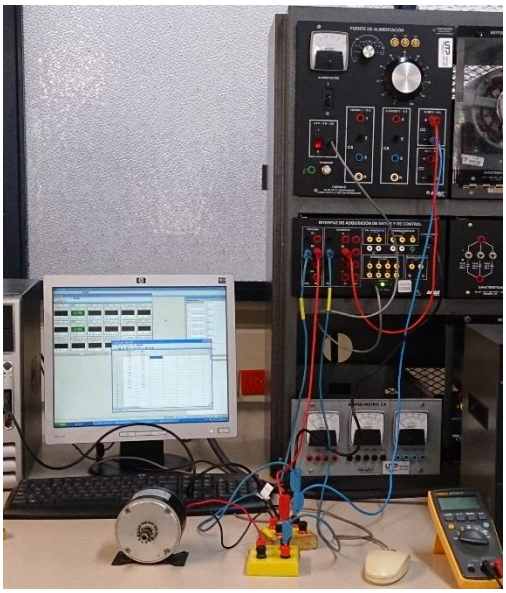

Figura 4.5 Pruebas determinación parámetros de motor DC

4.1.4.1 Resistencia de armadura

Para calcular la resistencia se alimentó el motor con una tensión de baja magnitud, justo antes de que comenzara a funcionar, y se tomaron diferentes medidas de tensión y corriente

variando la posición del rotor; a continuación se realizó el cálculo de la resistencia de armadura usando la ley de ohm $R = \frac{V}{I}$ y realizando un promedio de los resultados [13].

MOTOR 180W		
E (V)	**I (A)**	**Ra (Ω)**
0,595	0,253	2.352
0,608	0,251	2.422
0,624	0,246	2.537
0,637	0,244	2.611
0,655	0,236	2.775
0,667	0,235	2.838
0,667	0,234	2.850
0,719	0,221	3.253
0,725	0,218	3.326
0,746	0,215	3.470
0,758	0,216	3.509
0,781	0,208	3.755
0,818	0,199	4.111
0,833	0,192	4.339
0,869	0,185	0.470
1,080	0,132	8.182
		3.564

Tabla 4.2 Resistencia de armadura

4.1.4.2 Inductancia de armadura.

Para el cálculo de la inductancia, se alimentó el motor con una fuente AC y se varió teniendo en cuenta que no se excedieran los límites de corriente de los motores, se tomaron las lecturas

de tensión y corriente; a continuación se realizó el cálculo de la Impedancia usando la ley de ohm $Z = \frac{V}{I}$ y realizando un promedio de los resultados [13].

MOTOR 180W		
E (V)	**I (A)**	**Z (Ω)**
0,993	0,422	2,35308057
1,69	0,855	1,97660819
2,09	1,09	1,91743119
2,26	1,86	1,21505376
2,762	2,265	1,21942605
2,932	3,194	0,9179712
2,604	2,942	0,88511217
3,086	3,427	0,90049606
3,354	3,518	0,9533826
3,731	3,647	1,02303263
		1,33615944

Tabla 4.3 Inductancia de armadura

Para el cálculo de L (Inductancia de armadura) se debía usar la ecuación $L = \frac{\sqrt{z^2 - R^2}}{2\pi f}$ Obteniendo un valor de inductancia negativo, concluyendo así, que el valor de inductancia era demasiado pequeño para ser calculado por medio de esta prueba. Para el correcto cálculo de la Inductancia de armadura se utilizó un medidor de parámetros, obteniendo el siguiente resultado:

Figura 4.6 Inductancia motor 180W

4.1.4.3 Constantes intrínsecas del motor

Para determinar el valor de la constante de fuerza electromotriz K se puede partir de la ecuación: $K = \frac{Va - Ra * i}{\omega}$; para tratar de minimizar el error en el cálculo se deben hacer mediciones variando la tensión desde 1 hasta el voltaje máximo, después se calculará una K promedio [14].

Para determinar el valor del coeficiente de amortiguamiento B se puede partir de la ecuación: $B = \frac{Ke * i}{\omega}$ para tratar de minimizar el error en el cálculo se deben hacer mediciones variando la tensión desde 1 hasta el voltaje máximo, después se calculará una B promedio [14].

MOTOR 180W					
E (V)	**I (A)**	**W(rpm)**	**W(rad/s)**	**K(V*s/rad)**	**B(N*m*s/rad)**
2,132	0,345	203,2	21,279104	0,0424061	0,000687534
3,255	0,381	365,3	38,254216	0,04959072	0,000493908
4,375	0,431	511	53,51192	0,05305072	0,000427285
5,367	0,473	638,8	66,895136	0,05502868	0,000389095
6,278	0,506	758,8	79,461536	0,05631069	0,000358579
7,314	0,536	891,9	93,399768	0,05785463	0,000332015
8,525	0,559	1052	110,16544	0,05929839	0,000300891
9,596	0,58	1189	124,51208	0,06046631	0,000281663
10,65	0,622	1330	139,2776	0,06054879	0,000270405
11,54	0,62	1447	151,52984	0,06157348	0,000251934
20,81	0,727	2687	281,38264	0,06474761	0,000167286
				0,05644328	**0,000360054**

Tabla 4.4 Constantes intrínsecas del motor

4.1.4.4 Momento de inercia

Para estimar el momento de inercia se considera que el motor es un cilindro, y calcular el momento de inercia de un cilindro es muy fácil, y está dado por la siguiente ecuación: $J = \frac{M*R^2}{2}$, donde M es la masa del motor, R el radio estimado del motor [15].

$$J = \frac{1{,}8*0{,}049^2}{2} Kgm^2 = 2{,}1603x10^{-3} Kgm^2 \quad (4.13)$$

Valor de los parámetros obtenidos experimentalmente para el motor de 180W

MOTOR 180 W	
Ra (Ω)	3,56
K ($\frac{Vs}{Rad}$)	0,056
J (Kgm^2)	$2{,}1603x10^{-3}$
B ($\frac{Nms}{Rad}$)	3,6x10^{-4}
L (µH)	441,2

Tabla 4.5 Parámetros del motor DC obtenidos experimentalmente

Remplazando el valor de los parámetros en (4.12) y operando

$$G(S) = \frac{58737{,}82}{S^2 + 8069{,}07S + 4633{,}67} \quad (4.14)$$

Donde la **Ecuación 4.14** equivale a la función de transferencia de la planta.

4.1.5 Relación torque-corriente

Para determinar la relación entre Torque y Corriente del motor DC, ref. MY1016-180W, se realizaron pruebas de laboratorio, en donde variaba la carga vista por el motor utilizando un acoplamiento mecánico entre el motor de 180W y otro motor utilizado como dinamómetro. Se polarizó el motor a 24V y se hizo control de carga aumentando gradualmente el esfuerzo

entregado por el dinamómetro del módulo de LabVolt, ***Figura 4.7***. Se hizo seguimiento al amperímetro analógico DC y al visualizador de torque y velocidad del dinamómetro para la toma de datos, ***Figura 4.7***. Los resultados obtenidos se indican en la ***Tabla 4.6***.

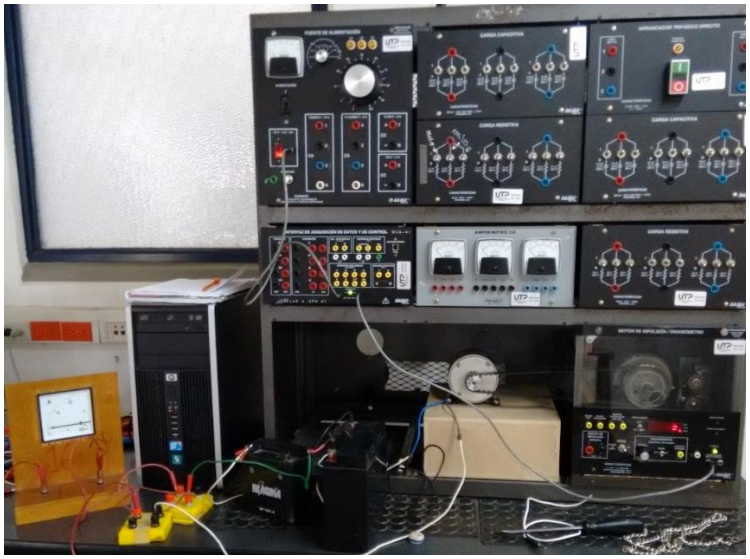

Figura 4.7 Prueba relación torque-corriente

MOTOR 180 W			
Corriente [A]	**Torque [N.m]**	**Velocidad [rpm]**	**Voltaje [V]**
2	0	1291	24.2
3.1	0.14	1221	24.2
4.2	0.4	1150	24.2
5.2	0.6	1079	24.2
6	0.72	1036	24.2
7.2	0.87	973	24.2
8	1	930	24.2
9.8	1.33	822	24.2
12	1.7	622	24.2
13.8	2.1	460	24.2
14	2.4	384	24.2

Tabla 4.6 Resultados pruebas relación torque-corriente

4.2 Control PID

Los controladores de tipo PID son con mucha diferencia los más frecuentemente utilizados en la industria de control de procesos, donde más del 95% de los lazos de control utilizan controladores PID. Sus sencillas, versatilidad y capacidad para resolver los problemas básicos que se presentan en los procesos con dinámica favorable y requisitos de funcionamiento modestos hacen de ellos una herramienta imprescindible en el control de procesos. A lo largo de las últimas décadas los controladores PID han sobrevivido diferentes cambios tecnológicos que van desde los primeros controladores desarrollados a partir de elementos neumáticos hasta los desarrollados con microprocesadores pasando por las válvulas electrónicas, los dispositivos analógicos, los transistores y los circuitos integrados. La incorporación de los microprocesadores ha tenido un impacto muy grande ya que ha permitido que los controladores PID se enriquezcan en funciones colaterales sin perder ninguna de sus propiedades. Posibilidades de ejecución de reglas lógicas o automatismo secuenciales. Por todo ello, y aunque existen técnicas de control más sofisticadas, en el nivel más bajo de control de muchos procesos sigue estando presente o incluso es el mayoritario [16] [17] [18].

El controlador ideal en tiempo continuo para un proceso SISO (Single input-Single output) está expresado en el dominio de Laplace de la siguiente forma:

$$U(S) = G_c(S)E(S) \quad (4.15)$$

La función de transferencia de un controlador PID está dada por **Ecuación 4.15**

$$G_c(s) = K_p + K_d S + \frac{K_i}{S} \quad (4.16)$$

Donde Kp = ganancia proporcional, Kd y Ki son constantes de valor real. El problema de diseño consiste en la determinación de los valores de estas tres constantes, de tal manera que el sistema de control satisfaga las condiciones de diseño [16] [17].

El control derivativo introduce una señal que es proporcional al efecto de la derivada en el sistema de lazo, cuyo objeto es mejorar el amortiguamiento del sistema de lazo cerrado. El control derivativo no afecta el estado estable del sistema de salida [16] [17].

El control integral introduce una señal que es proporcional al efecto de la integral en el sistema de lazo. El control integral mejora el error de estado estable del sistema de lazo cerrado y permite encontrar el valor apropiado de la constante Ki. El amortiguamiento de la respuesta transitoria también puede ser mejorado a costa del tiempo de subida y del tiempo de asentamiento [16] [17].

El diseño de un controlador PID, puede hacerse artificialmente dividiendo el controlador en dos partes; PD y PI el procedimiento es el siguiente:

Considerar que el controlador PID consta de una parte PI conectada en cascada con otra parte PD. La función de transferencia del controlador PID dada en la **Ecuación (4.15)** es particionada como [16] [17]:

$$G_c(s) = K_p + K_d S + \frac{K_i}{S}$$

$$G_c(s) = (1 + K_{d1} S)(K_{p2} + + \frac{K_{i2}}{S}) \quad (4.17)$$

La constante proporcional de la parte PD se coge como 1, ya que el controlador PID solo requiere de tres parámetros. Igualando ambos miembros de la **Ecuación (4.17)** se tiene:

$$K_d = K_{d1} K_{d2} \quad (4.18)$$

$$K_d = K_{d1} K_{p2} \quad (4.19)$$

$$K_i = K_{i2} \quad (4.20)$$

La parte PI selecciona los valores de K_{i2} y K_{p2} que son requeridos para que el tiempo de subida del sistema sea satisfecho. El error de estado estable del sistema de control PI es mejorado en un orden. El máximo sobre impulso en esta etapa no es considerado y puede ser más grande que el deseado [16] [17] [18].

Se usa la parte PD para reducir el máximo sobre impulso. Se selecciona el valor de K_{d1} para encontrar el máximo sobre impulso requerido. Los valores de Kp, Kd y Ki son encontrados usando las **Ecuaciones (4.18), (4.19) y (4.20)** [16] [17].

Existe un algoritmo PID llamado PID de velocidad el cual cuenta con la siguiente ley de control [19]:

$$\mathrm{U}(s) = \mathrm{K}\left(1 + \frac{1}{T_i S} + \frac{T_d S}{1+\frac{T_d}{\mathrm{N}} S}\right) E(s) \quad (4.21)$$

4.2.1 Sintonización ganancias PID

Para llevar a cabo la implementación del controlador PID se hace uso de un bloque contenido en SIMULINK/MATLAB llamado PID Controller; este bloque sintoniza las ganancias del controlador automáticamente [20].

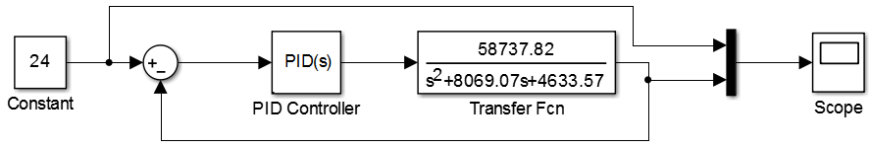

Figura 4.8 Diagrama de bloques controlador PID SIMULINK/MATLAB

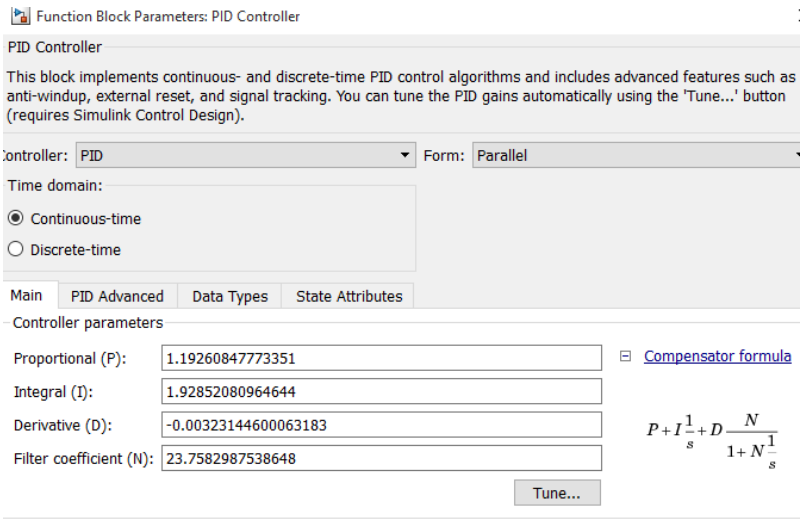

Figura 4.9 Sintonización automática de constantes SIMULINK/MATLAB

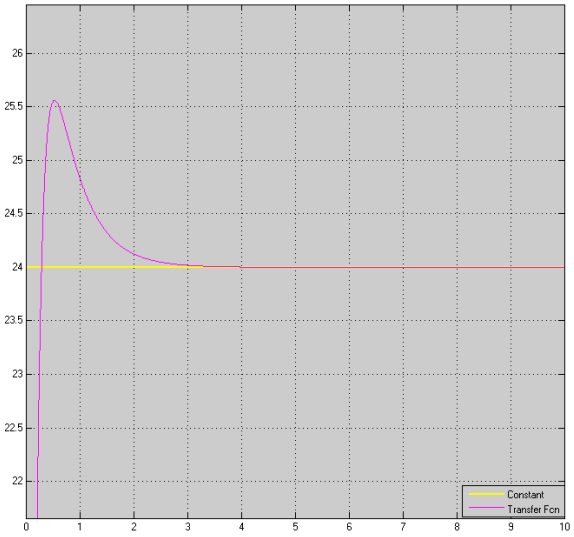

Figura 4.10 Respuesta de la planta controlada y señal de referencia en SIMULINK/MATLAB

4.2.2 Control PID digital

Para poder realizar un control a nivel digital se debe discretizar la ley de control, en este caso se debe discretizar la ley de control PID. Aplicando la transformada Z a la **Ecuación (4.21)**, se obtiene [19]:

$$\frac{\mathrm{U}(z)}{E(z)} = K_p\left(1 + \frac{T}{T_i(1-Z^{-1})} + \frac{\frac{\mathrm{N}*\mathrm{T}}{T_d+\mathrm{N}*\mathrm{T}}(1-Z^{-1})}{1-\frac{T_d}{T_d+\mathrm{N}*\mathrm{T}}Z^{-1}}\right) \quad (4.22)$$

Donde T es el periodo de muestreo elegido.

