

# **TRABAJO ESPECIAL DE GRADO**

**Controlador Difuso para el sistema Ball and Beam de  
Quanser**

Presentado ante la ilustre  
Universidad Central de Venezuela  
por el Br. Brandon Jiménez  
para optar al título de  
Ingeniero Electricista.

Caracas, noviembre 2020

# **TRABAJO ESPECIAL DE GRADO**

**Controlador Difuso para el sistema Ball and Beam de  
Quanser**

TUTOR ACADÉMICO: Panayotis Tremante

Presentado ante la ilustre  
Universidad Central de Venezuela  
por el Br. Brandon Alberto Jiménez Rivas  
para optar al título de Ingeniero  
Electricista.

Caracas, noviembre 2020



*A mi abuelo, estaría muy orgulloso por este gran logro*

## **RECONOCIMIENTOS Y AGRADECIMIENTOS**

Gracias a toda mi familia, en especial a mi madre y hermanas, ustedes siempre han creido en mi incluso en mis peores momentos. Mis amigos y compañeros que he conocido en la UCV, en especial a Carlos Lopez, Antunes, Marco, Mora, Adrian, Carlos Salas, Jesus Mambie, Alfredo Cerqueira, Alejandro Barone, Luis Acevedo, Alexander Yammie, Andres Lares y Agustin Negrin, son los mejores y cada día en la universidad fue especial gracias a ustedes. A mis compañeros de trabajo Ricardo Tascon y Daniel Lira, personas que hoy día me hacen un mejor profesional. Fabiana gracias por todo el apoyo, amor y el tiempo que me brindaste para afrontar este reto y finalizarlo con éxito sin ti las cosas hubiesen sido más difíciles.

A los profesores que a lo largo de la carrera se esforzaron por mantenerse en la universidad enseñando, en especial al profesor Alonso que sus conocimientos tanto académicos como de la vida me han ayudado mucho en el campo laboral, al profesor Rivero que siempre me ayudó y me motivo por la carrera de electrónica, al profesor Panayotis que me acepto en sus clases de postgrado y fue guía en este trabajo y al profesor José Antonio de la escuela de Física por su constante apoyo en mi tiempo en la facultad de ciencias.

Agradezco a mi casa de estudio donde tuve el honor de pertenecer a dos grandes facultades donde viví tantos momentos que nunca olvidare.

**Brandon A., Jiménez R.**

## **Controlador Difuso para el sistema Ball and Beam de Quanser**

**Tutor Académico: Ing. Panayotis Tremante. Tesis. Caracas, Universidad Central de Venezuela. Facultad de Ingeniería. Escuela de Ingeniería Eléctrica. Opción Electrónica. Trabajo de Grado. 2020. 149 páginas + anexos**

**Palabras Claves:** Ball and Beam, Quanser, control difuso, lógica difusa, Mamdani, método de patrón de búsqueda, control PD, control proporcional.

**Resumen.-** Este trabajo trata sobre la implementación y diseño de controladores difusos en el sistema Ball and Beam de Quanser. Partiendo por la descripción del sistema, se analizaron las etapas del sistemas, ecuaciones que describen el sistema y los distintos sensores y actuadores involucrados, donde se diseño un circuito acondicionador de señales y se selecciono el microcontrolador ESP32 para llevar a cabo el control, debido a sus prestaciones a nivel de procesamiento y su compatibilidad con las librerías de control difuso. Se plantearon dos topologías de control, donde en un caso se diseñó dos controladores difusos y en el otro caso un control proporcional y un PD, la segunda topología estuvo conformada por un controlador difuso con tres entradas. Se realizaron simulaciones en cada topología, donde las funciones de pertenencia de los controladores difusos fueron ajustadas por el método de optimización por patrón de búsqueda obteniendo un mejor desempeño considerando el sobre pico, tiempo de alza y tiempo de establecimiento con la topología de doble lazo con el controlador difuso ajustado. Luego se implementaron los controladores diseñados para la topología de control de doble lazo, evaluando el tiempo de alza, tiempo de establecimiento y sobre pico a través de una conexión serial con MATLAB, donde se obtuvo un mejor desempeño del sistema con el controlador difuso ajustado de la posición de la esfera.

## ÍNDICE GENERAL

<b>RECONOCIMIENTOS Y AGRADECIMIENTOS</b>	<b>III</b>
<b>ÍNDICE GENERAL</b>	<b>V</b>
<b>LISTA DE FIGURAS</b>	<b>XI</b>
<b>LISTA DE TABLAS</b>	<b>XVII</b>
<b>LISTA DE ACRÓNIMOS</b>	<b>XX</b>
<b>INTRODUCCIÓN</b>	<b>1</b>
<b>CAPÍTULO I</b>	<b>5</b>
<b>PLANTEAMIENTO DEL PROBLEMA Y OBJETIVOS</b>	<b>5</b>
1.1. PLANTEAMIENTO DEL PROBLEMA . . . . .	5
1.2. OBJETIVO GENERAL . . . . .	6
1.3. OBJETIVOS ESPECÍFICOS . . . . .	6
<b>CAPÍTULO II</b>	<b>7</b>
<b>MARCO TEÓRICO</b>	<b>7</b>
2.1. TEORÍA DIFUSA . . . . .	7
2.1.1. CONJUNTO DIFUSO . . . . .	7
2.1.2. LÓGICA DIFUSA . . . . .	9
2.2. FUNCIÓN DE PERTENENCIA . . . . .	10

2.2.1.	TIPOS DE FUNCIONES DE PERTENENCIA . . . . .	10
2.2.2.	FUNCIÓN DE PERTENENCIA TRIANGULAR . . . . .	10
2.2.3.	FUNCIÓN DE PERTENENCIA TRAPEZOIDAL . . . . .	11
2.2.4.	SINGLETON DIFUSO . . . . .	12
2.3.	OPERACIONES ENTRE CONJUNTOS DIFUSOS . . . . .	13
2.3.1.	IGUALDAD . . . . .	13
2.3.2.	INCLUSIÓN . . . . .	13
2.3.3.	COMPLEMENTO . . . . .	13
2.3.4.	UNIÓN . . . . .	14
2.3.5.	INTERSECCIÓN . . . . .	14
2.4.	VARIABLE LINGÜÍSTICA . . . . .	14
2.5.	OPERADORES LÓGICOS . . . . .	16
2.5.1.	OPERADOR S-NORMA . . . . .	16
2.5.2.	OPERADOR T-NORMA . . . . .	17
2.6.	REGLAS SI-ENTONCES . . . . .	18
2.7.	SISTEMA DIFUSO . . . . .	20
2.8.	TIPOS DE SISTEMAS DIFUSOS . . . . .	20
2.8.1.	SISTEMA DIFUSO PURO . . . . .	20
2.8.2.	SISTEMA DIFUSO TAKAGI-SUGENO-KANG (TSK) .	21
2.8.3.	SISTEMA DIFUSO CON FUSIFICACIÓN Y DEFUSIFICACIÓN TIPO MAMDANI . . . . .	22
2.8.4.	SISTEMA DIFUSO CON FUSIFICACIÓN Y DEFUSIFICACIÓN TIPO SUGENO . . . . .	22
2.9.	PARTES DE UN SISTEMA DIFUSO . . . . .	23
2.9.1.	FUSIFICACIÓN . . . . .	23

2.9.2. APLICACIÓN DEL OPERADOR DIFUSO . . . . .	24
2.9.3. APLICACIÓN DEL MÉTODO DE IMPLICACIÓN . . . . .	24
2.9.4. COMPOSICIÓN DE LAS REGLAS . . . . .	25
2.9.5. DEFUSIFICACIÓN . . . . .	26
2.9.6. MÉTODO DEL CENTRO DE GRAVEDAD, CENTROIDE O CENTRO DE ÁREA . . . . .	27
2.10. CONTROL DIFUSO . . . . .	28
2.11. REPRESENTACIÓN ÚNICA DE FUNCIONES DE PERTENENCIA . . . . .	30
2.12. FUNCIÓN OBJETIVO Y MULTIOBJETIVO . . . . .	32
2.13. OPTIMIZACIÓN POR BÚSQUEDA DIRECTA . . . . .	33
2.14. MÉTODO DE PATRÓN DE BÚSQUEDA . . . . .	33
2.15. AJUSTE DE FUNCIONES DE PERTENENCIA . . . . .	36
<b>CAPÍTULO III</b>	<b>42</b>
<b>DESCRIPCIÓN DEL SISTEMA BALL AND BEAM DE QUANSER</b>	<b>42</b>
3.1. EL SISTEMA BALL AND BEAM DE QUANSER . . . . .	42
3.2. MÓDULO BB01 . . . . .	44
3.3. MÓDULO SRV02 . . . . .	45
3.4. MÓDULO VOLTPAQ-X1 . . . . .	48
<b>CAPÍTULO IV</b>	<b>52</b>
<b>DESCRIPCIÓN DEL FUNCIONAMIENTO DEL SISTEMA BALL AND BEAM DE QUANSER</b>	<b>52</b>

4.1. DESARROLLO DE LAS ECUACIONES MATEMÁTICAS QUE DESCRIBEN LA PLANTA . . . . .	52
4.1.1. DESARROLLO DE LAS ECUACIONES MATEMÁTICAS PARA EL MÓDULO BB01 . . . . .	53
4.1.2. DESARROLLO DE LAS ECUACIONES MATEMÁTICAS PARA EL MÓDULO SRV02 . . . . .	58
4.1.3. ECUACIONES MATEMÁTICAS DEL SISTEMA BALL AND BEAM . . . . .	63
<b>CAPÍTULO V</b>	<b>65</b>
<b>DESARROLLO DEL HARDWARE Y SELECCIÓN DE UNA TARJETA DE DESARROLLO</b>	<b>65</b>
5.1. SELECCIÓN DE LA TARJETA DE DESARROLLO . . . . .	65
5.1.1. ARDUINO UNO . . . . .	65
5.1.2. ESP32 WROOM Devkit . . . . .	66
5.1.3. RASPBERRY PI 3 B+ . . . . .	68
5.1.4. SOFTWARE PARA TRABAJAR CON UN CONTROLADOR DIFUSO . . . . .	69
5.1.5. EVALUACIÓN DE TARJETAS DE DESARROLLO . . . . .	71
5.2. DISEÑO DE LA INTERFAZ ENTRE EL SISTEMA BALL AND BEAM Y ESP32 . . . . .	72
<b>CAPÍTULO VI</b>	<b>78</b>
<b>DISEÑO Y AJUSTE DE LOS CONTROLADORES DIFUSOS</b>	<b>78</b>
6.1. TOPOLOGÍA DE CONTROL DE DOBLE LAZO . . . . .	80

6.1.1. DISEÑO DEL CONTROLADOR DIFUSO PARA EL ÁNGULO DE LA BARRA . . . . .	81
6.1.2. SIMULACIÓN DEL CONTROLADOR DIFUSO PARA EL ÁNGULO DE LA BARRA . . . . .	84
6.1.3. DISEÑO DE CONTROLADOR DIFUSO PARA LA POSICIÓN DE LA ESFERA . . . . .	86
6.1.4. SIMULACIÓN DE CONTROLADOR DIFUSO PARA LA POSICIÓN DE LA ESFERA . . . . .	90
6.1.5. AJUSTE DE CONTROLADOR DIFUSO PARA LA POSICIÓN DE LA ESFERA . . . . .	91
6.1.6. SIMULACIÓN DEL CONTROLADOR DIFUSO AJUSTADO PARA LA POSICIÓN DE LA ESFERA . . . . .	94
6.2. TOPOLOGÍA DE CONTROL DE TRES ENTRADAS . . . . .	95
6.2.1. DISEÑO DE CONTROLADOR DIFUSO DE TRES ENTRADAS . . . . .	96
6.2.2. SIMULACIÓN DE CONTROLADOR DIFUSO DE TRES ENTRADAS . . . . .	101
6.2.3. AJUSTE DE CONTROLADOR DIFUSO DE TRES ENTRADAS . . . . .	102
6.2.4. SIMULACIÓN DE CONTROLADOR DIFUSO AJUSTADO DE TRES ENTRADAS . . . . .	105
6.3. CONTROLADOR CONVENCIONAL DE DOBLE LAZO . . . . .	106
6.3.1. DISEÑO DE CONTROLADOR PROPORCIONAL PARA EL ÁNGULO DE LA BARRA . . . . .	106
6.3.2. DISEÑO DE CONTROLADOR PD PARA LA POSICIÓN DE LA ESFERA . . . . .	109

6.4. COMPARACIÓN POR SIMULACIÓN DE CONTROLADOR DIFUSO AJUSTADO CON CONTROLADOR PD . . . . .	111
<b>CAPÍTULO VII</b>	<b>112</b>
<b>IMPLEMENTACIÓN DE CONTROLADORES EN EL SISTEMA</b>	
<b>BALL AND BEAM DE QUANSER</b> <b>112</b>	
7.1. ALGORITMO DE CONTROL . . . . .	112
7.2. FILTRO DIGITAL A LA ENTRADA DE LOS CONTROLADORES	116
7.3. RESULTADOS DEL CONTROL PD PARA LA POSICIÓN DE LA ESFERA . . . . .	117
7.4. RESULTADOS DEL CONTROLADOR DIFUSO SIN AJUSTAR DE LA POSICIÓN DE LA ESFERA . . . . .	118
7.5. RESULTADOS DEL CONTROLADOR DIFUSO AJUSTADO DE LA POSICIÓN DE LA ESFERA . . . . .	119
7.6. COMPARACIÓN ENTRE LOS CONTROLADORES . . . . .	120
<b>CONCLUSIONES</b>	<b>123</b>
<b>REFERENCIAS</b>	<b>125</b>
<b>CÓDIGO EN MATLAB PARA GRÁFICA</b> <b>A-1</b>	
<b>CÓDIGO EN MATLAB PARA AJUSTE DE FUNCIONES DE PERTENENCIA</b> <b>A-3</b>	
<b>CÓDIGO EN EL ESP32</b>	<b>A-29</b>
<b>DISEÑO DE PCB</b>	<b>A-84</b>

## LISTA DE FIGURAS

2.1.	Funciones de los días del fin de semana . . . . .	8
2.2.	Función de pertenencia triangular. . . . .	11
2.3.	Función de pertenencia Trapezoidal. . . . .	12
2.4.	Proceso gráfico del resultado de una regla en ejemplo de precio estimado de una casa. . . . .	19
2.5.	Configuración Básica de un Sistema Difuso Puro. . . . .	21
2.6.	Configuración Básica de un Sistema Difuso TSK. . . . .	21
2.7.	Configuración Básica y Diagrama de Bloque de un Sistema Difuso con fusificación y defusificación tipo Mamdani. . . . .	22
2.8.	Fusificación de una entrada. . . . .	23
2.9.	Aplicación de operador difuso de suma clásica. . . . .	24
2.10.	Método de Implicación . . . . .	25
2.11.	Composición de las reglas. . . . .	26
2.12.	Método de defusificación de Centro de Gravedad. . . . .	27
2.13.	Diagrama completo de un sistema difuso Tipo Mamdani. . . . .	28
2.14.	Diagrama de bloques de un controlador difuso en un Sistema a lazo cerrado. . . . .	30
2.15.	Función Única. . . . .	31
2.16.	Gráfico del Método de patrón de búsqueda con 5 iteraciones . . . . .	34
2.17.	Diagrama de Flujo del algoritmo de Optimización por el método de patrón de búsqueda . . . . .	36

2.18. Función de pertenencia trapezoidal que no cumple las restricciones <b>2.30, 2.31 y 2.32.</b>	37
2.19. Funciones de pertenencia que emulan la lógica booleana. . . . .	38
2.20. Funciones de pertenencia de entrada de un Controlador Difuso. . . . .	40
2.21. Funciones de pertenencia con simetría respecto al punto $x = x_i$ . . . . .	40
 3.1. El sistema Ball and Beam de Quanser. . . . .	42
3.2. Esquema de conexión del tren de engranajes del Ball and Beam. . . . .	43
3.3. Diagrama de bloques del sistema Ball and Beam de Quanser. . . . .	43
3.4. Partes del módulo BB01 y módulo SRV02. . . . .	44
3.5. Diagrama de bloques del Módulo SRV02. . . . .	46
3.6. Modulo SRV02 configuración High gear. . . . .	47
3.7. Partes del modulo SRV02. . . . .	47
3.8. Entradas y salidas del VoltPAQ-X1. . . . .	49
3.9. Cables del módulo VoltPAQ-X1. . . . .	51
 4.1. Diagrama de bloques del sistema ball and beam en tiempo. . . . .	52
4.2. Esquema físico del sistema Ball and Beam. . . . .	54
4.3. Diagrama de cuerpo libre de la esfera inicialmente. . . . .	55
4.4. Diagrama de cuerpo libre de la esfera. . . . .	56
4.5. Esquema del circuito de armadura y tren de engranajes. . . . .	59
4.6. Diagrama de bloques en tiempo con ecuaciones del sistema. . . . .	64
 5.1. Tarjeta de desarrollo Arduino UNO. . . . .	66
5.2. Tarjeta de desarrollo ESP32 WROOM Devkit. . . . .	68
5.3. Raspberry pi 3 B+. . . . .	69
5.4. Ventana del Editor del Sistema Difuso en el Fuzzy Logic Toolbox. . . . .	70

5.5.	Diagrama de bloques del sistema ball and beam con el circuito acondicionador de señales y el ESP32 WROOM Devkit. . . . .	72
5.6.	Característica de transferencia para señal de posición. . . . .	73
5.7.	Acondicionamiento de señal sensor de posición. . . . . . . . .	74
5.8.	Característica de transferencia para señal del ángulo de la barra. .	75
5.9.	Acondicionamiento de señal sensor de ángulo de la barra. . . . .	75
5.10.	Característica de transferencia de la señal del motor. . . . .	76
5.11.	Acondicionamiento de señal para el motor . . . . . . . . .	76
6.1.	Topología de control de doble lazo. . . . . . . . . . . . . . .	78
6.2.	Bloque de la planta. .	79
6.3.	Bloques internos de la planta, ángulo y posición. . . . . . . .	79
6.4.	Diagrama de bloques para el ángulo de la barra. . . . . . .	80
6.5.	Diagrama de bloques para la posición. . . . . . . . . . . . .	80
6.6.	Funciones de pertenencia para la entrada del error del ángulo. .	82
6.7.	Funciones de pertenencia de la derivada del error del ángulo. .	82
6.8.	Funciones de pertenencia para la salida del controlador difuso para el ángulo de la barra. . . . . . . . . . . . . . . . . . .	83
6.9.	Diagrama de bloques para la simulación con controlador difuso del ángulo de la barra. . . . . . . . . . . . . . . . . . .	85
6.10.	Simulación del ángulo de la barra, Set point: $45^\circ$ , tiempo de muestreo: 12ms, $\theta_l(0) = 0$ y $\dot{\theta}_l(0) = 0$ . . . . . . . . . . . . .	86
6.11.	Funciones de pertenencia para la entrada del error de posición. .	87
6.12.	Funciones de pertenencia para la entrada de la derivada del error de posición. . . . . . . . . . . . . . . . . . .	88
6.13.	Funciones de pertenencia para la salida del controlador difuso de la posición. . . . . . . . . . . . . . . . . . .	89

6.14. Diagrama de bloques para la simulación con el controlador difuso de la posición. . . . .	90
6.15. Simulación de posición de la esfera, Set point: 5cm, tiempo de muestreo: 12ms, $x(0) = 0$ , $\dot{x}(0) = 0$ , $\theta_l(0) = 0$ y $\dot{\theta}_l(0) = 0$ . . . . .	91
6.16. Diagrama de bloques para ajustar el controlador difuso de la posición de la esfera. . . . .	92
6.17. Funciones de pertenencia ajustadas para la entrada del error de posición. . . . .	92
6.18. Funciones de pertenencia ajustadas para la entrada de la derivada del error de posición. . . . .	93
6.19. Simulación de Posición con controlador difuso ajustado, Set Point: 5cm, Tiempo de muestreo: 12ms, $x(0) = 0$ , $\dot{x}(0) = 0$ , $\theta_l(0) = 0$ y $\dot{\theta}_l(0) = 0$ . . . . .	94
6.20. Comparación entre controlador de doble lazo ajustado y sin ajustar. . . . .	95
6.21. Topología de control de tres entradas. . . . .	95
6.22. Funciones de pertenencia del controlador difuso de tres entradas para la entrada del error de posición. . . . .	96
6.23. Funciones de pertenencia del controlador difuso de tres entradas para la entrada de la derivada del error de posición. . . . .	97
6.24. Funciones de pertenencia para el controlador difuso de tres entradas para la entrada del error del ángulo de la barra. . . . .	98
6.25. Funciones de pertenencia de la salida del controlador difuso de tres entradas. . . . .	98
6.26. Diagrama de bloques para la simulación con controlador difuso de tres entradas. . . . .	101
6.27. Simulación de posición con controlador difuso de tres entradas, Set point: 5cm, tiempo de muestreo: 12ms y condiciones iniciales nulas. . . . .	102

6.28. Diagrama de bloques para la simulación con controlador difuso de tres entradas ajustado. . . . .	103
6.29. Funciones de pertenencia ajustadas para la entrada del error de posición. . . . .	103
6.30. Funciones de pertenencia ajustadas para la entrada del error de ángulo. . . . .	104
6.31. Simulación de posición con controlador difuso de tres entradas ajustado, Set point: 5cm. . . . .	105
6.32. Lugar Geométrico de las Raíces con control proporcional. . . . .	107
6.33. Diagrama de bloques para la simulación con controlador proporcional para el ángulo de la barra. . . . .	108
6.34. Simulación de ángulo de la barra con controlador proporcional, Set point: $45^\circ$ , tiempo de muestreo: 12ms y condiciones iniciales nulas.	108
6.35. Diagrama de bloques para simulación de controlador PD para posición. . . . .	109
6.36. Simulación de posición con controlador PD, Set point: 5cm, tiempo de muestreo: 12ms, $x(0) = 0$ , $\dot{x}(0) = 0$ , $\theta_l(0) = 0$ y $\dot{\theta}_l(0) = 0$ . . . . .	110
6.37. Respuesta del sistema para controlador difuso ajustado y control PD de posición. . . . .	111
7.1. Evento de datos UART1. . . . .	113
7.2. Interrupciones de Timers. . . . .	115
7.3. Respuesta del sistema real para controlador PD. . . . .	118
7.4. Respuesta del sistema real para controlador difuso sin ajustar. . . . .	119
7.5. Respuesta del sistema real para controlador difuso ajustado. . . . .	120
7.6. Respuesta del sistema real para controlador difuso ajustado y controlador difuso sin ajustar. . . . .	121

7.7. Respuesta del sistema real para controlador difuso ajustado y controlador PD . . . . .	121
4.1. Ubicación de componentes en PCB . . . . .	A-84

## LISTA DE TABLAS

2.1. Condiciones y Restricciones para el Ajuste de las Funciones de Pertenencia (Tremante, 2019) . . . . .	39
3.1. Componentes del módulo BB01. . . . .	44
3.2. Componentes del módulo SRV02 . . . . .	46
3.3. Entradas y salidas de módulo VoltPAQ-X1. . . . .	49
4.1. Elementos del módulo BB01. . . . .	53
4.2. Elementos del módulo SRV02 . . . . .	58
5.1. Especificaciones del Arduino UNO. . . . .	66
5.2. Especificaciones del ESP32 . . . . .	67
5.3. Especificaciones Raspberry Pi 3 B+ . . . . .	69
5.4. Componentes para el acondicionamiento de señal de posición. . .	77
6.1. Parámetros de las funciones de pertenencia para la entrada del error del ángulo. . . . .	82
6.2. Parámetros de las funciones de pertenencia de la derivada del error del ángulo. . . . .	83
6.3. Parámetros de las funciones de pertenencia para la salida del controlador difuso para el ángulo de la barra. . . . .	83
6.4. Base de reglas para el controlador difuso del ángulo de la barra. .	84
6.5. Desempeño del sistema con controlador difuso del ángulo de la barra.	86
6.6. Parámetros de las funciones de pertenencia para la entrada del error de posición. . . . .	88

6.7. Parámetros de las funciones de pertenencia para la entrada de la derivada del error de posición. . . . .	88
6.8. Parámetros de las funciones de pertenencia para la salida del controlador difuso de la posición. . . . .	89
6.9. Base de reglas para el controlador difuso de la posición. . . . .	89
6.10. Desempeño del sistema con controlador difuso de la posición. . . . .	91
6.11. Parámetros de las funciones de pertenencia ajustadas para la entrada del error de posición. . . . .	93
6.12. Parámetros de las funciones de pertenencia ajustadas para la entrada de la derivada del error de posición. . . . .	93
6.13. Desempeño del sistema para controlador difuso de posición ajustado.	94
6.14. Parámetros de las funciones de pertenencia del controlador difuso de tres entradas para la entrada de error de posición. . . . .	97
6.15. Parámetros de las funciones de pertenencia del controlador difuso de tres entradas para la entrada de la derivada del error de posición. .	97
6.16. Parámetros de las funciones de pertenencia del controlador difuso de tres entradas para la entrada de error del angulo. . . . .	98
6.17. Parámetros de las funciones de pertenencia de la salida del controlador difuso de tres entradas. . . . .	99
6.18. Base de reglas para el controlador difuso de tres entradas, error de posición -2. . . . .	99
6.19. Base de reglas para el controlador difuso de tres entradas, error de posición -1. . . . .	99
6.20. Base de reglas para el controlador difuso de tres entradas, error de posición 0. . . . .	100
6.21. Base de reglas para el controlador difuso de tres entradas, error de posición 1. . . . .	100

6.22. Base de reglas para el controlador difuso de tres entradas, error de posición 2 . . . . .	100
6.23. Desempeño del sistema con el controlador difuso de tres entradas. . . . .	102
6.24. Parámetros de las funciones de pertenencia del controlador difuso de tres entradas ajustado para la entrada de error de posición. . . . .	104
6.25. Parámetros de las funciones de pertenencia del controlador difuso de tres entradas ajustado para la entrada de error del angulo. . . . .	104
6.26. Desempeño del sistema con el controlador difuso de tres entradas ajustado. . . . .	105
6.27. Desempeño del sistema con el controlador proporcional. . . . .	108
6.28. Desempeño del sistema con el controlador PD. . . . .	110
6.29. Desempeño del sistema para ambos controladores. . . . .	111
 7.1. Desempeño del sistema real con el controlador PD. . . . .	118
7.2. Desempeño del sistema real con el controlador difuso sin ajustar. . . . .	119
7.3. Desempeño del sistema real con el controlador difuso ajustado. . . . .	120
7.4. Desempeño del sistema real para ambos controladores. . . . .	122

## LISTA DE ACRÓNIMOS

**ADC:** Analog-to-Digital Converter.

**BLE:** Bluetooth Low Energy.

**CPU:** Central processing unit.

**CSI:** Camera Serial Interface.

**DAC:** Digital-to-Analog Converter.

**DSI:** Display Serial Interface.

**EEPROM:** Electrically Erasable Programmable Read-Only Memory.

**eFLL:** Embedded Fuzzy Logic Library.

**GPIO:** General Purpose Input/Output.

**GPU:** Graphics Processing Unit.

**GPL:** General public license.

**HDMI:** High-Definition Multimedia Interface.

**I2C:** Inter-Integrated Circuit.

**I2S:** Inter-IC Sound.

**ICSP:** In Chip Serial Programme.

**IDE:** Integrated Development Environment.

**IoT:** Internet of Things.

**PD:** Proporcional Derivativo.

**PWM:** Pulse-Width Modulation.

**SPI:** Serial Peripheral Interface.

**SRAM:** Static Random Access Memory.

**UART:** Universal Asynchronous Receiver Transmitter.

**USB:** Universal Serial Bus.

**ROM:** Read Only Memory.

**RTC:** Real Time Clock.

## INTRODUCCIÓN

El control difuso tiene sus orígenes desde la publicación del artículo “Fuzzy Sets” (Zadeh, 1965) por Lofti Zadeh profesor de Ingeniería Eléctrica y Ciencias de la Computación de la Universidad de Berkeley; este escrito trata sobre la teoría de conjuntos difusos, en la cual se define la pertenencia parcial de los elementos en un conjunto en un rango continuo entre “0” y “1”. La pertenencia parcial es una característica que permite plantear la imprecisión en donde se encuentran los elementos, haciendo que la teoría difusa se asemeje a la lógica del ser humano, ya que admite implicitamente la existencia de la incertidumbre.

En principio, Zadeh fue profesor de la teoría clásica de control, donde observó que la mayoría de los sistemas de control estaban orientados a buscar la precisión y se basaban en aproximaciones matemáticas, debido a ello no se manejaban sistemas complejos que tuvieran incertidumbre, por tal razón desarrolló su teoría de conjuntos difusos.

Entre los años 1974 y 1975 se implementó por primera vez un controlador difuso hecho por Mamdani y Assilian del Queen Mary College para controlar una máquina de vapor, Mamdani en primer lugar público “*Aplication of a fuzzy algorithms for control a simple dynamic plant*” (Mamdani, 1974), en este artículo muestra el esquema de un algoritmo difuso aplicado a la máquina de vapor, luego se expusieron los resultados de este controlador en el artículo “*An experiment in linguistic synthesis with a fuzzy logic controller*” (Mamdani y Assilian, 1975), donde se encontró que el controlador fue muy fácil de construir y los resultados fueron satisfactorios. Pero el mayor impacto de la lógica difusa se dio en Japón en los años ochenta, donde se aplicó el control difuso en gran cantidad de productos, el

principal motivo de la aplicación de este tipo de controlador en Japón fue debido a la ventaja de no necesitar un modelo de la planta para diseñar el controlador difuso y su fácil construcción. Una de las aplicaciones más importantes fue el control difuso predictivo por Yasunobo y Miyamoto (Yasunobo y Miyamoto, 1983), con este trabajo de investigación la empresa Hitachi hizo un sistema de control difuso para el metro de Sendai con el propósito de regular los cambios de velocidad, no solo en el momento de producirse sino anticiparse a ellos.

Los sistemas difusos hoy en día han avanzado en la teoría de control aportando gran variedad de técnicas de control, tales como: controladores neurodifusos, controladores PID difusos, control difuso supervisorio, entre otros. Este tipo de controladores como ya se mencionó, están diseñados sin un modelo matemático del proceso a controlar, característica que llama la atención para ser implementado en plantas complejas, que sean no lineales e inestables, como por ejemplo el sistema Ball an Beam.

El sistema Ball and Beam es el sistema de laboratorio más usado para verificar técnicas de control (Sameera, Arun, y Mathew, 2017), es decir, el propósito de este tipo de planta es académico, por lo tanto es conocido en la teoría de control al ser un desafío para cualquier controlador, debido a su no linealidad e inestabilidad, características que se aprovechan para evaluar el desempeño de cualquier controlador. Quanser es una empresa que se encarga de promover la enseñanza de los sistemas de control, esta empresa posee gran cantidad de maquetas que modelan sistemas de control, como el péndulo invertido, el doble péndulo, el Ball and Beam, entre otros. En la Escuela de Ingeniería Eléctrica de la Universidad Central de Venezuela se encuentra un sistema Ball and Beam de Quanser, el cual consta en una esfera metálica y una barra acoplada a un servo motor que se encarga de inclinar la barra para mover la esfera de posición, en otras palabras, el sistema de control de este tipo se basa en colocar la esfera en una posición

deseada. Es necesario para controlar este tipo de sistemas un microcontrolador con al menos dos entradas analógicas para medir donde se encuentra la esfera y la barra (mediante el ángulo que forma respecto a la horizontal). Además, se necesita una salida analógica para interactuar con el motor y proporcionar movimiento a la barra.

En este trabajo se diseñaron distintos controladores difusos para el control del sistema Ball and Beam de Quanser. El desarrollo del trabajo se ha estructurado en 7 capítulos donde se detalla desde la descripción de la planta, el diseño del hardware de control, las topologías de control, los controladores difusos, las simulaciones, el proceso de ajuste, la implementación y la evaluación del desempeño del sistema para cada controlador.

El capítulo 1 trata sobre el planteamiento del problema y se muestran los objetivos establecidos para este trabajo.

El capítulo 2 introduce el contenido teórico que fue necesario para poder realizar el diseño de un controlador difuso, donde los conceptos más importantes son: lógica difusa, sistema difuso, el método de patrón de búsqueda y el ajuste de funciones de pertenencia.

En el capítulo 3 se hace el estudio sobre el sistema Ball and Beam de Quanser describiendo los módulos que lo componen y se hace énfasis en cada una de las partes que conforman estos módulos y la función que cumplen en el sistema.

El capítulo 4 se desarrollan las ecuaciones matemáticas partiendo del comportamiento físico del sistema y se obtienen las ecuaciones que rigen cada uno de los módulos del sistema.

En el capítulo 5 se selecciona el microcontrolador adecuado para llevar el control del sistema, así como se muestra el diseño de hardware necesario para

acondicionar las señales del Ball and Beam.

El capítulo 6 trata sobre el diseño de dos topologías de control para el sistema Ball and Beam, donde para la primera topología (topología de doble lazo) existen dos casos el primer caso consta de dos controladores difusos y el segundo se diseña un controlador proporcional y un PD. Por otro lado, para la segunda topología se diseña un controlador difuso de tres entradas. Además, en este capítulo se realiza el proceso de ajuste de las funciones de pertenencia de dos controladores difusos (uno de la primera topología y otro de la segunda), para luego hacer la comparación mediante simulación entre los controladores diseñados y obtener con que controlador se obtiene un mejor desempeño.

En el capítulo 7 se implementan los controladores diseñados para la topología de control de doble lazo, donde son evaluados a través de MATLAB® en tiempo real, observando con que controlador se obtiene un mejor desempeño del sistema.

# CAPÍTULO I

## PLANTEAMIENTO DEL PROBLEMA Y OBJETIVOS

### 1.1. PLANTEAMIENTO DEL PROBLEMA

El funcionamiento del sistema Ball and Beam se basa en posicionar una esfera en un lugar exacto sobre una barra, donde la ecuación que define el movimiento de la esfera depende del desplazamiento angular de la barra, cuya relación representa una ecuación no lineal. Además, este sistema tiene la característica de ser inestable, debido a estos factores hace que el sistema Ball and Beam sea un reto para cualquier controlador, por lo tanto es ideal para comprobar el desempeño de cualquier técnica de control.

La Escuela de Ingeniería Eléctrica de la Universidad Central de Venezuela posee un sistema Ball and Beam del fabricante Quanser, el cual está conformado por un tacómetro, potenciómetro para identificar la posición del motor, encoder y un motor acoplado a la barra por medio de engranajes. Asimismo, también tiene un módulo encargado de monitorear la posición de la pelota por un sensor cuyo funcionamiento es análogo al de un potenciómetro y una fuente que se encarga de alimentar los sensores y el motor.

Ahora bien, sabiendo que este tipo de planta se usa prácticamente para comprobar cualquier tipo de controlador siempre y cuando se tenga acceso a sus sensores y actuadores como es el caso, surge la inquietud de diseñar un controlador difuso para el sistema Ball and Beam con el propósito de verificar su funcionamiento.

to y su desempeño a través de la comparación con un controlador convencional y adquirir conocimientos más allá de la teoría de control, sobre controladores difusos y su aplicación experimentalmente.

## **1.2. OBJETIVO GENERAL**

Diseñar un controlador difuso para el sistema Ball and Beam de Quanser.

## **1.3. OBJETIVOS ESPECÍFICOS**

- Describir el funcionamiento e identificar las variables que rigen al sistema Ball and Beam de Quanser.
- Desarrollar las ecuaciones matemáticas que describen la planta.
- Seleccionar y adecuar el hardware necesario para realizar el sistema de control de la planta.
- Diseñar distintas topologías para controlar la planta con un controlador difuso.
- Ajustar los parámetros necesarios del controlador difuso para obtener un buen desempeño del sistema.
- Implementar un controlador convencional y el controlador difuso ajustado en el sistema Ball and Beam.
- Evaluar el desempeño del sistema para ambos controladores.