Para poder implementar el Control PID en Raspberry pi es necesario transformar la **Ecuación (4.22),** que está en transformada Z, en ecuación en diferencias utilizando la transformada Z inversa, obteniendo [19]:

$$\mathrm{U}(k) = U_{k-1} + a * E_k + b * E_{k-1} + c * E_{k-2} \quad (4.23)$$

$$a = K_p + \frac{K_p*T}{T_i} + \frac{K_p*T_d}{\mathrm{T}} \quad (4.24)$$

$$b = K_p + \frac{2*K_p*T_d}{\mathrm{T}} \quad (4.25)$$

$$c = K_p + \frac{K_p*T_d}{\mathrm{T}} \quad (4.26)$$

4.3 Lógica difusa

El concepto de lógica difusa es muy común, está asociado con la manera en que las personas perciben el medio, por ejemplo ideas relacionadas con la altura de una persona, velocidad con la que se mueve un objeto, la temperatura dominante en una habitación, cotidianamente se formulan de manera ambigua y depende de quién percibe el efecto físico o químico, será enunciado acerca de tal fenómeno. Una persona puede ser alta o baja, algo puede moverse rápido o lento, una temperatura puede ser baja, moderada o alta, se dice que estas afirmaciones acerca de una variable son ambiguas porque rápido, alto o bajo son afirmaciones del observador, y estas pueden variar de un observador a otro [23].

El control difuso proporciona una metodología formal para representar, manipular e implementar el conocimiento heurístico del ser humano acerca de cómo controlar un sistema. Básicamente, se puede ver el controlador difuso con un organismo artificial que toma

decisiones, que opera en un sistema de ciclo cerrado, en tiempo real. Reúne información de la salida de la planta, lo compara con el valor de referencia, y después decide cuál entrada de la planta [21]

Los conjuntos difusos definen justamente estas ambigüedades, y son una extensión de la teoría clásica de conjuntos, donde un elemento pertenece o no a un conjunto, tal elemento tiene solo 2 posibilidades, pertenecer o no, un elemento es bi-valuado y no se definen ambigüedades. Con los conjuntos difusos se realizan afirmaciones lógicas del tipo "IF-THEN", definiéndose estas con lógica difusa. Este tema es propio de inteligencia artificial, donde se intenta emular el pensamiento humano. Nuestro campo de estudio es el control industrial, debemos tener en cuenta la experiencia o base de conocimiento del operario, esto será útil para emular el comportamiento humano con una máquina, a pesar de estar muy limitada [23].

Un diagrama de bloque de un sistema de control difuso digital se muestra en la ***Figura 4.11*****.** El controlador difuso se compone de los siguientes 6 elementos [21]:

1) Una *reglas base* (Conjunto de reglas "IF-THEN"), las cuales contienen una cuantificación lógica difusa de la descripción lingüística del experto de cómo conseguir un control óptimo.

2) Un *generador de reglas activas,* etapa en la cual cada variable de entrada que genera una dirección de memoria, es discretizada, puntualizada y asociada a su correspondiente regla activa almacenada en la memoria.

3) Un *mecanismo de inferencia* (También llamado "motor de inferencia" o "módulo de Inferencia difusa"), el cual emula la decisión del experto de hacer interpretación y aplicar el conocimiento acerca de la mejor manera de controlar la planta.

4) Una *interface de fusificación*, la cual convierte las entradas del controlador en información que el *mecanismo de inferencia* pueda fácilmente usar para activar y aplicar reglas.

5) Una etapa de *Agregado*, donde se agrupan los conjuntos difusos evaluados antes de ser desfusificados.

6) Una *Interface de desfusificación*, la cual convierte las conclusiones del *mecanismo de inferencia* a entradas adecuadas para el proceso.

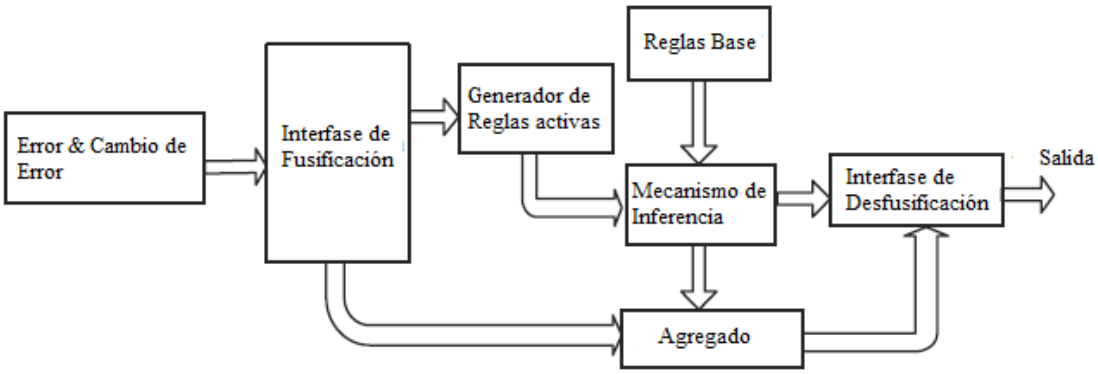

Figura 4.11 Controlador difuso

4.3.1 Interface de fusificación.

Es un procedimiento matemático en el que se convierte un elemento del universo de discurso (variable medida del proceso) en un valor para cada función de membresía a las cuales pertenece.

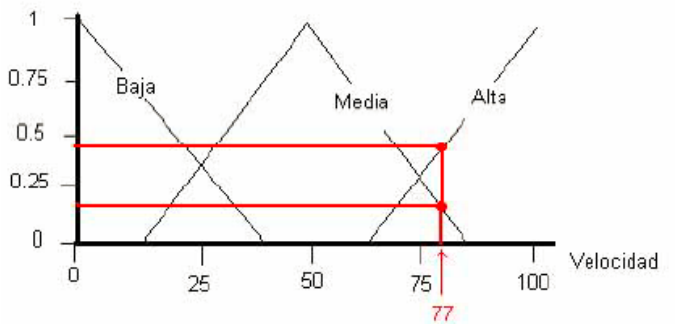

Figura 4.12 Ejemplo de fusificación de una variable

Para comprender mejor véase la ***Figura 4.12***. Analizando los siguientes datos: µAlta (77)=0.45, µMedia (77)=0.2, µBaja (77)=0. El valor de velocidad igual a 77 pertenece a dos conjuntos con distintos grados en cada uno.

A partir de ahora y durante el resto de las operaciones difusas, estos datos (0.45, 0.20 y 0, son valores de las funciones de membresía) representarán a las variables sensadas del proceso. A tales datos se les llamará **µ** en sentido genérico para diferenciarlos de otras funciones de membresía.

4.3.1.1 Reglas difusas de Mandani

"IF x1 es A AND x2 es B y x3 es C THEN µ1 es D, µ2 es E", donde x1, x2 y x3 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 (alto, medio, bajo), µ1 y µ2 son las acciones de control (aumento de corriente) en sentido genérico son todavía variables lingüísticas sin valores numéricos asociados, 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. La primera parte de la sentencia "IF x1 es A AND x2 es B y x3 es C" es el antecedente y la restante es el consecuente.

Un ejemplo es:
IF error es Positivo Grande AND derivada del error es Positiva Baja THEN µ es Positiva Chica.

4.3.1.2 Funciones de membresía

Las funciones de membresía representan el grado de pertenencia de un elemento a un subconjunto definido por una etiqueta. Las formas más comunes son del tipo trapezoidal, triangular y singleton.

$$A(x) = \begin{cases} 0 & si\ \ (x \leq a) o (x \geq d) \\ \dfrac{(x-a)}{(b-a)} & si\ \ x \in (a, b] \\ 1 & si\ x\ \in (b, c) \\ \dfrac{(d-x)}{(d-c)} & si\ \ x \in (b, d) \end{cases}$$

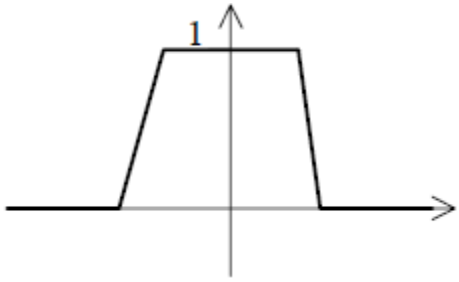

Figura 4.13 Función de membresía trapezoidal

$$A(x) = \begin{cases} 0 & si\ \ x \leq a \\ \dfrac{(x-a)}{(m-a)} & si\ x \in (a, m] \\ \dfrac{(b-x)}{(b-m)} & si\ x\ \in (m, b) \\ 1 & si\ \ x \geq b \end{cases}$$

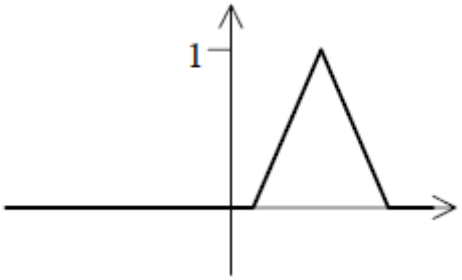

Figura 4.14 Función de membresía triangular

$$A(x) = \begin{cases} 1, & x = a \\ 0, & x \neq a \end{cases}$$

Figura 4.15 Función de membresía singleton

4.3.2 Reglas base

Los controladores difusos usan reglas, estas combinan uno o más conjuntos difusos de entrada llamados antecedentes o premisas y le asocian un conjunto difuso de salida llamado consecuente o consecuencia. A estas reglas se les llama reglas difusas. Son afirmaciones del tipo "IF-THEN". En los conjuntos difusos del antecedente se asocian mediante operaciones lógicas difusas AND, OR. [23]

Las reglas difusas 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 reglas base, es decir, la edición de esta base determina cuál será el comportamiento del controlador difuso y es aquí donde se emula el conocimiento o experiencia de la persona involucrada con el proceso y la correspondiente estrategia de control.

Las reglas base suelen representarse por tablas. Esta es clara en el caso de 2 variables de entrada y una de salida. En la medida que la cantidad de variables lingüísticas crece, también lo hará la tabla, y más difícil se hará su edición. Junto a cada regla puede estar asociado un valor entre cero y uno, esta distribución es importante cuando una regla tiene menor fuerza que otras de la base de reglas. [22][23]

Los dos grupos más importantes de tipo de reglas son: las reglas difusas de Mandani y las reglas difusas de Takagi-Sugeno [21]. El método seleccionado para dar solución al problema de control de Torque de un motor DC, son las reglas difusas de Mandani.

4.3.3 Mecanismo de inferencia

Las reglas difusas representan el conocimiento y la estrategia de control, pero cuando se asigna información específica a las variables de entrada en el antecedente, la inferencia difusa es necesaria para calcular el resultado de las variables de salida del consecuente. Este resultado es en términos difusos, es decir se obtiene un conjunto difuso de salida para cada regla, que posteriormente, junto con las demás salidas de reglas, se obtendrá la salida del sistema.

Existe una gran cantidad de métodos de inferencia difusa, pero hay cuatro que generan mejores resultados en el campo del control, estos son los enunciados en la ***Tabla 4.7***. Donde μ_W es la función de pertenencia del conjunto de salida w.

Método de inferencia	Definición
Inferencia de Mamdani por mínimos, R_M	$\min(\mu, \mu_W(z)), \forall z$
Inferencia del producto de Larsen, R_L	$\mu \times \mu_W(z), \forall z$
Inferencia del producto drástico, R_{DP}	$\begin{cases} \mu & \mid para\ \mu_W(z) = 1 \\ \mu_W(z) & \mid para\ \mu = 1 \\ 0 & \mid para\ \mu < 1\ y\ \mu_W(z) < 1 \end{cases}$
Inferencia del producto limitado, R_{BP}	$\max(\mu + \mu_W(z) - 1, 0)$

Tabla 4.7 Mecanismos de inferencia

4.3.4 Agregado

Cuando se evalúan las reglas se obtienen tantos conjuntos difusos como reglas existan. Para desfusificar, es necesario agrupar estos conjuntos, a esta etapa llamada agregado y existen varios criterios para realizar este paso. El criterio más empleado es el de agrupar los conjuntos inferidos mediante operación MAX.

4.3.5 Interface de desfusificación

La desfusificación 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 en términos difusos. Esta conclusión o salida difusa es obtenida por la etapa de inferencia difusa, la cual genera un conjunto difuso cuyo dato de salida 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 desfusificación y arrojan resultados distintos. El más común y ampliamente usado es el centroide, **Ecuación (4.27)**. Con este método se transforma la salida difusa en un número real el cual es la coordenada x del centro de gravedad de tal conjunto difuso de salida.

$$y_d = \frac{\int_S y \mu_Y(y) dy}{\int_S \mu_Y(y) dy} \quad (4.27)$$

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.

Uno de los desfusificadores más usados es el centro de área (COA, center of area) también llamado de altura, **Ecuación (4.28)**. El centro de gravedad es aproximado por el centro de gravedad de un arreglo de masas puntuales, las cuales son el centro de gravedad de cada conjunto de salida correspondiente a cada regla, con masa igual al grado de pertenencia en ese punto de su centro de gravedad. Si se le llama δi al centro de gravedad del conjunto difuso de salida Bi de la i-ésima regla, con R como número de reglas, el centro de gravedad queda determinado por:

$$y_d = \frac{\sum_{i=1}^{R} \delta_i \mu_{B_i(\delta_i)}}{\sum_{i=1}^{R} \mu_{B_i(\delta_i)}} \quad (4.28)$$

Tanto la Desfusificación como la Fusificación son el nexo del sistema difuso con el mundo real.

Figura 4.16 Desfusificación con el método centroide.

4.3.6 Implementación de un controlador difuso en SIMULINK/MATLAB

Para llevar a cabo la implementación del controlador difuso, tipo Mandani, se hace uso del toolbox de MATLAB Fuzzy Logic Designer [25]; este toolbox permite introducir las reglas de control, funciones de membresía, métodos de inferencia, fusificación y desfusificación [21] [22] [23] [24]. El código del controlador se encuentra en ANEXOS.

Para la implementación del controlador difuso se tomaron dos entradas llamadas ERROR y CAMBIO. El error se dividió en 5 grupos de la siguiente manera:

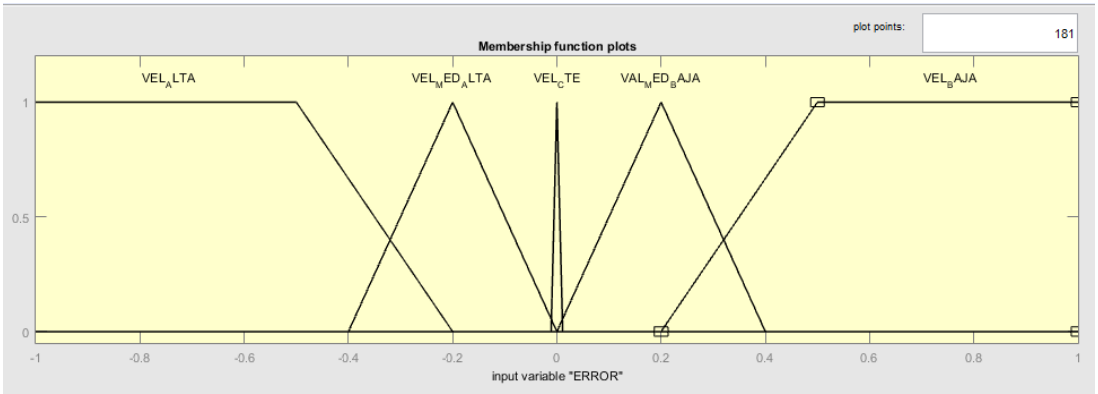

Figura 4.17 Función de membresía ERROR.

- Vel_Alta: Función Trapezoidal.
- Vel_Med_Alta: Función Triangular.
- Vel_Cte: Función Triangular.
- Vel_Med_Baja: Función Triangular.
- Vel_Baja: Función Trapezoidal.

*Ver subcapítulo 4.3.1.2 Funciones de membresía.

En la ***Figura 4.17*** se puede observar el esquema del error cuyo rango oscila entre -1 a 1, presentando estos valores en por unidad. Quiere decir que si la velocidad de referencia es 500 rpm, el valor de la retroalimentación es 600 rpm y el valor base es 1500 rpm, el error presentando sería:

$$Error = \frac{V_{ref} - V_{salida}}{V_{Base}} \quad (4.29)$$

$$Error = \frac{500 - 600}{1500} = -0.067$$

Según la ***Figura 4.16***, el resultado de la **Ecuación (4.29)**, el error estaría comprendido en la categoría Vel_Cte y Vel_Med_Baja. Esta etapa se conoce como fusificación e inferencia.

La derivada del error se dividió en dos grupos, ver ***Figura 4.18*** y representan los cambios súbitos, traducidos como cambio de error, que, cuando el motor gira a velocidad de referencia constante, puede presentar. Su rango oscila entre -1 y 1, presentando estos valores en por unidad.

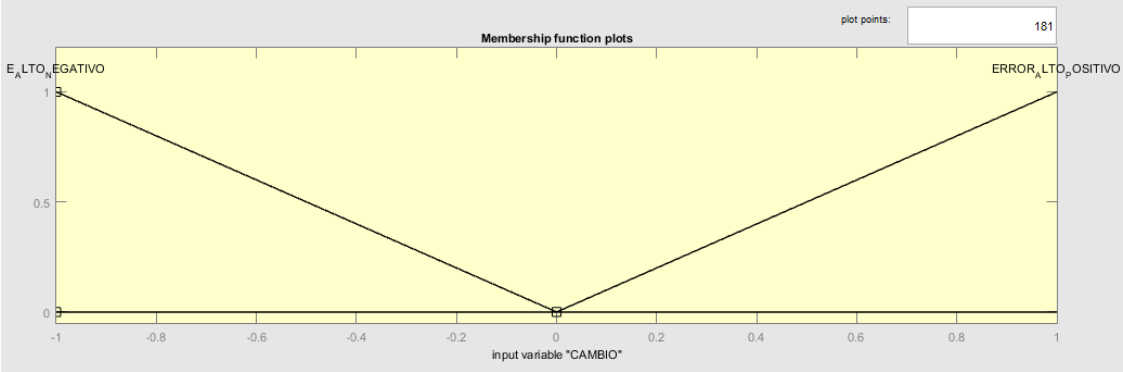

Figura 4.18 Función de membresía CAMBIO.

- E_Alto_Negativo: Función Triangular.
- E_Alto_Positivo: Función Triangular.

*Ver subcapítulo 4.3.1.2 Funciones de membresía.

La respuesta, o salida, del controlador difuso se denomina CONTROL. En la ***Figura 4.19*** se puede apreciar la distribución del universo de discurso para la acción de control, cuyo rango oscila entre 0 y 24 Volts:

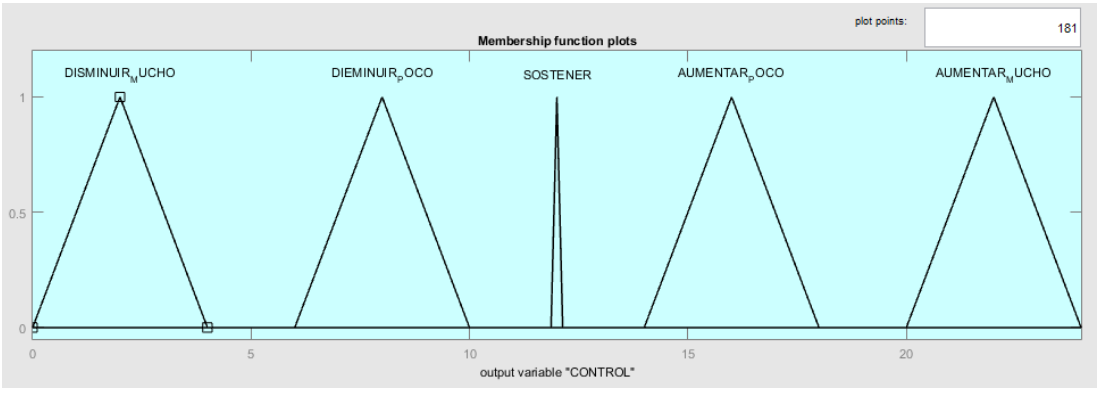

Figura 4.19 Función de membresía CONTROL.

- Disminuir_Mucho: Función Triangular.
- Disminuir_Poco: Función Triangular.
- Sostener: Función Triangular.
- Aumentar_Poco: Función Triangular.
- Aumentar_Mucho: Función Triangular.

*Ver subcapítulo 4.3.1.2 Funciones de membresía.

Ahora bien, resulta indispensable asignar un conjunto de reglas bases para que exista coherencia en la respuesta. En la ***Figura 4.21*** se muestra la acción de control a realizar dependiendo del valor recibido, por el controlador, como entrada. Como ejemplo se presenta que para un ERROR igual a 0 (Vel_Cte), CAMBIO igual a 0, la acción CONTROL deberá ser 12V (Sostener). Esta última etapa se conoce como desfusificación y MATLAB/SIMULINK emplea el método del centroide, analizado en el subcapítulo 4.3.5. Para fines prácticos, no se construyó una matriz de control como se recomienda en [22] [21], ya que el objetivo de este proyecto piloto es implementar el resultado de estas simulaciones en un sistema embebido y, demasiadas reglas de control traducen en un gasto computacional bastante elevado. La ***Figura 4.22*** enseña la superficie característica del controlador difuso y todas las posibles combinaciones que el sistema puede ofrecer.

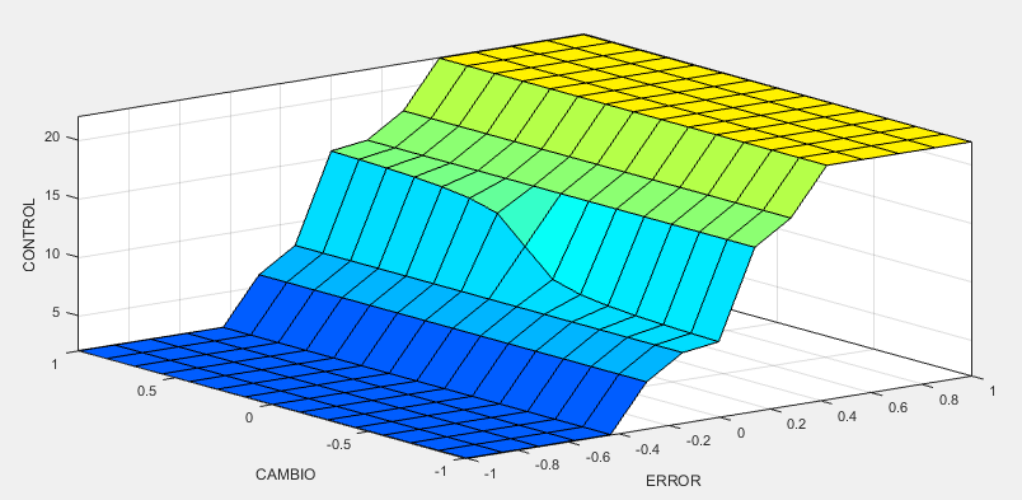

Figura 4.20 Superficie de control.

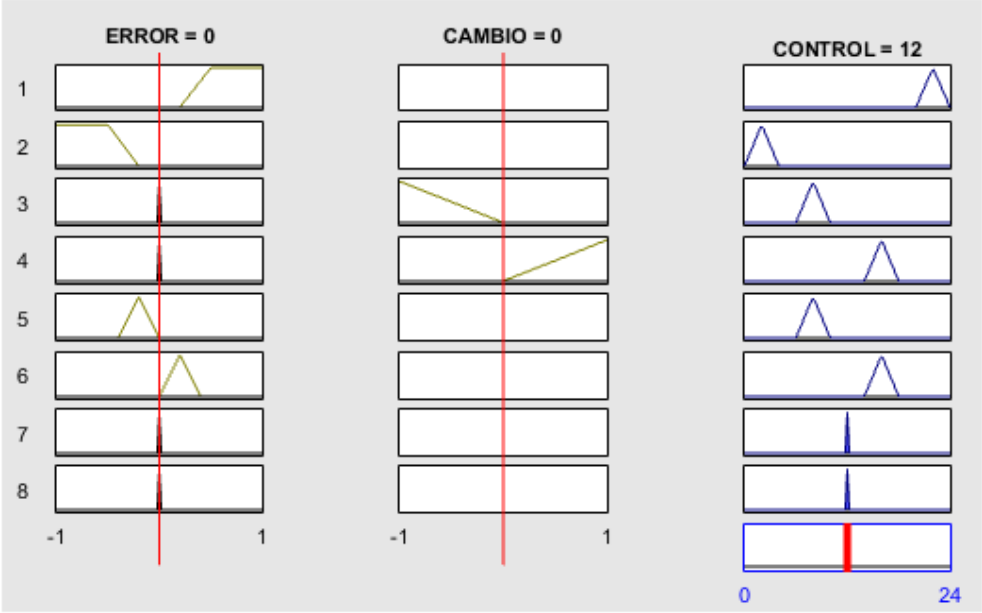

Figura 4.21 Reglas de control.

Finalmente, en la **Figura 4.22 y Figura 4.23** se pueden observar el diagrama de bloques propio del sistema de control difuso y la respuesta del sistema ante una perturbación. El bloque DC motor es el equivalente al motor DC descrito en el capítulo 4.1, cuyos valores de resistencia de armadura, inductancia de armadura, momento de inercia y parámetros intrínsecos fueron hallados por medio de pruebas de laboratorio y se pueden ver en la **Tabla 4.5**. Adicional a los parámetros para el modelamiento del motor, el bloque tiene una entrada de Torque para simular variaciones de este en tiempos distintos durante la simulación.

En efecto la **Figura 4.22** enseña la respuesta del sistema a un aumento considerable de torque a los 7 segundos de iniciada la simulación; y cómo la salida, a pesar de los transitorios, intenta seguir el valor de referencia.

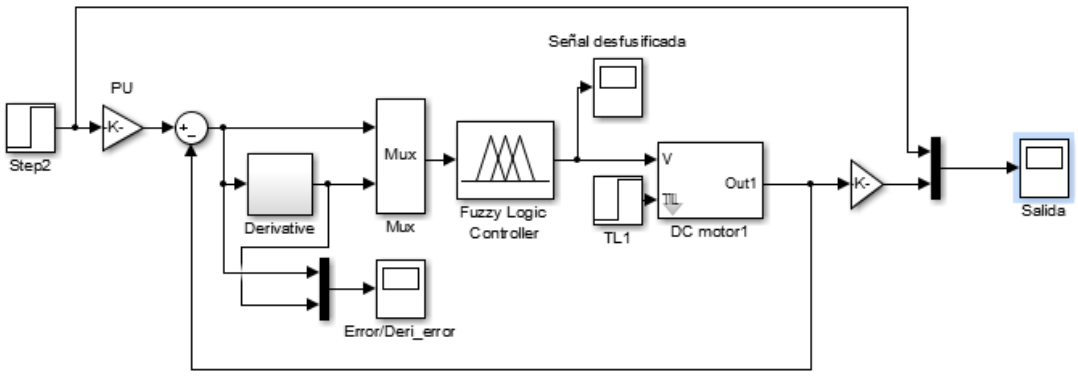

Figura 4.22 Diagrama de bloques controlador difuso.

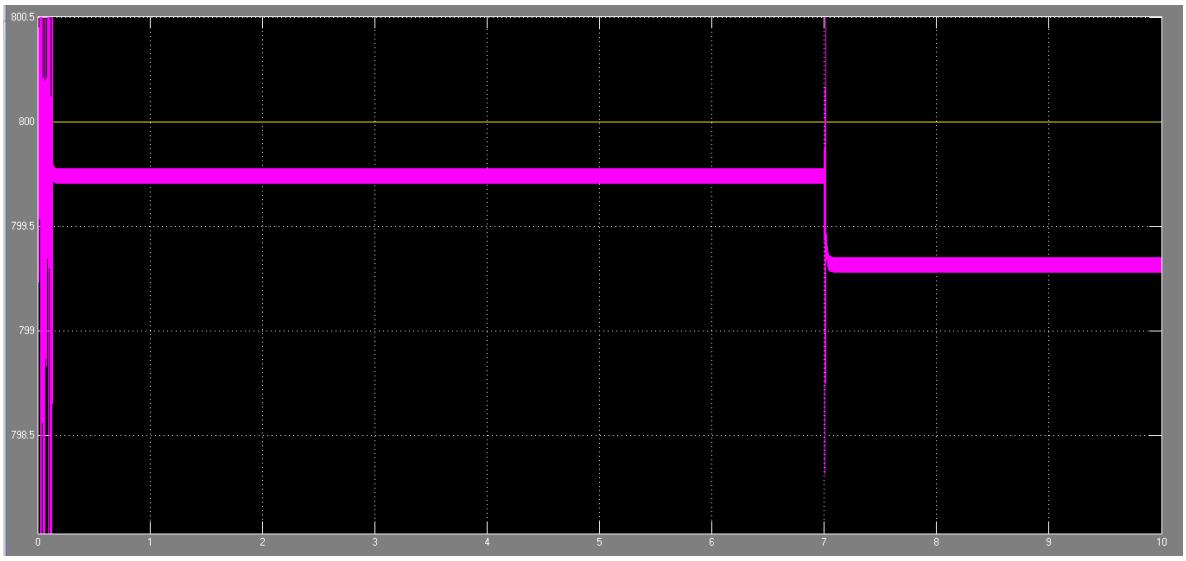

Figura 4.23 Grafica de señal de salida y referencia del controlador difuso.

4.4 Pruebas

Se realizaron pruebas, variando el torque y la velocidad de referencia, bajo los siguientes parámetros: torque constante y velocidad variable; torque variable y velocidad constante; torque y velocidad variables. Se hizo uso del modelamiento del motor DC y el esquema de control, presentados en la **Figura 4.22.** En las **Tabla 4.8**(resultados prueba control difuso) **y Tabla 4.9** (resultados pruebas control PID) se presentan 7 casos, así:

		Torque [Nm]	Vel. Refer [rpm]	Vel. Salida [rpm]	Corriente [A]	Tiempo [mS]
Caso 1	**Inicio**	1.35	100	100.10 - 100.04	10.03	7
	Fin	1.35	1500	1499.64 - 1499.57	10.03	
Caso 2	**Inicio**	2.1	100	99.78 - 99.71	13.72	8
	Fin	0	1500	1499.84 - 1499.77	2.12	
Caso 3	**Inicio**	0	1500	1499.84 - 1499.76	2.12	7.5
	Fin	0	100	99.93 - 99.96	2.12	
Caso 4	**Inicio**	0	1500	1499.84 - 1499.76	2.12	8
	Fin	2.1	100	99.62 - 99.54	13.72	
Caso 5	**Inicio**	0	1500	1499.84 - 1499.76	2.12	7
	Fin	1.35	1500	1499.64 - 1499.57	10.03	
Caso 6	**Inicio**	0	1000	999.76 - 999.69	2.12	7
	Fin	1.35	1000	999.54 - 999.49	10.03	
Caso 7	**Inicio**	0	500	499.86 - 499.78	2.12	7
	Fin	2.1	500	499.41 – 499.64	13.72	

Tabla 4.8 Variación de velocidad de referencia y torque aplicado al motor empleando lógica difusa.