## CAPÍTULO II

### MARCO TEÓRICO

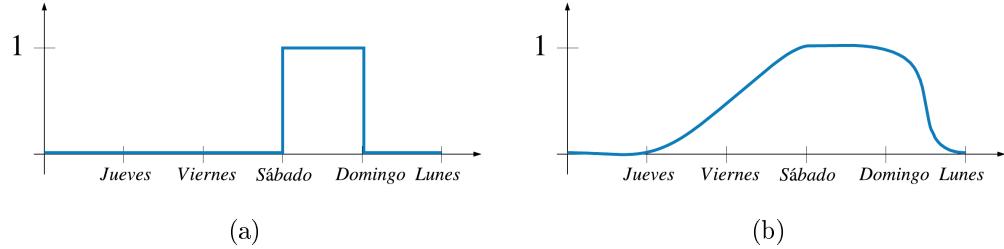
#### 2.1. TEORÍA DIFUSA

##### 2.1.1. CONJUNTO DIFUSO

Un conjunto difuso es un conjunto sin algo tajante, sin una frontera claramente definida. Puede contener elementos con un grado de pertenencia parcial (Tremante, 2006), donde el grado de pertenencia es determinado por una función de pertenencia y su valor se encuentra entre “0” y “1”. La teoría de conjuntos difusos es una extensión de la teoría de conjuntos clásicos (Zadeh, 1965). Existen varias situaciones que se pueden describir con la teoría de conjuntos difusos. Por ejemplo, el conjunto de los días del fin de semana, donde el día viernes puede estar contenido parcialmente en el conjunto dependiendo a la persona que se le pregunte, dado que para algunas personas el fin de semana comienza el viernes en la tarde o en la noche.

En la figura 2.1 se observan dos gráficas, en ambos casos, el eje horizontal representa al conjunto de los días de la semana y el eje vertical indica el grado de pertenencia al conjunto del fin de semana. En la figura 2.1a se puede ver como se representaría el conjunto del fin de semana usando la teoría de conjuntos clásicos, con una función con dos saltos discontinuos, una del viernes al sábado y otra del domingo al lunes, mientras que en la figura 2.1b se observa que la teoría de conjuntos difusos es más flexible, representando al fin de semana con una función

continua cuyo valor evaluado en el día viernes esta comprendido entre “0” y “1”, emulando una pertenencia parcial al conjunto.



**Figura 2.1.** Funciones de los días del fin de semana

La definición matemática para un conjunto difuso realizada por Zadeh en su publicación “*Fuzzy Sets*” (Zadeh, 1965) es:

Dado  $X$  un espacio de puntos, con un elemento genérico  $x$ . Así,  $X = \{x\}$ . Un conjunto difuso  $A$  en  $X$  está caracterizado por una función  $f_A(x)$  (o  $\mu_A(x)$ ), la cual asocia cada punto de  $X$  a un número real entre  $[0,1]$ , el valor de  $x$  en  $f_A(x)$  representa el “grado de pertenencia” de  $x$  en  $A$ . Así, el valor de 1 en  $f_A(x)$  es el más alto grado de pertenencia de  $x$  en  $A$ .  $X$  se conocen como el universo de discurso y  $f_A(x)$  (o  $\mu_A(x)$ ) se conocen también como función de membresía, función característica o simplemente función de pertenencia. En otras palabras, un conjunto difuso en un universo de discurso  $X$  está caracterizado por una función de pertenencia  $\mu_A(x)$  que toma valores en el intervalo  $[0, 1]$ .

Simbólicamente, un conjunto difuso  $A$  en  $X$  se suele denotar como un conjunto de pares ordenados de un elemento genérico  $x$  y su grado de pertenencia en la función  $\mu_A(x)$  esto es:

$$A = \{ (x, \mu_A(x)) \mid x \in X \} \quad (2.1)$$

Otra notación conveniente de  $A$  cuando el universo  $X$  es continuo ( $X = \mathbb{R}$ ), es:

$$A = \int_x \mu_A(x)/x \quad (2.2)$$

Donde el signo de la integral no denota integración, denota a todos los pares  $\mu_A(x)/x$ ,  $\forall x \in X$  es decir, esto denota la colección de todos los puntos  $x \in X$  con la función de pertenencia asociada a  $\mu_A(x)$ .

### 2.1.2. LÓGICA DIFUSA

Por lo general el ser humano tiende a usar adverbios de cantidad en el habla diaria como: “muy”, “poco”, “mas o menos”, “medio” o “demasiado”. Asimismo, estas palabras se utilizan para referirse a alguna cualidad de un objeto o persona, por ejemplo: “muy caliente”, “muy alto” o “medio cerca”, donde esta clase de términos implícitamente incorporan incertidumbre o vaguedad en lo que en realidad se quiere decir, ya que no son del todo claros y siempre van a depender de una referencia. La lógica difusa es la lógica que emula esta clase de comportamiento, definiendo dichos términos (términos lingüísticos) a través de funciones de pertenencia que asignan a un elemento del universo de discurso (rango de valores permitidos) un grado de pertenencia en un conjunto. La lógica difusa parte de la teoría de conjuntos difusos, donde un elemento puede pertenecer de forma parcial o absoluta a un conjunto, permitiendo la existencia de la incertidumbre.

## 2.2. FUNCIÓN DE PERTENENCIA

Dado un conjunto  $A$  contenido en el universo de discurso  $X$ , cuyos elementos se denotan por  $x$ , existe una función de pertenencia  $\mu_A(x)$  que determina la pertenencia de un elemento  $x$  en el conjunto  $A$  y dicho valor está comprendido entre 0 y 1.

$$0 \leq \mu_A(x) \leq 1 \quad x \in X \quad (2.3)$$

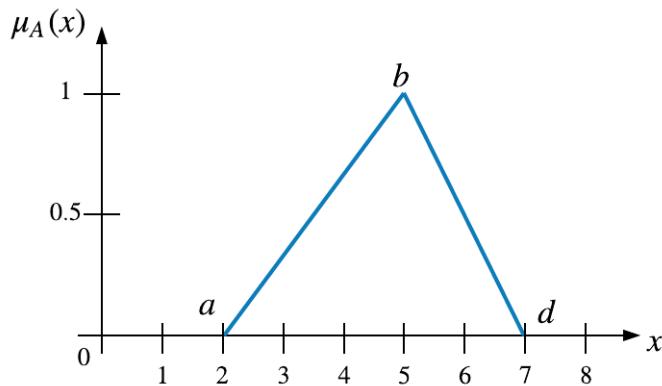
### 2.2.1. TIPOS DE FUNCIONES DE PERTENENCIA

Existe gran variedad de formas de representar una función de pertenencia las más comunes son: triangular, trapezoidal, Gaussiana, curvas sigmoides, curvas polinómicas: cuadráticas y cúbicas. Sin embargo, en la teoría de control las funciones mayormente usadas y que han obtenido buenos resultados son la triangular y la trapezoidal (Tremante, 2001).

### 2.2.2. FUNCIÓN DE PERTENENCIA TRIANGULAR

El tipo de función de pertenencia triangular es una función lineal a trozos representada en forma cartesiana, es decir, se define a través de tres puntos sobre el eje de coordenadas, partiendo de esos puntos se determinan las ecuaciones que definen a las rectas de un triángulo. Estos tres puntos se denotan con  $a$ ,  $b$  y  $d$ , como se muestra en la figura 2.4 Estos tres puntos deben cumplir con la condición:  $a < b < d$ . La ecuación que define esta función se muestra a continuación:

$$\mu_A(x) = \begin{cases} 0, & \forall x < a \\ \frac{x-a}{b-a}, & \forall a \leq x < b \\ \frac{d-x}{d-b}, & \forall b \leq x < d \\ 0, & \forall x \geq d \end{cases} \quad (2.4)$$



**Figura 2.2.** Función de pertenencia triangular.

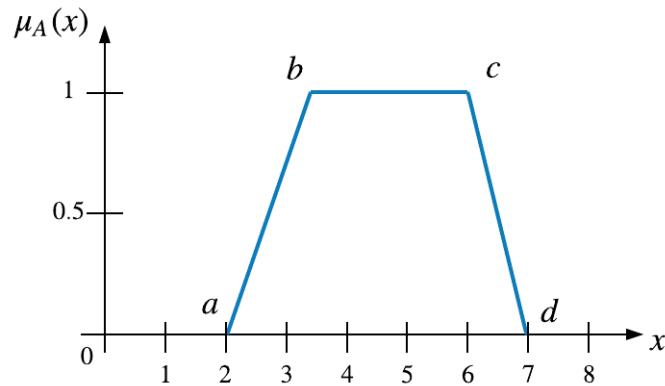
De la ecuación 2.4 y la figura 2.2 se puede observar que el parámetro “ $a$ ” se refiere al extremo izquierdo del triángulo, “ $b$ ” a la cima y “ $d$ ” el extremo derecho. Una característica importante de este tipo de funciones, es que un elemento en  $x$  alcanza el mayor grado de pertenencia en el conjunto  $A$  solo si este es igual a  $b$ .

### 2.2.3. FUNCIÓN DE PERTENENCIA TRAPEZOIDAL

La función de pertenencia trapezoidal se define de la misma forma que la triangular, con la diferencia que se añade un nuevo punto “ $c$ ” que expande el

dominio de los elementos que alcanzan el máximo grado de pertenencia (ver figura 2.3) y su ecuación matemática es:

$$\mu_A(x) = \begin{cases} 0, & \forall x < a \\ \frac{x-a}{b-a}, & \forall a \leq x < b \\ 1, & \forall b \leq x < c \\ \frac{d-x}{d-b}, & \forall c \leq x < d \\ 0, & \forall x \geq d \end{cases} \quad (2.5)$$



**Figura 2.3.** Función de pertenencia Trapezoidal.

#### 2.2.4. SINGLETON DIFUSO

Un singlenton difuso es aquel que alcanza el mayor grado de pertenencia solo en un valor  $x_i$  y en el resto del universo de discurso es cero, como se muestra en la siguiente ecuación:

$$\mu_A(x) = \begin{cases} 1, & \forall x = x_i \\ 0, & \forall x \neq x_i \end{cases} \quad (2.6)$$

## 2.3. OPERACIONES ENTRE CONJUNTOS DIFUSOS

Al igual que los conjuntos clásicos, los conjuntos difusos tienen operaciones entre conjuntos. Las operaciones más comunes son:

### 2.3.1. IGUALDAD

Dados dos conjuntos difusos  $A$  y  $B$  contenidos en el universo de discurso  $X$ , cuyas funciones de pertenencia son  $\mu_A(x)$  y  $\mu_B(x)$  respectivamente, son iguales si y sólo si:

$$\mu_A(x) = \mu_B(x) \quad \forall x \in X \quad (2.7)$$

### 2.3.2. INCLUSIÓN

Un conjunto difuso  $A$  está contenido o es subconjunto de un conjunto difuso  $B$  ( $A \subset B$ ) si y sólo si:

$$\mu_A(x) \leq \mu_B(x) \quad \forall x \in X \quad (2.8)$$

### 2.3.3. COMPLEMENTO

Dado un conjunto difuso  $A$  en el universo de discurso  $X$ , su complemento se denota como  $\bar{A}$  y es un conjunto difuso cuya función de pertenencia se define:

$$\mu_{\bar{A}}(x) = 1 - \mu_A(x) \quad \forall x \in X \quad (2.9)$$

### **2.3.4. UNIÓN**

Dados dos conjuntos difusos  $A$  y  $B$  en el universo de discurso  $X$ , su unión se denota como  $A \cup B$  y se define como:

$$\mu_A(x) \cup \mu_B(x) = \mu_{A \cup B}(x) = \max\{\mu_A(x), \mu_B(x)\} \quad \forall x \in X \quad (2.10)$$

### **2.3.5. INTERSECCIÓN**

Dados dos conjuntos difusos  $A$  y  $B$  en el universo de discurso  $X$ , su intersección se denota como  $A \cap B$  y se define como:

$$\mu_A(x) \cap \mu_B(x) = \mu_{A \cap B}(x) = \min\{\mu_A(x), \mu_B(x)\} \quad \forall x \in X \quad (2.11)$$

## **2.4. VARIABLE LINGÜÍSTICA**

Una variable lingüística se define como la variable que toma como valor palabras, frases u oraciones en un lenguaje natural o artificial. A estos valores se les conocen como valores lingüísticos o términos lingüísticos (Passino y Yurkovich, 1998). Por ejemplo, la temperatura es una variable lingüística y los distintos valores que ésta puede tener se pueden representar a través de términos lingüísticos, es decir, una temperatura de 40 °C puede definirse como una temperatura alta, donde alta es un término lingüístico. La forma matemática de representar a la variable lingüística es a través de un conjunto difuso, por tanto, una variable lingüística es un conjunto difuso, por supuesto un valor o término lingüístico también será

un conjunto difuso, pero, es un subconjunto de la variable lingüística (Tremante, 2006).

Para especificar las variables en un sistema difuso se denotan la variables lingüísticas como:

$$x_i \in X_i \quad (2.12)$$

Donde  $i = 1, 2, \dots, n$  denota el número de la variable de entrada y  $X_i$  representa el universo de discurso. Las variables lingüísticas de salida se denotan como:

$$y_i \in Y_i \quad (2.13)$$

Donde  $i = 1, 2, \dots, m$  denotan el número de la variable de salida.

Los términos lingüísticos se denotan como:

$$A_i^j \quad (2.14)$$

Siendo  $j$  el  $j^{th}$  término lingüístico de la variable lingüística  $x_i$  en el universo de discurso  $X_i$ . Si existen muchos términos lingüísticos definidos en  $X_i$ , entonces la variable lingüística  $x_i$  tomará varios términos lingüísticos denotados por:

$$A_i = \{A_i^j : j = 1, 2, \dots, N_i\} \quad (2.15)$$

Por ejemplo, sea  $A_1$  = “posición”, entonces sus términos lingüísticos son:

$$A_1^1 = \text{derecha}$$

$$A_1^2 = \text{centro}$$

$$A_1^3 7 = \text{izquierda}$$

## 2.5. OPERADORES LÓGICOS

Además de las operaciones lógicas previamente definidas en las ecuaciones **2.10** y **2.11**, las cuales definen la unión e intersección respectivamente, existen otras operaciones definidas por los operadores S-Norma para la unión y T-Norma para la intersección, esto debido a que en ciertos casos las operaciones clásicas de unión e intersección entre conjuntos difusos no generan el impacto deseado en un sistema.

### 2.5.1. OPERADOR S-NORMA

La unión de dos conjuntos difusos  $A$  y  $B$  se denota por un función  $S$ , tal que:

$$\mu_{A \cup B}(x) = S(\mu_A(x), \mu_B(x)) = \mu_A(x) \oplus \mu_B(x) \quad (2.16)$$

Donde  $\oplus$  es un operador binario para la función  $S$ . Estos operadores de la unión difusa a menudo se llama S-norma operadores que reúne los requisitos básicos siguientes:

Un operador S-norma es una función  $S$  definida en  $S : [0, 1] \times [0, 1] \rightarrow [0, 1]$ , lo que quiere decir que es una función de dos dimensiones y está contenida en el cuadrado unitario  $[0, 1] \times [0, 1]$  a valores  $[0, 1]$  que satisface las siguientes propiedades:

$$\text{Límite: } S(1, 1) = 1, S(a, 0) = S(0, a) = a$$

Monotonía:  $S(a, b) \leq S(c, d)$  si  $a \leq c$  y  $b \leq d$

Commutativa:  $S(a, b) = S(b, a)$

Asociativa:  $S(a, S(b, c)) = S(S(a, b), c)$

Donde  $a, b, c, d$  son funciones de pertenencia  $\mu_a, \mu_b, \mu_c$  y  $\mu_d$  respectivamente.

Algunas operaciones asociadas con las S-norma son: el operador clásico o suma clásica, la suma difusa Dombi, la suma difusa Dubois-Prade, la suma difusa Yager, la suma drástica, suma Einstein, la suma algebraica y la suma acotada (Tremante, 2001). La definición del operador clásico para todos los valores de  $(a, b)$  entre  $[0, 1]$  es:

$$S_c(a, b) = \max(a, b) \quad (2.17)$$

### 2.5.2. OPERADOR T-NORMA

La intersección de dos conjuntos difusos  $A$  y  $B$  está especificada por una función  $T$ :

$$\mu_{A \cap B}(x) = T(\mu_A(x), \mu_B(x)) = \mu_A(x) \otimes \mu_B(x) \quad (2.18)$$

Donde  $\otimes$  es un operador binario para la función  $T$ . Estos operadores de la intersección difusa normalmente son llamados operadores T-norma, que reúne los requisitos básicos siguientes:

Un operador T-Norma es una función  $T$  definida en  $T : [0, 1] \times [0, 1] \rightarrow [0, 1]$ , es decir, es una función de dos dimensiones definida sobre el cuadrado unitario cerrado  $[0, 1] \times [0, 1]$  a valores  $[0, 1]$  que satisface las siguientes propiedades:

Límite:  $T(0, 0) = 0, T(a, 1) = T(1, a) = a$

Monotonía:  $T(a, b) \leq T(c, d)$  si  $a \leq c$  y  $b \leq d$

Commutativa:  $T(a, b) = T(b, a)$

Asociativa:  $T(a, T(b, c)) = T(T(a, b), c)$

Donde  $a, b, c, d$  son funciones de pertenencia:  $\mu_A(x), \mu_B(x), \mu_C(x), \mu_D(x)$  respectivamente.

Algunas operaciones asociadas con la T-norma son: el operador clásico o producto clásico, el producto difuso Dombi, el producto difuso Dubois-Prade, el producto difuso Yager, el producto drástico, el producto Einstein, el producto algebraico y el producto acotado (Tremante, 2001). La definición del operador clásico para todos los valores de  $(a, b)$  entre  $[0, 1]$  es:

$$T_c(a, b) = \min(a, b) \quad (2.19)$$

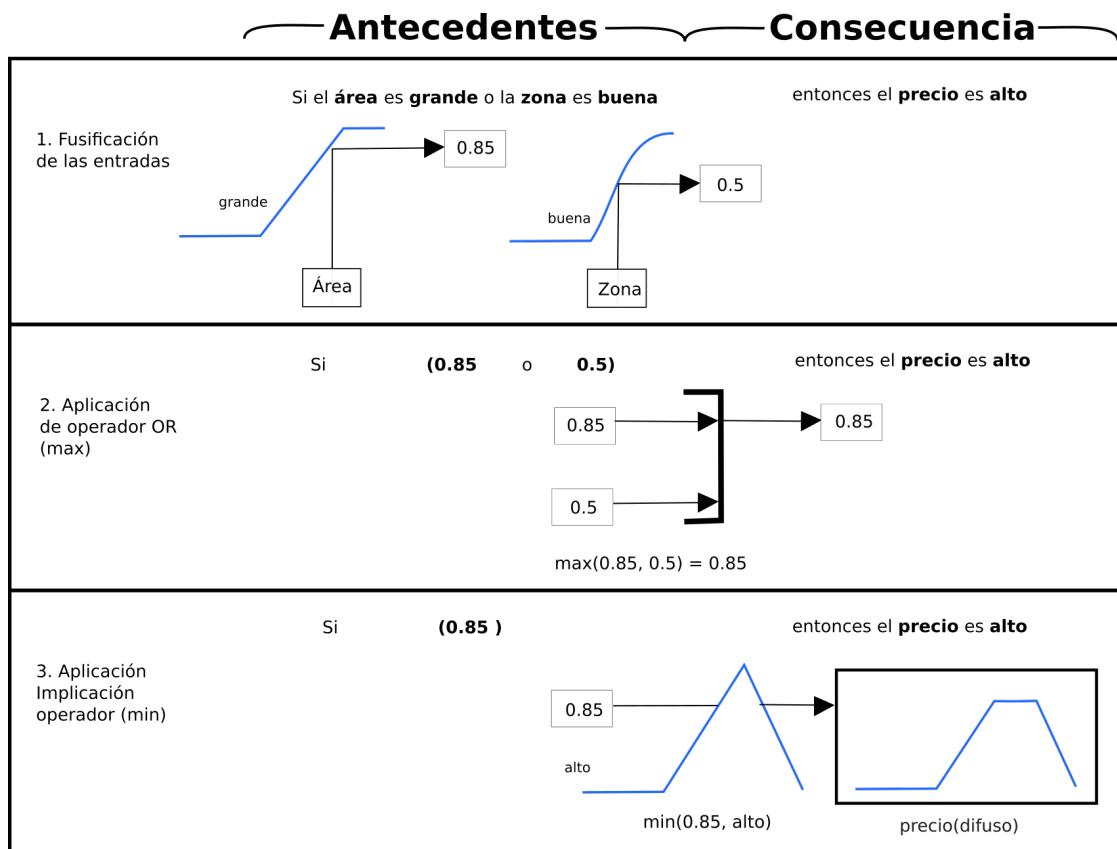
## 2.6. REGLAS SI-ENTONCES

Una parte fundamental en los sistemas difusos son las reglas SI-ENTONCES, estas son las que se relacionan con la toma de decisiones del ser humano, partiendo de una premisa o antecedente para ejecutar una acción o consecuencia, por ejemplo:

$$Si x \text{ es } A \text{ entonces } y \text{ es } B \quad (2.20)$$

Donde “ $x$ ” y “ $y$ ” son variables lingüísticas,  $A$  y  $B$  son términos lingüísticos, siendo la expresión “ $x$  es  $A$ ” el antecedente y “ $y$  es  $B$ ” la consecuencia. La evaluación de un antecedente se conoce como fusificación, donde se determina

el grado de verdad que pueda tener la variable lingüística respecto al término lingüístico y este siempre va a estar comprendido entre 0 y 1 para luego relacionarse con la salida, lo que se conoce como implicación. Es importante destacar, que la implicación no es más que una función que modifica un conjunto difuso a través del antecedente de la regla. La manera más común de modificar el conjunto difuso de la salida es truncarlo usando el operador *min*, el cual es ampliamente utilizado en el control difuso, en los sistemas difusos y en especial en el control difuso (Tremante, 2006). Existen reglas SI-ENTONCES, donde se integran más de un antecedente a través de operadores difusos. Asimismo, pueden haber más de una consecuencia. Un ejemplo donde hay múltiples antecedentes y una salida puede ser el precio estimado de una casa dependiendo del área que posee y la zona donde se encuentre, como se puede observar en la figura 2.4.



**Figura 2.4.** Proceso gráfico del resultado de una regla en ejemplo de precio estimado de una casa.

En la figura anterior se observa que las variables lingüísticas son el área, la zona y los términos lingüísticos son grande y buena respectivamente. Al fusificar estas entradas se obtienen dos valores difusos por cada función de pertenencia y al evaluarlos a través de la regla “Si el área es grande o la zona es buena entonces el precio es alto” se relacionan con el término lingüístico alto y así se obtiene un valor difuso del precio de la casa.

## 2.7. SISTEMA DIFUSO

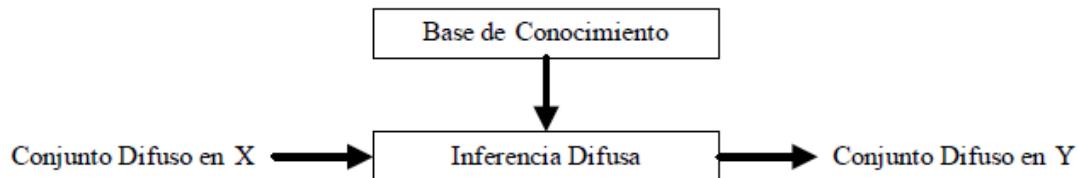
Un sistema difuso es un sistema basado en la lógica difusa, el cual está compuesto por la base de conocimiento, una o varias entradas y salidas. Una entrada o varias entradas son procesadas a través de la base de conocimiento para generar una o varias salidas. Asimismo, la base de conocimiento está constituida por gran cantidad de reglas SI-ENTONCES las cuales provienen de un experto humano conocedor del proceso que se quiere aplicar el sistema difuso. Este sistema debe ser lo suficientemente flexible para abarcar los distintos valores que se pueden presentar en las entradas y lo suficientemente robusto para establecer la salida deseada. Los sistemas difusos están hechos para afrontar sistemas complejos como lo son los sistemas no lineales, sistemas con variables difíciles de medir y sistemas sujetos a grandes perturbaciones.

## 2.8. TIPOS DE SISTEMAS DIFUSOS

### 2.8.1. SISTEMA DIFUSO PURO

Un sistema difuso puro es aquel donde sus entradas y salidas son variables lingüísticas representadas por conjuntos difusos. En la base de conocimientos se encuentran todas las reglas SI-ENTONCES obtenidas por un experto y en la inferencia difusa se procesan estas reglas respecto a las entradas del sistema para

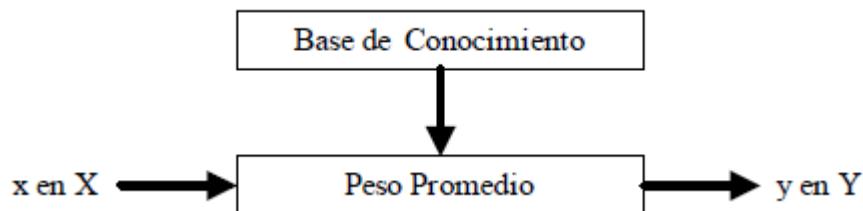
obtener la salida; el diagrama de bloque de este sistema se puede observar en la figura 2.5. Este tipo de sistemas no es el ideal para aplicar en la teoría de control ya que las entradas y salidas de los sistemas reales son valores numéricicos.



**Figura 2.5.** Configuración Básica de un Sistema Difuso Puro.

### 2.8.2. SISTEMA DIFUSO TAKAGI-SUGENO-KANG (TSK)

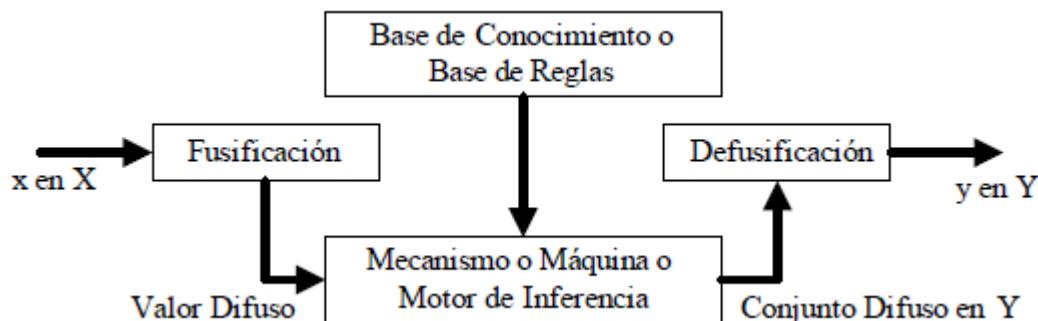
Un sistema difuso TSK se diferencia del sistema difuso puro en que sus entradas y salidas son un escalar y la consecuencia de sus reglas son una relación matemática, donde al procesar las entradas según la base de conocimiento la salida viene dada por el peso promedio de todas las reglas, como se puede ver en la figura 2.6. Sin embargo, en este tipo de sistema se pierde la esencia de los sistemas difusos, ya que la consecuencia viene dada por una relación matemática y no por un término lingüístico, por ende se pierde la versatilidad del sistema difuso.



**Figura 2.6.** Configuración Básica de un Sistema Difuso TSK.

### 2.8.3. SISTEMA DIFUSO CON FUSIFICACIÓN Y DEFUSIFICACIÓN TIPO MAMDANI

El sistema difuso con fusificación y desfusificación tipo Mamdani es aquel donde la entrada (un número) se fusifica, pasa por el motor de inferencia el cual procesa toda las reglas de la base de conocimiento, hace la implicación y defusifica para obtener un escalar a la salida. Este proceso se puede representar con el diagrama de bloques de la figura que se muestra a continuación:



**Figura 2.7.** Configuración Básica y Diagrama de Bloque de un Sistema Difuso con fusificación y defusificación tipo Mamdani.

### 2.8.4. SISTEMA DIFUSO CON FUSIFICACIÓN Y DEFUSIFICACIÓN TIPO SUGENO

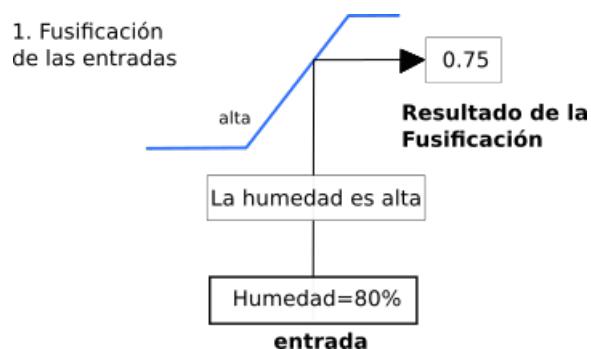
Este tipo de sistema cumple con el mismo diagrama de bloques que se muestra en la figura 2.7 con la diferencia que las funciones de pertenencia de la salida son del tipo singleton difuso (ver ecuación 2.6) o spike, este tipo de sistema posee una ventaja a nivel de cómputo, ya que el proceso de desfusificación es más rápido debido a la simplicidad de la salida.

## 2.9. PARTES DE UN SISTEMA DIFUSO

Un sistema difuso está compuesto por tres etapas, la fusificación, el motor de inferencia y la defusificación, donde el motor de inferencia se subdivide en tres partes que contienen: la aplicación del operador difuso, la aplicación del método de implicación y la composición de reglas.

### 2.9.1. FUSIFICACIÓN

Es el proceso en el cual se determina el grado de pertenencia que la entrada del sistema difuso tiene en los términos lingüísticos a través de las funciones de pertenencia, donde cabe acotar que esta entrada es un valor numérico contenido en el universo de discurso. El grado de pertenencia de este valor de entrada es un valor difuso que será la entrada para la siguiente etapa del sistema. Por ejemplo, en la figura 2.8 se puede observar que la variable lingüística humedad tiene un valor de 80 %, este evaluado sobre el término lingüístico alta tiene un grado de pertenencia de 0,75, siendo el valor de entrada fusificado.



**Figura 2.8.** Fusificación de una entrada.

## 2.9.2. APLICACIÓN DEL OPERADOR DIFUSO

Cuando hay más de un antecedente en una regla SI-ENTONCES de un sistema difuso se deben usar los operadores lógicos difusos, vistos en el apartado 2.5, con el propósito de determinar un valor final entre los grados de pertenencia de los antecedentes. En el caso de un OR se utiliza los operadores S-norma y en el caso de un AND se utiliza los T-norma. Por ejemplo, observando la figura 2.9 hay dos entradas temperatura y humedad las cuales son evaluadas en la funciones de pertenencia alta y baja de cada universo de discurso (los universos de discurso son distintos), el operador que se usa es el clásico de donde se obtiene un valor difuso de 0,75.

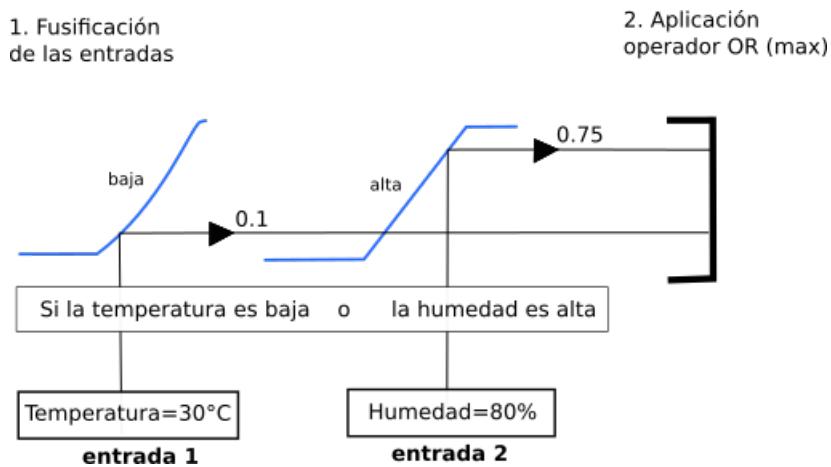


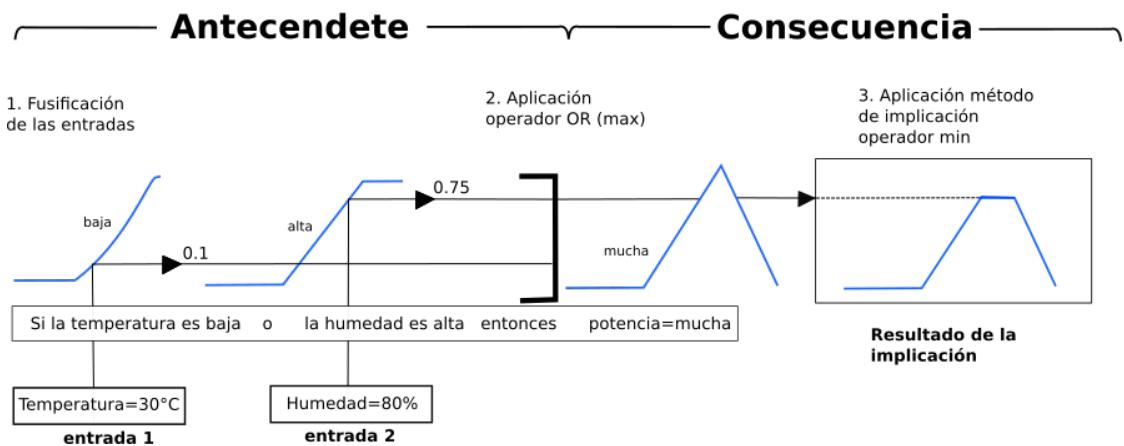
Figura 2.9. Aplicación de operador difuso de suma clásica.

## 2.9.3. APLICACIÓN DEL MÉTODO DE IMPLICACIÓN

Es sumamente importante considerar que las reglas de un sistema difuso poseen un peso comprendido entre cero y uno, pero por lo general este es 1 y no afecta a la implicación. La implicación no es más que el proceso de formar una consecuencia representada por un conjunto difuso según el antecedente. La representación matemática de la implicación se muestra a continuación:

$$\mu_{\hat{B}_q^i}(y_q) = \mu_i(x_1, x_2, \dots, x_n) \otimes \mu_{B_q^p}(y_q) \quad (2.21)$$

Donde  $\hat{B}_q^i$  es el conjunto difuso implícito de la regla  $i^{th}$ ,  $\mu_{\hat{B}_q^i}(y_q)$  especifica el grado de pertenencia de la salida. Por ejemplo, en la figura 2.10 se observa que las entradas temperatura y humedad luego de ser fusificadas y al aplicar el método de implicación con el operador de producto clásico, donde se trunca la función de pertenencia “mucha”.

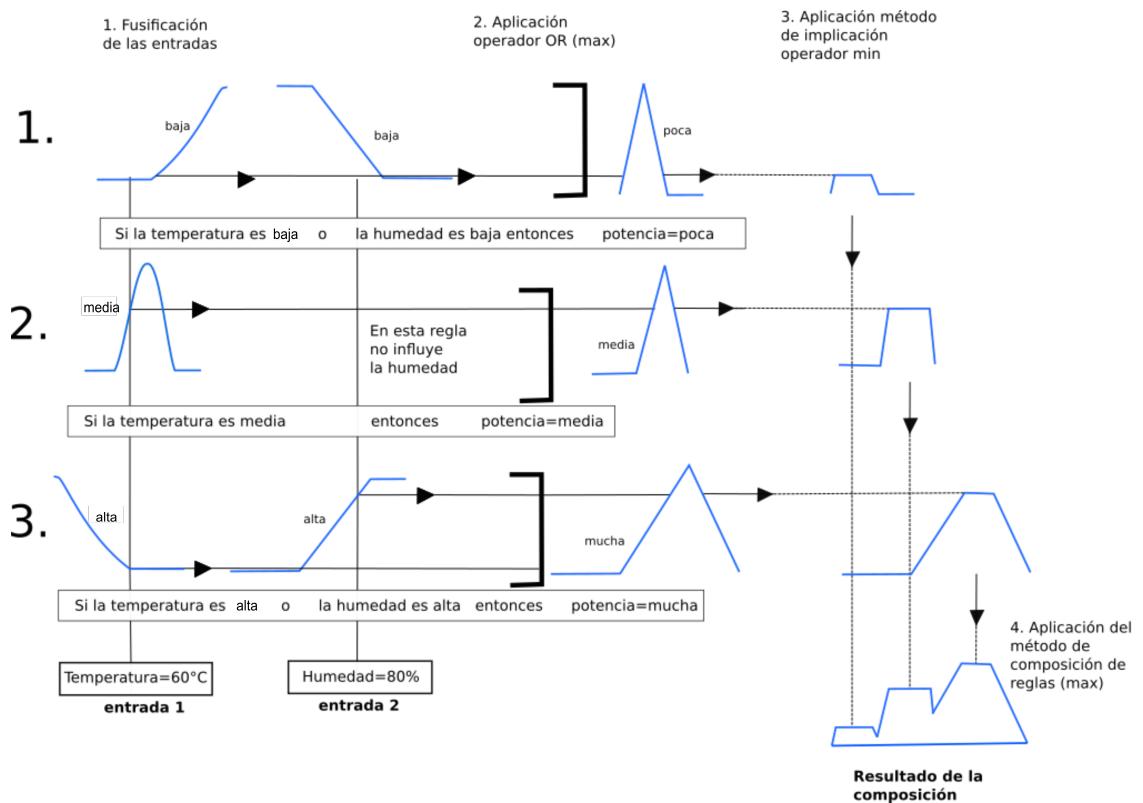


**Figura 2.10.** Método de Implicación

#### 2.9.4. COMPOSICIÓN DE LAS REGLAS

La composición de las reglas es la unión de todos los conjuntos difusos generados en cada regla. Una manera de representar la composición de las reglas de forma matemática es como se muestra en la ecuación 2.22, donde los conjuntos difusos generados por la implicación en cada regla se unen para generar un solo conjunto difuso. Un ejemplo del resultado que se obtiene luego de aplicar las reglas de un sistema difuso se observa en la figura 2.11, cuyas entradas son las variables temperatura, humedad y la salida potencia a disipar por una resistencia.

$$\mu_{\hat{B}_q}(y_q) = \mu_{\hat{B}_q^1}(y_q) \oplus \mu_{\hat{B}_q^2}(y_q) \oplus \dots \oplus \mu_{\hat{B}_q^n}(y_q) \quad (2.22)$$



**Figura 2.11.** Composición de las reglas.

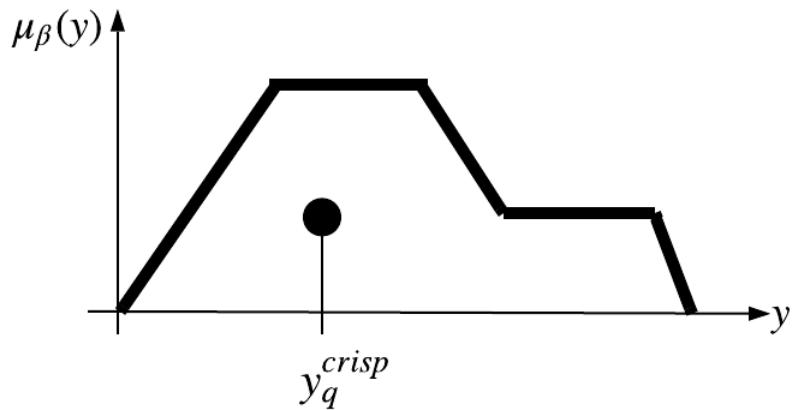
## 2.9.5. DEFUSIFICACIÓN

La defusificación es el proceso final del sistema difuso, este consiste en determinar el valor de la salida (un número) dependiendo de la composición de reglas representada como un conjunto difuso. Existen muchas maneras de difusificar un conjunto difuso pero el más utilizado es el cálculo del centroide (Tremante, 2006).