		Torque [Nm]	Vel. Refer [rpm]	Vel. Salida [rpm]	Corriente [A]	Tiempo [S]
Caso 1	**Inicio**	1.35	100	99.85	10.03	2.30
	Fin	1.35	1500	1500.5	10.03	
Caso 2	**Inicio**	2.1	100	99.85	13.72	2.30
	Fin	0	1500	1500.5	2.12	
Caso 3	**Inicio**	0	1500	1500.6	2.12	2.30
	Fin	0	100	99.5	2.12	
Caso 4	**Inicio**	0	1500	1501	2.12	2.30
	Fin	2.1	100	98	13.72	
Caso 5	**Inicio**	0	1500	1501.6	2.12	2.30
	Fin	1.35	1500	1499.4	10.03	
Caso 6	**Inicio**	0	1000	1002	2.12	2.30
	Fin	1.35	1000	99.5	10.03	
Caso 7	**Inicio**	0	500	500.8	2.12	2.30
	Fin	2.1	500	499.4	13.72	

Tabla 4.9 Variación de velocidad de referencia y torque aplicado al motor empleando PID.

Caso 1. El torque se mantuvo constante y la velocidad tuvo un incremento súbito, de 100 a 1500 rpm, a los 5 segundos de iniciada la simulación. Este caso simula una pendiente positiva con permanencia de carga.

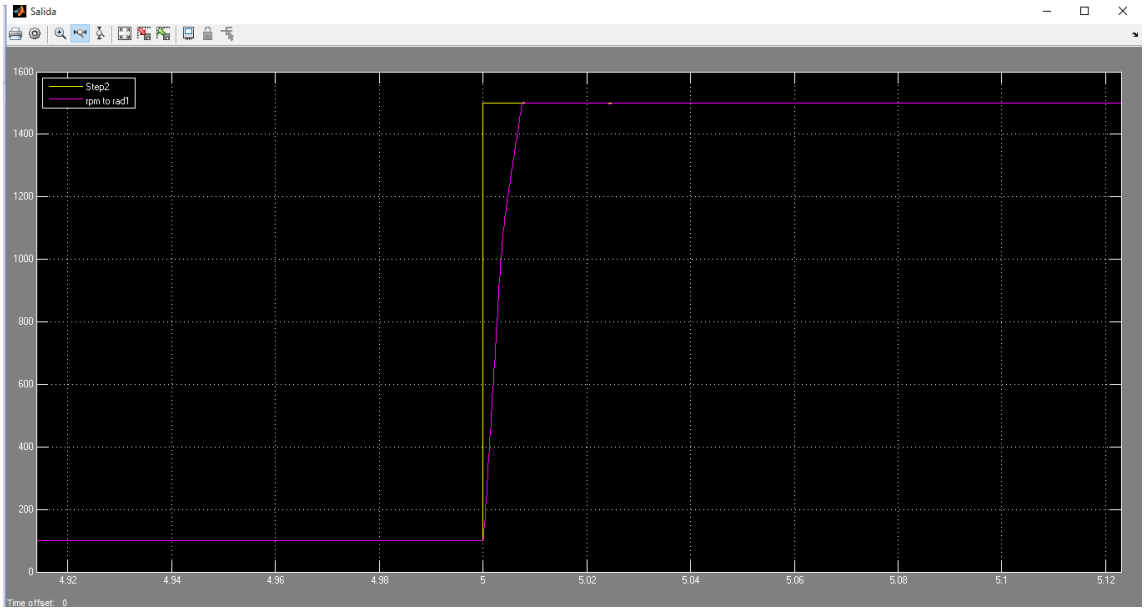

Figura 4.24 Caso 1 de la Tabla 4.6

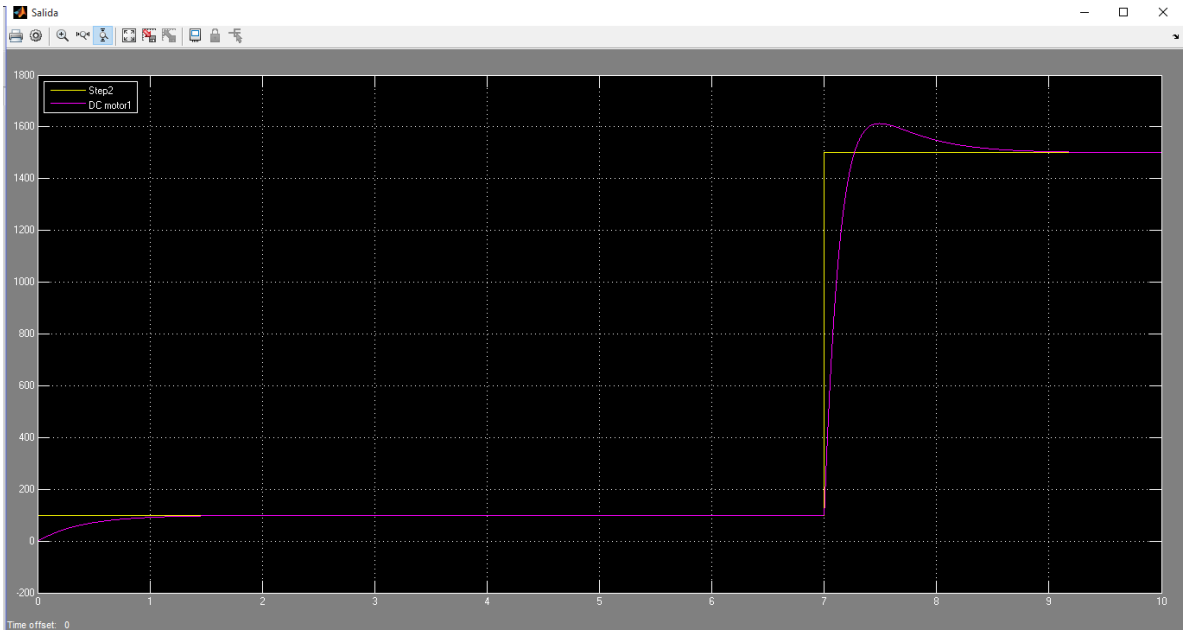

Figura 4.25 Caso 1 de la tabla 4.7

Caso 2. Tanto el torque como la velocidad varían. Inicia con un torque de 2.10 Nm y una velocidad de 100 rpm y finaliza con un torque de 0 Nm (vacío) y una velocidad de 1500 rpm. Este caso simula la respuesta del motor ante una pendiente positiva pronunciada y una liberación de carga al final de la misma, en un ambiente plano.

Caso 3. Existe una variación de velocidad manteniendo el torque constante. Este caso simula la respuesta del sistema, traducido en ciclo de trabajo, cuando se presenta un esfuerzo en el rotor.

Caso 4. Tanto el torque como la velocidad varían. Inicia con un torque de 0 Nm (vacío) y una velocidad de 1500 rpm, y finaliza con un torque de 2.10 Nm y una velocidad de 100rpm. Este caso simula la respuesta del motor ante una trayectoria plana seguida de una pendiente positiva muy pronunciada.

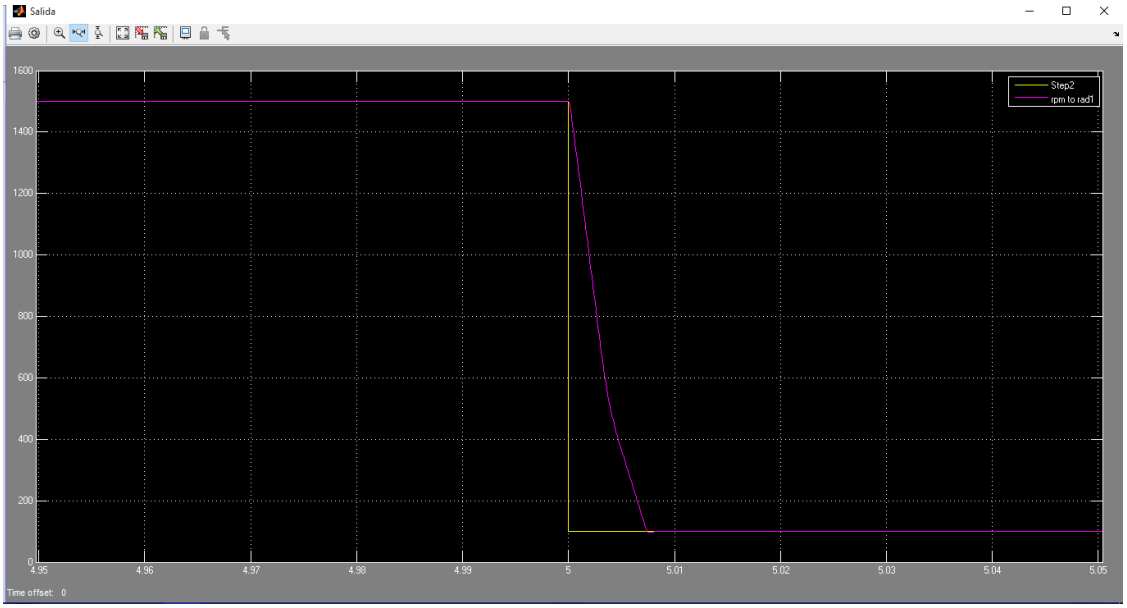

Figura 4.26 Caso 4 de la Tabla 4.6

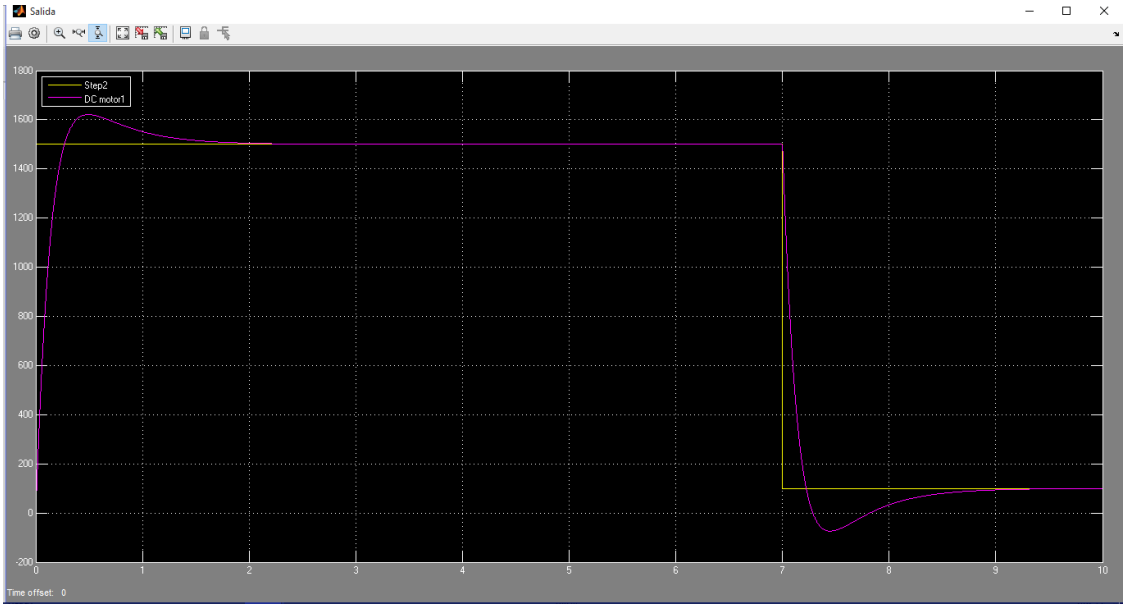

Figura 4.27 Caso 4 de la tabla 4.7

Caso 5. Existe una variación del torque manteniendo la velocidad constante. Este caso simula la respuesta del sistema en vacío ante un aumento de esfuerzo permaneciendo el sistema estable. Se aprecia en las

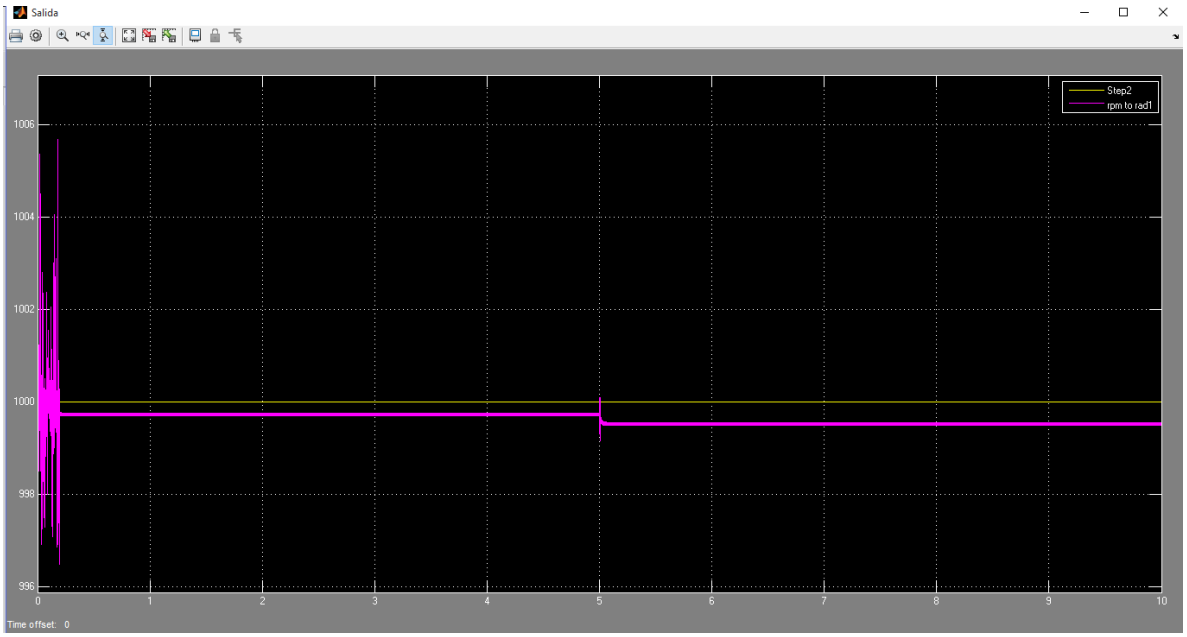

Figura *4.28* y **Figura 4.29** que ante una variación de carga, la velocidad se aproxima a la velocidad de referencia.