### 2.9.6. MÉTODO DEL CENTRO DE GRAVEDAD, CENTROIDE O CENTRO DE ÁREA

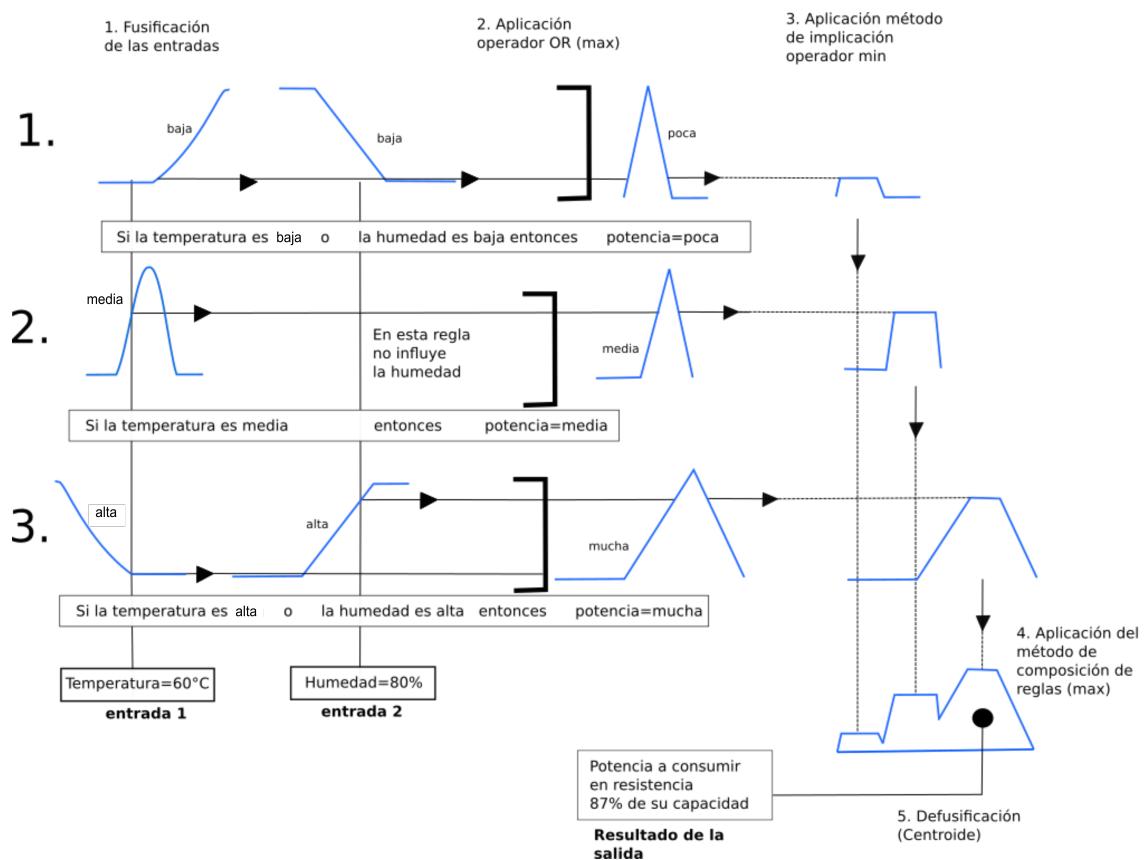
Este método consiste en encontrar el centro de gravedad o punto de equilibrio del conjunto difuso que se obtiene luego de la composición de las reglas y dicho punto será el valor de la salida del sistema difuso; éste se calcula de acuerdo a la ecuación 2.23. Un ejemplo de este valor se puede observar en la figura 2.12 donde es claro que el punto tiende a estar ubicado donde el área es mayor.

$$y_q^{crisp} = \frac{\int_{y_q} y_q \mu_{\hat{B}_q}(y_q) dy_q}{\int_{y_q} \mu_{\hat{B}_q}(y_q) dy_q} \quad (2.23)$$



**Figura 2.12.** Método de defusificación de Centro de Gravedad.

El proceso completo que describe un sistema difuso se puede observar en la figura 2.13, donde las entradas (números) primero se fusifican, se les aplica un operador difuso, se procesan según la base de reglas, luego se implican generando un conjunto difuso y para finalizar se desfusifica obteniendo un valor numérico.



**Figura 2.13.** Diagrama completo de un sistema difuso Tipo Mamdani.

## 2.10. CONTROL DIFUSO

El control difuso es aquel que se basa en la experiencia humana, este método de control no es prioridad conocer la planta de forma analítica, mas bien se analiza de forma cualitativa, con el propósito de obtener información a través de palabras en lugar de relaciones físicas o matemáticas. El control difuso es un sistema difuso en el cual sus entradas son variables numéricas que por lo general son el error entre el valor deseado y la salida y su segunda entrada es la derivada del error para obtener información respecto a la tendencia del error. Las ecuaciones que describen el error y la derivada se pueden observar a continuación:

$$Error = SetPoint - Salida \quad (2.24)$$

$$Derivada\ del\ error = \frac{Error(n) - Error(n - 1)}{T} \quad (2.25)$$

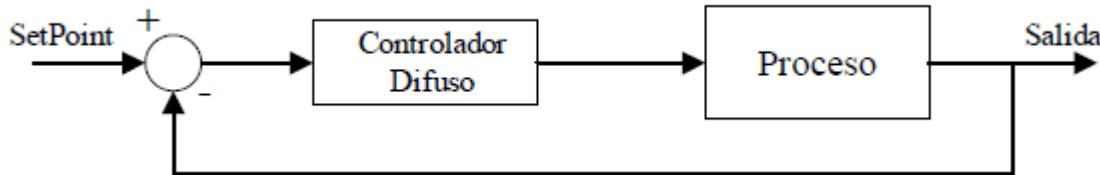
Donde  $Error(n)$  es el error en la muestra actual,  $Error(n - 1)$  es el error de la muestra anterior y  $T$  es el tiempo de muestreo.

En el control difuso siempre debe haber un experto conocedor del proceso que se desea controlar, esta persona debe estar en la capacidad de describir el funcionamiento de la planta con palabras para poder asignar términos lingüísticos que cumplan con la descripción del proceso. Una vez que el experto tenga definido las entradas, las salidas y los términos lingüísticos este debe diseñar la base de reglas que rigen las acciones del controlador, para hacer este diseño el experto debe hacerse ciertas preguntas claves para poder controlar el proceso. Por ejemplo, “¿Qué debo hacer si la temperatura baja un poco y su pendiente es negativa pero pequeña?, debo encender un poco la calefacción para volver al Set point”, con este tipo de interrogantes se procede armar la base de reglas para poder controlar el proceso.

Uno de los factores más importantes que afectan en el desempeño del sistema son las funciones de pertenencia, la manera de cuantificar la base de conocimientos que el experto diseña es a través de las funciones de pertenencia. Una de las desventajas del control difuso es el ajuste de las funciones de pertenencia en la mayoría de los casos viene dado por ensayo y error, donde las funciones de pertenencia toman un papel importante y el experto debe evaluar en qué posición debe colocarlas, qué forma deben tener, entre otros factores.

El diagrama de bloque que frecuentemente se emplea en los sistemas de control

difuso se muestra en la figura 2.14, donde se cierra el lazo con las variables de interés del proceso para proceder a controlar la variable deseada.



**Figura 2.14.** Diagrama de bloques de un controlador difuso en un Sistema a lazo cerrado.

## 2.11. REPRESENTACIÓN ÚNICA DE FUNCIONES DE PERTENENCIA

Una forma general de representar una variedad de funciones de pertenencia es a través de una función única, ésta debe tener la característica de representar varias formas y posiciones que una función de pertenencia pueda tener. Esta función viene dada por la ecuación 2.26 y se puede tener tanto una función triangular como una trapezoidal, incluso se pueden obtener variaciones de estas. Para ello se modifican los términos  $n_T$ ,  $n_1$  y  $n_2$ .

Partiendo de la ecuación 2.4 (función de pertenencia triangular)  $n_1$  es el parámetro que modifica las pendientes de las rectas que van de  $a$  hasta  $b$  y de  $b$  hasta  $c$  (ver figura 2.2). El parámetro de función trapezoidal  $n_T$  modifica el punto de mayor grado de pertenencia y también se puede extender formando un trapecio y el parámetro  $n_2$  modifica la forma de la función de pertenencia permitiendo que las rectas de los extremos de la función sean curvas y no rectas (Tremante, 2019). Por ejemplo, en la figura 2.15 se puede observar variaciones de estos parámetros para obtener distintas funciones de pertenencia.

$$\mu_A(x) = \begin{cases} 0, & \forall x < a \\ \left[1 + n_1 \frac{x - (b - n_T(b - a))}{b - a - n_T(b - a)}\right]^{n_2}, & \forall b - \frac{b - a}{n_1} \leq x < b - n_T(b - a) \\ 1, & \forall b - n_T(b - a) \leq x < c + n_T(d - c) \\ \left[1 + n_1 \frac{(c + n_T(d - c)) - x}{d - b + n_T(d - c)}\right]^{n_2}, & \forall c + n_T(d - c) \leq x < c + \frac{d - c}{n_1} \\ 0, & \forall x \geq d \end{cases} \quad (2.26)$$

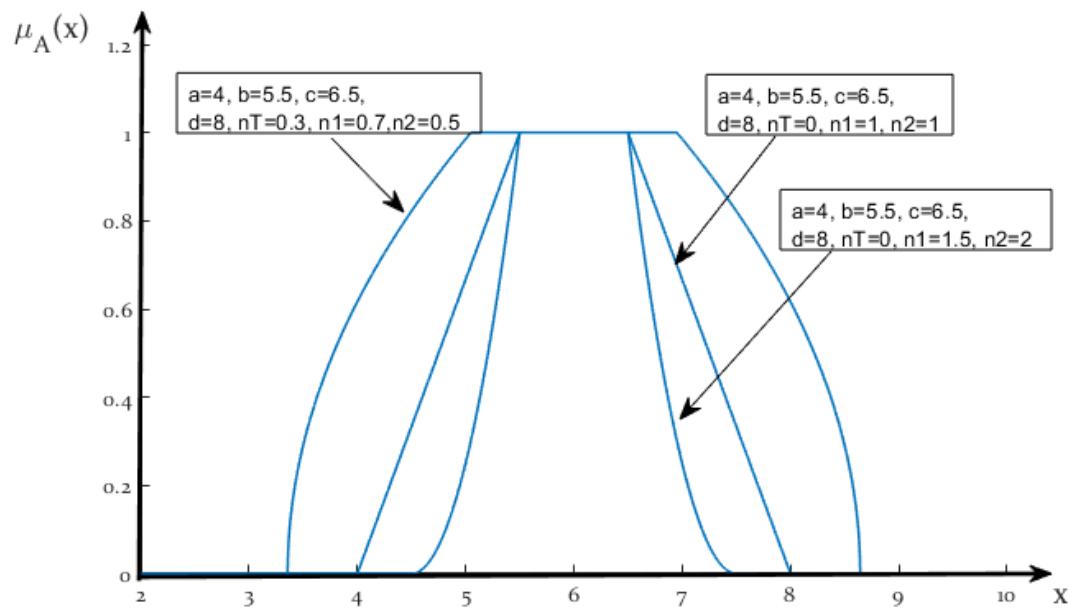


Figura 2.15. Función Única.

## 2.12. FUNCIÓN OBJETIVO Y MULTIOBJETIVO

Una función objetivo se utiliza cuando se emplea una metodología de optimización (Tremante, 2019). Existen gran variedad de funciones objetivos en la teoría de control, donde los parámetros que se consideran siempre van a depender del problema que se este estudiando, pero en general siempre se considera mas de un parámetro por lo tanto la función objetivo se convierte en una función multiobjetivo, estos parámetros son: sobre pico porcentual, tiempo de alza, tiempo de establecimiento y el error en estado estacionario son los factores claves para determinar el desempeño de un sistema. Por ejemplo, una función multiobjetivo que se puede emplear se muestra a continuación:

$$f = w_1 \sum_{i=1}^n e^2(n) + w_2 |E_{ss} - 0,01| + w_3 |s_{max} - r - 0,1| + w_4 \frac{t_s(E_{ss} - 0,01)}{T} + w_5 \frac{t_a}{T} \quad (2.27)$$

Observando la ecuación 2.27,  $\sum_{i=1}^n e^2(n)$  es el error cuadrático de las  $n$  muestras tomadas,  $E_{ss}$  es el error en estado estacionario,  $s_{max}$  es el sobre pico porcentual,  $r$  el valor de referencia o set point,  $t_s$  es el tiempo en el cual se llega al 1% del set point,  $t_a$  es el tiempo de alza,  $T$  es el tiempo empleado en la simulación y los términos  $w_1, w_2, w_3, w_4$  y  $w_5$  representan la ponderación de los parámetros en la función multiobjetivo, donde se cumple que:

$$\sum_{i=1}^n w_n = 1$$

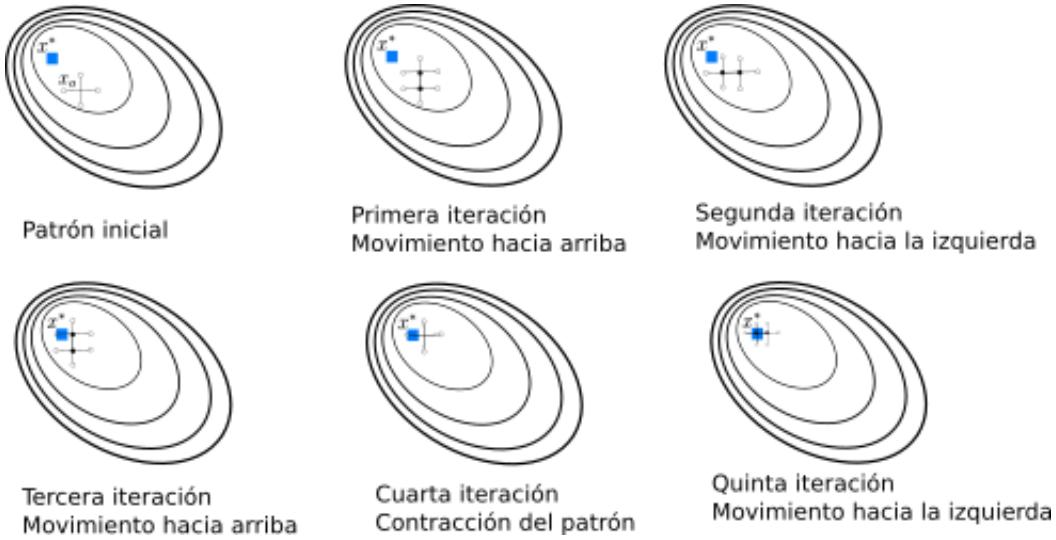
## **2.13. OPTIMIZACIÓN POR BÚSQUEDA DIRECTA**

Los métodos de optimización por búsqueda directa son aquellos que no utilizan derivadas para obtener los mínimos locales o globales de una función, este tipo de método usan evaluaciones sobre una función objetivo y según el valor que se obtenga se toman decisiones. La mayor desventaja de este tipo de optimización es que a nivel computacional demora mucho en converger a una solución debido a que es un método netamente iterativo y varía según las variables y restricciones que pueda tener el algoritmo. Sin embargo, si se tiene un sistema donde no se está claro de la estructura de la función objetivo y se tiene un modelo de la misma que pudo haberse obtenido por simulación u otras formas, entonces, las técnicas de búsqueda directa son los métodos apropiados para aplicar (Tremante, 2019). Existen gran variedad de métodos de optimización por búsqueda directa como el método de Nelder-Mead, el método de búsqueda aleatoria y el método de patrón de búsqueda.

## **2.14. MÉTODO DE PATRÓN DE BÚSQUEDA**

El método de patrón de búsqueda es un método de optimización por búsqueda directa creado por Hooke a Jeeves en 1961 (Kolda, Lewis, y Torczon, 2003), consiste en establecer un patrón, cuyo centro es un punto  $x_0$  y los extremos del patrón son puntos de prueba definidos por un paso  $\Delta_0$ , que se evaluarán en una función objetivo para determinar en qué punto se obtiene un valor menor o mayor dependiendo del caso. Por ejemplo, observando la figura 2.16, partiendo del punto  $x_0$  se establece un patrón inicial para proceder a hallar el valor deseado, una vez que se evalúan los puntos del patrón, se centra el patrón en el punto donde se encontró el valor mínimo (caso de minimización) obtenido por la función objetivo y se disminuye el paso del patrón  $\Delta_0$  y así sucesivamente, hasta que al evaluar en la función objetivo no se obtenga ningún valor menor respecto al punto central del

patrón, lo que quiere decir que se ha llegado al valor mínimo, entonces la búsqueda a terminado, como se puede ver en la figura 2.16 el método culminó a la quinta iteración donde se llega al punto  $x^*$ .



**Figura 2.16.** Gráfico del Método de patrón de búsqueda con 5 iteraciones

En este método siempre se establece una tolerancia  $\Delta_{tol}$ , la cual fija hasta donde se va a disminuir el paso del patrón  $\Delta_0$ . Asimismo, se forma el patrón a través de una matriz  $D_{\oplus}$  definida como:

$$D_{\oplus} = \{d_1, d_2, \dots, d_n, \dots, d_{2n}\} \quad (2.28)$$

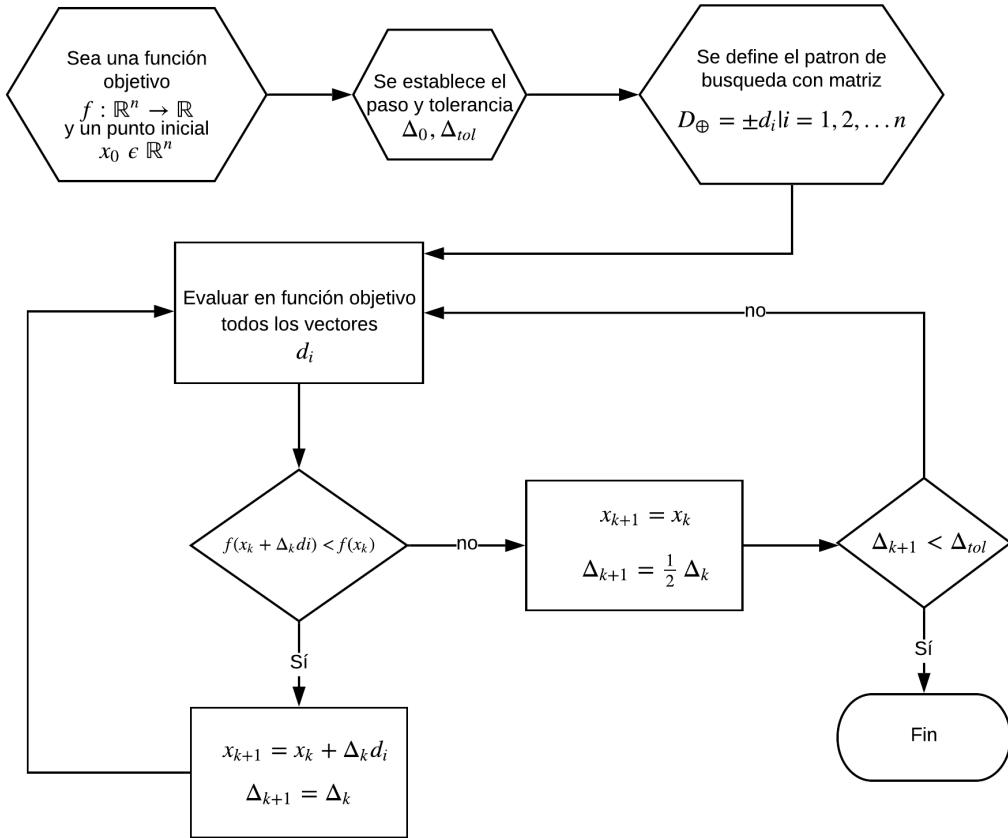
Donde  $n$  es el número de variables de la función objetivo y las dimensiones de la matriz es de  $n \times 2n$  y los  $d_i$  representan el conjunto de vectores denominados patrones. Una manera distinta de definir esta matriz es como lo hace Kolda (2003) por medio de vectores de coordenadas unitarias positivas y negativas, como se muestra a continuación:

$$D_{\oplus} = \{d_1, d_2, \dots, d_n, -d_1, -d_2, \dots, -d_n\} \quad (2.29)$$

Por ejemplo, si se tiene una función objetivo  $f$  que va de  $\mathbb{R}^2 \rightarrow \mathbb{R}$ , se puede representar como:

$$D_{\oplus} = \begin{pmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{pmatrix}$$

En resumen, el método de patrón de búsqueda es un proceso iterativo, donde en cada  $k$ -ésima iteración se calcula  $f(x_k + \Delta_k d_i)$  para todos los  $2n$  vectores de  $D_{\oplus}$ , hasta encontrar un vector  $d_i$  tal que  $f(x_k + \Delta_k d_i) < f(x_k)$ . Si, ninguno de los vectores de  $D_{\oplus}$  redujo el valor de la función objetivo, entonces se contrae el paso del patrón de lo contrario, si al menos un vector logra reducir el valor de la función objetivo, se mueve el patrón a esa dirección. El método termina cuando la longitud del paso sea lo suficientemente pequeño (Tremante, 2019). Una forma de representar este método como un diagrama de flujo se puede hacer de la siguiente manera:



**Figura 2.17.** Diagrama de Flujo del algoritmo de Optimización por el método de patrón de búsqueda

## 2.15. AJUSTE DE FUNCIONES DE PERTENENCIA

En el control difuso una parte fundamental son las funciones de pertenencia que influyen de manera significativa en el desempeño de un controlador difuso (Simon, 2002). Para poder ajustar las funciones de pertenencia se deben considerar ciertas restricciones que no comprometan la estabilidad del sistema y no generen incongruencias en la base de conocimientos del controlador. Partiendo de una función de pertenencia cuyos parámetros son  $[a_i^j, b_i^j, c_i^j, d_i^j] \in \mathbb{R}$ , con  $i = 1, 2, \dots, n$  es el número de la entrada y  $j$  es el término lingüístico, para que este tipo de función sea una función de pertenencia se debe cumplir los siguientes requisitos

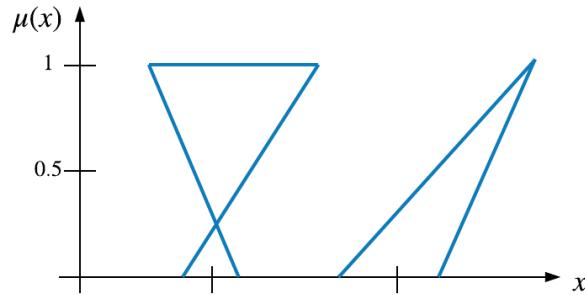
(Tremante, 2019):

$$a_i^j < b_i^j \quad (2.30)$$

$$b_i^j \leq c_i^j \quad (2.31)$$

$$c_i^j < d_i^j \quad (2.32)$$

Las restricciones **2.30**, **2.31** y **2.32** son necesarias para no obtener dos grados de pertenencia en una misma función de pertenencia como se puede observar en la figura **2.18**.



**Figura 2.18.** Función de pertenencia trapezoidal que no cumple las restricciones **2.30**, **2.31** y **2.32**.

Los puntos  $a_i^j$  y  $d_i^j$  de los extremos deben estar contenidos en el universo de discurso, para ello se deben cumplir las siguientes condiciones:

$$x_{i_{min}} \leq a_i^j \quad (2.33)$$

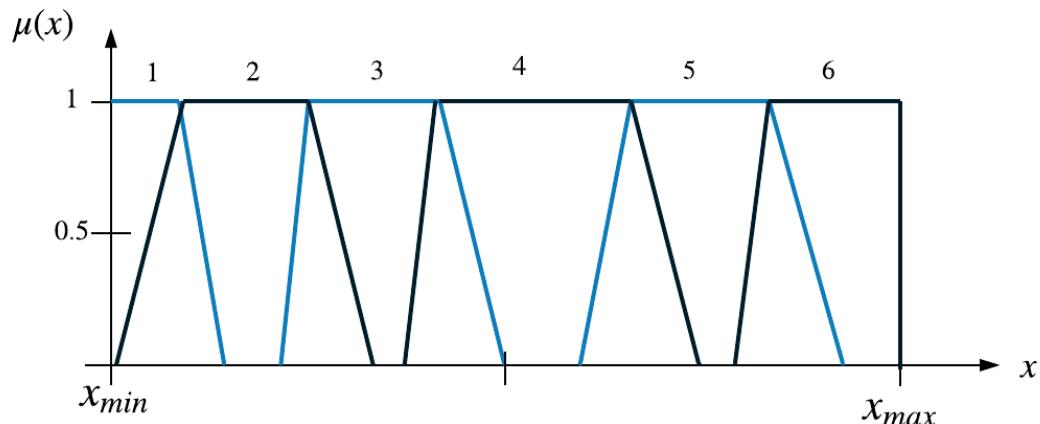
$$d_i^j \leq x_{i_{max}} \quad (2.34)$$

Siendo  $x_{i_{min}}$  el valor mínimo y  $x_{i_{max}}$  el valor máximo del universo de discurso.

Todos los valores pertenecientes al universo de discurso deben estar contemplados en al menos una función de pertenencia, es decir, “un valor de una variables lingüística de entrada, dentro del universo de discurso, debe tener un grado de pertenencia en un término lingüístico” (Tremante, 2019), para cumplir este requerimiento es necesario que exista el simple solapamiento entre dos funciones de pertenencia adyacentes, entonces se debe cumplir que:

$$a_i^{j+1} < d_i^j \quad (2.35)$$

Otro factor que se debe evitar es introducir la lógica booleana a través de las funciones de pertenencia, esto pasa cuando dos funciones de pertenencia adyacentes tienen su punto de máxima pertenencia uno junto el otro. Un ejemplo de este caso se puede observar en la figura 2.19, donde las regiones con el mayor grado de pertenencia se encuentran consecutivas, esto hace que la lógica difusa se pierda ya que la pertenencia a cualquier conjunto se vuelve binaria.



**Figura 2.19.** Funciones de pertenencia que emulan la lógica booleana.

Para evitar este caso se debe cumplir la ecuación:

$$c_i^j < b_i^{j+1} \quad (2.36)$$

Una última condición para las funciones de pertenencia es que no puede existir el múltiple solapamiento entre ellas, es decir, no puede existir un caso donde un valor pertenezca a tres conjuntos a la vez. Por ejemplo, si se está haciendo una medición de temperatura es normal discernir entre si la temperatura es “muy caliente” o “caliente”, pero no tiene sentido pensar en si esta “fría” también, para evitar este tipo de problemas se debe seguir la siguiente restricción:

$$d_i^j < a_i^{j+2} \quad (2.37)$$

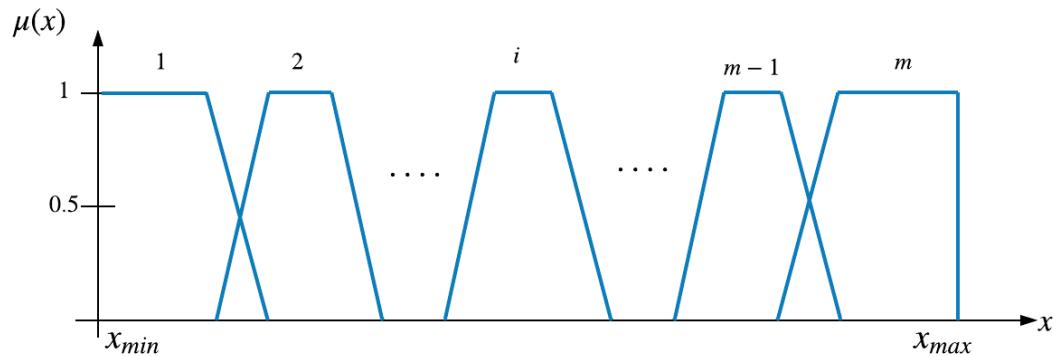
En resumen para llevar a cabo un ajuste de funciones de pertenencia se deben cumplir las condiciones que se encuentran contempladas en la tabla **2.1** (Tremante, 2019).

**Tabla 2.1.** Condiciones y Restricciones para el Ajuste de las Funciones de Pertenencia (Tremante, 2019).

Condición	Restricción
1. Restricción propia de la función de pertenencia	$a_i^j < b_i^j$
	$b_i^j \leq c_i^j$
	$c_i^j < d_i^j$
2. Restricción de los extremos	$x_{i_{min}} < a_i^j$
	$d_i^j \leq x_{i_{max}}$
3. Garantizar simple solapamiento	$a_i^{j+1} < d_i^j$
4. Evitar pertenencia total	$c_i^j < b_i^{j+1}$
5. Evitar múltiple solapamiento	$d_i^j < a_i^{j+2}$

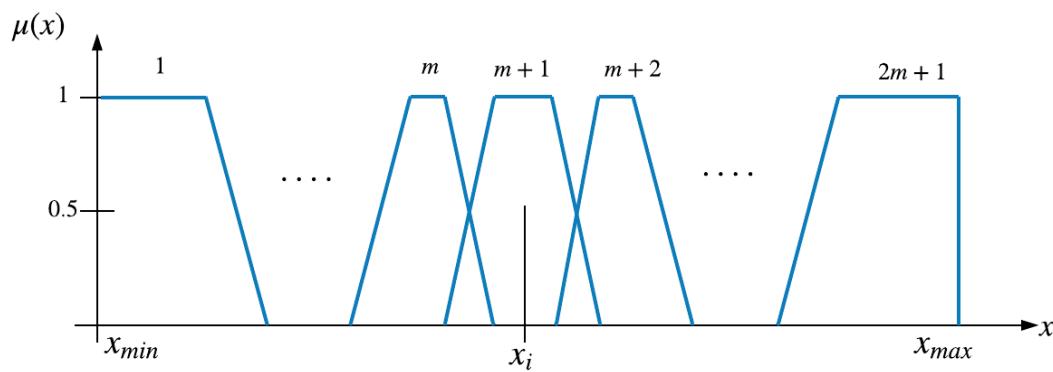
Es notorio que para ajustar un controlador con  $m$  funciones de pertenencia se deben tomar en cuenta 4 parámetros de cada una de ellas (si son trapezoidales),

dando un total de  $4m$  con  $m \geq 2$ , pero si a los extremos del universo de discurso las funciones de pertenencia son abierta a la izquierda y a la derecha como se observa en la figura 2.20.



**Figura 2.20.** Funciones de pertenencia de entrada de un Controlador Difuso.

Entonces la cantidad de parámetros a ajustar se disminuiría a  $4m - 4$ . Ahora bien, en algunos casos puede existir simetría respecto a las funciones de pertenencia, para estos casos se pueden reducir aun más los puntos a ajustar, observando la figura 2.21



**Figura 2.21.** Funciones de pertenencia con simetría respecto al punto  $x = x_i$ .

Se puede ver que ahora se tiene un total de  $2m+1$  de funciones de pertenencia,

lo que da un total de  $4(2m + 1) - 4$  parámetros a ajustar, pero al existir simetría se cumple que:

$$a_i^{m+1} = -d_i^{m+1}, \quad b_i^{m+1} = -c_i^{m+1} \quad (2.38)$$

Así como en las funciones de pertenencia  $m$  y  $m + 2$ .

$$a_i^m = -d_i^{m+2}, \quad b_i^m = -c_i^{m+2} \quad (2.39)$$

$$c_i^m = -b_i^{m+2}, \quad d_i^m = -a_i^{m+2} \quad (2.40)$$

Asimismo, en los extremos del universo de discurso se obtiene:

$$c_i^1 = -b_i^{2m+1}, \quad d_i^1 = -a_i^{2m+1} \quad (2.41)$$

$$x_{i_{min}} = a_i^1 = b_i^1 = -c_i^{2m+1} = d_i^{2m+1} = -x_{i_{max}} \quad (2.42)$$

Por lo tanto, dada la simetría se reduce la cantidad de parámetros a ajustar a  $4m$ , donde solo se ajustan las funciones de pertenencia de un lado del universo de discurso (Tremante, 2019).

## CAPÍTULO III

### DESCRIPCIÓN DEL SISTEMA BALL AND BEAM DE QUANSER

#### 3.1. EL SISTEMA BALL AND BEAM DE QUANSER

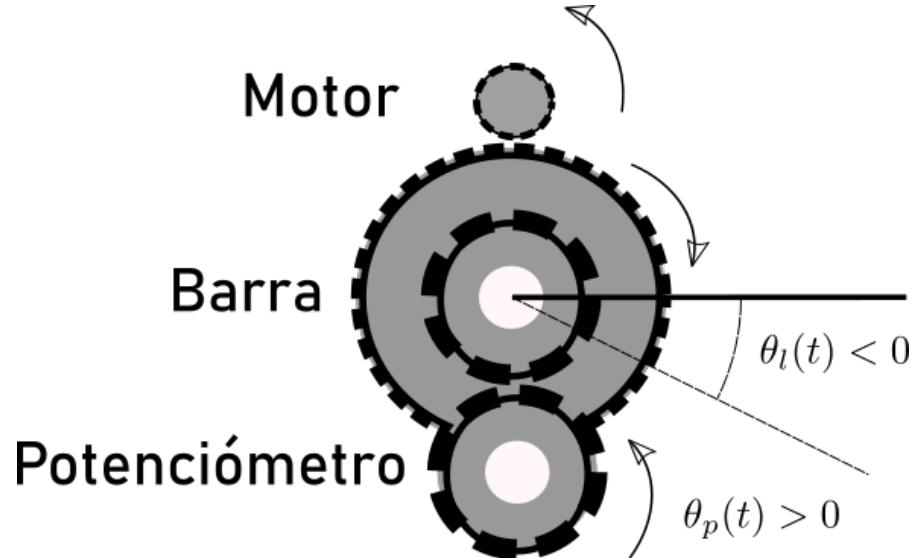
El sistema Ball and Beam es un sistema no lineal e inherentemente inestable, muy utilizado a nivel de laboratorios académicos para evaluar el desempeño de cualquier controlador (Sameera y cols., 2017). El sistema Ball and Beam de Quanser está constituido por tres módulos: BB01 conformado por la barra y la esfera, SRV02 constituido por el servo motor y el VoltPAQ-X1 es la interfaz del sistema. El sistema real se puede observar en la figura 3.1.