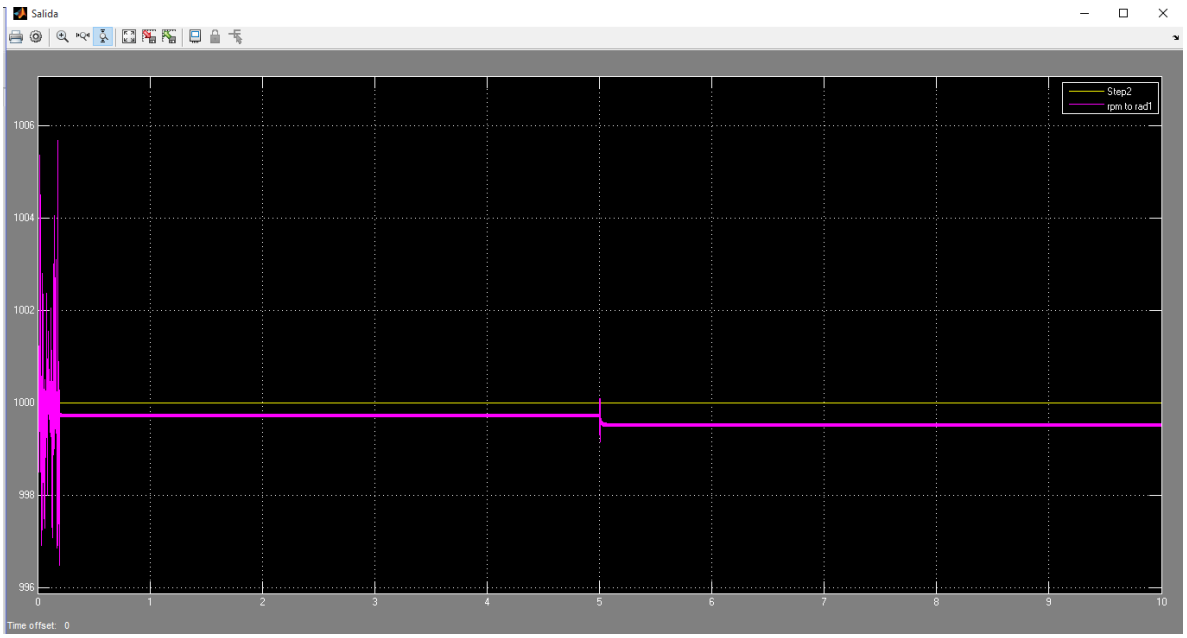

Figura 4.28 Caso 5 de la Tabla 4.6

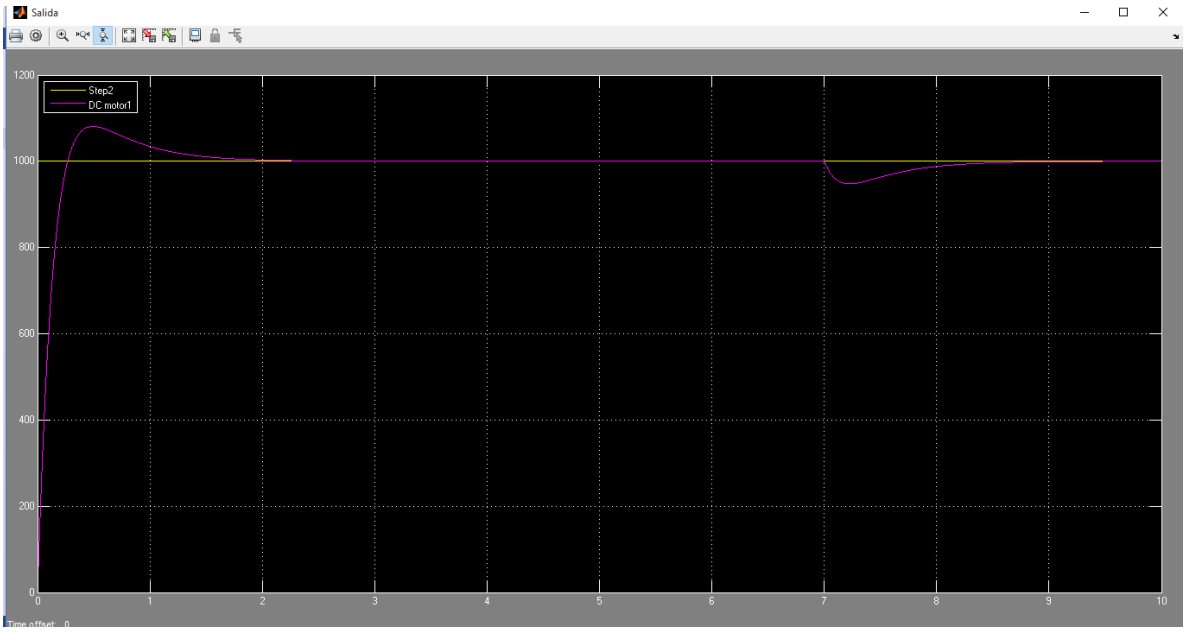

Figura 4.29 Caso 5 de la Tabla 4.7

Caso 6. Existe una variación del torque manteniendo la velocidad constante en 1000 rpm. Este caso simula la respuesta del sistema, bajo carga, ante un aumento de esfuerzo permaneciendo el sistema estable. La velocidad de salida se aproxima muy bien a la velocidad de referencia, como se observa en las **Tabla 4.8** y **Tabla 4.9**

Caso 7. Existe una variación del torque manteniendo la velocidad constante en 500 rpm. Este caso simula la respuesta del sistema, bajo carga considerable, ante un aumento de esfuerzo permaneciendo el sistema estable. La velocidad de salida se aproxima muy bien a la velocidad de referencia, como se observa en las **Tabla 4.8** y **Tabla 4.9.**

4.4.1 Análisis simulaciones

Basados en los resultados obtenidos en las **Tabla 4.8** y **Tabla 4.9** se puede concluir que el controlador difuso responde más rápido que el controlador PID. El primero tiene una respuesta en milisegundos, mientras que el controlador PID tiene una respuesta en segundos. Además, en cuanto al análisis de transitorios en las señales de respuesta de ambos controladores, se puede observar en la **Figura 4.26** que en el controlador difuso es casi

inexistente, mientras que en la **Figura 4.27** el controlador PID genera un transitorio más prolongado.

Ahora bien, el controlador difuso presenta interferencia en la

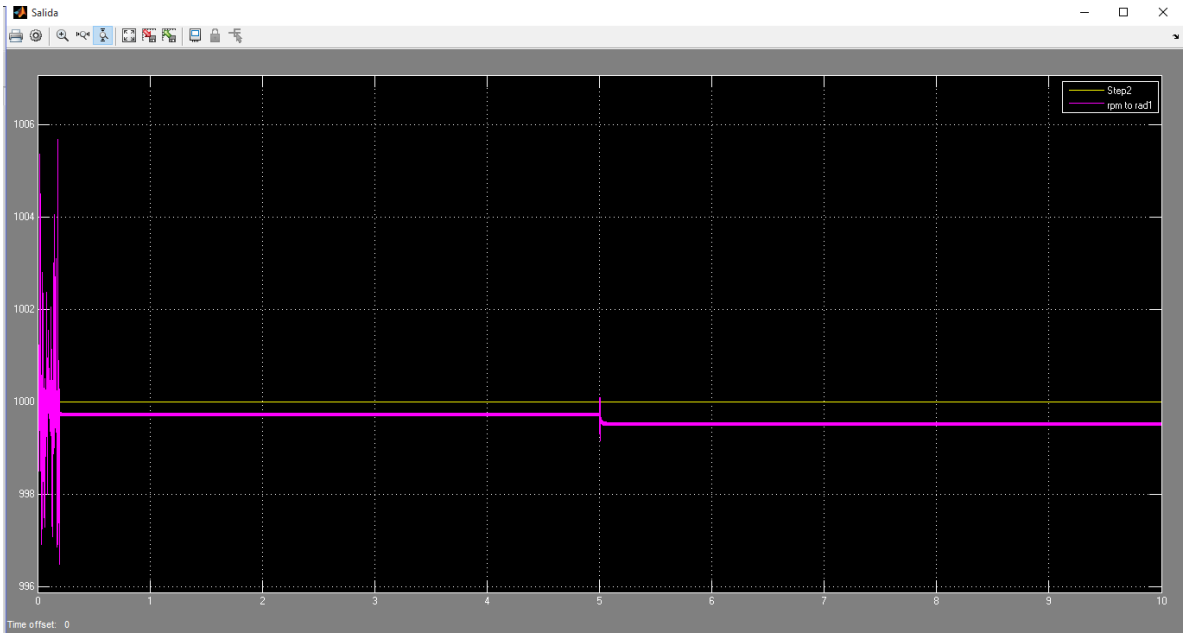

Figura *4.28* caso en el cual se simula la respuesta del sistema en vacío ante un aumento de esfuerzo, permaneciendo el sistema invariante en velocidad. Mientras que en la **Figura 4.29** se aprecia que la respuesta del controlador PID es mucho más limpia y suave. Se plantea la posibilidad de que la interferencia sea mitigada por los elementos circuitales utilizados en el prototipo.

5. PROGRAMACIÓN DE ALGORITMOS DE CONTROL EN LENGUAJE PYTHON PARA RASPBERRY PI

5.1 Lenguaje de programación Python

Python es un poderoso lenguaje de programación computacional multiparadigma, optimizado para la productividad del programador, legibilidad del código y calidad de software, por su naturaleza libre, es usado para programación autónoma y aplicaciones de script en una amplia variedad de dominios, además permite una calidad de software que proporciona una ventaja estratégica en proyectos largos y cortos [26].

Se enfoca en la legibilidad, coherencia y calidad de software, un código Python es diseñado para ser legible y además sea reusable y mantenible. La uniformidad de un código Python lo hace fácil de comprender además tiene un soporte profundo para uso de software más avanzados, como la programación orientada a objetos y la programación funcional; mejora la productividad del desarrollador comparado con lenguajes como C, C++, y Java, un código Python es típicamente un tercio o un quinto de su equivalente en un código C++ o Java, esto es traducido a que como hay menos que digitar hay menos que corregir, revisar o mejorar, programas Python corren inmediatamente sin la compilación lenta de otras herramientas, mejorando la velocidad del programa [26].

5.1.1 Ventajas de Python

Es Orientado a objetos y funcional; Python es un lenguaje de programación orientado a objetos (OOP), desde el principio, sus modelos de clase soportan nociones avanzadas como lo son polimorfismo, sobrecarga del operador y múltiple herencia; aun, en el contexto de un Python de sintaxis y digitación simples, OOP es considerablemente fácil de aplicar. De hecho, la programación orientada a objetos es más fácil de aplicar en Python que en cualquier otro lenguaje OOP disponible. Aparte de servir como un poderoso código estructural y dispositivo reusable, la naturaleza OOP de Python lo hace ideal como una herramienta de scripting para otros sistemas de lenguajes orientado a objetos. Por ejemplo, con un código de

unión apropiado, programas Python pueden hacer subclases implementadas en C++, Java y C. De igual significancia, OOP es sólo una opción en Python, como C++, Python soporta tanto el modo de programación por procedimientos como el orientado a objetos. Su herramienta orientada a objetos puede ser aplicada siempre y cuando las restricciones lo permitan; esto es especialmente útil en modos de desarrollo tácticos los cueles impiden el diseño de fases. En adición a sus originales paradigmas, modo de procedimientos (basado en declaraciones) y su modo orientado a objetos (basado en clases), Python recientemente ha adquirido un soporte incorporado para programación funcional que puede servir como complemento o alternativa a su herramienta OOP [26].

Es poderoso; desde una perspectiva de las características, Python es una especie de híbrido, su conjunto de herramientas lo coloca entre lenguajes de script tradicionales (como Tcl, Scheme, y Perl) y lenguajes de desarrollo de sistemas (como C, C ++ y Java). Python proporciona toda la sencillez y la facilidad de uso de un lenguaje de programación, junto con las herramientas más avanzadas de ingeniería de software que se encuentran típicamente en lenguajes compilados. A diferencia de otros lenguajes de script, esta combinación hace Python útil para los proyectos de desarrollo a gran escala [26].

Es combinable, Programas Python pueden ser fácilmente "unidos" a los componentes escritos en otros idiomas en una variedad de maneras. Por ejemplo, “C API” de Python permite a los programas en C llamar y ser llamados por programas Python de manera flexible. Eso significa que se puede agregar funcionalidad al sistema Python según sea necesario, y utilizar programas de Python en otros entornos o sistemas. Mezclar Python con bibliotecas codificadas en lenguajes como C o C ++, por ejemplo, lo hace una fácil de usar interfaz y herramientas de personalización. Como se mencionó anteriormente, esto también hace a Python bueno en prototipado rápido, sistemas pueden ser implementados en Python primero, para aprovechar su velocidad de desarrollo, y más tarde trasladarlo a C para la entrega, de una sola pieza a la vez, de acuerdo a las demandas de rendimiento [26].

5.2 Programación de algoritmo PID en Python

El código Python desarrollado para la implementación de la acción de control PID explicada en el capítulo 4.2, fue escrito en el compilador IDLE (Python GUI), probado en el sistema embebido Raspberry PI y se encuentra en el ANEXO 2.

5.3 Programación de algoritmo de lógica difusa en Python

El código Python desarrollado para la implementación de la acción de control de lógica difusa explicada en el capítulo 4.3, fue escrito en el compilador IDLE (Python GUI), probado en el sistema embebido Raspberry PI y se encuentra en el ANEXO 3.

6. IMPLEMENTACIÓN DEL PROTOTIPO DE CONTROL DE VELOCIDAD Y TORQUE DEL MOTOR DC UTILIZANDO RASPBERRY PI

6.1 Esquema de control de velocidad y torque

El modelo de control de velocidad y torque del motor DC puede ser representado por medio de un diagrama de bloques, exponiendo todos los componentes que integran el sistema:

Figura 6.1 Esquema de control

Para realizar el control del Motor DC por medio de Raspberry Pi es necesario el uso de componentes que permitan el reconocimiento por parte del motor, de la señal de salida emitida por Raspberry Pi (pulso PWM) y la lectura análoga de la corriente con su correspondiente acondicionamiento y conversión digital para que pueda ser leída por el sistema embebido.

En la **Figura 6.2** se aprecia la conexión de todo el prototipo. Se describe la conexión del conjunto de elementos que conforman el sistema: Motor DC, Puente H, Sensor de Corriente, Raspberry Pi, Convertidor Análogo/Digital. La descripción detallada de cada uno de los componentes electrónicos se verá a lo largo de este capítulo.

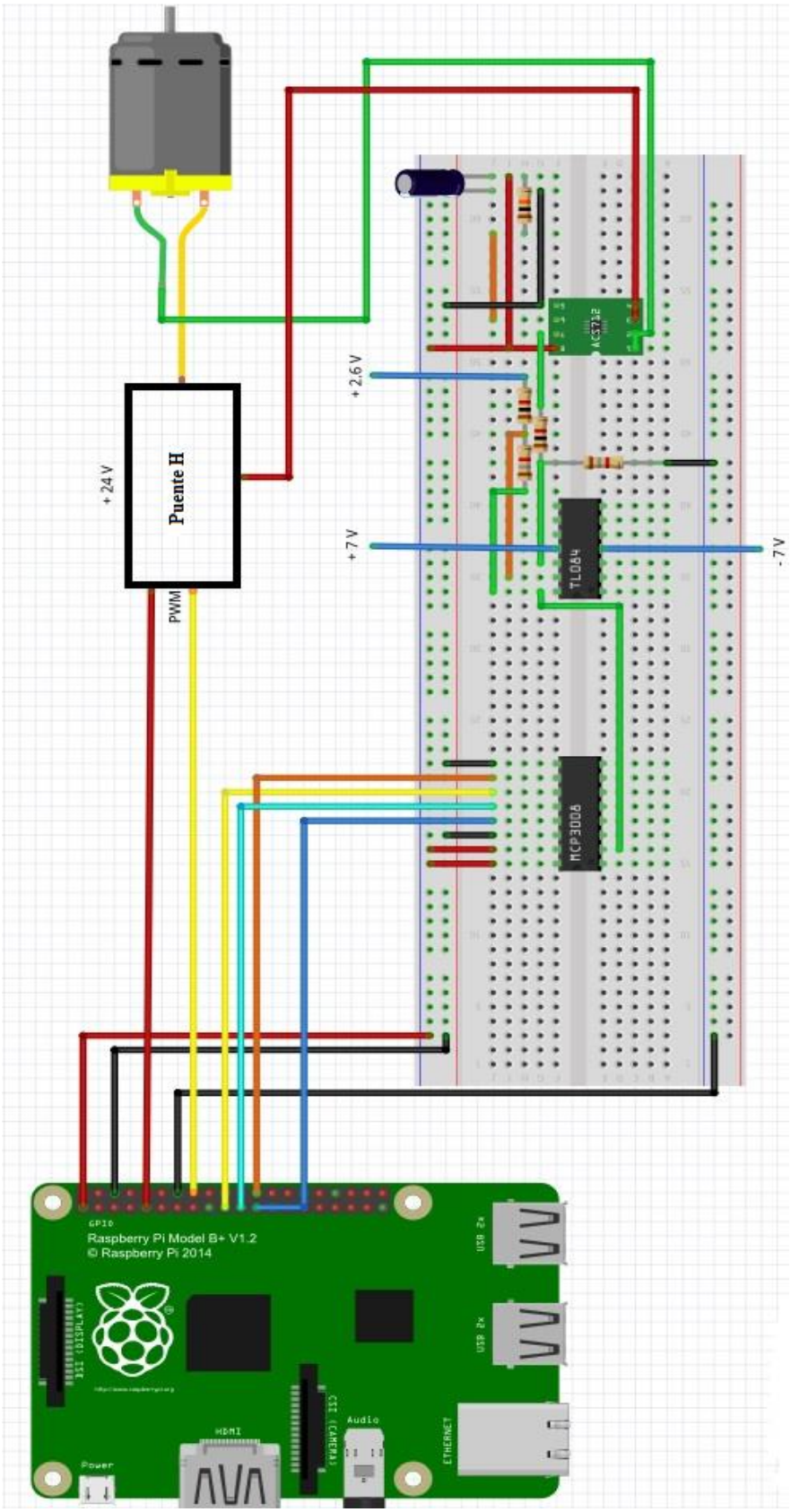

Figura 6.2 Conexión prototipo completo

6.1.1 Etapa de potencia

El puente H es un circuito típico utilizado para el control de motores. En la **Figura 6.2** se muestra una representación esquemática simplificada del circuito. Para que el motor gire, se activan dos de los transistores opuestos diagonalmente. En función del par de transistores activados, la corriente fluye en uno u otro sentido, lo que permite controlar el sentido de giro del motor [27].

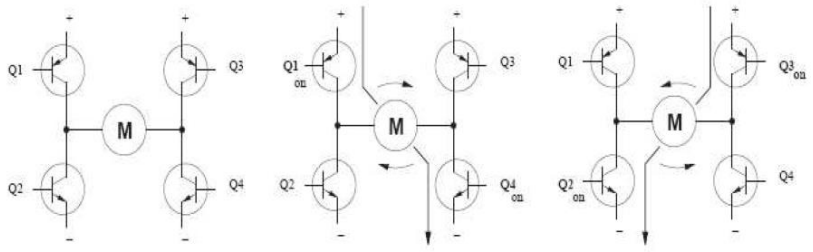

Figura 6.3 Representación Puente H

El sentido de giro del motor es controlado activando y desactivando pares de transistores diagonalmente opuestos. Así, la corriente fluye a través del motor por dos caminos distintos: desde Q1 a Q4 o desde Q3 a Q2. La corriente fluye a través del motor en un sentido u otro, traduciéndose en un giro horario o antihorario del motor [27].

La modulación por ancho de pulsos (pulse-width modulation o PWM) es una técnica que se basa en la modificación del ciclo de trabajo de una señal periódica (por ejemplo sinusoidal o cuadrada). El ciclo de trabajo de una señal periódica es el ancho relativo de su parte positiva en relación al período [27].

$$D = \frac{\tau}{T} \quad \textbf{(6.1)}$$

Donde:

D: Es el ciclo de trabajo.

τ: Es el ancho de pulso, esto es, el tiempo en que la función es positiva.

T: Es el período de la función.

En un motor de continua, la magnitud de entrada que determina la cantidad de par que va a aplicar es el valor medio de la tensión de la señal de entrada, esto es, el valor de tensión continua que tiene la entrada (esta afirmación sólo es cierta si se está en régimen permanente). El valor medio de una señal cualquiera $x(t)$ se define como [27]:

$$x(t) = \frac{1}{T}\int_0^T x(t)dt \quad \textbf{(6.2)}$$

El puente H usado en este proyecto, cuenta con los siguientes esquemáticos elaborados en el programa EAGLE, recepción de PWM, etapa de opto acoplamiento, amplificación e inversión **Figura 6.4,** y Puente H, **Figura 6.5.**

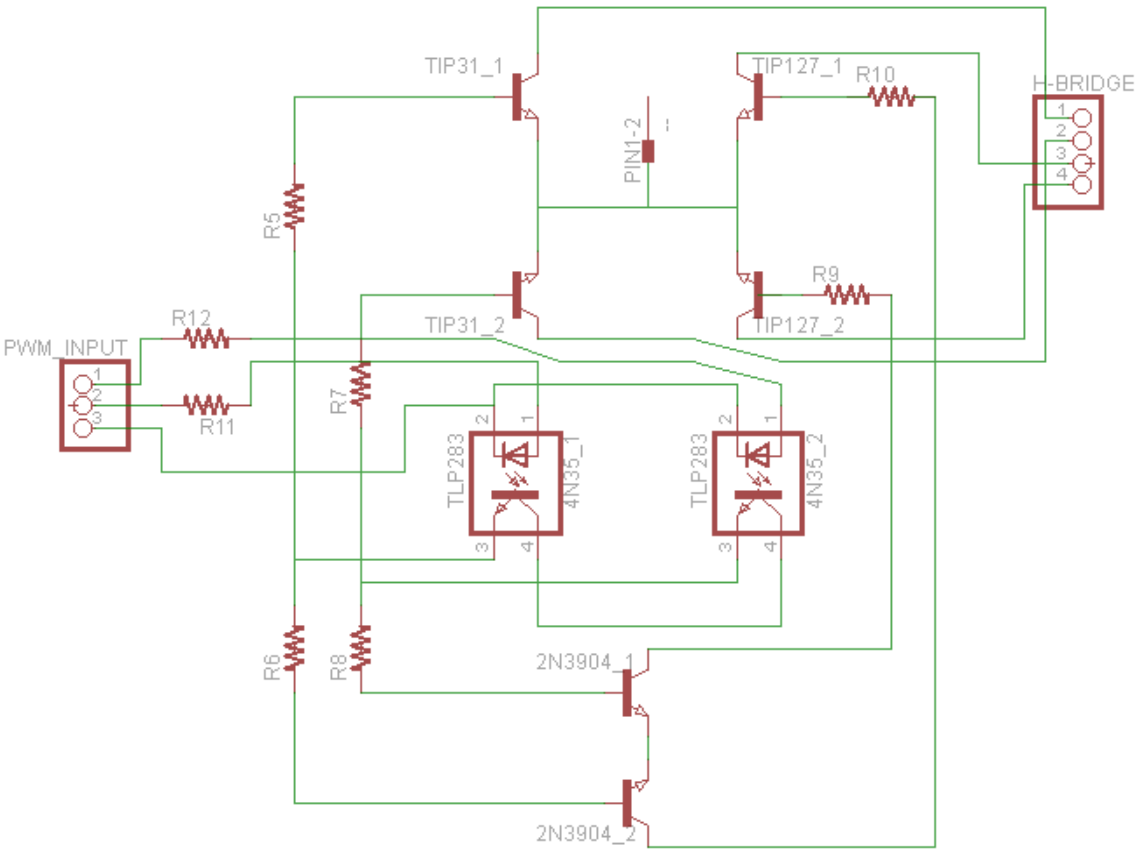

Figura 6.4 Esquema driver Puente H

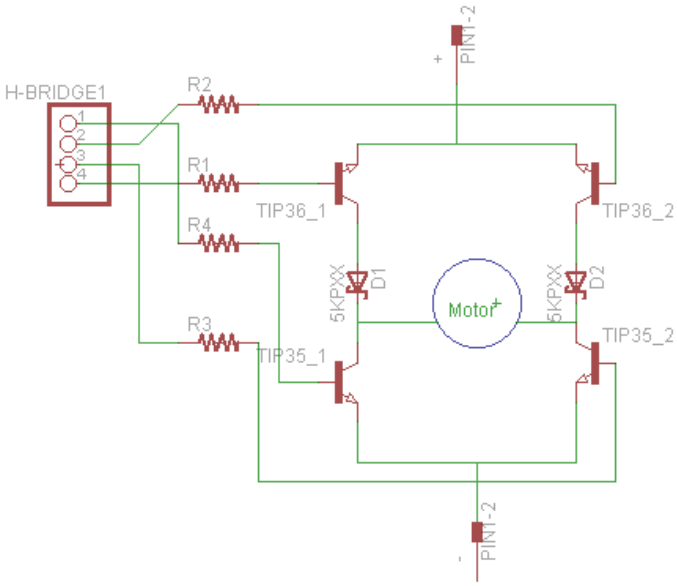

Figura 6.5 Esquema Puente H

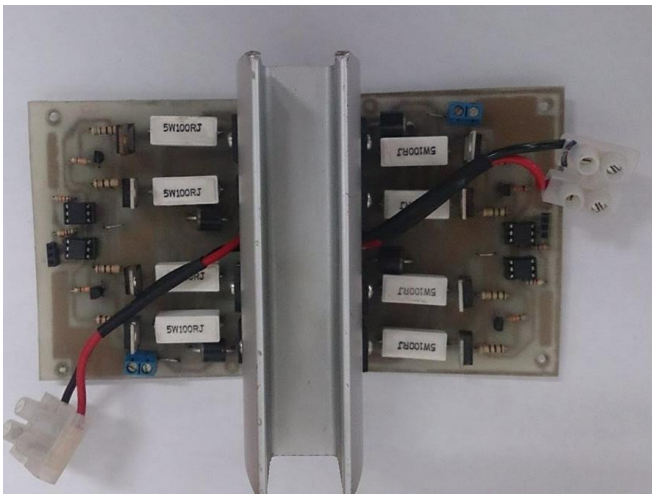

Figura 6.6 Etapa de potencia implementada

6.1.2 Sensor de corriente

El sensor usado en este proyecto es un sensor de corriente lineal basado en el efecto Hall ACS711KEXLT-15AB-T de Allegro, viene en un tablero de soporte o tablero de desbloqueo, con salida de fallo de sobrecorriente; este sensor tiene una tensión de 3 V a 5,5 V y una sensibilidad de salida de 90 mV / A cuando es Vcc 3,3 V (o 136 mV / A cuando Vcc es 5 V) [28].

El sensor requiere una fuente de alimentación de 3V a 5.5V conectada entre Los terminales Vcc y GND. La salida del sensor es una tensión análoga, linealmente proporcional a la corriente de entrada. La tensión de salida de reposo es Vcc/2 y cambia en relación a 90 mV por Amper de corriente de entrada (cuando Vcc = 3.3 V), la relación entre la corriente instantánea de entrada i, y la tensión de salida del sensor, VOUT, puede ser representada por la siguiente ecuación [28]:

$$Vout = \frac{Vcc}{2} + i * \frac{Vcc}{36.7} \quad \textbf{(6.3)}$$

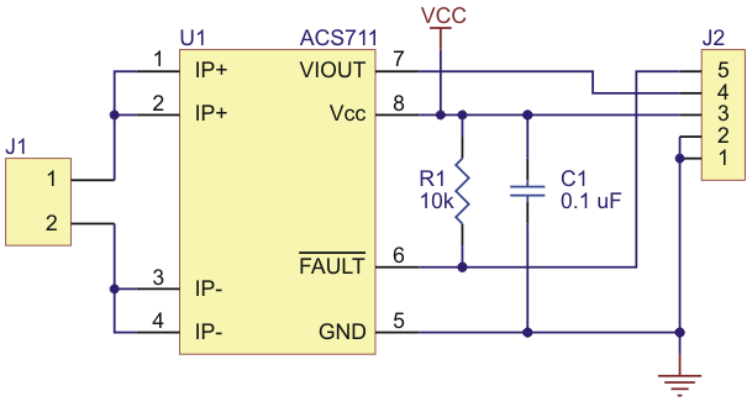

Figura 6.7 Esquema sensor de corriente

6.1.3 Convertidor analógico-digital MCP3008

El dispositivo Mcp3008 es un convertidor analógico digital de aproximaciones sucesivas de 10 bit, es programable para proporcionar cuatro pares de entradas pseudo-diferenciales u ocho entradas de terminación única. La comunicación con este dispositivo es lograda usando una simple interfaz serial compatible con el protocolo SPI (en este caso Raspberry Pi). Es capaz de alcanzar una tasa de conversión de hasta 200 KSPS, opera sobre un amplio rango de voltaje (2,7 V - 5,5 V), su diseño permite el funcionamiento con corrientes típicas de reserva de sólo 5 nA y corrientes activas típicas de 320 uA [29].

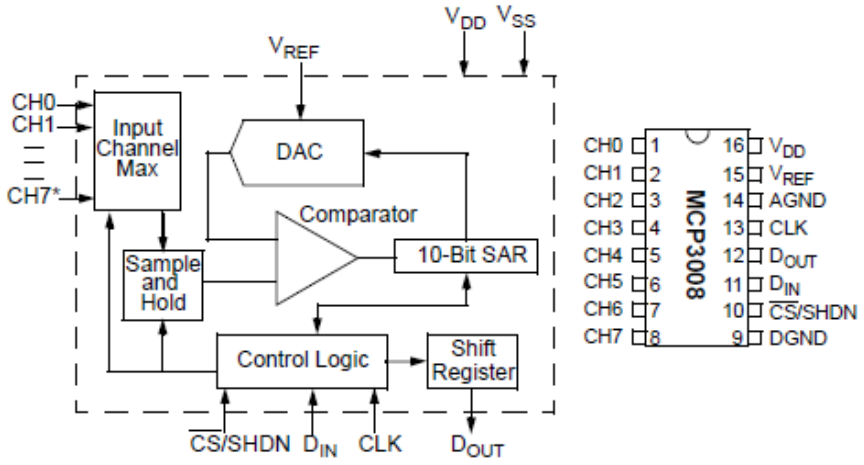

Figura 6.8 Esquema convertidor de señal analógico digital

6.1.4 Acondicionamiento de Señal

Para realizar una lectura adecuada de la corriente por medio del sensor, es necesario acondicionar la tensión de salida a los valores permitidos por el sistema embebido Raspberry Pi, los cuales se encuentran entre 0V y 3.3V, para ello, es necesario un circuito acondicionador de señal que adecue la tensión de salida.