**Figura 3.1.** El sistema Ball and Beam de Quanser.

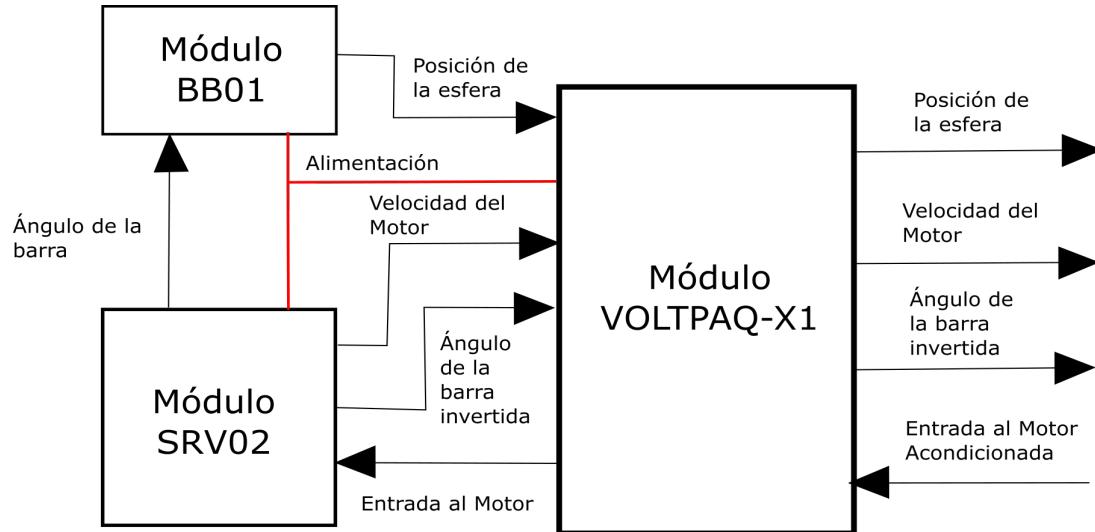
Cabe destacar que el ángulo de la barra en el sistema Ball and Beam se obtiene a través de un potenciómetro y este valor va en sentido contrario a como se desplaza la barra debido a como se encuentra conectado el tren de engranaje como se muestra en la figura 3.2, donde  $\theta_l$  es el ángulo de la barra y  $\theta_p$  es el

ángulo del potenciómetro.



**Figura 3.2.** Esquema de conexión del tren de engranajes del Ball and Beam.

Un diagrama de bloques que representa al sistema se puede observar en la figura 3.3.



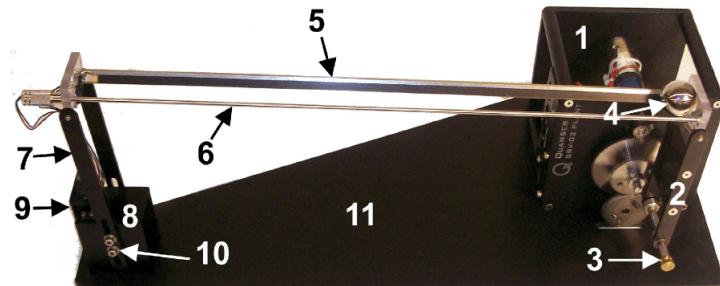
**Figura 3.3.** Diagrama de bloques del sistema Ball and Beam de Quanser.

### 3.2. MÓDULO BB01

En este módulo se encuentran la esfera metálica, el sensor de posición y la barra, cuyo tornillo de acople se encuentra unido al movimiento del servo motor del módulo SRV02 por medio de un tren de engranajes. En la base de este módulo, en el extremo opuesto al actuador se encuentra un conector DIN6 para conectarse con el módulo VoltPAQ-X1, donde se obtienen la alimentación y la salida del sensor de posición. El módulo BB01 se puede observar en la figura 3.4 y ver el nombre de sus partes en la tabla 3.1.

**Tabla 3.1.** Componentes del módulo BB01.

ID	Componente del BB01
1	Módulo SRV02
2	Brazo Palanca
3	Tornillo de acople
4	Esfera de acero
5	Potenciómetro
6	Varilla de acero
7	Brazo de soporte
8	Base
9	Conector del sensor del BB01
10	Tornillos de soporte
11	Base principal



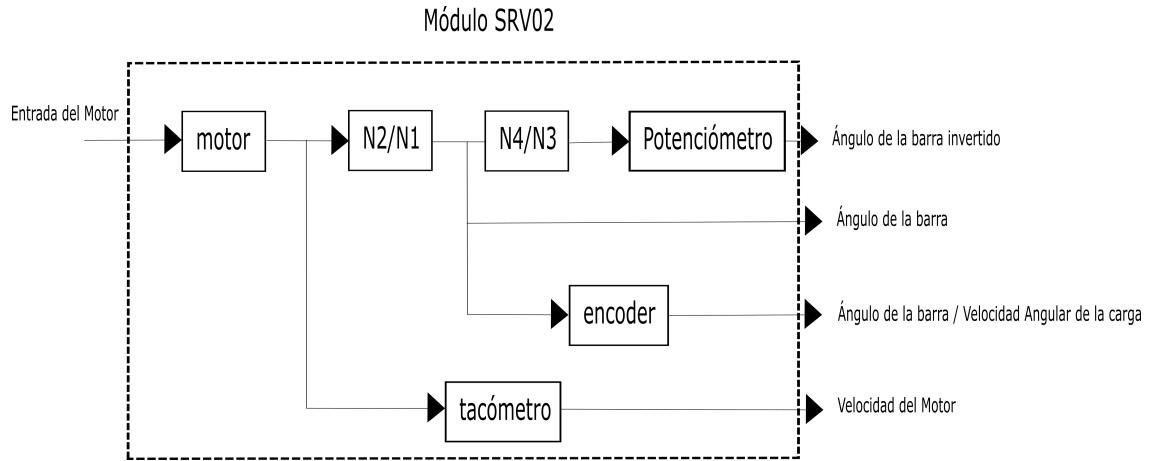
**Figura 3.4.** Partes del módulo BB01 y módulo SRV02.

Ahora bien, respecto al principio de funcionamiento del sensor de posición, la barra está formada por un lado de acero y otro de nickel cromo, este material posee una alta resistencia eléctrica, el cual se usa como transductor. Cuando se coloca la esfera sobre la barra se cierra el circuito y al estar alimentado se obtiene un valor analógico distinto para cada posición donde este ubicada la esfera, comportamiento análogo al de un potenciómetro, cuyo valor en tensión está comprendido entre -4,5V y 4,5V.

### 3.3. MÓDULO SRV02

La mayoría de los sistemas desarrollados por Quanser utilizan este módulo como actuador en dos posibles configuraciones del tren de engranajes: Low gear y High gear, estas se diferencian en el diámetro de los engranajes. En el sistema Ball and Beam se usa la configuración High gear recomendada por el manual del fabricante (Apkarian, Lévis, y Gurocak, 2011a).

El módulo SRV02 está compuesto por un motor DC, un tacómetro analógico que mide la velocidad del motor en ambas direcciones, un encoder óptico digital para medir la posición relativa de la barra y un potenciómetro analógico para obtener la posición absoluta de la barra. Un diagrama de bloques de este módulo se puede observar en la figura 3.5, donde  $N2/N1$  y  $N4/N3$  representan la relación entre los diámetros de los engranajes, debido a como se encuentran los engranajes  $N3$  y  $N4$  el potenciómetro mide la posición de la barra en sentido opuesto.



**Figura 3.5.** Diagrama de bloques del Módulo SRV02.

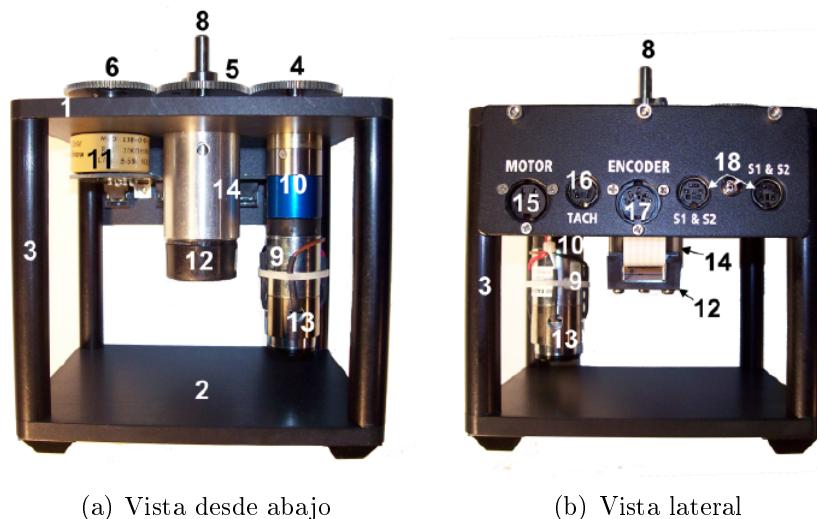
El módulo SRV02 se puede observar en las figuras **3.6** (configuración High Gear) y **3.7**. Además los nombres de cada una sus partes se pueden ver en la tabla **3.2**.

**Tabla 3.2.** Componentes del módulo SRV02

ID	Componente	ID	Componente
1	Tapa superior	13	Tacómetro
2	Tapa inferior	14	Bloque transductor
3	Pilares	15	Conector de motor
4	Engranaje del motor	16	Conector de tacómetro
5	Engranaje de la barra	17	Conector de encoder
6	Engranaje de potencíometro	18	Conector de potencíometro
7	Muelles anti retroceso	19	Engranaje de motor 2
8	Asa de carga	20	Engranaje de la barra 2
9	Motor DC		
10	Caja de engranajes		
11	Potencíometro		
12	Encoder		



**Figura 3.6.** Modulo SRV02 configuración High gear.



(a) Vista desde abajo

(b) Vista lateral

**Figura 3.7.** Partes del modulo SRV02.

El potenciómetro del módulo SRV02 se puede observar en la figura 3.7, ID 11, fue fabricado por Vishay Spectrol, este consiste en una resistencia de  $10k\Omega$  sin tope mecánico y tiene un rango de  $352^\circ$  (Apkarian y cols., 2011a), conectado al módulo VoltPAQ-X1 a través de un conector DIN6. Este sensor mide la posición

angular de la barra pero debido a como está acoplado mecánicamente al motor este mide la posición en sentido opuesto.

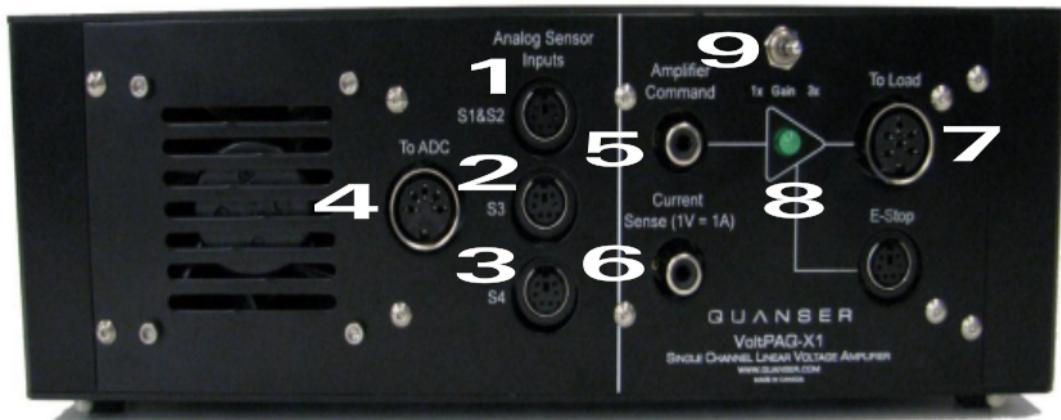
El tacómetro está acoplado al eje del motor y este mide su velocidad en ambas direcciones, el valor máximo de tensión de este sensor es de 9V cuando el motor gira a su velocidad máxima de 6000 rpm (Apkarian y cols., 2011a). Se puede tener acceso al valor analógico del tacómetro a través de un conector DIN6 por el terminal que indica TACH al costado del módulo SRV02 (ver figura **3.7b**, ID 16).

El encoder óptico cumple la función de enviar información respecto a la barra, puede medir tanto la posición angular como la velocidad angular. La información de la barra es enviada de forma serial por dos canales A y B pero puede tener variaciones respecto a la frecuencia, ya que, ésta es proporcional a la velocidad en la barra, por lo tanto se debe disponer de un equipo adecuado a este tipo de señal para procesarla. El uso de este componente es opcional, por ende se puede llevar a cabo el control sin el mismo.

El motor DC que posee el sistema que se observa en la figura **3.7**, ID 9, es un producto de Faulhaber Coreless, modelo 2338S006, cuya entrada máxima en tensión es de  $\pm 6V$ , la velocidad máxima de giro es de 6000 rpm, su corriente nominal en continuo es de 1A y se conecta al módulo VoltPAQ-X1 a través de un conector tipo DIN5 para su accionamiento.

### 3.4. MÓDULO VOLTPAQ-X1

Este módulo es la fuente de la planta, gran cantidad de los productos de Quanser usan este módulo para alimentar los sensores y ejecutar acciones de control a través de su amplificador lineal de potencia. El módulo VoltPAQ-X1 se muestra en la figura **3.8** y en la tabla **3.3** se describen las entradas y salidas detalladamente.



**Figura 3.8.** Entradas y salidas del VoltPAQ-X1.

**Tabla 3.3.** Entradas y salidas de módulo VoltPAQ-X1.

ID	I/O	Descripción	Rango
1	S1, S2	Lee la salida de dos sensores analógicos y también suministra $\pm 12V$	Entrada: [-10,10]V
2	S3	Lee la salida del sensor analógico y también suministra $\pm 12V$	Entrada: [-10,10]V
3	S4	Lee la salida del sensor analógico y también suministra $\pm 12V$	Entrada: [-10,10]V
4	to ADC	Salida a ADC para leer valores de sensores	Salida: [-10,10]V
5	Amplificador	Voltaje de entrada a ser amplificado	Entrada: [-10,10]V
6	Sensor de corriente	Detecta la corriente que circula en el motor	1 A/V
7	Carga	Salida a la carga Salida= Ganancia $\times$ Voltaje de entrada	[-24,24] V
8	LED	Apagado: Sobre calentamiento/- Parada de emergencia/No hay alimentación, Encendido: Amplificador activo	
9	Ganancia	Multiplicador de ganancia para x1 y x3	

Los sensores del sistema Ball and Beam se alimentan a través de este módulo

por medio de un cable con conectores DIN6 en ambos extremos que se muestra en la figura **3.9b**, cada uno de los valores analógicos de estos sensores son procesados y se acceden a ellos por medio de un cable con conectores DIN5 y RCA (ver figura **3.9a**) conectado a la salida del módulo “to ADC” que se observa en la figura **3.8**, ID 4 , cuyos valores límites son -10V y 10V. Por otro lado, las acciones de control en la planta también pasan por este módulo a través de la entrada “Amplifier Command” que se muestra en la figura **3.8**, ID 5, usando un cable con conectores RCA en ambos lados que se encuentra en la figura **3.9d** y la salida “to Load” (ver figura **3.8**, ID 7) con un cable con conectores DIN6 y DIN4 (ver figura **3.9c**), el amplificador lineal de potencia tiene una capacidad de amplificar una entrada hasta por un factor de tres, con un límite por saturación de -24V y 24V, pero en el caso del Ball and Beam este factor multiplicativo es 1. Además, el VoltPAQ-X1 también ofrece un sensor de corriente para implementar posible acciones de control para resguardar la seguridad del equipo a la cual se accede por el cable con conectores RCA de la figura **3.9d**.



(a) Cable DIN5 a RCA.



(b) Cable para sensores DIN6.



(c) Cable del motor DIN4 a DIN6.



(d) Cable RCA para amplificador y sensor de corriente.

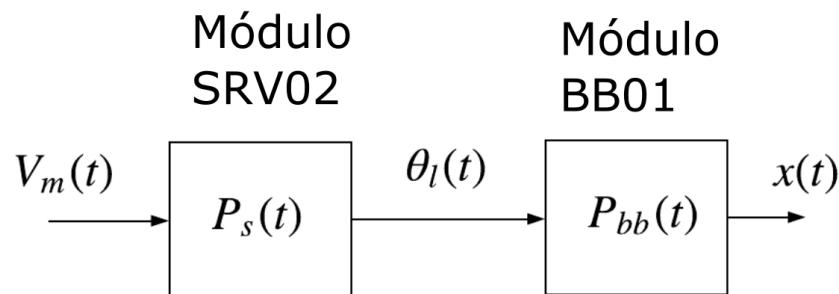
**Figura 3.9.** Cables del módulo VoltPAQ-X1.

## CAPÍTULO IV

### DESCRIPCIÓN DEL FUNCIONAMIENTO DEL SISTEMA BALL AND BEAM DE QUANSER

#### 4.1. DESARROLLO DE LAS ECUACIONES MATEMÁTICAS QUE DESCRIBEN LA PLANTA

El funcionamiento del sistema Ball and Beam de Quanser se puede describir por las ecuaciones matemáticas que rigen la planta a través de dos etapas, la primera describe las ecuaciones eléctricas y mecánicas del motor (módulo SRV02) y la segunda el movimiento de la esfera sobre la barra (módulo BB01). Es importante resaltar que para ambas etapas se consideran que los parámetros de la planta son concentrados, invariantes en tiempo y con tolerancia despreciable. Una representación del sistema en diagrama de bloques se muestra en la figura 4.1, donde  $P_s(t)$  y  $P_{bb}(t)$  representan la primera y segunda etapa respectivamente.



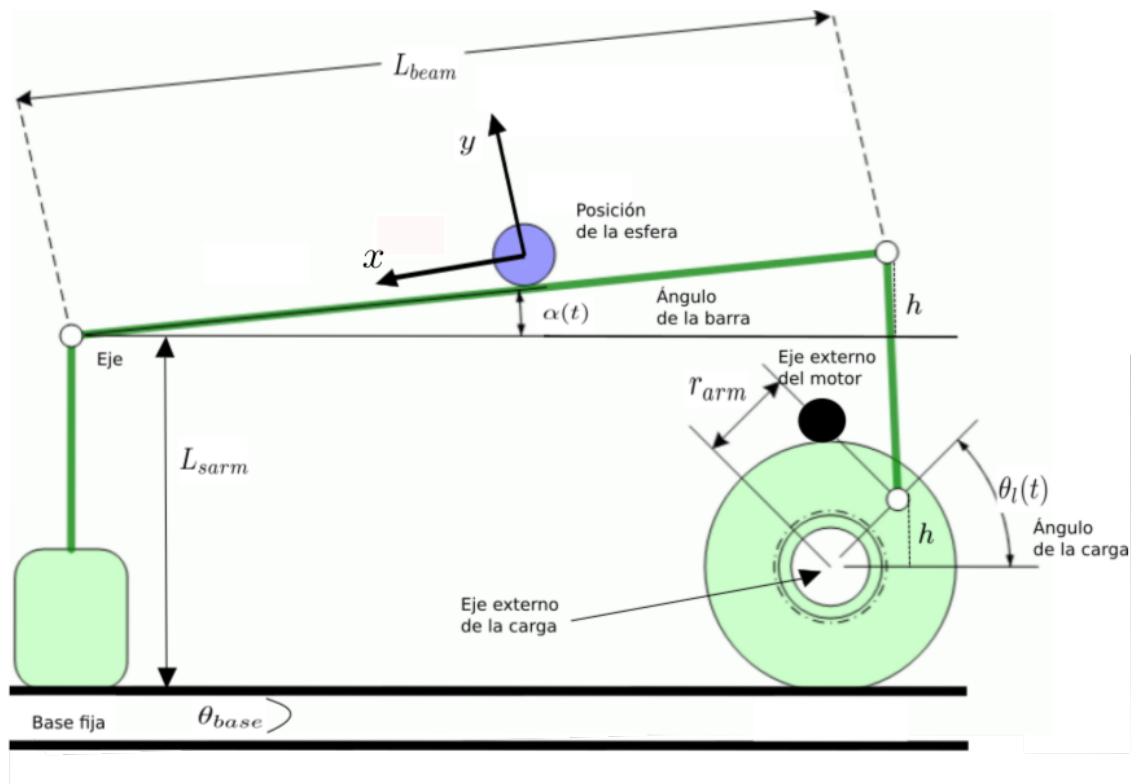
**Figura 4.1.** Diagrama de bloques del sistema ball and beam en tiempo.

#### 4.1.1. DESARROLLO DE LAS ECUACIONES MATEMÁTICAS PARA EL MÓDULO BB01

En la figura 4.2 se puede observar un esquema físico del sistema,  $\theta_{base}$  representa el ángulo de la plataforma de calibración del módulo BB01 respecto al suelo,  $\theta_l(t)$  es el ángulo de la barra y su rango está comprendido entre  $45^\circ$  y  $-45^\circ$  respecto a la horizontal según las especificaciones del fabricante (Apkarian, Lévis, y Gurocak, 2011b) y  $\alpha(t)$  representa el ángulo de la barra, donde el cero de  $\theta_l$  se define como la posición donde la esfera no se mueve y la barra está horizontal, es decir,  $\alpha(t) = 0^\circ$ ; una lista detallada de cada uno de los elementos que se presentan en la figura 4.2 se muestran en la tabla 4.1.

**Tabla 4.1.** Elementos del módulo BB01.

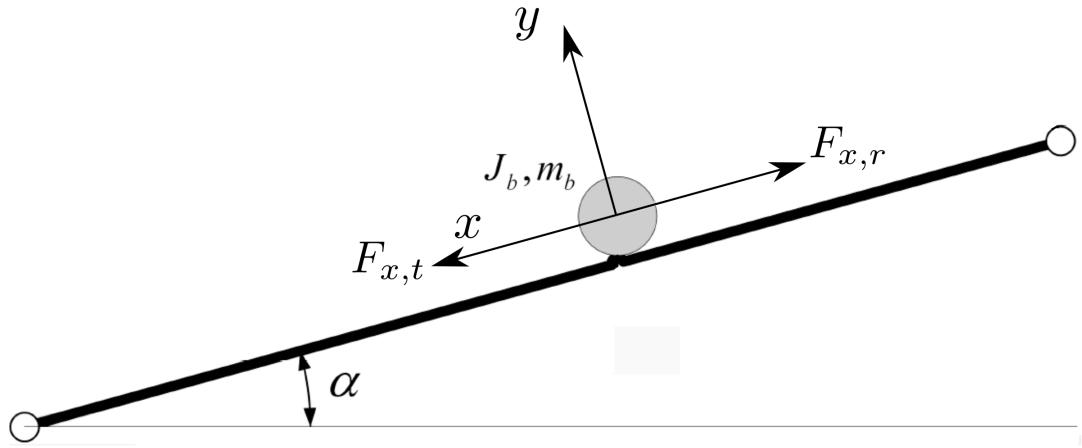
Variable	Descripción	Valor
$L_{beam}$	Longitud de la barra	42,55 cm
$L_{arm}$	Longitud del brazo de palanca	12 cm
$r_{arm}$	Distancia entre eje de barra y el acople de brazo palanca	2,54 cm
$L_{sarm}$	Longitud del brazo de soporte	16 cm
$m_b$	Masa de la esfera	0,064 kg
$r_b$	Radio de la esfera	1,27 cm
$J_b$	Momento de inercia de la esfera	$4,13 \times 10^{-6} \text{ kg m}^2$
$K_{bs}$	Sensibilidad de sensor de posición	4,25 cm/V
$V_{bias}$	Alimentación del sensor	$\pm 12 V$
$V_{range}$	Rango de tensión de salida del sensor de posición	$\pm 5 V$



**Figura 4.2.** Esquema físico del sistema Ball and Beam.

Para describir el movimiento de la esfera se debe realizar un diagrama de cuerpo libre y ver las distintas fuerzas que interactúan sobre ella, partiendo de la segunda ley de Newton y observando la figura 4.3 se tiene que las fuerzas que se encuentran en el eje x son:

$$m_b \left( \frac{d^2 x(t)}{dt} \right) = \sum F \quad (4.1)$$



**Figura 4.3.** Diagrama de cuerpo libre de la esfera inicialmente.

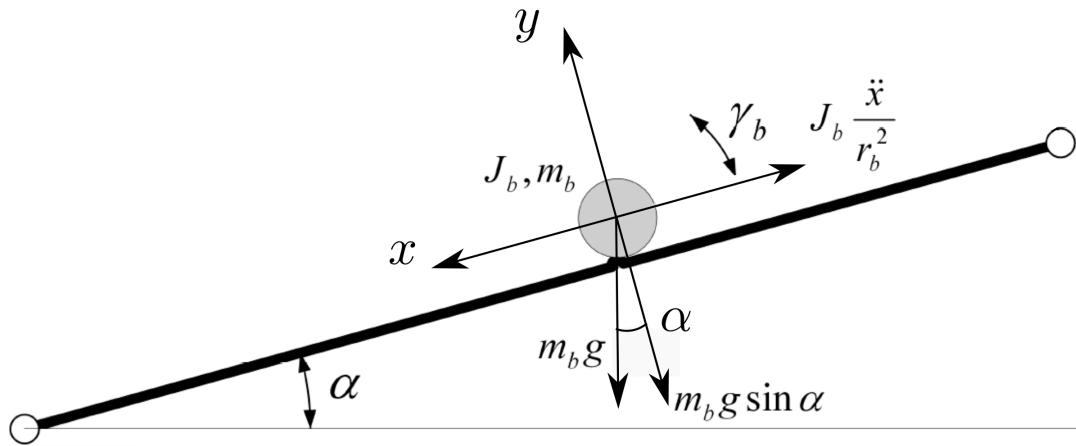
Donde  $m_b$  representa la masa de la esfera,  $J_b$  es el momento de inercia de la esfera,  $\sum F$  representa la suma de todas las fuerzas que interactúan sobre la esfera, las cuales son:  $F_{x,t}$  representa la fuerza translacional generada por la gravedad y  $F_{x,r}$  es la inercia de la pelota, por lo tanto la ecuación 4.2 se convierte en:

$$m_b \left( \frac{d^2 x(t)}{dt} \right) = F_{x,t} - F_{x,r} \quad (4.2)$$

Donde la fuerza  $F_{x,t}$  se debe a una fracción del peso actuando sobre el plano inclinado en la dirección  $x$  positiva y esta relación se obtiene del triángulo rectángulo formado por el vector de peso y el eje de coordenadas inclinado (ver figura 4.4). Asimismo, sabiendo que el torque es igual al producto entre el radio de la esfera y la fuerza aplicada se obtiene una relación entre la fuerza debido a la inercia y el torque aplicado  $\tau_b$ , por ende:

$$F_r, t = \frac{\tau_b}{r_b} \quad (4.3)$$

$$F_{x,t} = m_b g \operatorname{sen}(\alpha) \quad (4.4)$$



**Figura 4.4.** Diagrama de cuerpo libre de la esfera.

Donde la relación del torque  $\tau_b$  con el momento de inercia  $J_b$  por la segunda ley de Newton es:

$$\tau_b = J_b \left( \frac{d^2}{dt^2} \gamma_b(t) \right) \quad (4.5)$$

Siendo  $\gamma_b(t)$  el ángulo que se mueve la esfera respecto a su centro. Ahora, considerando lo que recorre la esfera en  $x$  respecto al desplazamiento angular se obtiene que  $x(t) = \gamma_b(t)r_b$  y sustituyendo en la ecuación 4.2 las ecuaciones 4.3, 4.4 y 4.5 se obtiene:

$$m_b \left( \frac{d^2x(t)}{dt^2} \right) = m_b g \operatorname{sen}(\alpha(t)) - \frac{J_b \left( \frac{d^2x(t)}{dt^2} \right)}{r_b^2} \quad (4.6)$$

Despejando la aceleración de la esfera

$$\frac{d^2x(t)}{dt^2} = \frac{m_b g \operatorname{sen}(\alpha(t)) r_b^2}{m_b r_b^2 + J_b} \quad (4.7)$$

Luego, observando la figura **4.2**, la altura  $h$  proporciona una relación entre los ángulos  $\theta_l(t)$  y  $\alpha(t)$  dada por trigonometría.

$$\operatorname{sen}(\alpha(t)) = \frac{h}{L_{beam}} \quad (4.8)$$

$$\operatorname{sen}(\theta_l(t)) = \frac{h}{r_{arm}} \quad (4.9)$$

Despejando  $h$  de la ecuación **4.9** y sustituyendo en la ecuación **4.8**.

$$\operatorname{sen}(\alpha(t)) = \frac{\operatorname{sen}(\theta_l(t)) r_{arm}}{L_{beam}} \quad (4.10)$$

Ahora sustituyendo la ecuación **4.10** en **4.7**.

$$\frac{d^2x(t)}{dt^2} = \frac{m_b g r_{arm} \operatorname{sen}(\theta_l(t)) r_b^2}{L_{beam} (m_b r_b^2 + J_b)} \quad (4.11)$$

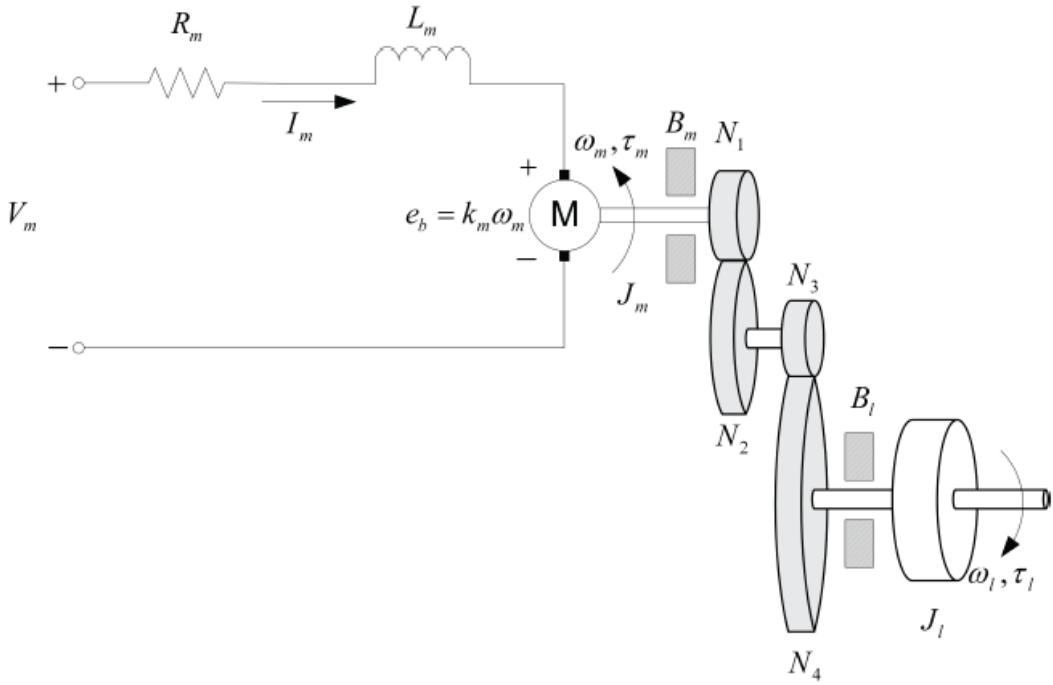
Resultando que la relación entre la posición de la esfera y el ángulo de la barra es una ecuación no lineal.

#### 4.1.2. DESARROLLO DE LAS ECUACIONES MATEMÁTICAS PARA EL MÓDULO SRV02

Partiendo del esquema que se muestra en la figura 4.5, donde se encuentra el circuito de armadura y su relación mecánica con la carga a través del tren de engranajes, cada una de las piezas involucradas en esta parte del sistema se observan en la tabla 4.2.

**Tabla 4.2.** Elementos del módulo SRV02

Variable	Descripción	Valor
$V_{nom}$	Voltaje nominal del motor	6 V
$R_m$	Resistencia de armadura	2,6 $\Omega$
$L_m$	Inductancia de armadura	0,18 mH
$n_m$	Eficiencia del motor	0,69
$n_g$	Eficiencia de los engranajes	0,9
$f_{max}$	Frecuencia máxima de entrada	50 Hz
$\omega_{max}$	Velocidad angular del motor máxima	628,3 rad/s
$K_{enc}$	Resolución del encoder SRV02-EHR	8192 conteos/rev
$B_{eq}$	Coeficiente de fricción viscosa	0,015 N m/(rad/s)
$J_m$	Momento de inercia del motor	$3,9 \times 10^{-7} kg/m^2$
$J_l$	Momento de inercia de la carga	$7,06 \times 10^{-8}$
$K_g$	Razón de vueltas de los engranajes	70
$k_m$	Constante EMF del motor	$7,68 \times 10^{-3} Vs/rad$
$k_t$	Constante proporcional de corriente	$7,68 \times 10^{-3} N m/A$



**Figura 4.5.** Esquema del circuito de armadura y tren de engranajes.

Aplicando la ley de Kirchhoff en el circuito de armadura se obtiene:

$$V_m(t) - I_a R_m - L_m \frac{dI_a}{dt} - k_m \omega_m(t) = 0 \quad (4.12)$$

Donde  $V_m$  es la tensión aplicada al motor,  $I_a$  la corriente que circula por el mismo,  $k_m$  es la fuerza contra electromotriz,  $R_m$  la resistencia del embobinado,  $L_m$  la inductancia de la bobina y  $\omega_m$  es la velocidad angular del motor. Considerando que la resistencia es mucho mayor que la inductancia se puede anular este parámetro de la ecuación y se obtiene:

$$V_m(t) - I_a R_m - k_m \omega_m(t) = 0 \quad (4.13)$$

Despejando la corriente  $I_a$ .

$$I_a(t) = \frac{V_m(t) - k_m \omega_m(t)}{R_m} \quad (4.14)$$

Por otro lado, partiendo del movimiento del motor respecto a la carga, se tiene que:

$$\theta_m(t) = K_g \theta_l(t) \quad (4.15)$$

Siendo  $\theta_m$  el ángulo que se desplaza el eje del motor y  $\theta_l$  el ángulo que se mueve la barra. Derivando respecto al tiempo ambos lados de la ecuación 4.15.

$$\omega_m(t) = K_g \omega_l(t) \quad (4.16)$$

Donde la relación de vueltas de los engranajes  $K_g$  se obtiene de la relación  $N_1$  con  $N_2$  definida como  $K_{gi}$  y  $N_3$  con  $N_4$  como  $K_{ge}$ .

$$K_{gi} = \frac{N_2}{N_1} \quad (4.17)$$

$$K_{ge} = \frac{N_4}{N_3} \quad (4.18)$$

$$K_g = K_{gi} K_{ge} = 70 \quad (4.19)$$

Asimismo, el torque en la barra  $\tau_l$  se relaciona con el torque del motor  $\tau_m$  a través del torque aplicado del motor hacia la barra  $\tau_{ml}$ , la eficiencia  $n_g$  y la relación de vueltas de los engranajes.

$$\tau_l(t) = n_g K_g \tau_{ml}(t) \quad (4.20)$$

Luego, la ecuación que describe el movimiento de la barra viene dada por:

$$J_l \frac{d\omega_l(t)}{dt} + B_l \omega_l(t) = \tau_l(t) \quad (4.21)$$

Donde  $J_l$  y  $B_l$  representan el momento de inercia y el coeficiente de viscosidad respectivamente. Sustituyendo la ecuación 4.20 en 4.21 se obtiene:

$$J_l \frac{d\omega_l(t)}{dt} + B_l \omega_l(t) = n_g K_g \tau_{ml}(t) \quad (4.22)$$

Analogamente de la ecuación 4.21 se describe el movimiento angular del motor.

$$J_m \frac{d\omega_m(t)}{dt} + B_m \omega_m(t) + \tau_{ml}(t) = \tau_m(t) \quad (4.23)$$

Ahora multiplicando ambos lados de la ecuación por  $n_g K_g$  y sustituyendo la ecuación 4.22 y 4.16 en 4.23 se obtiene:

$$(n_g K_g^2 J_m + J_l) \frac{d\omega_l(t)}{dt} + (n_g K_g^2 B_m + B_l) \omega_l(t) = n_g K_g \tau_m(t) \quad (4.24)$$

Donde se definen los términos  $J_{eq}$  y  $B_{eq}$  como:

$$J_{eq} = n_g K_g^2 J_m + J_l$$

$$B_{eq} = n_g K_g^2 B_m + B_l$$

Resultando

$$J_{eq} \frac{d\omega_l(t)}{dt} + B_{eq}\omega_l(t) = n_g K_g \tau_m(t) \quad (4.25)$$

Ahora bien, el torque es proporcional a la corriente que circula por el motor (Apkarian y cols., 2011a) según la siguiente ecuación:

$$\tau_m(t) = n_m k_t I_a \quad (4.26)$$

Donde  $n_m$  y  $k_t$  son la eficiencia y constante proporcional de corriente del motor. Sustituyendo la ecuación 4.14 en 4.26 se tiene:

$$\tau_m(t) = \frac{n_m k_t (V_m(t) - k_m \omega_m(t))}{R_m} \quad (4.27)$$

Luego, sustituyendo esta ecuación en 4.25.

$$J_{eq} \frac{d\omega_l(t)}{dt} + B_{eq}\omega_l(t) = n_g K_g \frac{n_m k_t (V_m(t) - k_m \omega_m(t))}{R_m} \quad (4.28)$$

Haciendo operaciones algebraicas sobre esta ecuación y definiendo dos nuevos términos  $A_m$  y  $B_{eq,v}$ .

$$J_{eq} \frac{d\omega_l(t)}{dt} + B_{eq,v}\omega_l(t) = A_m V_m(t) \quad (4.29)$$

$$A_m = \frac{n_g K_g n_m k_t}{R_m}$$

$$B_{eq,v} = \frac{n_g K_g^2 n_m k_t k_m + B_{eq} R_m}{R_m}$$

Haciendo el cambio  $\frac{d\theta_l(t)}{dt} = \omega_l(t)$ .

$$J_{eq} \frac{d^2\theta_l(t)}{dt^2} + B_{eq,v} \frac{d\theta_l(t)}{dt} = A_m V_m(t) \quad (4.30)$$

Resultando una ecuación lineal, que describe la relación entre el ángulo de la barra y el voltaje aplicado al motor.

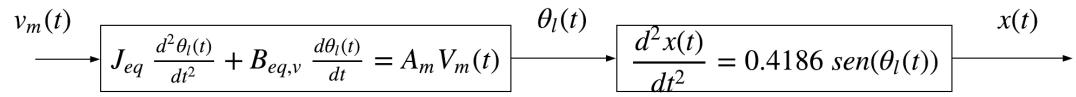
#### 4.1.3. ECUACIONES MATEMÁTICAS DEL SISTEMA BALL AND BEAM

Sustituyendo en las ecuaciones 4.30 y 4.11, por los valores en las tablas 4.2 y 4.1, se obtiene:

$$\frac{d^2\theta_l(t)}{dt^2} + 48,8554 \frac{d\theta_l(t)}{dt} = 74,6546 V_m(t) \quad (4.31)$$

$$\frac{d^2x(t)}{dt^2} = 0,4186 \operatorname{sen}(\theta_l(t)) \quad (4.32)$$

Donde se puede observar que el sistema Ball and Beam esta constituido por una parte lineal y otra no lineal que corresponden a los módulos SRV02 y BB01 respectivamente, lo que lo hace un sistema no lineal, representado por el diagrama de bloques que se muestra a continuación:



**Figura 4.6.** Diagrama de bloques en tiempo con ecuaciones del sistema.

Con el sistema dividido en dos etapas Quanser recomienda para realizar el control usar dos lazos, uno para el control del ángulo de la barra y otro para la posición de la esfera. Sin embargo, para poder realizar este control se necesita de un circuito que acondicione las señales de los sensores para un microcontrolador. Así como, maneje la salida analógica para llevar el valor de tensión de salida del microcontrolador a valores en tensión que permita mover el motor en ambas direcciones.

# CAPÍTULO V

## DESARROLLO DEL HARDWARE Y SELECCIÓN DE UNA TARJETA DE DESARROLLO

### 5.1. SELECCIÓN DE LA TARJETA DE DESARROLLO

Para llevar a cabo el controlador difuso se deben cumplir ciertos requerimientos respecto al hardware, los cuales son: salida con modulación por ancho de pulso (PWM) o convertidor digital analógico (DAC) para poder realizar acciones sobre el motor, al menos dos entradas con convertidor analógico digital (ADC) para obtener los valores de los sensores de posición y ángulo de la barra. Además, un puerto serial para envío y recepción de datos para exportar datos respecto al sistema. El controlador difuso se implementará con una librería que existe para tal fin. A continuación se presentan distintas tarjetas de desarrollo compatibles con las librerías de control difuso que se hallaron.

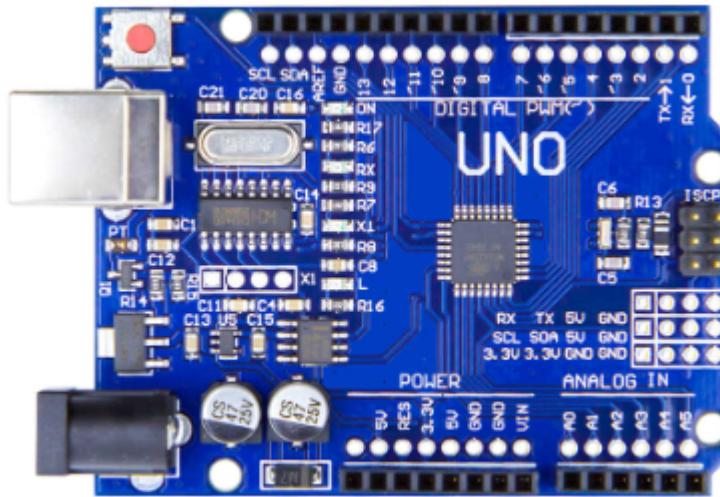
#### 5.1.1. ARDUINO UNO

El Arduino UNO es una tarjeta de desarrollado (ver figura 5.1) basada en el microcontrolador ATMega328 con arquitectura de 8 bits, conocida debido a su facilidad de programación y variedad de librerías desarrolladas para realizar diferentes tipos de proyectos. Arduino es una compañía Open Source bajo la licencia GNU GPL (*General Public License*), por lo tanto todos sus diseños de hardware y software están abiertos a la comunidad, cualquiera puede construir

sus diseños e implementar alguna de sus librerías. La tarjeta Arduino UNO se programa a través del Arduino IDE, el cual es una interfaz de programación para desarrollar código en C++. A continuación en la tabla **5.1** se muestran las especificaciones de hardware del Arduino UNO.

**Tabla 5.1.** Especificaciones del Arduino UNO.

Alimentación	USB 5V, Entrada $V_{in}$ 7 – 12V
Almacenamiento	32kbytes Flash, 2kBytes SRAM, 1kbyte EEPROM
Perifericos	14 pines de Entrada/Salida, Salidas PWM, 6 Entradas ADC de 10 bits
Comunicaciones	USB, ICSP, I2C, SPI, UART
Boton Reset	Boton para reinicio del programa
Procesamiento	ATMega328, velocidad 16MHz



**Figura 5.1.** Tarjeta de desarrollo Arduino UNO.

### 5.1.2. ESP32 WROOM Devkit

El ESP32 WROOM Devkit es una tarjeta de desarrollo basado en el microcontrolador ESP32 con arquitectura de 32 bits (ver figura **5.2**) diseñado por la

empresa ESPRESSIF en 2016. Esta tarjeta está orientada a aplicaciones de Internet de la cosas (IoT) debido a su facilidad de conexión inalámbrica, ya que dispone de conexión WiFi y Bluetooth sin necesidad de módulos adicionales. Además, esta tarjeta tiene la capacidad de trabajar con múltiples núcleos, manteniendo dos núcleos principales que permiten la aplicación de dos procesos en forma simultanea y un tercero para llevar a cabo tareas de bajo consumo. Esta tarjeta solo puede ser programada bajo la integración de un sistema operativo llamado FreeRTOS. A continuación se muestran las especificaciones técnicas:

**Tabla 5.2.** Especificaciones del ESP32

Alimentación	USB 5V, Entrada $V_{in}$ 5V, Entrada 3,3V
Almacenamiento	4MB Flash, 448KByte ROM, 520KByte SRAM, 16KByte SRAM con RTC
Perifericos	36 GPIO, Salidas PWM, 16 entradas ADC de 12 bits, 2 DAC de 8 bits, Entradas Touch
Comunicaciones	USB, SPI, I2C, I2S, 3 UART, Wifi: 802.11 b/g/n/e/i (802.11n @ 2.4 GHz hasta 150 Mbit/s), Bluetooth, BLE
Procesamiento	Tensilica Xtensa 32-bit LX6, velocidad de reloj entre 160 MHz y 240 MHz

Por otra parte, existen tres posibles entornos de desarrollo para programar este dispositivo, la primera es Eclipse, en este IDE se pueden incorporar tanto las librerías de FreeRTOS como las desarrolladas por ESPRESSIF en lenguaje C, otra opción de programación es el Arduino IDE, Arduino desarrollo librerías compatibles con el ESP32 que le permiten poder programar este dispositivo en C++ desde su IDE y poder ajustar sus librerías según las especificaciones de hardware del dispositivo. La tercera opción de programación es Micropython, que no es más que un interprete de Python para microcontroladores, es decir, es una versión de Python para dispositivos con bajos recursos a nivel de memoria.



**Figura 5.2.** Tarjeta de desarrollo ESP32 WROOM Devkit.

### 5.1.3. RASPBERRY PI 3 B+

El Raspberry Pi 3 B+ es un dispositivo electrónico que posee las mismas capacidades de un computador regular (ver figura 5.3), fue creado por Raspberry Pi Foundation y su propósito principal es motivar la enseñanza en el área informática de los institutos de educación. Su escala reducida ofrece muchas ventajas para trabajar con equipos de bajo costo pudiendo contar con los beneficios de un computador por menos de \$100, permite la conectividad mediante Ethernet, puertos USB, salida de audio analógica, salida de video HDMI, terminales para entradas y salidas digitales de propósito general (GPIO) y varios puertos que admiten varios protocolos de comunicación serial muy comunes para aquellos ya familiarizados con microcontroladores. Al ser prácticamente un computador, este dispositivo debe poseer un sistema operativo, pero a diferencia de los computadores regulares el sistema operativo estará almacenado en un memoria micro SD que se debe insertar en el Raspberry. Los sistemas operativos compatibles con este dispositivo son: Raspbian, Windows 10 IoT, Lubutu, entre otros. Además, otra característica respecto a la programación, al poseer un sistema operativo basado en Linux, por defecto ya se puede programar en los lenguajes Python y C/C++. A continuación se muestran las especificaciones técnicas de este dispositivo:

**Tabla 5.3.** Especificaciones Raspberry Pi 3 B+

Alimentación	Conecotor Jack 12V
Almacenamiento	Depende de MicroSD
Perifericos	40 terminales GPIO, Salida PWM
Comunicaciones	I2C, SPI, WiFi doble banda: 2,4 GHz y 5 GHz, Ethernet: Gigabit Ethernet sobre USB 2.0 (300 Mbps), HDMI, USB 2.0, CSI y DSI para conexión de camara, Bluetooth 4.2, BLE
Procesamiento	CPU + GPU: Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC @ 1.4GHz

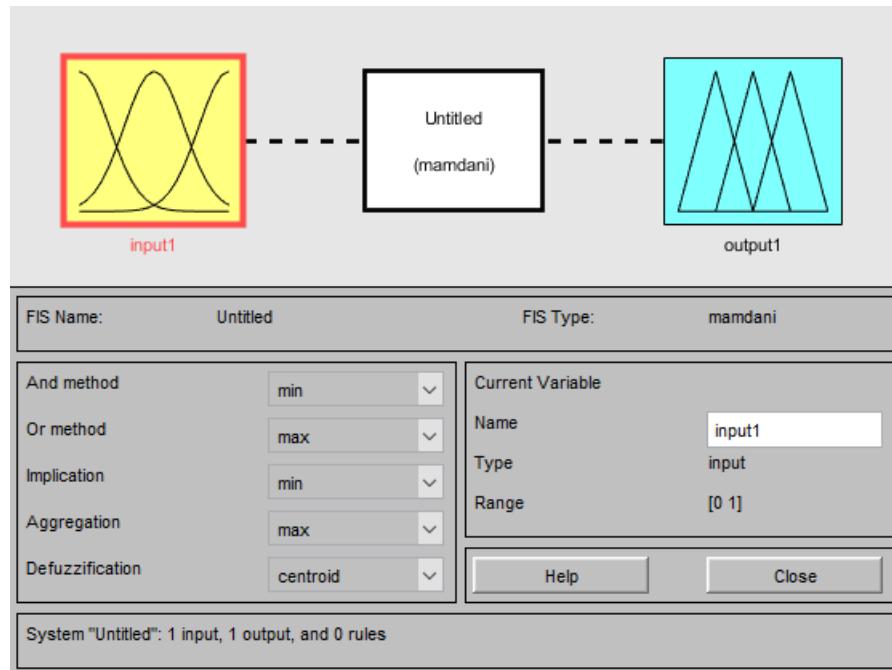


**Figura 5.3.** Raspberry pi 3 B+.

#### 5.1.4. SOFTWARE PARA TRABAJAR CON UN CONTROLADOR DIFUSO

En la tarjeta de desarrollo se debe cargar una librería de control difuso para poder realizar las operaciones matemáticas relacionadas con el sistema difuso, pero primero se debe diseñar el sistema difuso usando el software MATLAB® y su herramienta “*Fuzzy Logic ToolBox*”; en la figura 5.4 se puede observar la ventana editor para el diseño del sistema difuso en el Fuzzy Logic Toolbox. Esta herramienta brinda una interfaz donde el usuario puede definir el tipo de sistema difuso y todas las partes del sistema difuso, vistos en el capítulo 2. Además, esta

herramienta posee la particularidad de poder definir funciones de pertenencia personalizadas.



**Figura 5.4.** Ventana del Editor del Sistema Difuso en el Fuzzy Logic Toolbox.

## eFLL

El software eFLL (Embedded Fuzzy Logic Library) es una librería Open Source desarrollada en C++ por el Grupo de Investigación de Robótica de la Universidad del estado de Piaui en Brasil (Alvez, 2019). En esta librería se pueden implementar sistemas difusos tanto Mandami como Sugeno, las funciones de pertenencia permitidas son triangular, trapezoidal y singleton, el método de implicación que usa es el clásico y la defusificación es a través del método del centroide.

## **SKFUZZY**

SKFuzzy es una librería Open Source que posee una colección de algoritmos difusos desarrollados para Scipy, en principio SKFuzzy fue desarrollado por Josh Warner (Warner, 2019) y a través de los años ha sido modificada por la comunidad de Scipy, agregando módulos para Machine Learning, procesamiento de imágenes y control. Se puede diseñar sistemas difusos tanto Mamdani como Sugeno y permite realizar simulaciones sobre estos sistemas. Asimismo, puede integrar funciones de pertenencia distintas a la triangular y trapezoidal, asignar peso a las reglas y la defuzzificación es por el método del centroide.

### **5.1.5. EVALUACIÓN DE TARJETAS DE DESARROLLO**

Observando las tarjetas de desarrollo y las librerías disponibles, se tomó la decisión respecto a la tarjeta a utilizar considerando la menor cantidad de hardware a integrar y la capacidad de integrar al menos una de estas librerías en el microcontrolador.

El Raspberry Pi 3 B+ es una tarjeta que puede ejecutar programas en Python y C++, por lo tanto se pueden usar cualquiera de las dos librerías sugeridas. Sin embargo, a pesar de que este dispositivo cuenta con muchos atributos a nivel de procesamiento, no cuenta con entradas ADC, por ende se debe integrar un convertidor externo, por lo tanto se descarta esta opción, debido a que no se dispone de este recurso.

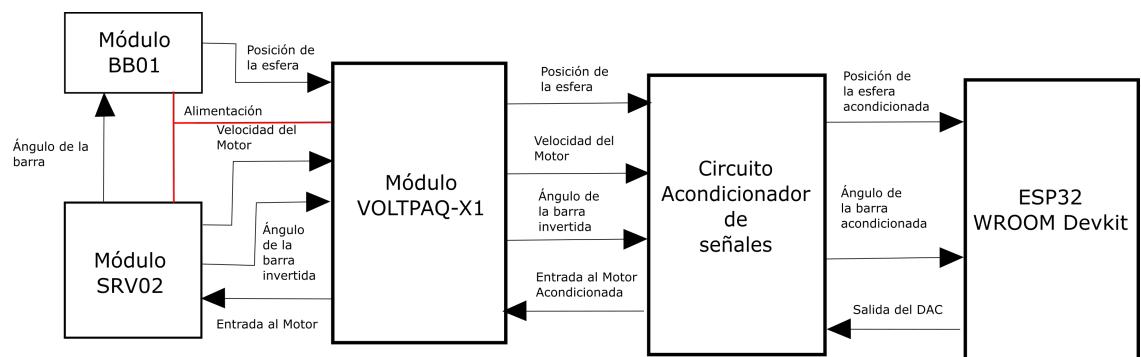
Usando el Arduino UNO se puede llevar a cabo el control con la librería de eFLL, este cuenta con al menos dos entradas con ADC, tiene al menos una salida PWM y puede enviar y transmitir datos por UART. Dados estos atributos, hace parecer a la tarjeta Arduino UNO como una opción viable a emplear, pero, por medio de pruebas, se encontró que presenta problemas a la hora de crear más

de dos sistemas difusos usando la librería eFFL, el comportamiento es inestable y comienza a retornar valores erroneos en la defusificación. La estructura del programa para realizar el control se usan dos sistemas difusos, por lo tanto esta opción también fue descartada.

La decisión fue escoger al ESP32 debido a que posee todos los requerimientos necesarios para realizar el control, DAC, ADC y puerto serial. Además, se puede programar en Micropython y C/C++, por lo tanto se tiene la posibilidad de usar cualquiera de las librerías mencionadas anteriormente. Para asegurar el control con este dispositivo se realizaron las mismas pruebas hechas con la tarjeta Arduino UNO pero este no presentó problemas, por lo tanto con esta tarjeta se decidió implementar el controlador.

## 5.2. DISEÑO DE LA INTERFAZ ENTRE EL SISTEMA BALL AND BEAM Y ESP32

Las salidas de los sensores del Ball and Beam sobrepasan los niveles de tensión de las entradas del ADC del ESP32, por lo tanto se realizó un acondicionamiento de señal siguiendo el diagrama de bloques de la siguiente figura.

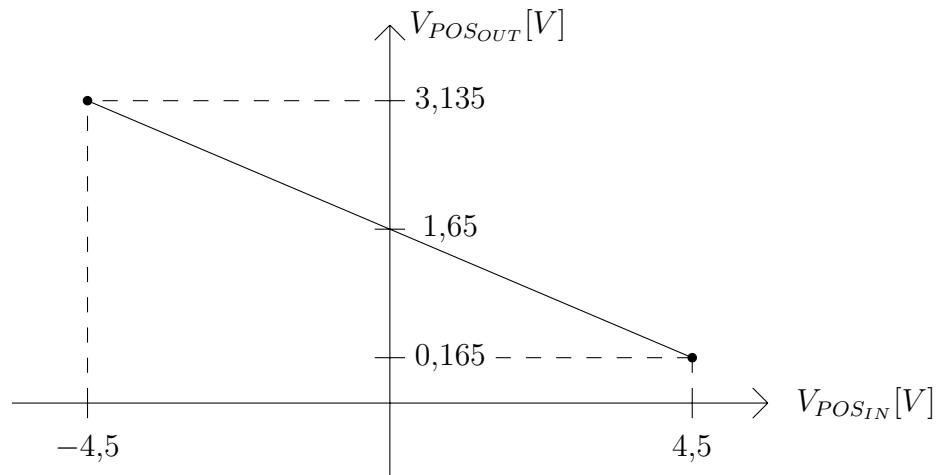


**Figura 5.5.** Diagrama de bloques del sistema ball and beam con el circuito acondicionador de señales y el ESP32 WROOM Devkit.

Para realizar el circuito acondicionador de señales, se partió del diseño de

un controlador digital realizado por Hernandez (Hernández, 2018), en el cual se consideró que un filtro pasa bajo de segundo orden con frecuencia de corte de 86 Hz (valor obtenido por Hernandez experimentalmente), es capaz de seguir a la posición de la esfera y el ángulo de la barra sin incorporar ruido significativo a la medición. Sin embargo, este diseño está en función de 0 a 5V y el nivel DC de ajuste fue el mismo para ambas variables (posición de la esfera y ángulo de la barra). Entonces, en este caso el diseño se basó en las características de transferencias que se observan en las figuras 5.6 y 5.8, por ende siguiendo estas funciones la posición de la esfera se representa con la ecuación de la recta que se muestra a continuación:

$$V_{POS_{OUT}} = 0,33V_{POS_{IN}} + 1,65 \quad (5.1)$$



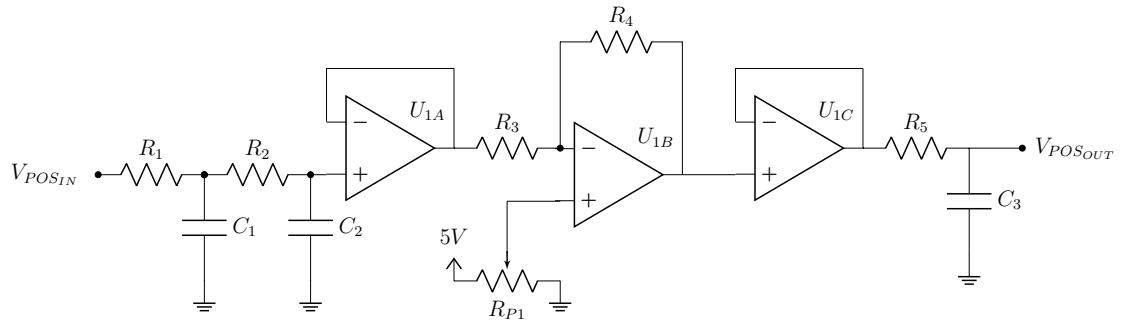
**Figura 5.6.** Característica de transferencia para señal de posición.

En la implementación de esta ecuación se uso un amplificador operacional como se puede observar en la figura 5.7, donde es claro que para cumplir la ecuación  $R_4/R_3$  debe ser igual a 0,33 y la ganancia del nivel DC  $V_r$ , cuya función es trasladar la curva, es igual a 1,65, entonces:

$$V_r \left( 1 + \frac{R_4}{R_3} \right) = 1,65 \quad (5.2)$$

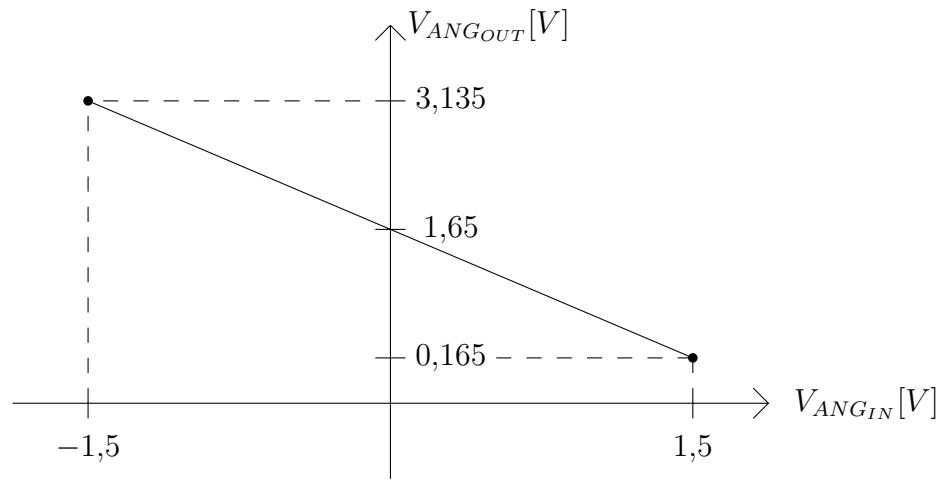
$$V_r = 1,24V \quad (5.3)$$

Otro factor importante a la hora de diseñar el circuito acondicionador de señales, fue incorporar a cada salida un filtro pasa bajo con frecuencia de corte 1kHz, por recomendaciones del fabricante del ESP32 para mejorar la lectura del ADC.



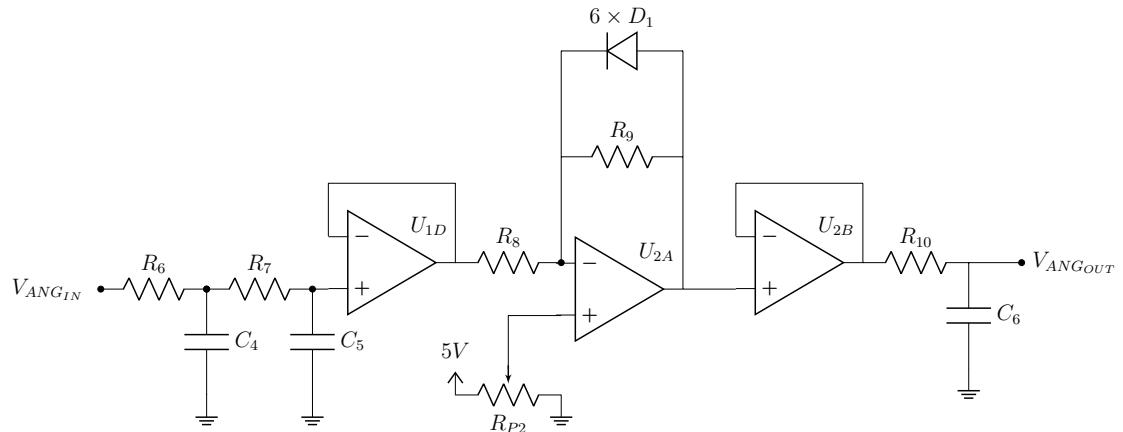
**Figura 5.7.** Acondicionamiento de señal sensor de posición.

Análogamente, para el ángulo de la barra el procedimiento fue el mismo, solo que se considera la característica de transferencia del ángulo de la barra.



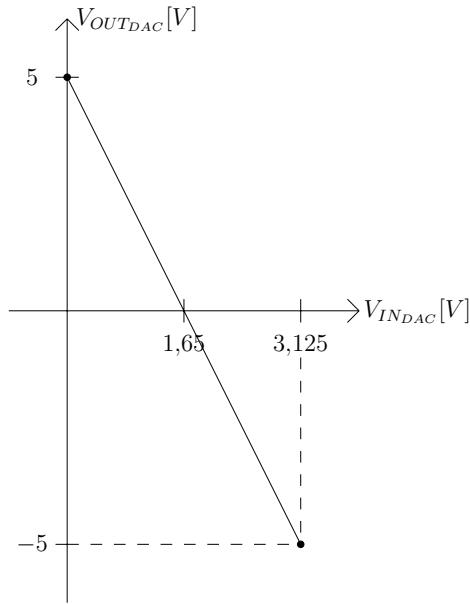
**Figura 5.8.** Característica de transferencia para señal del ángulo de la barra.

Sin embargo, para este diseño como se muestra en la figura 5.9, se integraron seis diodos para retener los valores de tensión hasta 3,3V, debido a que cuando el angulo sobrepasa los  $45^\circ$ , se excede el valor límite del ADC. La cantidad de diodos fue obtenida experimentalmente.

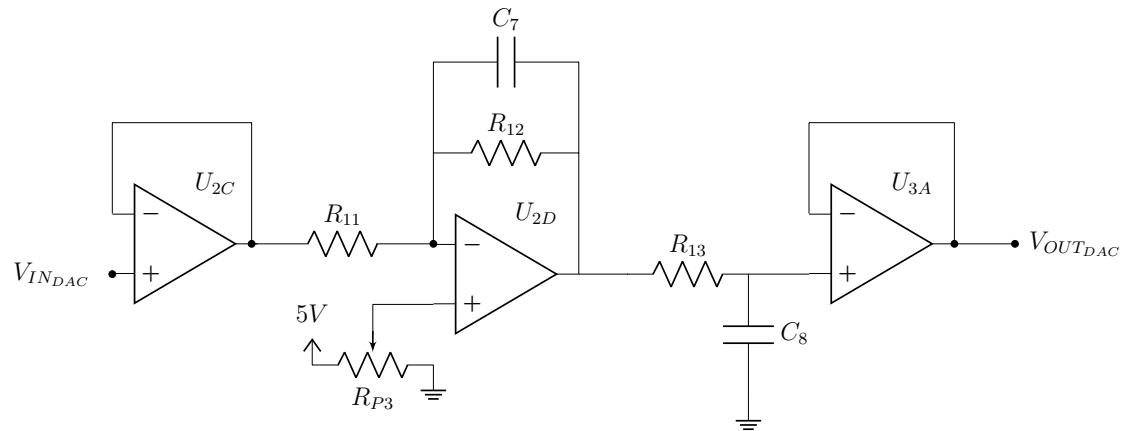


**Figura 5.9.** Acondicionamiento de señal sensor de ángulo de la barra.

La salida del DAC del ESP32 va de 0 a 3,125V y la entrada del motor va de -5V a 5V, entonces se diseñó el circuito de la figura 5.11 siguiendo la característica de transferencia de la figura 5.10, donde se integró un filtro activo y uno pasivo pasa bajo, cuya frecuencia de corte es de 48Hz, esto con el propósito de cumplir los límites en frecuencia a la entrada del motor como lo especifica el manual (Apkarian y cols., 2011a).



**Figura 5.10.** Característica de transferencia de la señal del motor.



**Figura 5.11.** Acondicionamiento de señal para el motor

En resumen, para poder cumplir con todas las características de transferencia, frecuencias de corte de los filtros pasa bajos y ganancias necesarias para acondicionar la señal para el microcontrolador, se usaron los siguientes componentes:

**Tabla 5.4.** Componentes para el acondicionamiento de señal de posición.

Símbolo	Descripción	Valor
$R_1, R_2, R_6, R_7$	Resistencia de carbón, 5 %	$5,6k\Omega$
$R_3$	Resistencia de carbón, 5 %	$330\Omega$
$R_4$	Resistencia de carbón, 5 %	$1k\Omega$
$R_5, R_8, R_9, R_{10}$	Resistencia de carbón, 5 %	$1,5k\Omega$
$R_{11}, R_{13}$	Resistencia de carbón, 5 %	$10k\Omega$
$R_{12}$	Resistencia de carbón, 5 %	$33k\Omega$
$C_1, C_2, C_4, C_5, C_8$	Condensador de Poliéster	$330nF$
$C_3, C_6, C_7$	Condensador Axial Cerámico	$100nF$
$D_1$	Diodo de señal	1N4148
$U_1, U_2$	Cuádruple Amplificador Operacional	LM324

## CAPÍTULO VI

### DISEÑO Y AJUSTE DE LOS CONTROLADORES DIFUSOS

Partiendo del manual de laboratorio de Quanser para el sistema Ball and Beam, donde se sugiere que la topología de control se haga mediante dos lazos, como se pude observar en la figura 6.1 (Apkarian y cols., 2011b), el primer lazo se encarga del control del ángulo de la barra y el segundo de la posición de la esfera, con esta topología la Consigna del control del ángulo de la barra viene dado por la salida del control de la posición de la esfera, por lo tanto para poder controlar el sistema se realizaron dos sistemas difusos, uno para cada lazo.

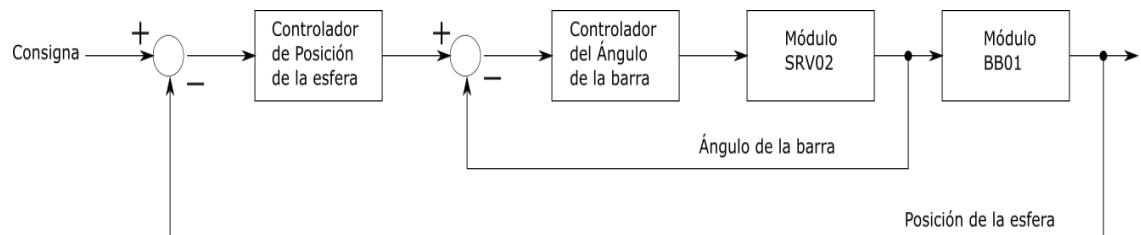
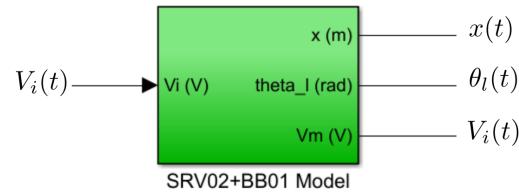
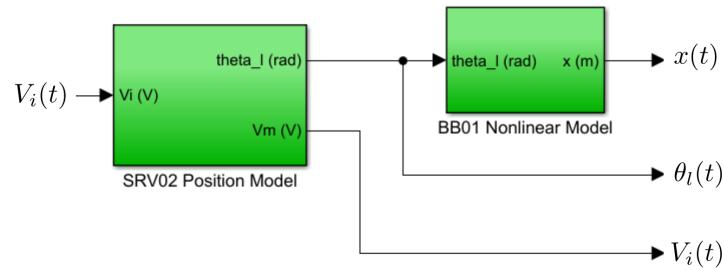


Figura 6.1. Topología de control de doble lazo.

Las pruebas del controlador difuso se realizaron en Simulink empleando las ecuaciones planteadas en el capitulo 4, la ecuación no lineal que describe el movimiento de la esfera sobre la barra (ecuación 4.11) y la ecuación lineal del servo motor(ecuación 4.31). En la simulación la planta se representa por el bloque de la figura 6.2, en el cual se encuentra dos sub-bloques, uno para el ángulo de la barra y otro para la posición de la esfera como se muestra en la figura 6.3.

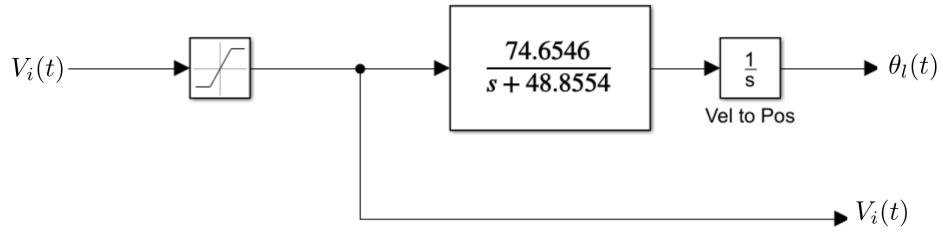


**Figura 6.2.** Bloque de la planta.



**Figura 6.3.** Bloques internos de la planta, ángulo y posición.

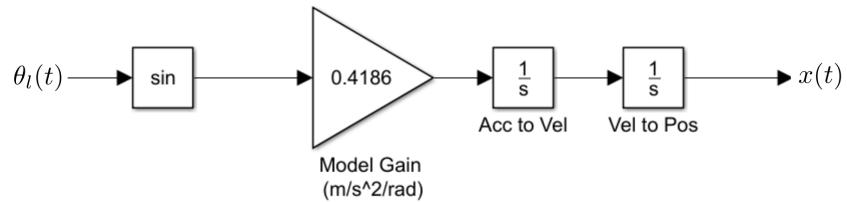
Dentro del bloque que representa el ángulo de la barra de la figura 6.3 se encuentra la función de transferencia equivalente al aplicar la transformada de Laplace en la ecuación 4.31 (ver figura 6.4). Además, también se agregó una saturación, esto se debe a que en las condiciones reales del sistema la entrada máxima al motor es 5V y la mínima es -5V, de acuerdo al manual del usuario (Apkarian y cols., 2011a).



**Figura 6.4.** Diagrama de bloques para el ángulo de la barra.

A diferencia de la figura 6.4 en la figura 6.5 no se puede representar por medio de la transformada de Laplace al ser un sistema no lineal, por lo tanto se representó de forma explícita con la ecuación no lineal, la cual despejando la posición de la ecuación 4.32, se obtiene:

$$x(t) = \iint 0,4186 \operatorname{sen}(\theta_l(t)) dt^2 \quad (6.1)$$



**Figura 6.5.** Diagrama de bloques para la posición.

## 6.1. TOPOLOGÍA DE CONTROL DE DOBLE LAZO

En esta topología como se muestra en la figura 6.1, el control se realiza mediante dos lazos, el primero para el control del ángulo de la barra y el segundo para el control de la posición de la esfera, donde la salida del controlador de posición

de la esfera es la consigna del controlador del ángulo de la barra lo que quiere decir que ambos controladores están conectados en cascada.