El circuito acondicionador empleado es un "restador" que opera bajo la siguiente ecuación:

$$Vdig = 2.7\ (Vanal - 1.65) \quad (\mathbf{6.4})$$

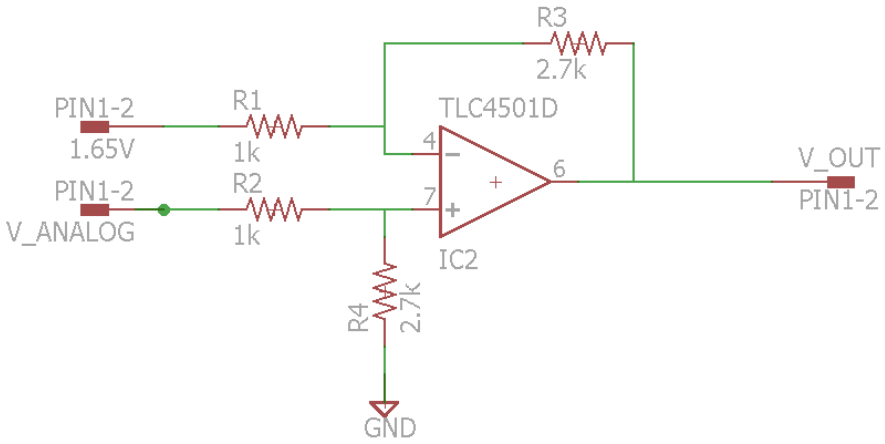

Figura 6.9 Circuito acondicionador de señal

7. RESULTADOS

Inicialmente, se hizo el montaje del prototipo, incluyendo el motor y su sistema de frenado, Raspberry Pi, puente h, sensor de corriente, acondicionamiento de señal y convertidor analógico/digital, **Figura 7.1**; los elementos de medición, tales como multímetro digital, amperímetro análogo, osciloscopio; y los diferentes voltajes de polarización necesarios para la correcta operación de los elementos electrónicos, **Figura 6.2.**

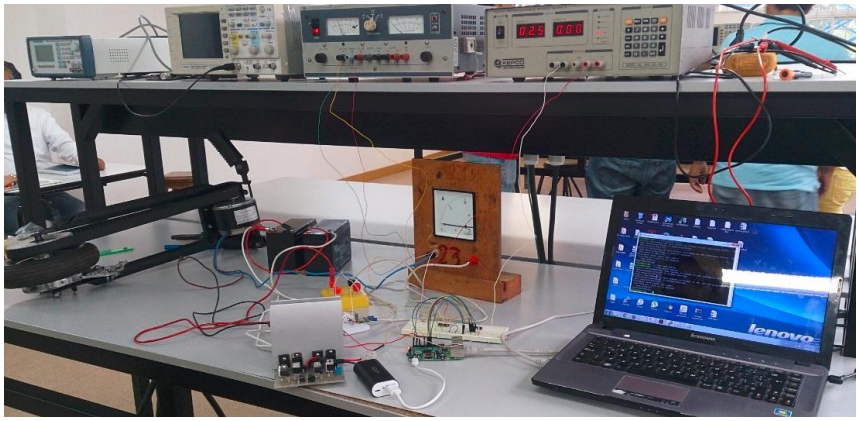

Figura 7.1 Implementación del prototipo de control de velocidad y torque a un motor DC

Después de comprobar continuidad de todo el sistema, inspeccionar las conexiones y comprobar la tensión entregada por las baterías, se procedió a efectuar las pruebas de los controladores con base a los resultados obtenidos en las simulaciones, **Tabla 4.8** y **Tabla 4.9**

Se percibió que la respuesta del controlador difuso no era constante aun operando el motor en vacío, **Figura 7.2.** Se hizo un recorrido por el circuito y se determinó que la salida del convertidor analógico/digital oscilaba alrededor de +/- 0.3V el valor de lectura. Esto se debe a qué el sensor de corriente entrega un voltaje analógico a su salida con bastantes perturbaciones. El sistema se acondiciona de tal forma que el convertidor analógico/digital reciba un voltaje entre 0 y 3.3 V, **Capitulo 6.1.4**, una variación de 0.3V representa el 9% del rango de operación. Para blindar los controladores de estas perturbaciones propias del circuito se procedió a instalar un capacitor de 0,1 µF a la entrada del convertidor y se reprogramaron los códigos de tal forma que la variable de salida del convertidor, la cual es

una posición entre 0 y 1023, **Capítulo 6.1.3,** no retorne un ciclo de trabajo distinto en la n-iteración del programa, siempre y cuando la variable medida no salga de un rango de +/- 50 posiciones. Ver **Anexos 2 y 3**.

```
pi@raspberrypi ~ $ sudo python fuzzy3.py
Cambio
Nivel 511
--------------------------{}--------------------------------
CicloTrabajo 8   Voltaje A/D 1.65   Error -0.5
Nivel 511
--------------------------{}--------------------------------
CicloTrabajo 8   Voltaje A/D 1.65   Error -0.5
Nivel 511
--------------------------{}--------------------------------
CicloTrabajo 8   Voltaje A/D 1.65   Error -0.5
Nivel 511
--------------------------{}--------------------------------
CicloTrabajo 8   Voltaje A/D 1.65   Error -0.5
Nivel 511
--------------------------{}--------------------------------
CicloTrabajo 8   Voltaje A/D 1.65   Error -0.5
Nivel 511
--------------------------{}--------------------------------
CicloTrabajo 8   Voltaje A/D 1.65   Error -0.5
Nivel 511
--------------------------{}--------------------------------
CicloTrabajo 8   Voltaje A/D 1.65   Error -0.5
Cambio
Nivel 675
--------------------------{}--------------------------------
CicloTrabajo 16   Voltaje A/D 2.18   Error -0.339393939394
Cambio
Nivel 510
--------------------------{}--------------------------------
CicloTrabajo 8   Voltaje A/D 1.65   Error -0.5
Nivel 510
--------------------------{}--------------------------------
CicloTrabajo 8   Voltaje A/D 1.65   Error -0.5
Nivel 510
--------------------------{}--------------------------------
CicloTrabajo 8   Voltaje A/D 1.65   Error -0.5
```

Figura 7.2 Perturbación estabilidad el controlador

Las **Figura 7.3** y **Figura 7.4** muestran el comportamiento de los controladores PID y Difuso, respectivamente. Se observa que ambos responden eficientemente ante la estabilidad del sistema. El primero se probó con una velocidad de referencia de 700 rpm cuya respuesta teórica debe ser conservar un ciclo de trabajo del 50%, tal como se observa en la **Figura 7.3.** El controlador difuso se probó con un voltaje de referencia de 0.65V, que se traduce en un error negativo cuya respuesta debe ser disminuir el ciclo de trabajo a menos del 10%, Ver **Figura 4.21,** como se aprecia en la **Figura 7.4.** Cabe recalcar que las medidas implementadas para blindar el sistema de perturbaciones en la entrada del convertidor analógico/digital fueron efectivas.

```
pi@raspberrypi ~ $ sudo python PID25.py
---------------- 1 --------------------
Velocidad 700 rpm
Ciclo de trabajo 50 %
Cambio
---------------- 2 --------------------
a 12.6916896959 b 1.19106754995 c -0.0007704250264 Error 0.0141365824 Data 544
Current : (1.75V) 1.22 Amp Velocidad 700.85092196 RPM Referencia 700
Ciclo de trabajo 50 %
---------------- 3 --------------------
a 12.6916896959 b 1.19106754995 c -0.0007704250264 Error 0.0141365824 Data 544
Current : (1.75V) 1.22 Amp Velocidad 700.16854342 RPM Referencia 700
Ciclo de trabajo 50 %
---------------- 4 --------------------
a 12.6916896959 b 1.19106754995 c -0.0007704250264 Error 0.0141365824 Data 544
Current : (1.75V) 1.22 Amp Velocidad 700.66380612 RPM Referencia 700
Ciclo de trabajo 50 %
---------------- 5 --------------------
a 12.6916896959 b 1.19106754995 c -0.0007704250264 Error 0.0141365824 Data 544
Current : (1.75V) 1.22 Amp Velocidad 700.66380612 RPM Referencia 700
Ciclo de trabajo 50 %
---------------- 6 --------------------
a 12.6916896959 b 1.19106754995 c -0.0007704250264 Error 0.0141365824 Data 544
Current : (1.75V) 1.22 Amp Velocidad 700.66380612 RPM Referencia 700
Ciclo de trabajo 50 %
---------------- 7 --------------------
a 12.6916896959 b 1.19106754995 c -0.0007704250264 Error 0.0141365824 Data 544
Current : (1.75V) 1.22 Amp Velocidad 700.66380612 RPM Referencia 700
Ciclo de trabajo 50 %
---------------- 8 --------------------
a 12.6916896959 b 1.19106754995 c -0.0007704250264 Error 0.0141365824 Data 544
Current : (1.75V) 1.22 Amp Velocidad 700.66380612 RPM Referencia 700
Ciclo de trabajo 50 %
---------------- 9 --------------------
a 12.6916896959 b 1.19106754995 c -0.0007704250264 Error 0.0141365824 Data 544
Current : (1.75V) 1.22 Amp Velocidad 700.66380612 RPM Referencia 700
Ciclo de trabajo 50 %
---------------- 10 --------------------
a 12.6916896959 b 1.19106754995 c -0.0007704250264 Error 0.0141365824 Data 544
Current : (1.75V) 1.22 Amp Velocidad 700.66380612 RPM Referencia 700
Ciclo de trabajo 50 %
---------------- 11 --------------------
a 12.6916896959 b 1.19106754995 c -0.0007704250264 Error 0.0141365824 Data 544
Current : (1.75V) 1.22 Amp Velocidad 700.66380612 RPM Referencia 700
```

Figura 7.3 Resultado pruebas controlador PID en vacío

```
pi@raspberrypi ~ $ sudo python fuzzy3.py
Cambio
Nivel 510
--------------------------{}--------------------------------
CicloTrabajo 8   Voltaje A/D 1.65   Error -0.5
Nivel 510
--------------------------{}--------------------------------
CicloTrabajo 8   Voltaje A/D 1.65   Error -0.5
Nivel 510
--------------------------{}--------------------------------
CicloTrabajo 8   Voltaje A/D 1.65   Error -0.5
Nivel 510
--------------------------{}--------------------------------
CicloTrabajo 8   Voltaje A/D 1.65   Error -0.5
Nivel 510
--------------------------{}--------------------------------
CicloTrabajo 8   Voltaje A/D 1.65   Error -0.5
Nivel 510
--------------------------{}--------------------------------
CicloTrabajo 8   Voltaje A/D 1.65   Error -0.5
Nivel 510
--------------------------{}--------------------------------
CicloTrabajo 8   Voltaje A/D 1.65   Error -0.5
Nivel 510
--------------------------{}--------------------------------
CicloTrabajo 8   Voltaje A/D 1.65   Error -0.5
Nivel 510
--------------------------{}--------------------------------
CicloTrabajo 8   Voltaje A/D 1.65   Error -0.5
Nivel 510
--------------------------{}--------------------------------
CicloTrabajo 8   Voltaje A/D 1.65   Error -0.5
```

Figura 7.4 Resultado pruebas controlador difuso

Al momento de exigir el motor, por medio de la acción de frenado, se presentaron complicaciones con el voltaje entregado por el Puente H ya que no correspondía con el valor del ciclo de trabajo entregado por el sistema embebido, Raspberry Pi. Se concluyó que las baterías habían perdido capacidad de respuesta.

Aun así al hacer seguimiento con el multímetro digital a las tensiones de salida del puerto multipropósito configurado como salida del PWM programado en Python, al voltaje de salida del Puente H; y a los valores presentados en pantalla se puede determinar que ambos controladores satisfacían los requerimientos del sistema aumentando el ciclo de trabajo. Ver **Figura 7.5**.

```
pi@raspberrypi ~ $ sudo python fuzzy3.py
Cambio
Nivel 511
--------------------------{}--------------------------------
CicloTrabajo 75   Voltaje A/D 1.65   Error 0.30303030303
Nivel 511
--------------------------{}--------------------------------
CicloTrabajo 75   Voltaje A/D 1.65   Error 0.30303030303
Cambio
Nivel 656
--------------------------{}--------------------------------
CicloTrabajo 100   Voltaje A/D 2.12   Error 0.445454545455
Nivel 656
--------------------------{}--------------------------------
CicloTrabajo 100   Voltaje A/D 2.12   Error 0.445454545455
Nivel 656
--------------------------{}--------------------------------
CicloTrabajo 100   Voltaje A/D 2.12   Error 0.445454545455
Nivel 656
--------------------------{}--------------------------------
CicloTrabajo 100   Voltaje A/D 2.12   Error 0.445454545455
Nivel 656
--------------------------{}--------------------------------
CicloTrabajo 100   Voltaje A/D 2.12   Error 0.445454545455
Nivel 656
--------------------------{}--------------------------------
CicloTrabajo 100   Voltaje A/D 2.12   Error 0.445454545455
Nivel 656
--------------------------{}--------------------------------
CicloTrabajo 100   Voltaje A/D 2.12   Error 0.445454545455
```

Figura 7.5 Resultado pruebas controlador difuso bajo carga

8. CONCLUSIONES Y TRABAJOS FUTUROS

- Se realizó el diseño del control conociendo y modelando los distintos parámetros físicos que intervinieron en el sistema a controlar (**Tabla 4.5**); además, la metodología necesaria a la hora de diseñar el control, así como la base teórica sobre la que se sustentan los cálculos, ya que, sin un modelado del sistema físico, sería imposible realizar dicho diseño o las simulaciones pertinentes para comprobar la validez del mismo (**Capítulos 4.2 y 4.3**).

- Se presentó un acercamiento de un controlador de lógica difusa y PID, para un motor DC usando programación Python y un sistema digital embebido, Raspberry Pi.

- En la ejecución de los algoritmos de control de ambas técnicas empleadas se evidenció que el lenguaje de programación más adecuado para la implementación en el sistema embebido es Python debido a su facilidad de escritura, baja demanda de recurso computacional y flexibilidad en cuanto a uso de librerías y declaración de variables (**Capítulo 5.1**).

- A partir de los resultados adquiridos para los sistemas simulados (**Capítulos 4.2 y 4.3)** se puede afirmar que la metodología con mejor comportamiento es la difusa ya que, además de proporcionar la respuesta con mayor rapidez, se logra una estimación más precisa que la presentada por el PID.

- Con base en los resultados obtenidos en las pruebas finales (**Capítulo 7)** se puede deducir que, pese a que el controlador difuso tiene un tiempo de respuesta mucho menor comparado con el del PID, la estabilidad de este último resulta más útil en la aplicación de sistemas robustos, ya que el principio de traslape de zonas utilizado en la desfusificación presenta problemas en la programación para sistemas con presencia de perturbaciones y oscilaciones no naturales.

Para trabajos futuros se plantea investigar la implementación de un controlador difuso/PID adaptativo en Raspberry Pi

9. BIBLIOGRAFÍA

[1] Raspberry Pi Blog, [En línea]. Disponible en: http://www.Raspberry.org

[2] J. Teeter, "Use of a Fuzzy Gain Tuner for Improved Control of a DC Motor System with Nonlinearities," pp. 258–262, 1994.

[3] B. Behnam and M. Mansouryar, "Modeling and simulation of a DC motor control system with digital PID controller and encoder in FPGA using Xilinx system generator," Proc. 2011 2nd Int. Conf. Instrum. Control Autom. ICA 2011. November, pp. 104–108, 2011.

[4] K. Ogata, Ingeniería de control moderna, pp. 567-596. 2003.

[5] S. B. M. Noor, S. M. Uashi, and M. K. Hassan, "Microcontroller Performance for DC Motor Speed Control System," pp. 104–109, 2003.

[6] Detrás del pinguino. Guía de Raspberry Pi y Raspberry Pi 2 [En línea], 2013, A. Cobo. Disponible en: http://dplinux.net/guia-raspberry-pi/

[7] Dynamo Electronics, [En línea]. Disponible en: http://www.dynamoelectronics.com

[8] A. Robinson, Raspberry Pi Projects, vol. 1, pp. 24-50. 2015.

[9] Frambuesa Pi Colombia. Conexión remota al Raspberry Pi usando SSH [En línea]. 2013, M. Bejarano. Disponible en: http://www.frambuesapi.co/2013/ 09/25/tutorial-5-conexion-remota-al-raspberry-pi-usando-ssh/

[10] J. Leonardo and R. Gaviria, "Control PID para el control de velocidad de un motor DC," 2014.

[11] S. Chapman, Máquinas eléctricas Chapman, Edición 5, pp. 345-353. 2013.

[12] D. Giraldo, Teoría De Control, pp. 89-91. 1997.

[13] H. S. Isaza, “Estimación básica de los parámetros del circuito equivalente de la máquina de corriente directa,” pp. 1–4, 2015.

[14] S. Salvador, “Determinación de los parámetros de un motor de CD por medición física directa,” pp. 5–8, 2014.

[15] R. Alexander Montenegro, C. Alberto, and F. Perdomo, “Diseño e Implementación de un Control PID Dígital para Motor DC,” Segundo Congr. Virtual , Microcontroladores y sus Apl., pp. 1–9, 2010.

[16] L. Moreno, S. Garrido, and C. Balaguer, “Ingeniería de control. Modelado y control de sistemas dinámicos,” no. February, p. 460, 2003.

[17] A. O’Dwyer, Handbook of PI and PID Controller Tuning Rules, vol. 26, no. 1. 2006.

[18] SCIENTIA ET TECHNICA, Universidad Tecnológica de Pereira, Pereira-Colombia, Junio, pp. 10-13. 1997

[19] A. Ramirez, "Diseño e implementacion de un control pid digital para motor dc, usando dispositivos embebidos psoc cy8c29466-24pvxi", pp. 1-9. 2010

[20] MathWorks. PID Controller [En línea]. Disponible en: http://www.mathworks. com/help/simulink/slref/pidcontroller.html

[21] K. Passino and S. Yurkovich, Fuzzy control. Addison Wesley Longman, 1998.