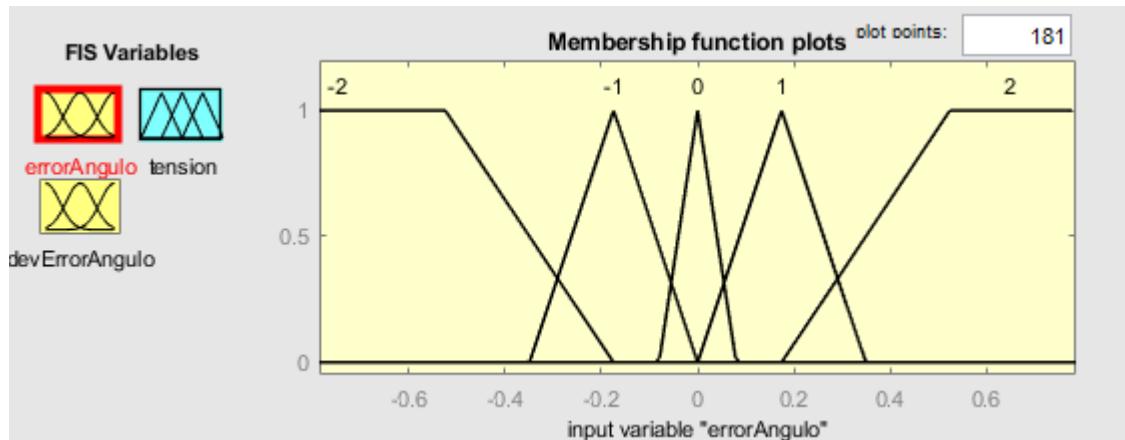
### **6.1.1. DISEÑO DEL CONTROLADOR DIFUSO PARA EL ÁNGULO DE LA BARRA**

Las entradas para el controlador difuso para el ángulo de la barra son el error del ángulo, la derivada del error del ángulo y su salida es la tensión que se aplica al motor. En el error del ángulo se usan los términos lingüísticos: “muy abajo”, “abajo”, “cero”, “arriba” y “muy arriba” para identificar donde se encuentra la posición angular de la barra respecto al ángulo deseado.

Por otro lado, para la derivada del error se usaron términos lingüísticos que reflejan que tan rápido cambia este error y con que signo: “positivo rápido”, “positivo lento”, “cero”, “negativo lento” y “negativo rápido”. Respecto a la salida del controlador se usaron los términos lingüísticos: “horario rápido”, “horario lento”, “cero”, “anti-horario lento” y “anti-horario rápido”, los cuales representan el sentido en que se mueve el motor y que tan rápido se desea.

Las funciones de pertenencia se representaron a través de la función única (ecuación 2.15). Asimismo, la base de reglas se representa a través de una matriz que contiene un valor numérico lingüístico, el cual representa el término lingüístico de cada función de pertenencia.

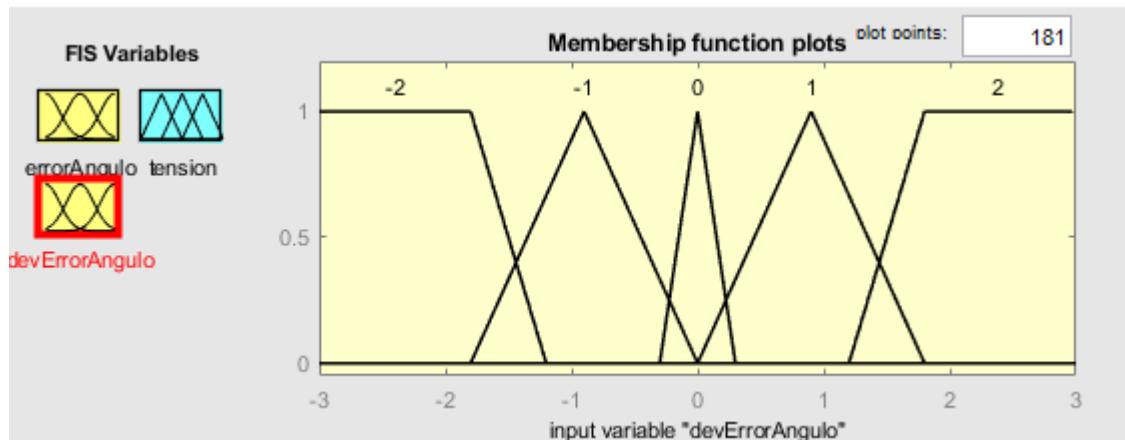
A continuación se muestran las funciones de pertenencia para cada entrada y salida y la base de reglas del controlador difuso del ángulo de la barra:



**Figura 6.6.** Funciones de pertenencia para la entrada del error del ángulo.

**Tabla 6.1.** Parámetros de las funciones de pertenencia para la entrada del error del ángulo.

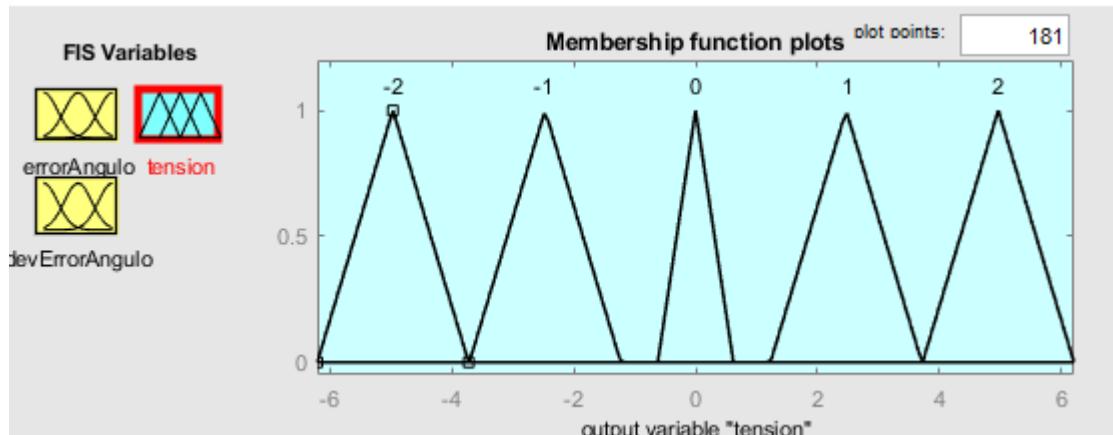
Valor Numérico Lingüístico	Término Lingüístico	Parámetros [a, b, c, d, nT, n1, n2]
-2	Muy abajo	[-0.785, -0.785, -0.5236, -0.1745, 0, 1, 1]
-1	Abajo	[-0.349, -0.174, -0.174, 0, 0, 1, 1]
0	Cero	[-0.08, 0, 0, 0.08, 0, 1, 1]
1	Arriba	[0, 0.174, 0.174, 0.349, 0, 1, 1]
2	Muy arriba	[0.1745, 0.5236, 0.785, 0.785, 0, 1, 1]



**Figura 6.7.** Funciones de pertenencia de la derivada del error del ángulo.

**Tabla 6.2.** Parámetros de las funciones de pertenencia de la derivada del error del ángulo.

Valor Numérico Lingüístico	Término Lingüístico	Parámetros [a, b, c, d, nT, n1, n2]
-2	Positivo Rápido	[ -3, -3, -1.8, -1.2, 0, 1, 1 ]
-1	Positivo Lento	[ -1.8, -0.9, -0.9, 0, 1, 1 ]
0	Cero	[ -0.3, 0, 0, 0.3, 0, 1, 1 ]
1	Negativo Lento	[ 0, 0.9, 0.9, 1.8, 0, 1, 1 ]
2	Negativo Rápido	[ 1.2, 1.8, 3, 3, 0, 1, 1 ]



**Figura 6.8.** Funciones de pertenencia para la salida del controlador difuso para el ángulo de la barra.

**Tabla 6.3.** Parámetros de las funciones de pertenencia para la salida del controlador difuso para el ángulo de la barra.

Valor Numérico Lingüístico	Término Lingüístico	Parámetros [a, b, c, d]
-2	Antihorario Rápido	[ -6.2, -4.96, -4.96, -3.72 ]
-1	Antihorario Lento	[ -3.707, -2.467, -2.467, -1.227 ]
0	Cero	[ -0.62, 0, 0, 0.62 ]
1	Horario Lento	[ 1.227, 2.467, 2.467, 3.707 ]
2	Horario Rápido	[ 3.72, 4.96, 4.96, 6.2 ]

**Tabla 6.4.** Base de reglas para el controlador difuso del ángulo de la barra.

Tensión		Derivada del Error del Ángulo				
		-2	-1	0	1	2
Error del Ángulo	-2	-2	-2	-2	-2	-2
	-1	-2	-1	-1	-1	-1
	0	-1	0	0	0	1
	1	1	1	1	1	2
	2	2	2	2	2	2

### 6.1.2. SIMULACIÓN DEL CONTROLADOR DIFUSO PARA EL ÁNGULO DE LA BARRA

Una vez establecida las funciones de pertenencia y la base de reglas del controlador del ángulo de la barra, se realizó la simulación mediante el diagrama de bloques que se muestra en la figura 6.9, donde se agregaron los filtros del hardware diseñado. Para el sensor de posición de la esfera y el ángulo de la barra el filtro de segundo orden correspondiente tiene una frecuencia de corte de 86Hz, por lo tanto su función de transferencia es:

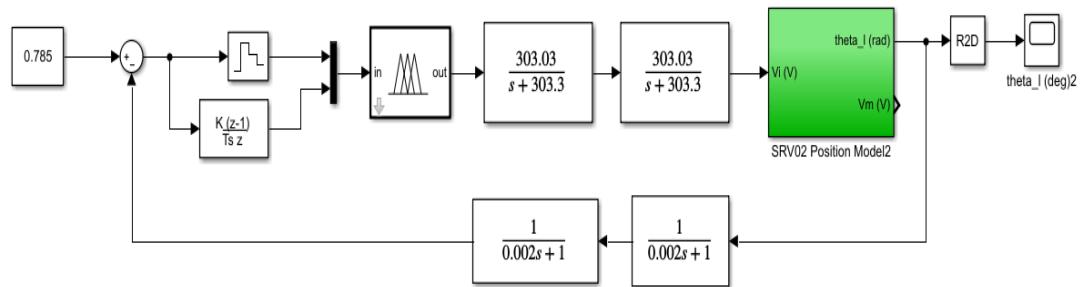
$$\frac{V_{out}(s)}{V_{in}(s)} = \frac{1}{s^2 0,002 + 1} \quad (6.2)$$

En la salida del controlador se colocaron dos filtros de frecuencias de corte 48Hz como se menciono en el capítulo 5 cuya función de transferencia es:

$$\frac{V_{out}(s)}{V_{in}(s)} = \frac{303,73}{s + 303,73} \quad (6.3)$$

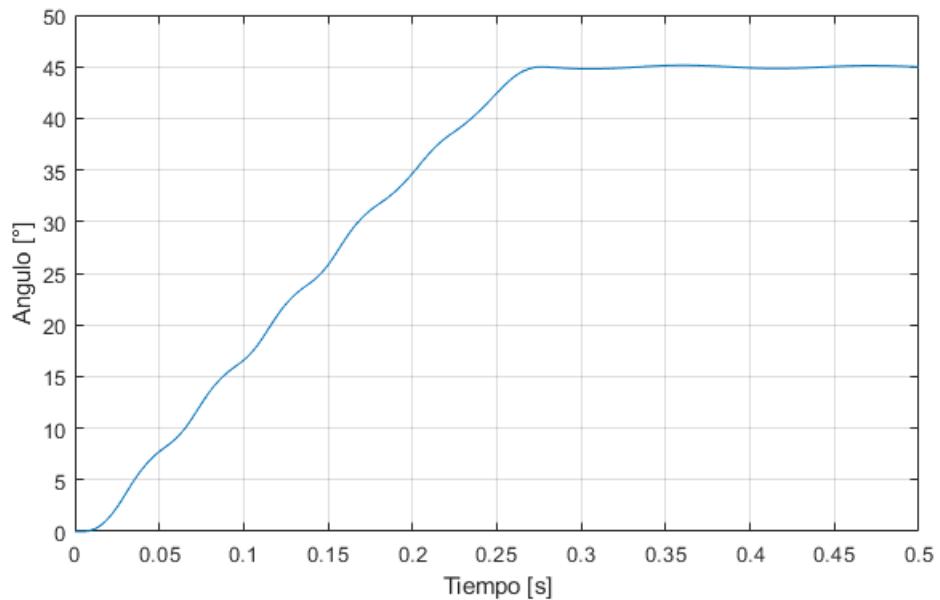
La simulación solo se considero el retardo de los filtros dados por las funciones de transferencia de las ecuaciones 6.2 y 6.3. El ángulo de la barra inicialmente

en cero ( $\theta_l(0) = 0$ ) y sin velocidad angular ( $\dot{\theta}_l(0) = 0$ ), con un valor deseado de  $45^\circ$  y un tiempo de muestreo de 12 ms, tiempo obtenido del teorema de Shanon, ya que el armónico fundamental de la placa se encuentra en 7.7 Hz, entonces el tiempo de muestreo debe ser de aproximadamente 77 Hz lo que en el dominio del tiempo es 12.9 ms, valor con el cual se partió y se obtuvieron buenos resultados con 12 ms.



**Figura 6.9.** Diagrama de bloques para la simulación con controlador difuso del ángulo de la barra.

Luego de realizar la simulación se obtuvo la respuesta del sistema que se observa en la figura 6.10 y el desempeño del sistema se muestra en la tabla 6.5.



**Figura 6.10.** Simulación del ángulo de la barra, Set point:  $45^\circ$ , tiempo de muestreo: 12ms,  $\theta_l(0) = 0$  y  $\dot{\theta}_l(0) = 0$ .

**Tabla 6.5.** Desempeño del sistema con controlador difuso del ángulo de la barra.

Tiempo de establecimiento [s]	0.2645
Sobre Pico [%]	0.9649
Tiempo de alza [s]	0.2057

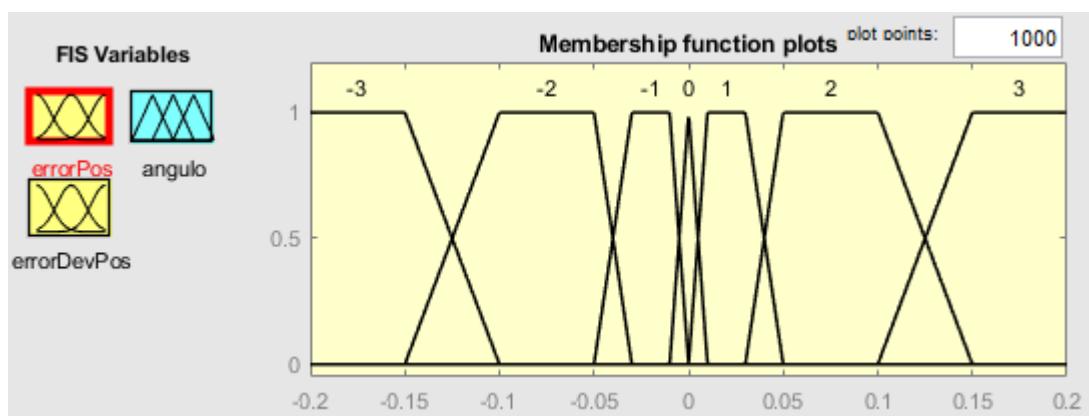
### 6.1.3. DISEÑO DE CONTROLADOR DIFUSO PARA LA POSICIÓN DE LA ESFERA

El diseño del controlador difuso de la posición de la esfera se basa en dos entradas y una salida, la primera entrada es el error de la posición y la segunda es la derivada del error de la posición, la salida es el ángulo necesario para llevar la esfera a la consigna.

El error de la posición se representó con los términos lingüísticos: “muy derecha”, “derecha”, “poco derecha”, “cero”, “poco izquierda”, “izquierda” y “muy

izquierda” para identificar donde se encuentra la esfera del valor deseado, por otro lado, la derivada del error al igual que el controlador del ángulo de la barra muestra que tan rápido se mueve la esfera y en que dirección, donde los términos lingüísticos que se usan son: “positivo rápido”, “positivo lento”, “cero”, “negativo lento” y “negativo rápido”. Para finalizar la salida se identifica con los términos lingüísticos: “mucho horario”, “horario”, “poco horario”, “cero”, “poco anti-horario”, “anti-horario” y “mucho anti-horario” lo que muestra hacia donde se debe mover el ángulo de la barra para mover la esfera al valor deseado.

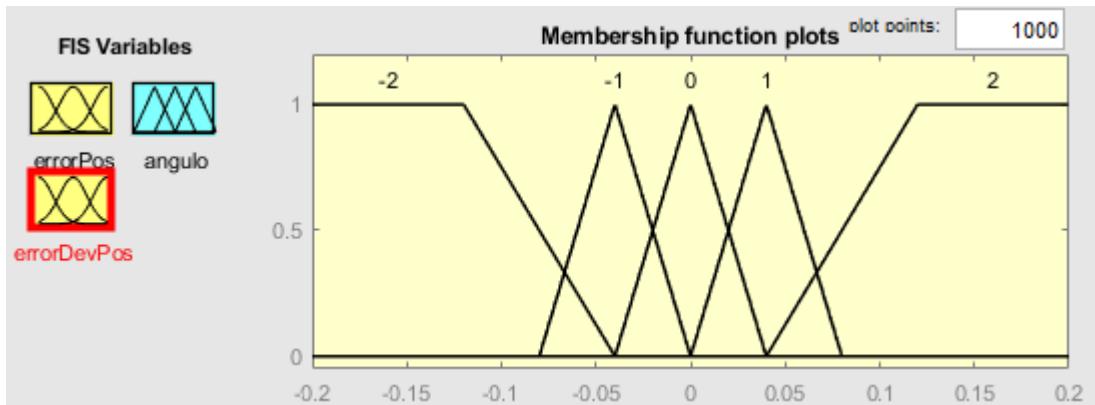
De la misma forma que el controlador anterior, las funciones de pertenencia se representan con la función única y la base de reglas con una matriz, como se muestra a continuación:



**Figura 6.11.** Funciones de pertenencia para la entrada del error de posición.

**Tabla 6.6.** Parámetros de las funciones de pertenencia para la entrada del error de posición.

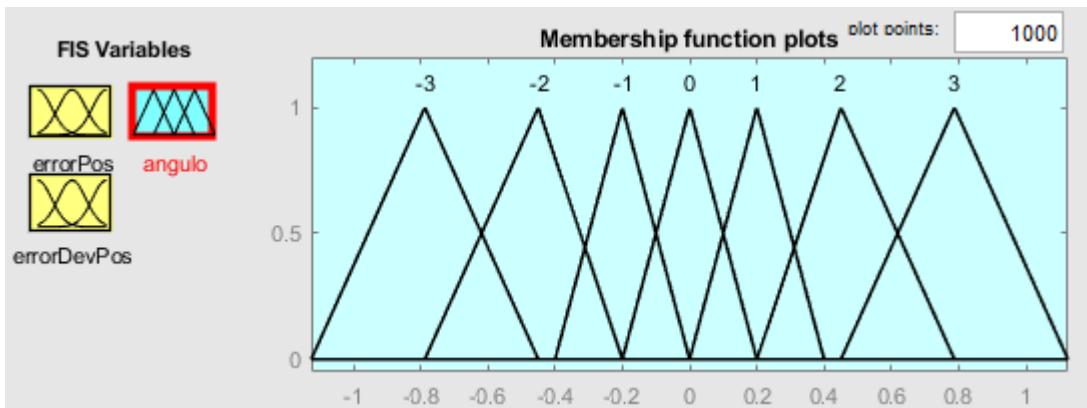
Valor Numérico Lingüístico	Término Lingüístico	Parámetros [a, b, c, d, nT, n1, n2]
-3	Muy derecha	[-0.2, -0.2, -0.15, -0.1, 0, 1, 1]
-2	Derecha	[-0.15, -0.1, -0.05, -0.03, 0, 1, 1]
-1	Poco derecha	[-0.05, -0.03, -0.01, 0, 0, 1, 1]
0	Cero	[-0.01, 0, 0, 0.01, 0, 1, 1]
1	Poco izquierda	[0, 0.01, 0.03, 0.05, 0, 1, 1]
2	Izquierda	[0.03, 0.05, 0.1, 0.15, 0, 1, 1]
3	Muy izquierda	[0.1, 0.15, 0.2, 0.2, 0, 1, 1]



**Figura 6.12.** Funciones de pertenencia para la entrada de la derivada del error de posición.

**Tabla 6.7.** Parámetros de las funciones de pertenencia para la entrada de la derivada del error de posición.

Valor Numérico Lingüístico	Término Lingüístico	Parámetros [a, b, c, d, nT, n1, n2]
-2	Positivo Rápido	[-0.2, -0.2, -0.12, -0.04, 0, 1, 1]
-1	Positivo Lento	[-0.08, -0.04, -0.04, 0, 0, 1, 1]
0	Cero	[-0.04, 0, 0, 0.04, 0, 1, 1]
1	Negativo Lento	[0, 0.04, 0.04, 0.08, 0, 1, 1]
2	Negativo Rápido	[0.04, 0.12, 0.2, 0.2, 0, 1, 1]



**Figura 6.13.** Funciones de pertenencia para la salida del controlador difuso de la posición.

**Tabla 6.8.** Parámetros de las funciones de pertenencia para la salida del controlador difuso de la posición.

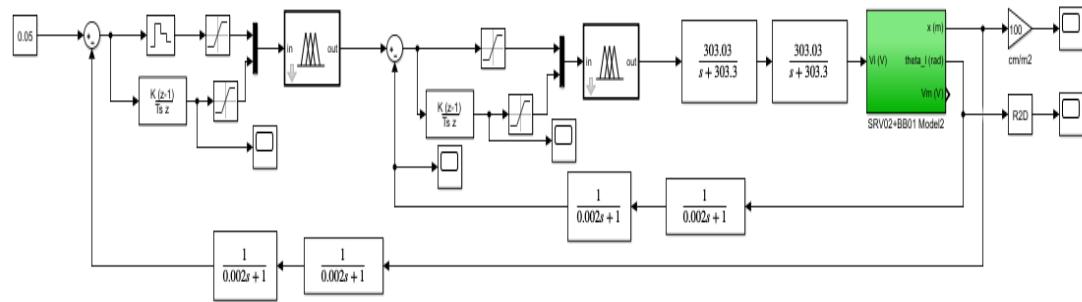
Valor Numérico Lingüístico	Término Lingüístico	Parámetros [a, b, c, d]
-3	Mucho Antihorario	[-1.125, -0.7875, -0.7875, -0.45]
-2	Antihorario	[-0.7875, -0.45, -0.45, -0.2]
-1	Poco Antihorario	[-0.4, -0.2, -0.2, 0]
0	Cero	[-0.2, 0, 0, 0.2]
1	Poco Horario	[0, 0.2, 0.2, 0.4]
2	Horario	[0.2, 0.45, 0.45, 0.7875]
3	Mucho Horario	[0.45, 0.7875, 0.7875, 1.125]

**Tabla 6.9.** Base de reglas para el controlador difuso de la posición.

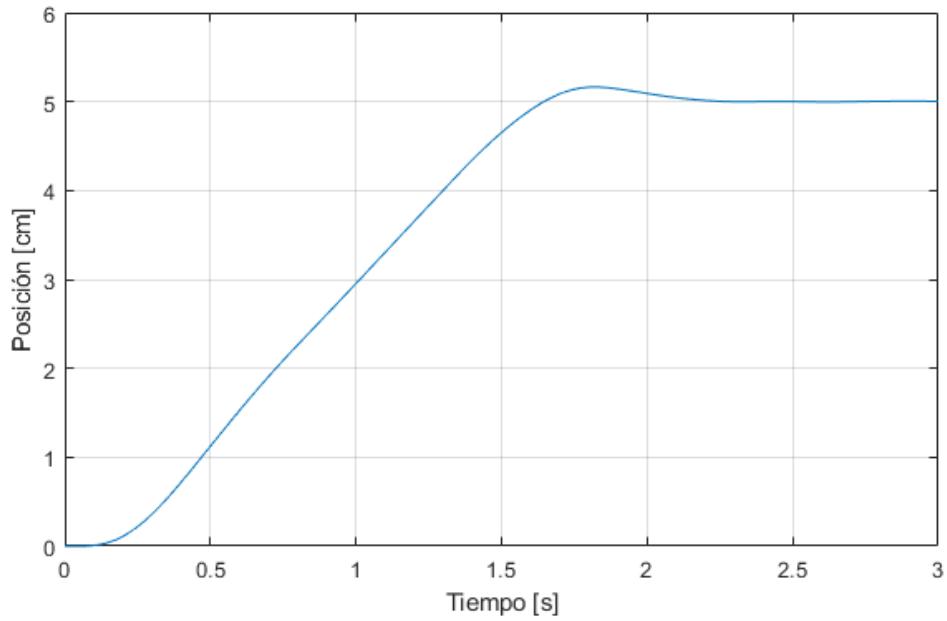
Ángulo		Derivada del Error de Posición				
		-2	-1	0	1	2
Error de Posición	-3	-3	-3	-3	-3	-2
	-2	-3	-3	-2	-1	1
	-1	-3	-3	-2	1	3
	0	-3	-2	0	2	3
	1	-3	-1	2	3	3
	2	-1	1	2	3	3
	3	2	3	3	3	3

#### 6.1.4. SIMULACIÓN DE CONTROLADOR DIFUSO PARA LA POSICIÓN DE LA ESFERA

La simulación se hizo a través del diagrama de bloques que se muestra en la figura 6.14, donde se agregaron saturaciones en las entradas de ambos controladores para asegurar que estas se encontraran dentro del universo de discurso de los controladores. Las condiciones iniciales para esta simulación fueron  $x(0) = 0$ ,  $\dot{x}(0) = 0$ ,  $\theta_l(0) = 0$  y  $\dot{\theta}_l(0) = 0$ , el tiempo de muestreo fue de 12 ms, la consigna fue 5 cm, su resultado se puede observar en la figura 6.15 y el desempeño del sistema en la tabla 6.10.



**Figura 6.14.** Diagrama de bloques para la simulación con el controlador difuso de la posición.



**Figura 6.15.** Simulación de posición de la esfera, Set point: 5cm, tiempo de muestreo: 12ms,  $x(0) = 0$ ,  $\dot{x}(0) = 0$ ,  $\theta_l(0) = 0$  y  $\dot{\theta}_l(0) = 0$ .

**Tabla 6.10.** Desempeño del sistema con controlador difuso de la posición.

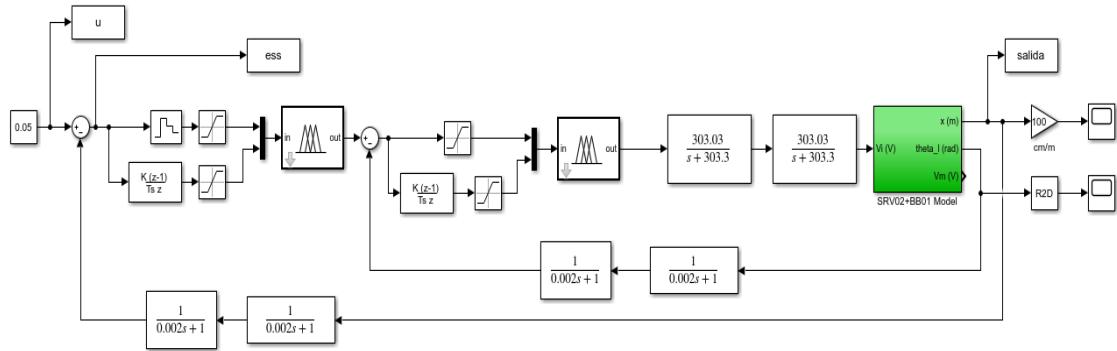
Tiempo de establecimiento [s]	1.9777
Sobre Pico [%]	2.9989
Tiempo de alza [s]	1.1214

### 6.1.5. AJUSTE DE CONTROLADOR DIFUSO PARA LA POSICIÓN DE LA ESFERA

El ajuste del controlador difuso de la posición de la esfera se hizo mediante el diagrama de bloques de la figura 6.16, donde se extrae la data del error de posición, la consigna y la salida con los bloques *ess*, *u* y *salida* respectivamente, esto con el propósito de ejecutar el algoritmo de optimización por búsqueda directa (Tremante, 2019), donde se variaron las funciones de pertenencia en cada iteración siguiendo el diagrama de flujo de la figura 2.17, donde la función objetivo que se

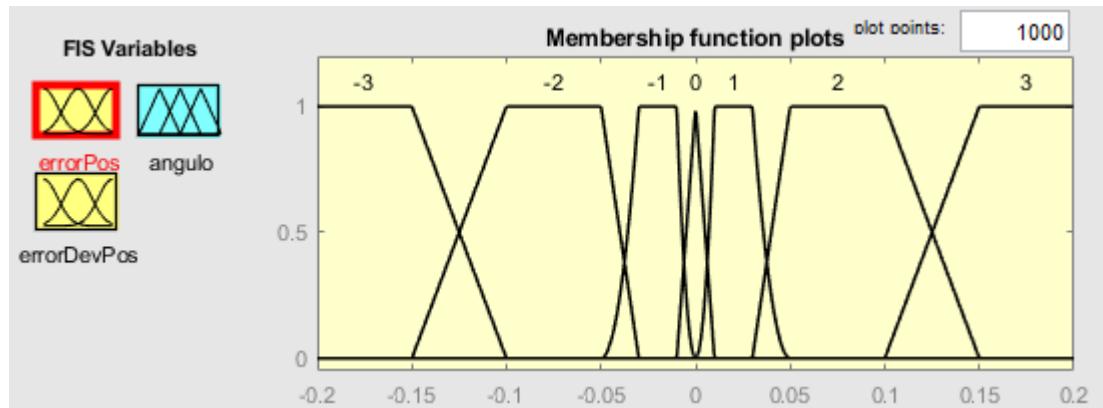
usó fue la siguiente:

$$f = 0,5 \sum_{i=1}^n e^2(n) + 0,25|E_{ss}-0,01| + 0,15|s_{max}-r-0,1| + 0,05 \frac{t_s(E_{ss} - 0,01)}{T} + 0,05 \frac{t_a}{T} \quad (6.4)$$



**Figura 6.16.** Diagrama de bloques para ajustar el controlador difuso de la posición de la esfera.

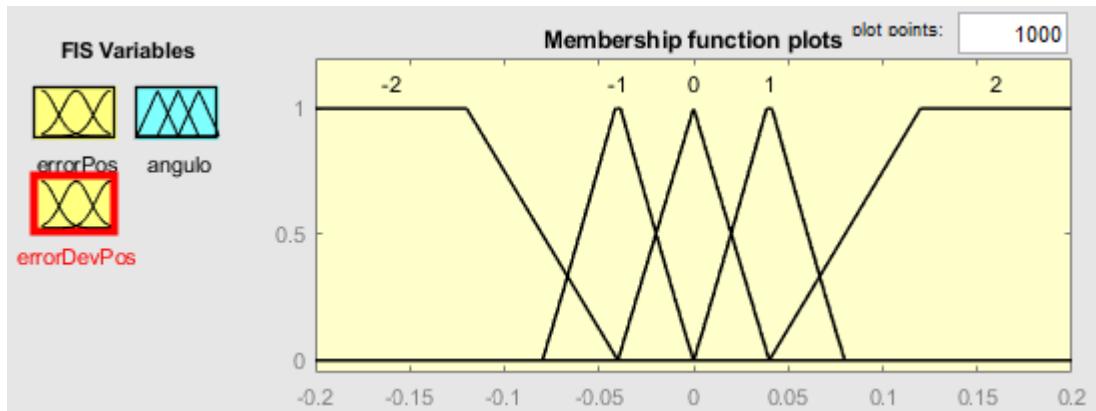
Una vez ajustadas las funciones de pertenencia de las entradas del error de posición y la derivada del error de posición, éstas cambiaron como se muestra a continuación:



**Figura 6.17.** Funciones de pertenencia ajustadas para la entrada del error de posición.

**Tabla 6.11.** Parámetros de las funciones de pertenencia ajustadas para la entrada del error de posición.

Valor Numérico Lingüístico	Término Lingüístico	Parámetros [a, b, c, d, nT, n1, n2]
-3	Muy derecha	[-0.2, -0.2, -0.15, -0.1, 0, 1, 1]
-2	Derecha	[-0.15, -0.1, -0.05, -0.03, 0, 1, 1]
-1	Poco derecha	[-0.05, -0.03, -0.01, 0, 0, 1, 2]
0	Cero	[-0.0098, 0, 0, 0.0098, 0, 1.002, 1]
1	Poco izquierda	[0, 0.01, 0.03, 0.05, 0, 1, 2]
2	Izquierda	[0.03, 0.05, 0.1, 0.15, 0, 1, 1]
3	Muy izquierda	[0.1, 0.15, 0.2, 0.2, 0, 1, 1]



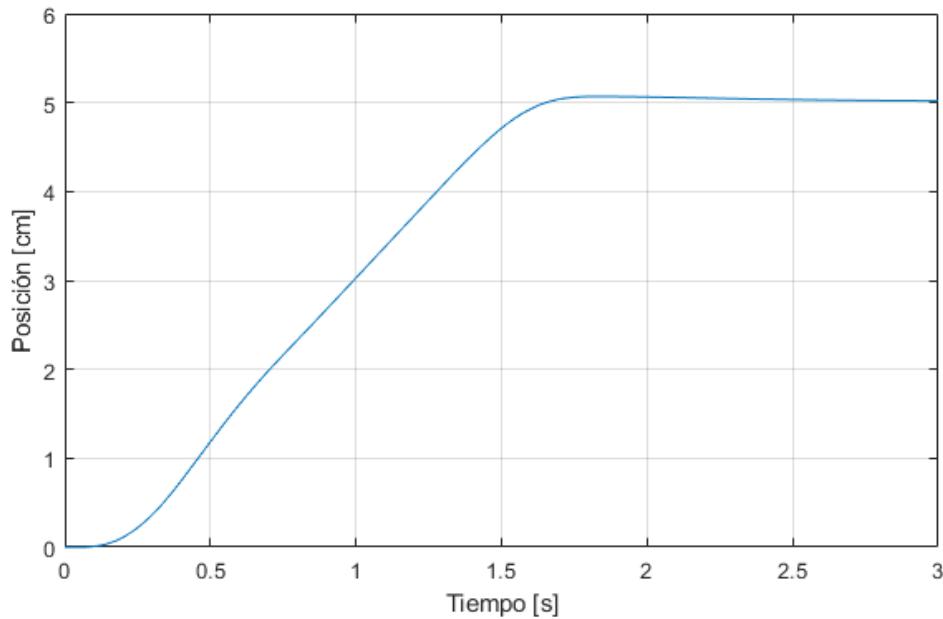
**Figura 6.18.** Funciones de pertenencia ajustadas para la entrada de la derivada del error de posición.

**Tabla 6.12.** Parámetros de las funciones de pertenencia ajustadas para la entrada de la derivada del error de posición.

Valor Numérico Lingüístico	Término Lingüístico	Parámetros [a, b, c, d, nT, n1, n2]
-2	Positivo Rápido	[-0.2, -0.2, -0.12, -0.04, 0, 1, 1]
-1	Positivo Lento	[-0.08, -0.0412, -0.0387, 0, 0.0039, 1, 1.031]
0	Cero	[-0.04, 0, 0, 0.04, 0, 1, 1]
1	Negativo Lento	[0, 0.0387, 0.0412, 0.08, 0.0039, 1, 1.031]
2	Negativo Rápido	[0.04, 0.12, 0.2, 0.2, 0, 1, 1]

### 6.1.6. SIMULACIÓN DEL CONTROLADOR DIFUSO AJUSTADO PARA LA POSICIÓN DE LA ESFERA

Cambiando las funciones de pertenencia iniciales por las obtenidas en el proceso de ajuste, se obtuvo el resultado que se muestra en la figura 6.19 y un desempeño del sistema que se observa en la tabla 6.13.

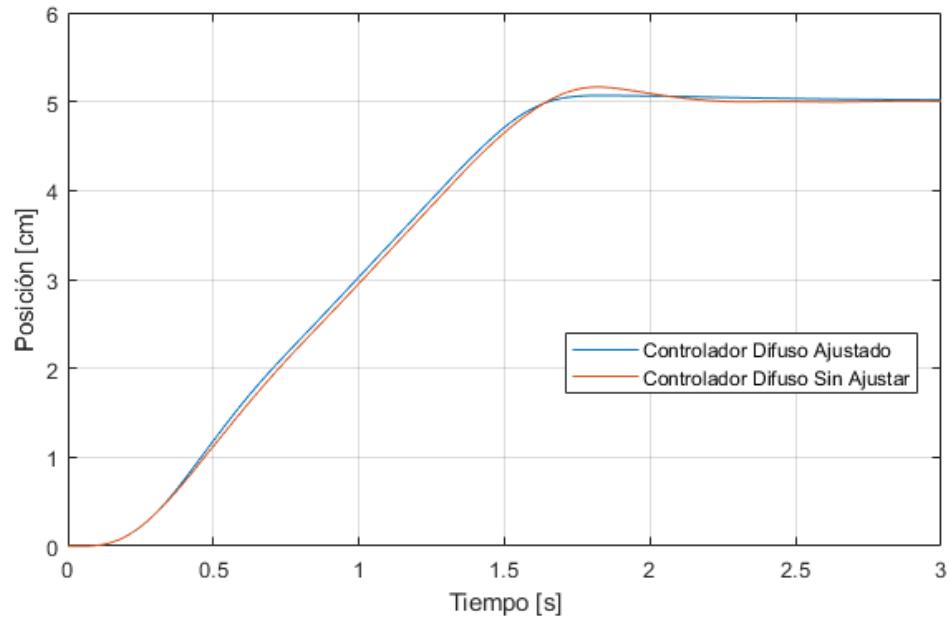


**Figura 6.19.** Simulación de Posición con controlador difuso ajustado, Set Point: 5cm, Tiempo de muestreo: 12ms,  $x(0) = 0$ ,  $\dot{x}(0) = 0$ ,  $\theta_l(0) = 0$  y  $\dot{\theta}_l(0) = 0$ .

**Tabla 6.13.** Desempeño del sistema para controlador difuso de posición ajustado.

Tiempo de establecimiento [s]	1.6107
Sobre Pico [%]	0.9372
Tiempo de alza [s]	1.1030

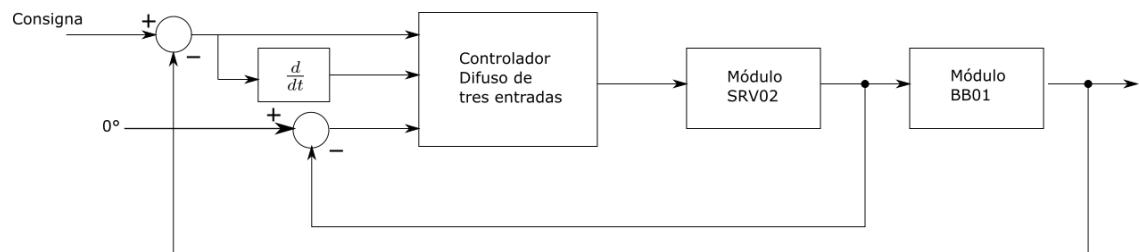
A continuación se muestra en la figura 6.20 una comparación entre el controlador difuso de la posición de la esfera sin ajustar y ajustado, donde se puede observar que el sobre pico porcentual y el tiempo de establecimiento se redujeron.



**Figura 6.20.** Comparación entre controlador de doble lazo ajustado y sin ajustar.

## 6.2. TOPOLOGÍA DE CONTROL DE TRES ENTRADAS

La topología de control de tres entradas consta de un controlador difuso con tres entradas, el error de la posición de la esfera, la derivada del error de posición de la esfera, el ángulo de la barra y una salida que es la tensión del motor como se puede observar en la siguiente figura.

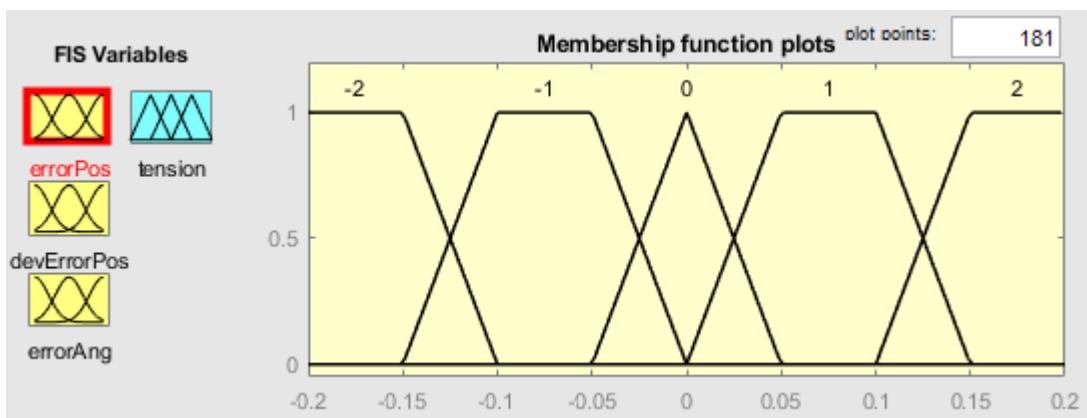


**Figura 6.21.** Topología de control de tres entradas.

### 6.2.1. DISEÑO DE CONTROLADOR DIFUSO DE TRES ENTRADAS

En el diseño del controlador difuso de tres entradas se consideraron como entradas: el error de posición, la derivada del error de posición y el error del ángulo de la barra, donde la consigna de esta última entrada siempre será cero. Respecto a la salida del controlador es la tensión que se aplica al motor para mover la barra.

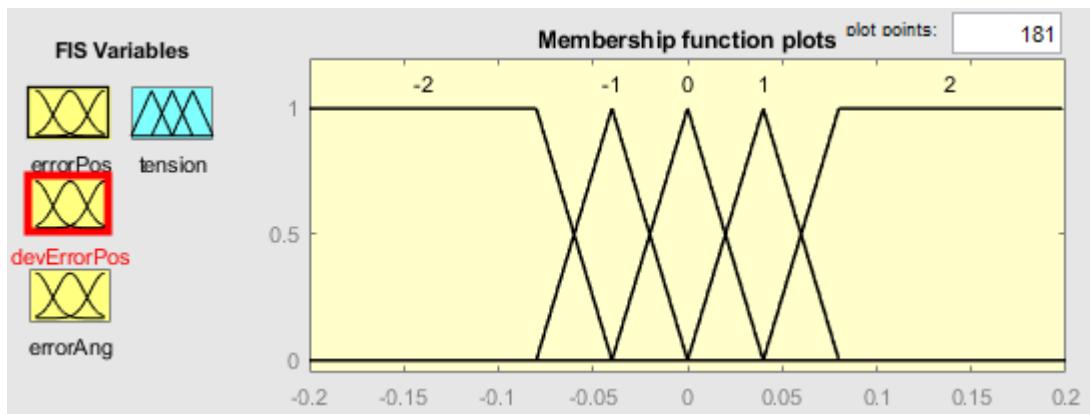
Para la entrada del error de posición los términos lingüísticos son: “muy derecha”, “derecha”, “cero”, “izquierda” y “muy izquierda”, los cuales representan donde se encuentra la esfera respecto a la consigna. Por otro lado, las entradas de la derivada del error de posición y el error del ángulo usan los mismos términos lingüísticos del controlador de la posición de la esfera y el controlador difuso del ángulo de la barra respectivamente. La salida del controlador usa los términos lingüísticos: “anti-horario rápido”, “anti-horario lento”, “cero”, “horario lento” y “horario rápido”. A continuación se presentan las funciones de pertenencia representadas con la función única y la base de reglas de forma matricial, donde se realizaron 5 matrices, una para cada término lingüístico del error de la posición.



**Figura 6.22.** Funciones de pertenencia del controlador difuso de tres entradas para la entrada del error de posición.

**Tabla 6.14.** Parámetros de las funciones de pertenencia del controlador difuso de tres entradas para la entrada de error de posición.

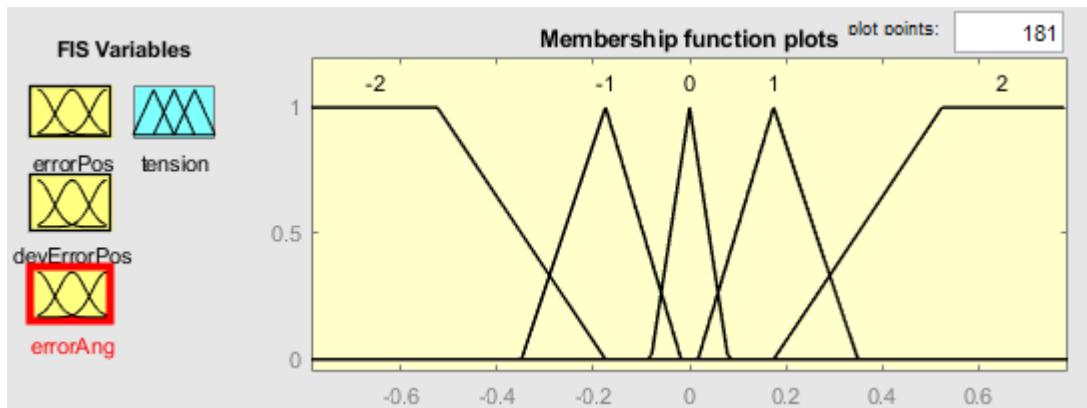
Valor Numérico Lingüístico	Término Lingüístico	Parámetros [a, b, c, d, nT, n1, n2]
-2	Muy Derecha	[-0.2, -0.2, -0.15, -0.1, 0, 1, 1]
-1	Derecha	[-0.15, -0.1, -0.05, 0, 0, 1, 1]
0	Cero	[-0.05, 0, 0, 0.05, 0, 1, 1]
1	Izquierda	[0, 0.05, 0.1, 0.15, 0, 1, 1]
2	Muy Izquierda	[0.1, 0.15, 0.2, 0.2, 0, 1, 1]



**Figura 6.23.** Funciones de pertenencia del controlador difuso de tres entradas para la entrada de la derivada del error de posición.

**Tabla 6.15.** Parámetros de las funciones de pertenencia del controlador difuso de tres entradas para la entrada de la derivada del error de posición.

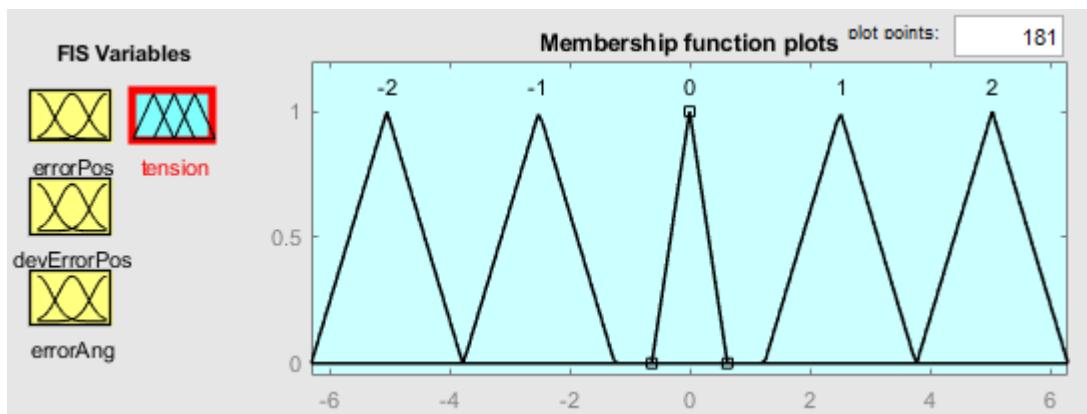
Valor Numérico Lingüístico	Término Lingüístico	Parámetros [a, b, c, d, nT, n1, n2]
-2	Positivo Rápido	[-0.2, -0.2, -0.08, -0.04, 0, 1, 1]
-1	Positivo Lento	[-0.08, -0.04, -0.04, 0, 0, 1, 1]
0	Cero	[-0.04, 0, 0, 0.04, 0, 1, 1]
1	Negativo Lento	[0, 0.04, 0.04, 0.08, 0, 1, 1]
2	Negativo Rápido	[0.04, 0.12, 0.2, 0.2, 0, 1, 1]



**Figura 6.24.** Funciones de pertenencia para el controlador difuso de tres entradas para la entrada del error del ángulo de la barra.

**Tabla 6.16.** Parámetros de las funciones de pertenencia del controlador difuso de tres entradas para la entrada de error del angulo.

Valor Numérico Lingüístico	Término Lingüístico	Parámetros [a, b, c, d, nT, n1, n2]
-2	Muy abajo	[-0.785, -0.785, -0.5236, -0.1745, 0, 1, 1]
-1	Abajo	[-0.349, -0.174, -0.174, -0.01745, 0, 1, 1]
0	Cero	[-0.08, 0, 0, 0.08, 0, 1, 1]
1	Arriba	[0.01745, 0.174, 0.174, 0.349, 0, 1, 1]
2	Muy arriba	[0.1745, 0.5236, 0.785, 0.785, 0, 1, 1]



**Figura 6.25.** Funciones de pertenencia de la salida del controlador difuso de tres entradas.

**Tabla 6.17.** Parámetros de las funciones de pertenencia de la salida del controlador difuso de tres entradas.

Valor Numérico Lingüístico	Término Lingüístico	Parámetros [a, b, c, d]
-2	Antihorario Rápido	[-6.2, -4.96, -4.96, -3.72]
-1	Antihorario Lento	[-3.707, -2.467, -2.467, -1.227]
0	Cero	[-0.62, 0, 0, 0.62]
1	Horario Lento	[ 1.227, 2.467, 2.467, 3.707]
2	Horario Rápido	[ 3.72, 4.96, 4.96, 6.2]

**Tabla 6.18.** Base de reglas para el controlador difuso de tres entradas, error de posición -2.

Tensión		Error del Ángulo				
		-2	-1	0	1	2
Derivada del error de posición	-2	-2	-2	-2	-1	0
	-1	-2	-2	-2	-1	0
	0	-2	-2	-2	-1	0
	1	-2	-2	-1	-1	0
	2	-2	-1	-1	0	0

**Tabla 6.19.** Base de reglas para el controlador difuso de tres entradas, error de posición -1.

Tensión		Error del Ángulo				
		-2	-1	0	1	2
Derivada del error de posición	-2	-2	-2	-1	-1	0
	-1	-2	-1	-1	0	1
	0	-2	-1	-1	0	1
	1	-2	-1	0	0	1
	2	-2	-1	0	1	2

**Tabla 6.20.** Base de reglas para el controlador difuso de tres entradas, error de posición 0.

Tensión		Error del Ángulo				
		-2	-1	0	1	2
Derivada del error de posición	-2	-2	-2	-1	1	2
	-1	-2	-1	-1	1	2
	0	-2	-2	0	2	2
	1	-2	-1	1	1	2
	2	-2	-1	1	2	2

**Tabla 6.21.** Base de reglas para el controlador difuso de tres entradas, error de posición 1.

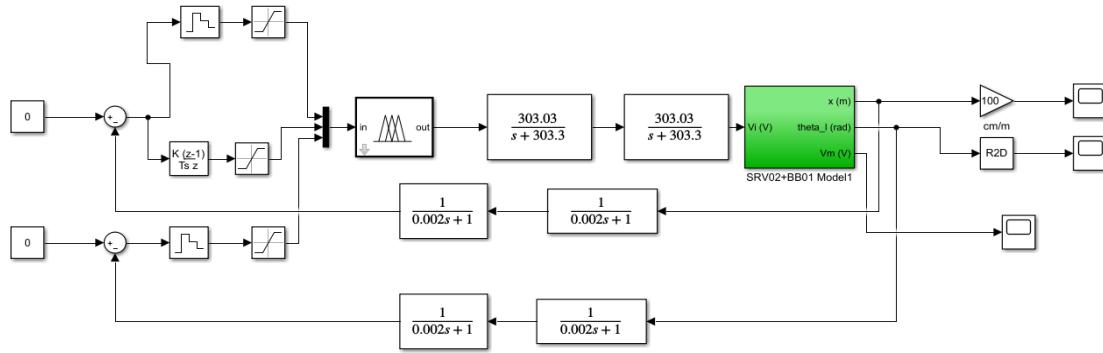
Tensión		Error del Ángulo				
		-2	-1	0	1	2
Derivada del error de posición	-2	-2	-1	0	1	2
	-1	-1	0	0	1	2
	0	-1	0	1	1	2
	1	-1	0	1	1	2
	2	0	1	1	2	2

**Tabla 6.22.** Base de reglas para el controlador difuso de tres entradas, error de posición 2.

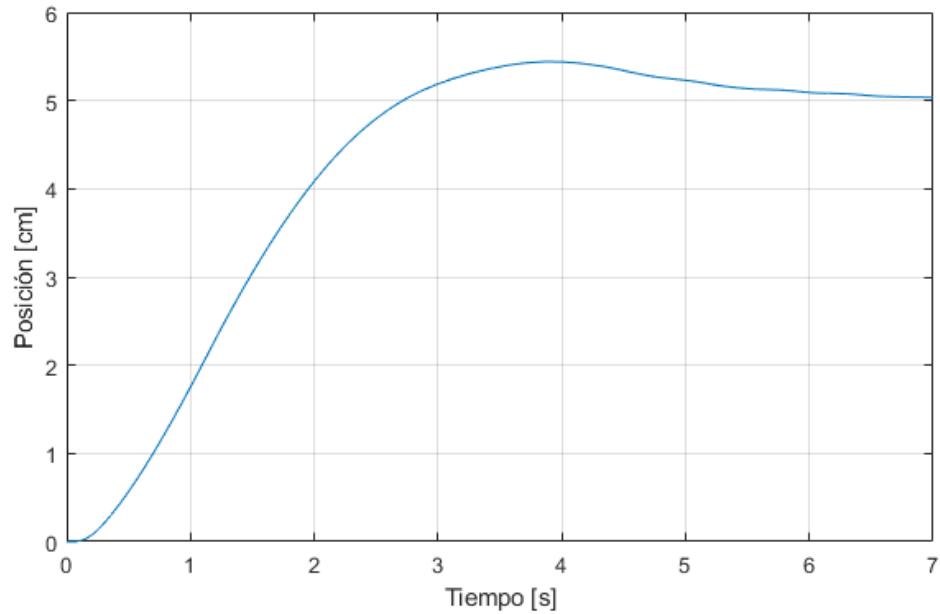
Tensión		Error del Ángulo				
		-2	-1	0	1	2
Derivada del error de posición	-2	0	0	1	1	2
	-1	0	1	1	2	2
	0	0	1	2	2	2
	1	0	1	2	2	2
	2	0	1	2	2	2

### 6.2.2. SIMULACIÓN DE CONTROLADOR DIFUSO DE TRES ENTRADAS

La simulación se hizo de acuerdo al diagrama de bloques de la figura 6.26, donde las condiciones iniciales fueron  $x(0) = 0$ ,  $\dot{x}(0) = 0$ ,  $\theta(0) = 0$  y  $\dot{\theta}(0) = 0$ . Asimismo, las consignas para la posición y el ángulo de la barra fueron de 5 cm y  $0^\circ$  y el tiempo de muestreo fue de 12 ms. Los resultados de esta simulación se pueden apreciar en la figura 6.2.2, así como el desempeño del sistema obtenido en la tabla 6.26.



**Figura 6.26.** Diagrama de bloques para la simulación con controlador difuso de tres entradas.



**Figura 6.27.** Simulación de posición con controlador difuso de tres entradas, Set point: 5cm, tiempo de muestreo: 12ms y condiciones iniciales nulas.

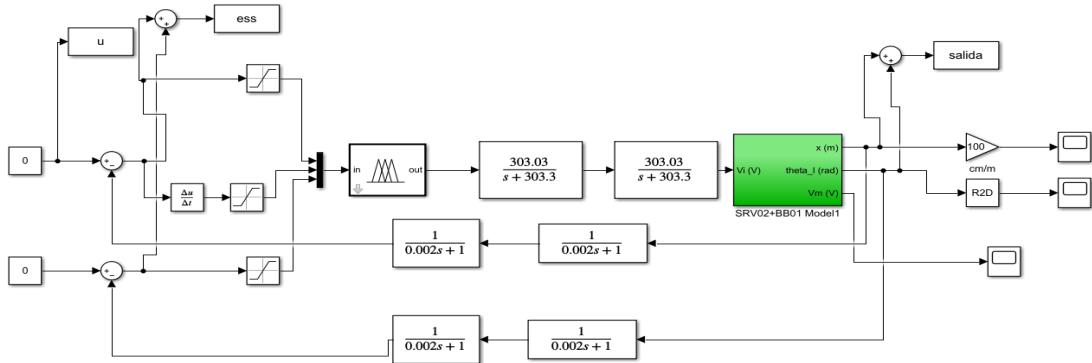
**Tabla 6.23.** Desempeño del sistema con el controlador difuso de tres entradas.

Tiempo de establecimiento [s]	5.9297
Sobre Pico [%]	8.8051
Tiempo de alza [s]	1.7969

### 6.2.3. AJUSTE DE CONTROLADOR DIFUSO DE TRES ENTRADAS

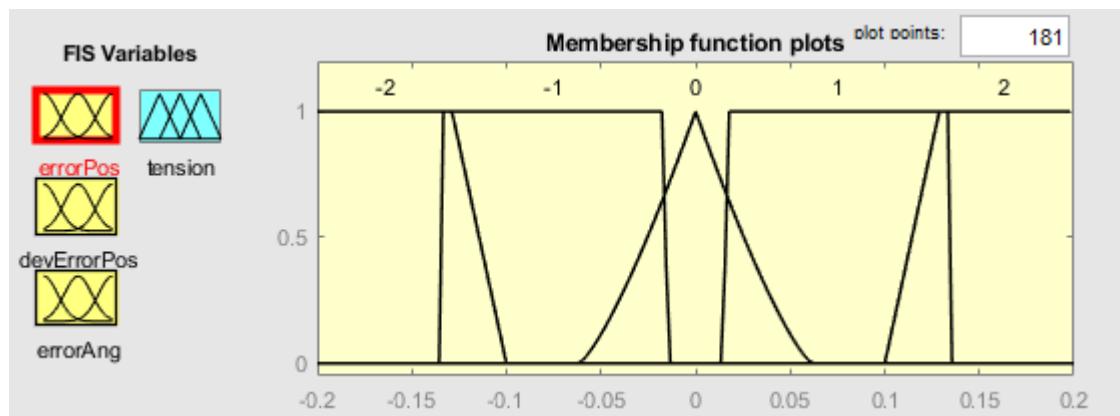
El ajuste del controlador difuso de tres entradas se hizo mediante el diagrama de bloques de la figura 6.28, siguiendo el mismo procedimiento de ajuste del controlador difuso de la posición de la esfera, con la diferencia que el peso del sobre pico en la función objetivo fue aumentado en un 5 %, como se muestra en la siguiente ecuación:

$$f = 0,5 \sum_{i=1}^n e^2(n) + 0,2|E_{ss} - 0,01| + 0,2|s_{max} - r - 0,1| + 0,05 \frac{t_s(E_{ss} - 0,01)}{T} + 0,05 \frac{t_a}{T} \quad (6.5)$$



**Figura 6.28.** Diagrama de bloques para la simulación con controlador difuso de tres entradas ajustado.

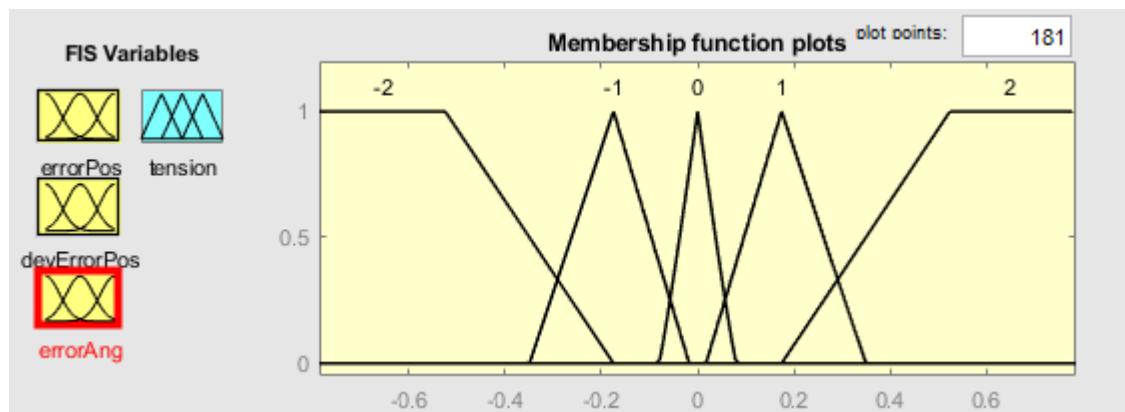
Una vez ajustadas las funciones de pertenencia de las entradas del error de posición y el error del ángulo éstas cambiaron como se muestra a continuación:



**Figura 6.29.** Funciones de pertenencia ajustadas para la entrada del error de posición.

**Tabla 6.24.** Parámetros de las funciones de pertenencia del controlador difuso de tres entradas ajustado para la entrada de error de posición.

Valor Numérico Lingüístico	Término Lingüístico	Parámetros [a, b, c, d, nT, n1, n2]
-2	Muy Derecha	[-0.2, -0.2, -0.136, -0.1, 0.2, 1, 1]
-1	Derecha	[-0.135, -0.1334, -0.0166, -0.01495, 0.2, 1.1, 1]
0	Cero	[-0.06173, 0, 0, 0.06173, 0, 1, 1.3]
1	Izquierda	[0.01495, 0.0166, 0.1334, 0.135, 0.2, 1.1, 1]
2	Muy Izquierda	[0.1, 0.136, 0.2, 0.2, 0.2, 1, 1]



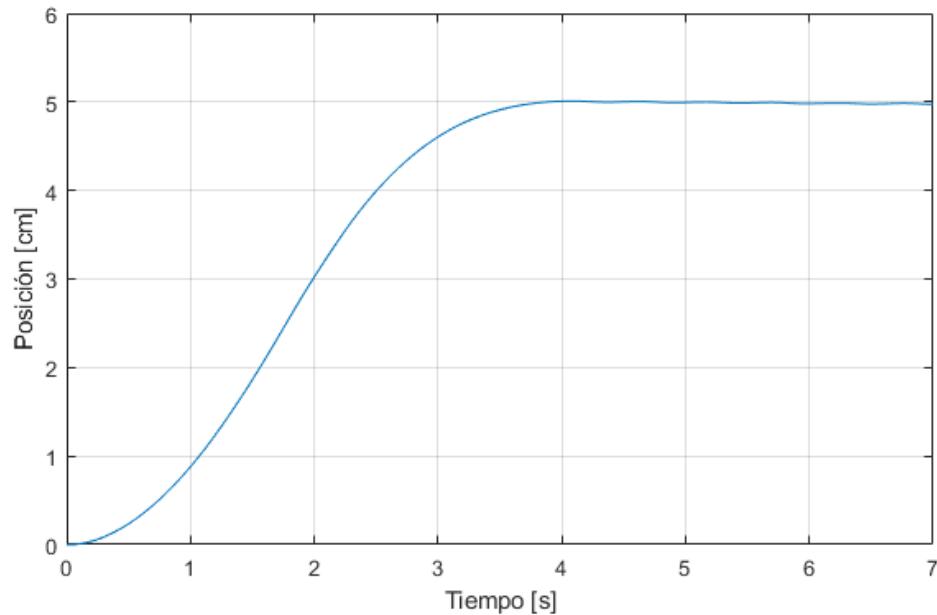
**Figura 6.30.** Funciones de pertenencia ajustadas para la entrada del error de ángulo.

**Tabla 6.25.** Parámetros de las funciones de pertenencia del controlador difuso de tres entradas ajustado para la entrada de error del angulo.

Valor Numérico Lingüístico	Término Lingüístico	Parámetros [a, b, c, d, nT, n1, n2]
-2	Muy abajo	[-0.785, -0.785, -0.5236, -0.1745, 0, 1, 1]
-1	Abajo	[-0.349, -0.174, -0.174, -0.01745, 0, 1, 1]
0	Cero	[-0.08, 0, 0, 0.08, 0, 1, 1.1]
1	Arriba	[0.01745, 0.174, 0.174, 0.349, 0, 1, 1]
2	Muy arriba	[0.1745, 0.5236, 0.785, 0.785, 0, 1, 1]

#### 6.2.4. SIMULACIÓN DE CONTROLADOR DIFUSO AJUSTADO DE TRES ENTRADAS

Cambiando las funciones de pertenencia iniciales por las obtenidas en el proceso de ajuste, se obtuvo el resultado que se muestra en la figura 6.31 y un desempeño del sistema que se observa en la tabla 6.26.



**Figura 6.31.** Simulación de posición con controlador difuso de tres entradas ajustado, Set point: 5cm.

**Tabla 6.26.** Desempeño del sistema con el controlador difuso de tres entradas ajustado.

Tiempo de establecimiento [s]	3.4723
Sobre Pico [%]	0.1820
Tiempo de alza [s]	2.1497

### **6.3. CONTROLADOR CONVENCIONAL DE DOBLE LAZO**

Para el diseño del controlador convencional, se partió igualmente de la propuesta de Quanser de usar doble lazo, por ende primero se diseño un control proporcional para el ángulo de la barra.

#### **6.3.1. DISEÑO DE CONTROLADOR PROPORCIONAL PARA EL ÁNGULO DE LA BARRA**

En esta etapa se realizó un diseño con una respuesta sin sobre pico, por ende se buscó una respuesta donde los polos se encuentren sobre el eje real del lado izquierdo del plano  $s$  con un control proporcional.

Para hallar un tiempo de establecimiento menor a 0.26s usando el criterio del 2%, el cual indica que la constante de tiempo  $\tau$  del polo dominante del sistema debe ser:

$$4\tau < 0,26s \quad (6.6)$$

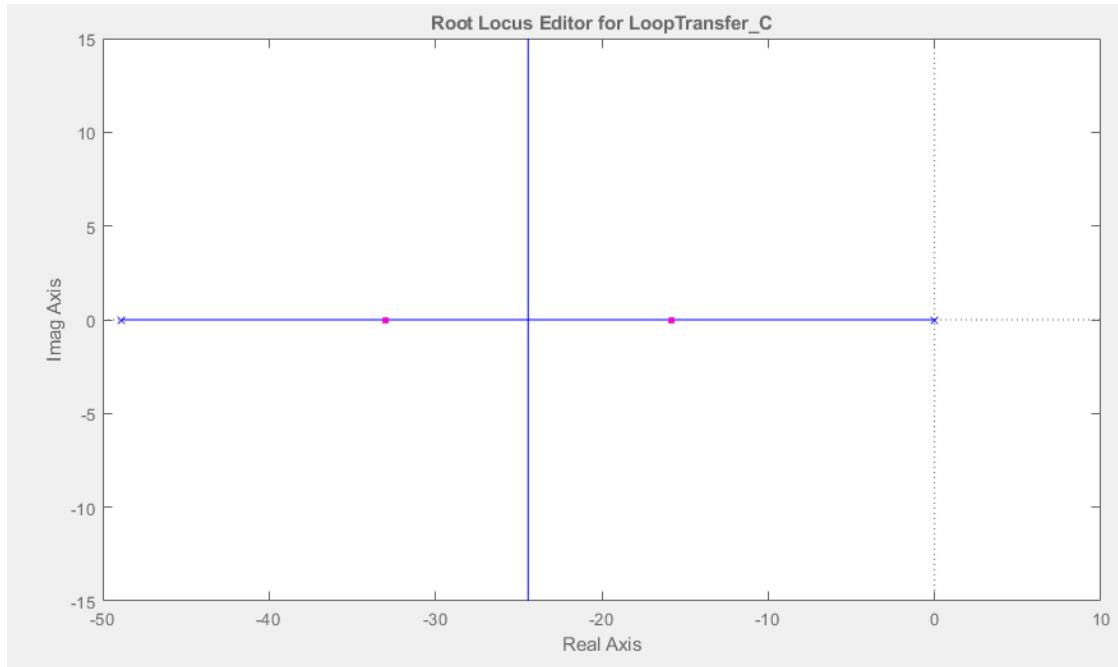
Despejando  $\tau$

$$\tau < 0,065s \quad (6.7)$$

Lo cual indica que el polo  $p_1$  (suponiendo que es el dominante), debe estar ubicado en:

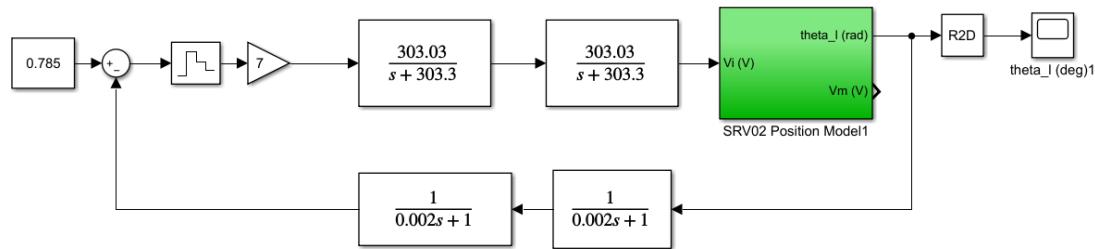
$$p_1 < -15,38 \quad (6.8)$$

Trazando el lugar geométrico de las raíces con MATLAB como se muestra en la figura 6.32, donde se ubicó un polo del sistema en -15.38 con una constante proporcional  $K = 6,8724$ .

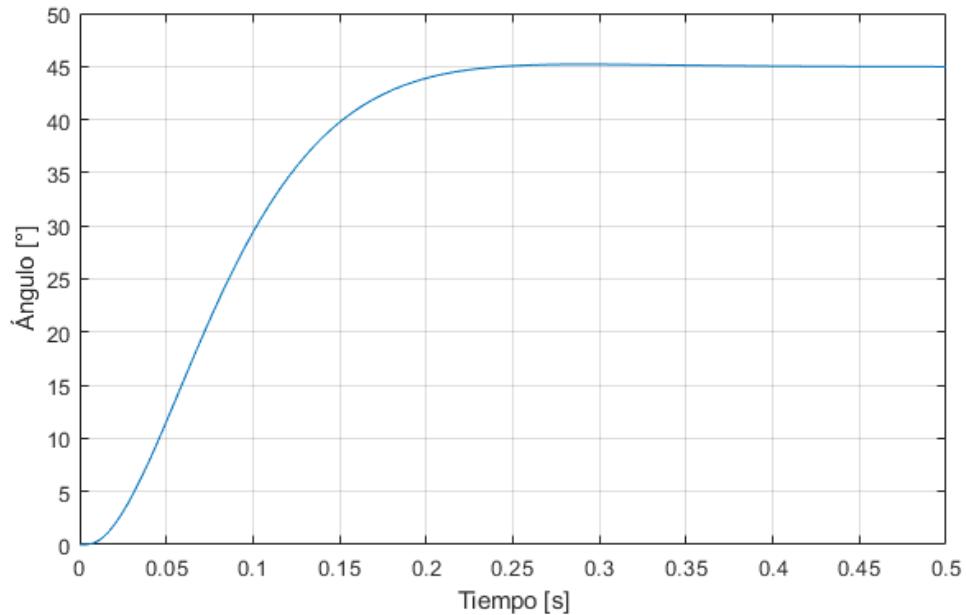


**Figura 6.32.** Lugar Geométrico de las Raíces con control proporcional.

Por medio de simulación usando el diagrama de bloques de la figura 6.33, se discretizó la ecuación del controlador y se observó el desempeño del sistema. Sin embargo, se tuvo que ajustar la constante proporcional  $K$  del controlador, dado que en este caso no existe un polo dominante, ya que para una ganancia  $K$  que permita tener un polo en -15,38, los polos del sistema se encuentran lo suficientemente cerca para que ninguno sea dominante, por lo tanto se ajustó el controlador con LGR hasta obtener una respuesta del sistema que se observa en la figura 6.34 y un desempeño del sistema de la tabla 6.27 con una constante proporcional  $K = 7$ .



**Figura 6.33.** Diagrama de bloques para la simulación con controlador proporcional para el ángulo de la barra.



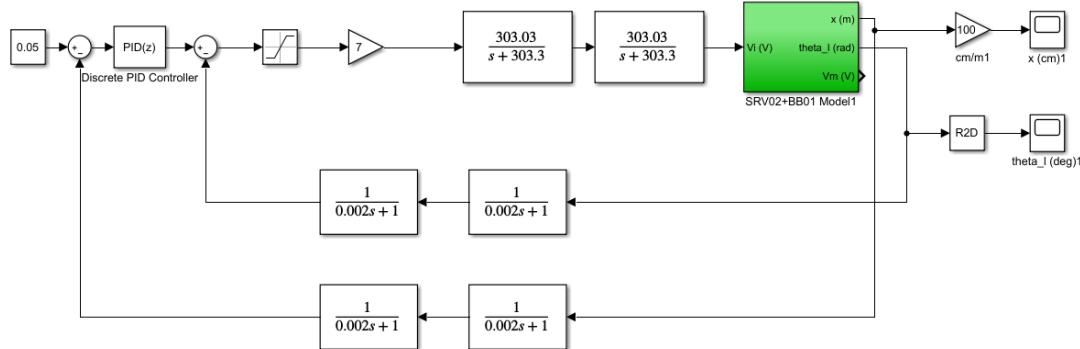
**Figura 6.34.** Simulación de ángulo de la barra con controlador proporcional, Set point: 45°, tiempo de muestreo: 12ms y condiciones iniciales nulas.