[22] L. Zadeh. Fuzzy Seys, Fuzzy Logic, and Fuzzy Systems. World Scientific, 1996.

[23] C. D. De Los Ríos and W. Ipanaqué, “EVALUACIÓN DE ESTRUCTURAS Y MÉTODOS DE AJUSTE DE REGULADORES PID-DIFUSOS,” pp. 4–26, 2004.

[24] L. Reznik. Fuzzy Controller. Oxford. p 287. 1997

[25] MathWorks. Fuzzy Logic Toolbox [Documento en línea].<>. http://www.mathworks.com/help/fuzzy/

[26] M. Lutz, "Learning Python", pp. 3-20. 2013

[27] F. Moreno, “Diseño de un sistema de control de velocidad de un motor de corriente continua basado en acelerómetros,” p. 350, 2010.

[28] Allegro. "ASC711", pp. 1-16. 2013

[29] Microchip. "MCP3004/3008", p. 1–40. 2008

ANEXO 1 CÓDIGO MATLAB CONTROLADOR DIFUSO

```
[System]
Name='control1'
Type='mamdani'
Version=2.0
NumInputs=2
NumOutputs=1
NumRules=7
AndMethod='prod'
OrMethod='max'
ImpMethod='prod'
AggMethod='sum'
DefuzzMethod='centroid'

[Input1]
Name='ERROR'
Range=[-1 1]
NumMFs=5
MF1='VEL_BAJA':'trapmf',[0.2 0.5 1 1]
MF2='VEL_CTE':'trimf',[-0.01 0 0.01]
MF3='VEL_ALTA':'trapmf',[-1 -1 -0.5 -0.2]
MF4='VAL_MED_BAJA':'trimf',[0 0.2 0.4]
MF5='VEL_MED_ALTA':'trimf',[-0.4 -0.2 -0]

[Input2]
Name='CAMBIO'
Range=[-1 1]
NumMFs=2
MF1='E_ALTO_NEGATIVO':'trimf',[-1 -1 0]
MF2='ERROR_ALTO_POSITIVO':'trimf',[0 1 1]

[Output1]
Name='CONTROL'
Range=[0 24]
NumMFs=5
MF1='DISMINUIR_MUCHO':'trimf',[0 2 4]
MF2='AUMENTAR_MUCHO':'trimf',[20 22 24]
MF3='DIEMINUIR_POCO':'trimf',[6 8 10]
MF4='SOSTENER':'trimf',[11.96 12 12.04]
MF5='AUMENTAR_POCO':'trimf',[14 16 18]

[Rules]
1 0, 2 (1) : 1
3 0, 1 (1) : 1
2 1, 3 (1) : 1
2 2, 5 (1) : 1
5 0, 3 (1) : 1
4 0, 5 (1) : 1
2 0, 4 (1) : 1
```

ANEXO 2 CÓDIGO PYTHON CONTROLADOR PID

```
#PID Controller
#!/usr/bin/python

import spidev
import time
import os
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setup(23,GPIO.OUT)
# Open SPI bus and PWM
spi = spidev.SpiDev()
spi.open(0,0)
Sal1 = GPIO.PWM(23,60)
Sal1.start(75)

# Function to read SPI data from MCP3008 chip
# Channel must be an integer 0-7
def ReadChannel(channel):
  adc = spi.xfer2([1,(8+channel)<<4,0])
  data = ((adc[1]&3) << 8) + adc[2]
  return data

# Function to convert data to voltage level,
# rounded to specified number of decimal places.
def ConvertVolts(data,places):
  volts = ((data * 3.3) / float(1023))
  volts = round(volts,places)
  return volts

# Function to conver Current Sensor  data, rounded to specified
def ConvertCurr(data,places):
  Vanal = (data * 3.3)/float(1023)
  Curr  = 36.7 *Vanal/3.3  - 18.3
  Curr  = round(Curr,places)
  return Curr

Curr_Channel  = 1

# Define delay between readings
delay    = 5
T        = 5
Vref     = 700
Count    = 0
Kp       = 1.1926084
Td       = -0.00323
```

```
Ti       = 1/1.928521
a        = Kp + Kp*T/Ti + Kp*Td/T
b        = Kp + 2*Kp*Td/T
c        = Kp*Td/T
Upas     = 0
Epaspas  = 0
Epas     = 0
D        = 80
Curr_level = 0
while True:

  # Read the Current sensor data
  Count      = Count +1
  if Count==1:
     U    = Vref
     Upas = U
     print ("---------------- {} --------------------".format(Count))
     print ("Velocidad {} rpm".format(U))
     print ("Ciclo de trabajo {} %".format(D))
     Sal1.ChangeDutyCycle(D)
  else:
     Curr_level_pas = Curr_level
     Curr_level = ReadChannel(Curr_Channel)
     if (Curr_level_pas - 100 < Curr_level and Curr_level_pas + 100 >Curr_level):
        Curr_level = Curr_level_pas
     else:
        print ("Cambio")
     Curr_volts = ConvertVolts(Curr_level,2)
     Curr       = ConvertCurr(Curr_level,2)
     I          = float(Curr)
     Vel        = -0.0096*I*I*I*I-0.1255*(I*I*I)+4.8919*(I*I)-92.9431*I+1461.5
     E          = Vref-Vel

     U       = Upas + a*E + b*Epas + c*Epaspas
     Upas    = U
     Epaspas = Epas
     Epas    = E
     print ("---------------- {} --------------------".format(Count))
     print("a {} b {} c {} Error {} Data {} ".format(a,b,c,E,Curr_level))
     # Print out results
     print("Current : ({}V) {} Amp Velocidad {} RPM Referencia
{}".format(Curr_volts,Curr,U,Vref))

     if (Curr < 2):
        Upas = Vref
        print ("Ciclo de trabajo {} %".format(D))
        Sal1.ChangeDutyCycle(D)
```

```
    elif (U>Vref):
        D = D-0.5
        print ("Ciclo de trabajo {} %".format(D))
        Sal1.ChangeDutyCycle(D)
    elif (U<Vref):
        D = D+0.5
        print ("Ciclo de trabajo {} %".format(D))
        Sal1.ChangeDutyCycle(D)
    else:
        print "-----iguales-------"
        Sal1.ChangeDutyCycle(D)
time.sleep(delay)
```

ANEXO 3 CÓDIGO PYTHON CONTROLADOR DIFUSO

```
#!/usr/bin/python

from math import*
import spidev
import time
import os
import RPi.GPIO as GPIO

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup(23,GPIO.OUT)

# Open SPI bus and PWM
spi = spidev.SpiDev()
spi.open(0,0)
Sal1 = GPIO.PWM(23,60)
Sal1.start(70)

# Function to read SPI data from MCP3008 chip
# Channel must be an integer 0-7
def ReadChannel(channel):
  adc = spi.xfer2([1,(8+channel)<<4,0])
  data = ((adc[1]&3) << 8) + adc[2]
  return data

# Function to convert data to voltage level,
# rounded to specified number of decimal places.
def ConvertVolts(data,places):
  volts = (data * 3.3) / float(1023)
  volts = round(volts,places)
  return volts

# Function to conver Current Sensor  data, rounded to specified
def ConvertCurr(data,places):
  Vanal = (data * 3.3) / float(1023)
  Curr  = (Vanal/(-0.05))+15.26
  Curr  = round(Curr,places)
  return Curr

Curr_Channel=1
delay = 3
Curr_level = 0
while True:
      Curr_level_pas = Curr_level
      Curr_level = ReadChannel(Curr_Channel)
```

```
if (Curr_level_pas - 100 < Curr_level and Curr_level_pas + 100 >Curr_level):
   Curr_level = Curr_level_pas
else:
   print ("Cambio")
print ("Nivel {}".format(Curr_level))
   Curr_volts = ConvertVolts(Curr_level,2)
   Curr       = ConvertCurr(Curr_level,2)

   ref        = 1.65
   error      = (Curr_volts-ref)/3.3
   derror     = 0
   z          = list()

   class fuzzy:
            _sigs_in={}
            _rules= []

            def __init__(self, name):
                     self._name=name

            def addInSignal(self,signal):
                     self._sigs_in[signal._name]=signal

            def addOutSignal(self,signal):
                     self._sig_out=signal

            def addRule(self,rule):
                     self._rules.append(rule)

            def evaluate(self,sigs_in):
                     ret={}

                     for sig in sigs_in.keys():
                              ret[sig]=self._sigs_in[sig].evaluate(sigs_in[sig])

                     vret={}

                     for key in self._sig_out._regions.keys():
                              vret[key] = 0.0

                     for rule in self._rules:
                              aa =  rule.evaluate(ret)
                              z.append(aa[1])
                              vret[aa[0]] += aa[1]
                              #print aa
                     #print vret
```

```
                if (z[6]!=1 and z[5]!=0 and z[6]>z[5] and z[0]==0 and z[1]==0 and
z[2]==0 and z[3]==0 and z[4]==0):
                        Vout=14
                elif (z[6]!=0 and z[5]!=1 and z[6]<z[5] and z[6]!=0 and z[0]==0 and
z[1]==0 and z[2]==0 and z[3]==0 and z[4]==0):
                        Vout=15
                elif (z[5]!=1 and z[0]!=0 and z[5]>z[0] and z[1]==0 and z[2]==0 and
z[3]==0 and z[4]==0 and z[6]==0):
                        Vout=18
                elif (z[5]!=0 and z[0]!=1 and z[5]<z[0] and z[5]!=0 and z[1]==0 and
z[2]==0 and z[3]==0 and z[4]==0 and z[6]==0):
                        Vout=20
                elif (z[0]>=0.5 and z[1]==0 and z[2]==0 and z[3]==0 and z[4]==0
and z[5]==0 and z[6]==0):
                        Vout=24
                elif (z[5]>=0.5 and z[0]==0 and z[1]==0 and z[2]==0 and z[3]==0
and z[4]==0 and z[6]==0):
                        Vout=16
                elif (z[6]>=0.5 and z[0]==0 and z[1]==0 and z[2]==0 and z[3]==0
and z[4]==0 and z[5]==0):
                        Vout=12
                elif (z[6]!=1 and z[4]!=0 and z[6]>z[4] and z[0]==0 and z[1]==0 and
z[2]==0 and z[3]==0 and z[5]==0):
                        Vout=11
                elif (z[6]!=0 and z[4]!=1 and z[6]<z[4] and z[6]!=0 and z[0]==0 and
z[1]==0 and z[2]==0 and z[3]==0 and z[5]==0):
                        Vout=9
                elif (z[4]!=1 and z[1]!=0 and z[4]>z[1] and z[0]==0 and z[2]==0 and
z[3]==0 and z[5]==0 and z[6]==0):
                        Vout=6
                elif (z[4]!=0 and z[1]!=0 and z[4]<z[1] and z[4]!=0 and z[0]==0 and
z[2]==0 and z[3]==0 and z[5]==0 and z[6]==0):
                        Vout=4
                elif (z[4]>=0.5 and z[0]==0 and z[1]==0 and z[2]==0 and z[3]==0
and z[5]==0 and z[6]==0):
                        Vout=8
                elif (z[1]>=0.5 and z[0]==0 and z[2]==0 and z[3]==0 and z[4]==0
and z[5]==0 and z[6]==0):
                        Vout=2
                elif (z[6]==1 and z[2]!=0 and z[0]==0 and z[1]==0 and z[3]==0 and
z[4]==0 and z[5]==0):
                        Vout=13
                elif (z[6]==1 and z[3]!=0 and z[0]==0 and z[1]==0 and z[2]==0 and
z[4]==0 and z[5]==0):
                        Vout=11
                else:
                        Vout=2
```

```
                D=Vout*100/24

                #while True:

            Sal1.ChangeDutyCycle(D)

                print("-------------------------{}-------------------------------")
                print("CicloTrabajo {}   Voltaje A/D {}   Error {}
".format(D,Curr_volts,error))

                time.sleep(delay)

    class rule:
        'Regla Difusa'

        def __init__(self,ireg,oreg):
            self._ireg = ireg
            self._oreg = oreg

        def evaluate(self,mvals):
            val = 1.0

            for rkey in self._ireg.keys():
                mval = mvals[rkey]
                rval = self._ireg[rkey]
                val = val* mval[rval]
            return [self._oreg,val]


    class fuzzySignal:
        'Senal difusa de entrada/salida'

        _regions = {}

        def __init__(self,name):
            'A constructor'

            self._name = name

        def addRegion(self,name,mf):
            'Ingresar una nueva region difusa, segun datos de entrada, con
nombre y funcion de membresia'

            self._regions[name]=mf

        def evaluate(self,x):
```

```
                regs = self._regions.keys()
                ret = {}
                for key in regs:
                        mf = self._regions[key]
                        ret[key] = mf.evaluate(x)
                return ret

class mf:

        def __init__(self, p):
                'A constructor'

                self._p = p[:]


class trimf(mf):
        'Funcion de membresia Triangular'

        def evaluate(self,x):
                if x<=self._p[0]:
                        return 0
                elif self._p[0] < x <= self._p[1]:
                        return (x - self._p[0])/(self._p[1] - self._p[0])
                elif self._p[1] < x <= self._p[2]:
                        return 1 - (x - self._p[1])/(self._p[2] - self._p[1])
                else:
                        return 0

class trapmf(mf):
        'Funcion de membresia Trapezoidal'

        def evaluate(self,x):
                if x<=self._p[0]:
                        return 0
                elif self._p[0] < x <= self._p[1]:
                        return (x - self._p[0])/(self._p[1] - self._p[0])
                elif self._p[1] < x <= self._p[2]:
                        return 1
                elif self._p[2] < x <= self._p[3]:
                        return 1 - (x - self._p[2])/(self._p[3] - self._p[2])
                else:
                        return 0


aa = fuzzySignal('Error')

aa.addRegion('Vel_Alta',trapmf([-1,-1,-0.5,-0.2]))
```

```
        aa.addRegion('Vel_Med_Alta',trimf([-0.4,-0.2,0]))
        aa.addRegion('Vel_Cte',trimf([-0.1,0,0.1]))
        aa.addRegion('Vel_Med_Baja',trimf([0,0.2,0.4]))
        aa.addRegion('Vel_Baja',trapmf([0.2,0.5,1,1.65]))

        bb = fuzzySignal('Cambio')

        bb.addRegion('E_Alto_Negativo',trimf([-1,-1,0]))
        bb.addRegion('E_Alto_Positivo',trimf([0,1,1]))

        cc = fuzzySignal('Control')

        cc.addRegion('Disminuir_Mucho',trimf([0,2,4]))
        cc.addRegion('Disminuir_Poco',trimf([6,8,10]))
        cc.addRegion('Sostener',trimf([11.5,12,12.5]))
        cc.addRegion('Aumentar_Poco',trimf([14,16,18]))
        cc.addRegion('Aumentar_Mucho',trimf([20,22,24]))

        ff = fuzzy('Fuzzy')

        ff.addInSignal(aa)
        ff.addInSignal(bb)
        ff.addOutSignal(cc)

        ff.addRule(rule({'Error':'Vel_Baja'},'Aumentar_Mucho'))
        ff.addRule(rule({'Error':'Vel_Alta'},'Disminuir_Mucho'))
        ff.addRule(rule({'Error':'Vel_Cte','Cambio':'E_Alto_Positivo'},'Aumentar_Poco'))
        ff.addRule(rule({'Error':'Vel_Cte','Cambio':'E_Alto_Negativo'},'Disminuir_Poco'))
        ff.addRule(rule({'Error':'Vel_Med_Alta'},'Disminuir_Poco'))
        ff.addRule(rule({'Error':'Vel_Med_Baja'},'Aumentar_Poco'))
        ff.addRule(rule({'Error':'Vel_Cte'},'Sostener'))

        ff.evaluate({'Error':error,'Cambio':derror})

time.sleep(delay)
```