**Tabla 6.27.** Desempeño del sistema con el controlador proporcional.

Tiempo de establecimiento [s]	0.2344
Sobre Pico [%]	0
Tiempo de alza [s]	0.1370

### 6.3.2. DISEÑO DE CONTROLADOR PD PARA LA POSICIÓN DE LA ESFERA

Partiendo del diseño de Reza Moezzi, Vu Trieu Minh y Mart Tamre de un controlador PID en su trabajo de investigación “Fuzzy Logic Control for a Ball and Beam System i” (Moezzi, Tamre, y Minh, s.f.) donde las constantes que usaron fueron  $K_p = 5$ ,  $K_d = 15$  y  $K_i = 0,1$ , constantes con las cuales se realizaron simulaciones usando el diagrama de bloques de la figura 6.35 y no se obtuvieron resultados cercanos a los controladores difusos, por lo que se ajustaron por simulación con valores alrededor de estos y se obtuvo un mejor desempeño con  $K_p = 12$ ,  $K_d = 7$  y  $K_i = 0$ .



**Figura 6.35.** Diagrama de bloques para simulación de controlador PD para posición.

La función de transferencia de este controlador viene dada por:

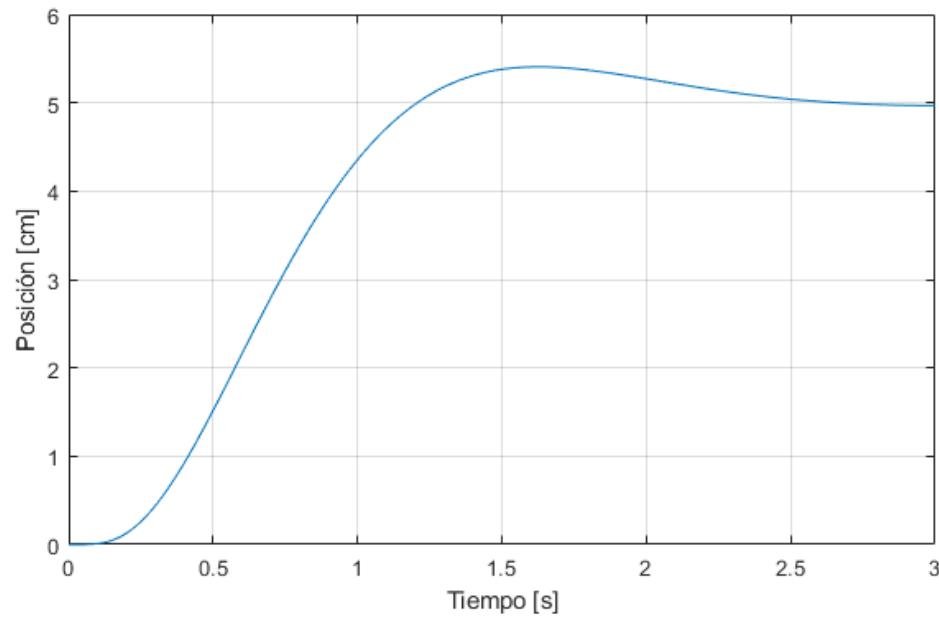
$$\frac{\theta_{l_d}(s)}{E_x(s)} = K_p + K_d s \quad (6.9)$$

Donde la variable  $E_x(s)$  representa el error de posición y  $\theta_{l_d}$  el ángulo de la barra deseado.

Aplicando la transformada inversa de Laplace, discretizando la ecuación resultante y aplicando la transformada Z con una frecuencia de muestreo de 12ms y usando la aproximación forward de Euler se obtiene:

$$\frac{\theta_l(z)}{E_x(z)} = K_p + K_d \frac{(1 - z^{-1})}{T_s} \quad (6.10)$$

Esta ecuación se implementa en el diagrama de bloques de la figura 6.35, donde el resultado se muestra en la figura 6.36 y el desempeño del sistema en la tabla 6.28.



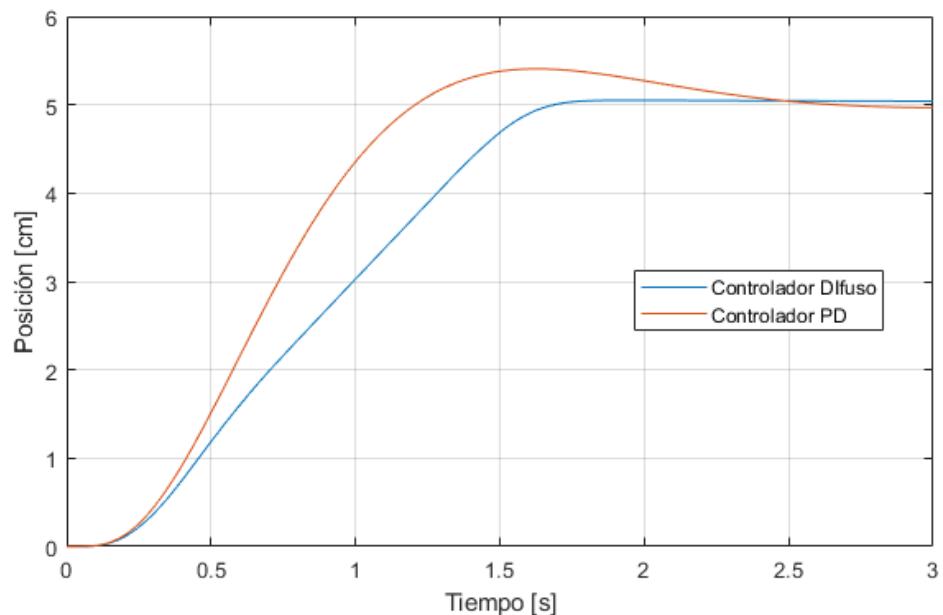
**Figura 6.36.** Simulación de posición con controlador PD, Set point: 5cm, tiempo de muestreo: 12ms,  $x(0) = 0$ ,  $\dot{x}(0) = 0$ ,  $\theta_l(0) = 0$  y  $\dot{\theta}_l(0) = 0$ .

**Tabla 6.28.** Desempeño del sistema con el controlador PD.

Tiempo de establecimiento [s]	2.3404
Sobre Pico [%]	12.0648
Tiempo de alza [s]	0.6812

#### 6.4. COMPARACIÓN POR SIMULACIÓN DE CONTROLADOR DIFUSO AJUSTADO CON CONTROLADOR PD

A continuación se presentan en la figura 6.37 una gráfica de la salida del sistema para el controlador difuso ajustado y el controlador PD ambos para el control de la posición de la esfera. Asimismo, en la tabla 6.29 se muestra una comparación del desempeño del sistema obtenido para ambos controladores.



**Figura 6.37.** Respuesta del sistema para controlador difuso ajustado y control PD de posición.

**Tabla 6.29.** Desempeño del sistema para ambos controladores.

Controlador	Convencional	Difuso	Diferencia [%]
Tiempo de establecimiento [s]	2.3404	1.6107	31.12
Sobre Pico [%]	12.0648	0.9372	92.25
Tiempo de alza [s]	0.6812	1.1030	38.24

## CAPÍTULO VII

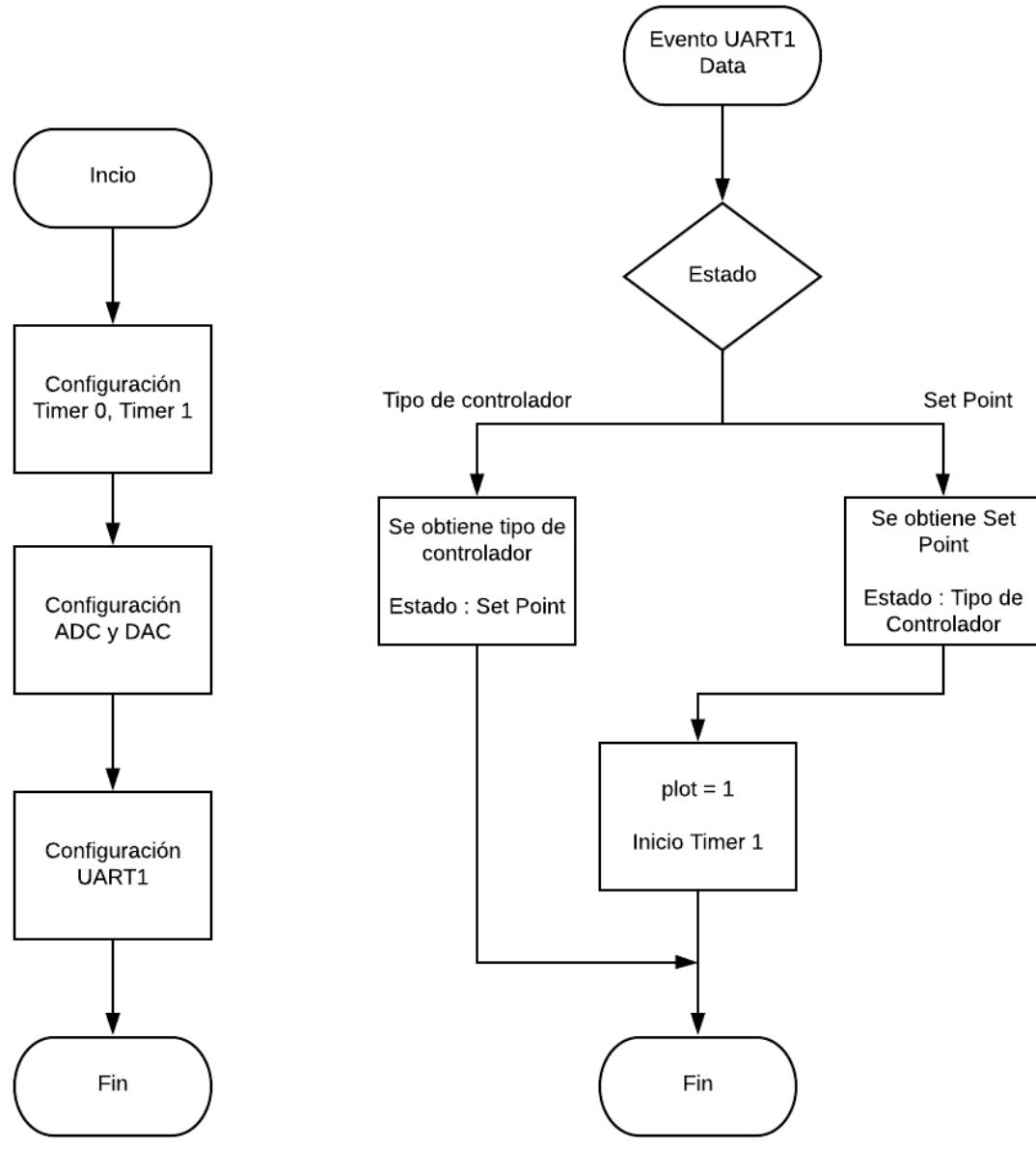
### IMPLEMENTACIÓN DE CONTROLADORES EN EL SISTEMA BALL AND BEAM DE QUANSER

#### 7.1. ALGORITMO DE CONTROL

Para implementar los controladores se realizó un programa que contiene tres controladores: el controlador difuso sin ajustar, el controlador difuso ajustado y el controlador PD los tres para el control de la posición de la esfera. Siguiendo el diagrama de flujo de la figura 7.1a, primero se inicializa todas las configuraciones generales del microcontrolador ADC, DAC, UART1 y Timers.

Se configura ADC a 12 bits, el DAC a 8 bits, el UART1 a una velocidad de 115200 baudios y se activan los eventos del UART1 para poder entrar en un evento cada vez que se reciban datos, se configuran dos interrupciones por timer, una interrupción en el timer 1 para la adquisición de datos para la gráfica de la implementación configurada a 7 segundos y el timer 0 configurado a 12ms para indicar cuando se debe muestrear y realizar la acción de control.

Las gráficas de la implementación se hicieron con MATLAB®, donde se creó un script con la función de enviar dos bytes al UART1 del ESP32, donde el primer byte indica que tipo de controlador se desea ejecutar (controlador difuso sin ajustar:0, controlador difuso ajustado:1 y control PD:2) y el segundo byte indica el Set Point, un diagrama de flujo que representa como es procesado esta información en el ESP32 se muestra en el diagrama de flujo de la figura 7.1b.

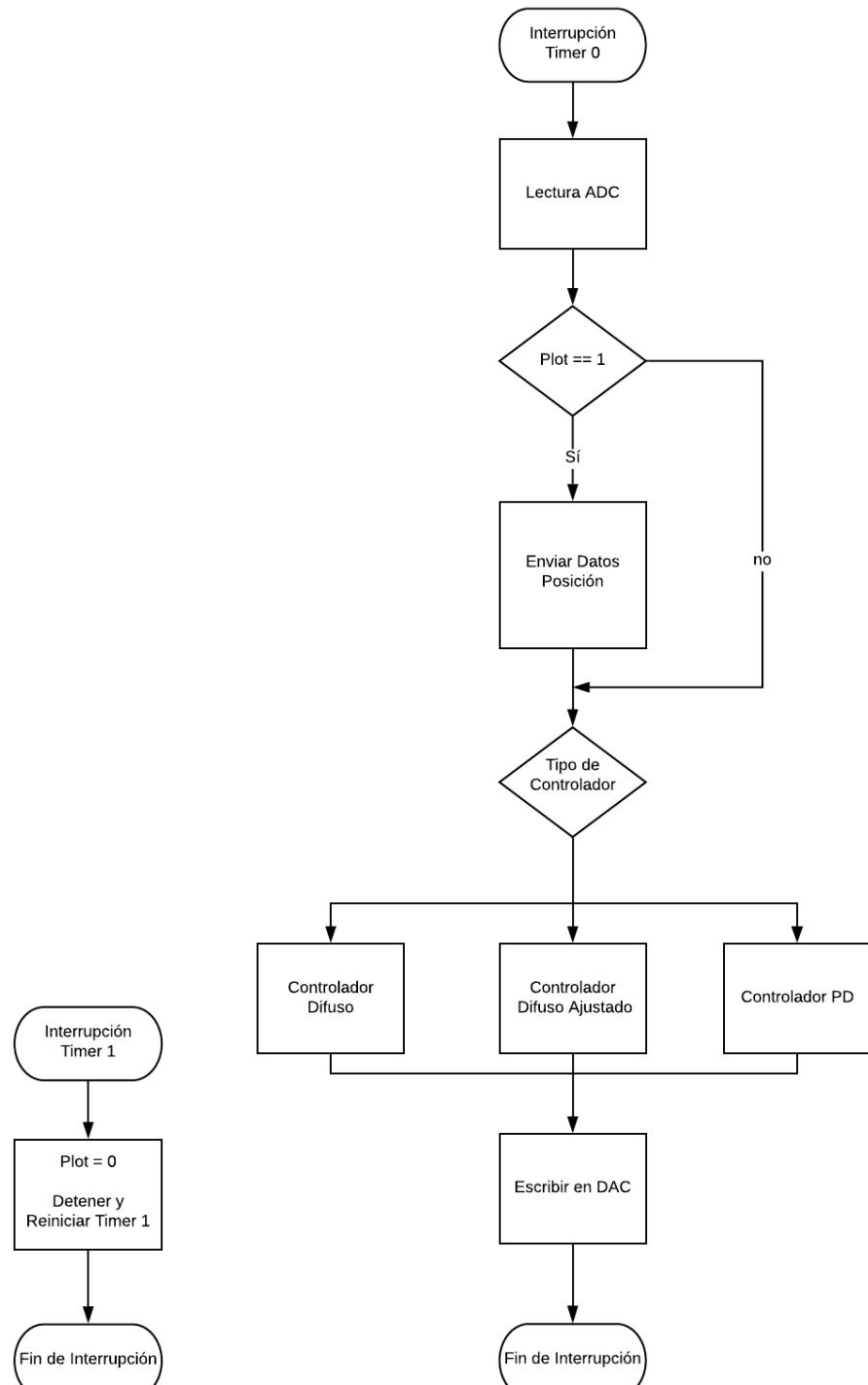


**Figura 7.1.** Evento de datos UART1.

El diagrama de flujo de la figura **7.1b** básicamente es una máquina de estados. Cuando el script de MATLAB® es ejecutado se envía el primer byte correspondiente al tipo de controlador, esta máquina de estados procesa byte a byte por lo tanto al llegar el primer byte se configura tipo del controlador, cambia de estado a Set Point, lo que indica que ya se espera que el siguiente byte indique la nueva consigna, al recibir este valor el activa una variable booleana llamada plot, activa el timer 1, modifica el Set Point y cambia nuevamente al estado tipo de controlador.

Luego, al ser activado la variable plot, en el diagrama de flujo de la figura **7.2b**, el cual corresponde con la interrupción del timer 0, se envía a través del UART1 los datos del sensor de posición para que MATLAB® al cabo de 7 segundos lo que equivale a recibir 583 datos, recordando que el tiempo de muestreo es 12ms, grafique los valores obtenidos y muestre por consola el desempeño del sistema.

Por último, la variable plot vuelve a ser 0, cuando la interrupción por el timer 1 es activada luego de haber pasado los 7 segundos desde que se activo, como se muestra en el diagrama de flujo de la figura **7.2a**.



(a) Interrupción Timer 1

(b) Interrupción Timer 0

**Figura 7.2.** Interrupciones de Timers.

## 7.2. FILTRO DIGITAL A LA ENTRADA DE LOS CONTROLADORES

De acuerdo al manual práctico de Quanser, el sensor de posición presenta un ruido inherente y puede generar un mal funcionamiento del controlador (Apkarian y cols., 2011b), por lo general en este tipo de situaciones se aplica un filtro pasa bajo para limitar el factor que multiplica al ruido de alta frecuencia (Anström y Hägglund, 2009), para ello Quanser sugiere agregar un filtro pasa bajo a la acción derivativa de  $5Hz$ , por lo tanto en el caso del controlador PD la ecuación **6.9** cambia a:

$$\frac{\theta_{l_d}(s)}{E_x(s)} = K_p + K_d \frac{31,4 s}{s + 31,4} \quad (7.1)$$

Aplicando la transformada inversa de Laplace, discretizando la ecuación resultante, aplicando la transformada Z con una frecuencia de muestreo de 12ms y agregando la ecuación del retenedor de orden cero se obtiene

$$\frac{\theta_l(z)}{E_x(z)} = K_p + K_d \frac{31,4 (1 - z^{-1})}{1 - 0,6805 z^{-1}} \quad (7.2)$$

Por otro lado, para los controladores difusos diseñados este ruido en el sensor de posición también afecta en el funcionamiento del controlador, para ello la ecuación **2.25** vista en el capítulo 2, debe ser modificada colocando el filtro pasa bajo que sugiere Quanser resultando:

$$\frac{\text{Error dev pos}(z)}{\text{Error pos}(z)} = \frac{31,4 (1 - z^{-1})}{1 - 0,6805 z^{-1}} \quad (7.3)$$

Donde  $Error dev pos$  y  $Error pos$  representan la derivada del error de posición y el error de posición respectivamente. Ahora, despejando y aplicando la transformada inversa Z, se obtiene:

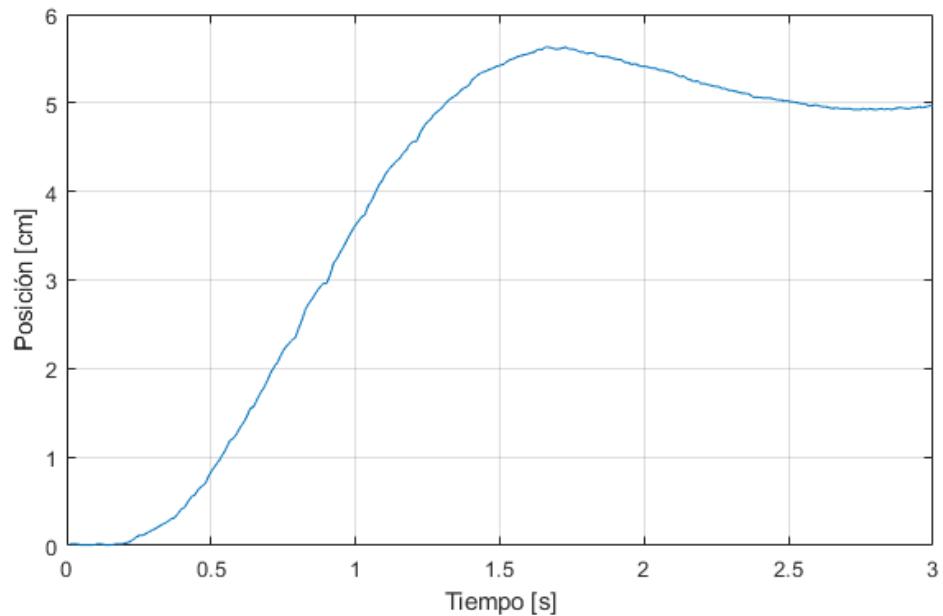
$$Error dev pos(k) = 31,4 (Error pos(k) - Error pos(k-1)) + 0,6805 Error dev pos(k-1) \quad (7.4)$$

Programando las ecuaciones **7.2** y **7.4** en los controladores, el ruido que presenta el sensor de posición bajo considerablemente.

Debido a imperfecciones mecánicas que posee el sistema (Apkarian y cols., 2011a) y a la zona muerta del motor que va de [-0,7 - 0,2] V (valores obtenidos experimentalmente), al inicio de implementar los controladores se obtenía un error en estado estacionario de 1cm para el control PD y de 0.5cm con los controladores difusos. Para solventar este error en estado estacionario Quanser sugiere para el control del ángulo sumar un ángulo de offset a la consigna, esto con la finalidad de sacar la salida del controlador de la zona muerta del motor y cubrir el error que puede venir por las imperfecciones mecánicas del sistema. Este valor se obtuvo por ensayo y error y varió para ambos controladores, el controlador convencional el ángulo de offset fue de 10°, pero en el controlador difuso este ángulo fue de 1°. Sin embargo, a pesar de corregir este error por software el error de posición de la esfera está comprendido entre ±0,1cm.

### 7.3. RESULTADOS DEL CONTROL PD PARA LA POSICIÓN DE LA ESFERA

Al implementar el controlador PD en el sistema Ball and Beam real se obtuvo la gráfica de la figura **7.3** y el desempeño del sistema de la tabla **7.1**.



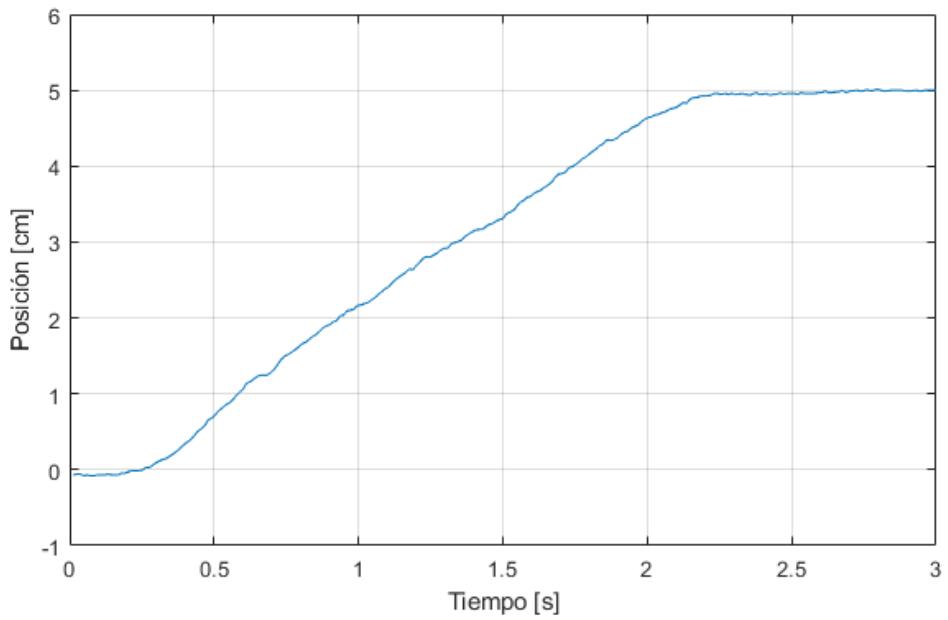
**Figura 7.3.** Respuesta del sistema real para controlador PD.

**Tabla 7.1.** Desempeño del sistema real con el controlador PD.

Tiempo de establecimiento [s]	2.3618
Sobre Pico [%]	12.5931
Tiempo de alza [s]	0.7576

#### 7.4. RESULTADOS DEL CONTROLADOR DIFUSO SIN AJUSTAR DE LA POSICIÓN DE LA ESFERA

Al implementar el controlador difuso de la posición de la esfera sin ajustar en el sistema Ball and Beam real se obtuvo la gráfica de la figura 7.4 y el desempeño del sistema de la tabla 7.2.



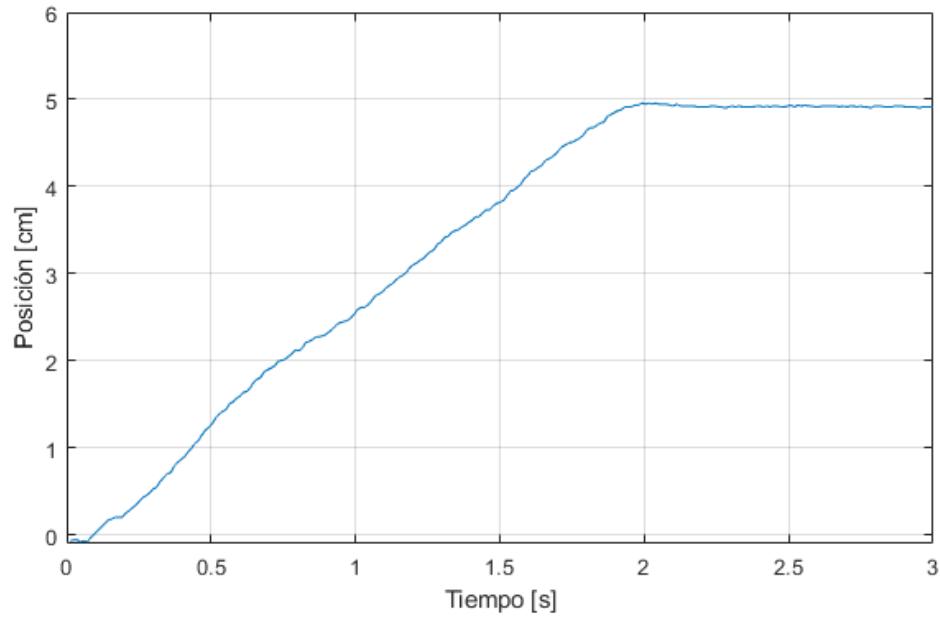
**Figura 7.4.** Respuesta del sistema real para controlador difuso sin ajustar.

**Tabla 7.2.** Desempeño del sistema real con el controlador difuso sin ajustar.

Tiempo de establecimiento [s]	2.1578
Sobre Pico [%]	0.4373
Tiempo de alza [s]	1.5130

## 7.5. RESULTADOS DEL CONTROLADOR DIFUSO AJUSTADO DE LA POSICIÓN DE LA ESFERA

Al implementar el controlador difuso de doble lazo ajustado en el sistema Ball and Beam real se obtuvo la gráfica de la figura 7.5 y el desempeño del sistema de la tabla 7.3.



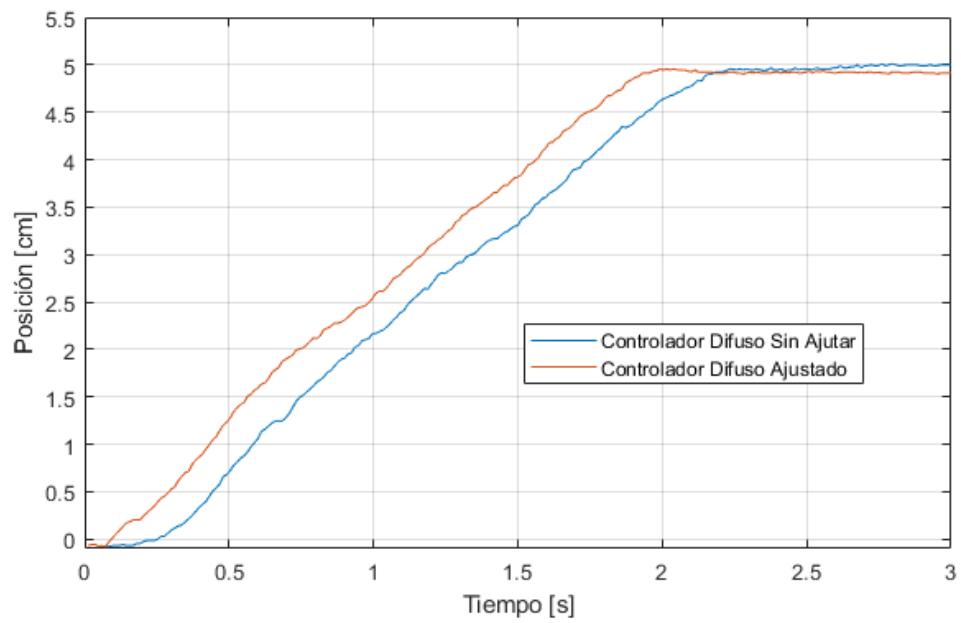
**Figura 7.5.** Respuesta del sistema real para controlador difuso ajustado.

**Tabla 7.3.** Desempeño del sistema real con el controlador difuso ajustado.

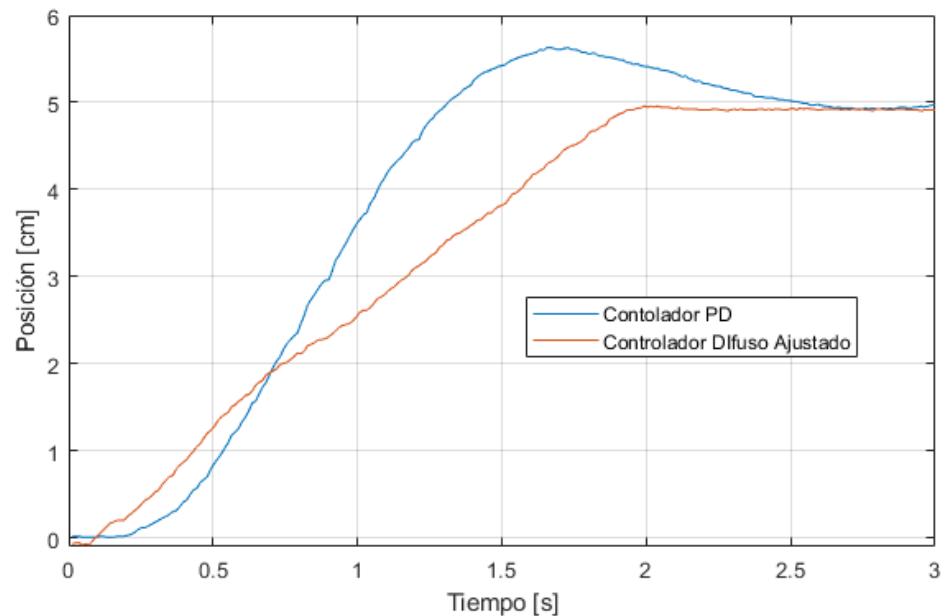
Tiempo de establecimiento [s]	1.9159
Sobre Pico [%]	0
Tiempo de alza [s]	1.4304

## 7.6. COMPARACIÓN ENTRE LOS CONTROLADORES

Luego de implementar los tres controladores, se realizaron dos gráficas, una que muestra el controlador difuso de la posición de la esfera ajustado y sin ajustar (ver figura 7.6) y otra que muestra el controlador difuso y el controlador PD para la posición de la esfera (ver figura 7.7). Asimismo, en la tabla 7.4 se muestra el desempeño del sistema para estos controladores.



**Figura 7.6.** Respuesta del sistema real para controlador difuso ajustado y controlador difuso sin ajustar.



**Figura 7.7.** Respuesta del sistema real para controlador difuso ajustado y controlador PD.

**Tabla 7.4.** Desempeño del sistema real para ambos controladores.

Controlador	Convencional	Difuso	Diferencia [%]
Tiempo de establecimiento [s]	2.3618	1.9159	18,87
Sobre Pico [%]	12.5931	0	-
Tiempo de alza [s]	0.7576	1.4304	47.05

La forma de la respuesta del sistema para los tres controladores se asemeja a la obtenida por simulación, por lo que los resultados son aceptables. Sin embargo, existen diferencias entre lo obtenido por simulación con lo obtenido en la implementación, esto puede venir dado por las imperfecciones de la máquina y el ruido inherente del sensor de posición que a pesar que se redujeron por software estos aun afectan, lo cual influye en el sistema.

Para finalizar, se evaluó entre los tres controladores y se obtuvo que el controlador difuso de la posición de la esfera ajustado se obtiene un mejor desempeño del sistema, ya que el controlador PD posee un mejor tiempo de alza por 47.05 % pero su sobre pico es mayor, al igual que su tiempo de establecimiento esta por debajo en un 18.87 %.

Los controladores implementados se pueden observar en la serie de videos que se encuentra en este [enlace](#) (Jiménez, 2020), donde se encuentra un video introductorio que describe el sistema de control, tres videos de pruebas ante perturbaciones y tres videos que muestran como se obtuvo el desempeño del sistema para cada controlador.

## CONCLUSIONES

Con los controladores implementados y el diseño de hardware para el control se cumplieron todos los objetivos propuestos en este trabajo de grado, donde se puede concluir que:

- Se realizó el análisis del sistema Ball and Beam dividiéndolo en etapas, describiendo la función de cada etapa, obteniendo las ecuaciones que describen al sistema para así conocer las variables de interés para realizar el control.
- Partiendo de los parámetros que rigen al sistema Ball and Beam, se seleccionó al microcontrolador ESP32 debido a su compatibilidad con las librerías de lógica difusa. Además, se diseñó el circuito acondicionador de señales para poder conectar los sensores y el servo motor al ESP32.
- Mediante el software MATLAB se diseñaron dos topologías de control: una de doble lazo y la segunda basada en un controlador difuso de tres entradas. Luego partiendo de estas topologías se diseñaron los controladores difusos y el controlador convencional, se comprobó su funcionamiento mediante simulación y se evaluó el desempeño del sistema para cada controlador.
- Las funciones de pertenencia de los controladores difusos fueron ajustadas mediante el método de optimización por búsqueda directa implementado en MATLAB, donde se evaluó nuevamente el desempeño del sistema obteniendo que con el controlador difuso para la posición de la esfera se obtiene un mejor desempeño del sistema.
- En la implementación se tuvo que realizar un ajuste con un filtro digital para obtener un mejor desempeño del sistema. Asimismo, se hizo una conexión

serial con el software MATLAB para poder obtener la posición de la esfera en tiempo real.

- Se evaluó el desempeño del sistema en el sistema real usando cada uno de los controladores, considerando el sobre pico porcentual, el tiempo de alza y el tiempo de establecimiento, donde se obtuvo un mejor desempeño con el controlador difuso para la posición de la esfera ajustado.
- El diseño de controladores difusos parte de cierto conocimiento del sistema, saber que rangos comprende las variables. A pesar de llegar a un buen resultado es de gran ayuda contar con un método de ajuste que va más allá de la experiencia y conocimiento del experto, con el método de patrón de búsqueda se obtienen buenos resultados y se evita gran cantidad de tiempo de ajuste, siempre y cuando se tenga una forma de simular el sistema.
- En el método de patrón de búsqueda se deben considerar las condiciones iniciales y el set point en el cual se desea obtener un buen desempeño, ya que si estos varian a la hora de realizar pruebas es posible no obtener los valores esperados.
- Indiferentemente de la técnica de control, el proceso de acondicionamiento es fundamental a la hora de implementar el controlador, donde se necesita cierta habilidad y comprensión del sistema, lo cual con lleva tiempo realizando pruebas del controlador. Los cálculos son fundamentales pero cuando se desea implementar el controlador, existen parámetros que no se consideran en el diseño, sea diseñado analíticamente (PID) o cualitativamente (control difuso), por ejemplo imperfecciones mecánicas, ruido en sensores, saturación, etc.

## REFERENCIAS

- Alvez, A. (2019). *Embedded Fuzzy Logic Library, Reource Group Robotic State University of Piauí Brasil*, visitado 15-05-2020. <https://github.com/zerokol/eFLL>.
- Anström, K., y Hägglund, T. (2009). Control PID Avanzado, Pearson Prentice Hall.
- Apkarian, J., Lévis, M., y Gurocak, H. (2011a). *Quanser Ball and Beam, User Manual*. Quanser Inc.
- Apkarian, J., Lévis, M., y Gurocak, H. (2011b). *SRV02 Ball and Beam User Manual*. Quanser Inc.
- Hernández. (2018). Diseño de un Controlador Digital para un Sistema Ball and Beam usando una tarjeta Controladora Intel Galileo, Escuela de Ingeniería Eléctrica, Universidad Central de Venezuela.
- Jiménez, B. (2020). *Controlador Difuso para el Sistema Ball and Beam de Quanser*, visitado 17-12-2020. [https://www.youtube.com/watch?v=K9Vu6w\\_IInE&list=PL-Z\\_5Zx\\_UY59NFn0tXmD-8x4Mgz\\_DTYgh](https://www.youtube.com/watch?v=K9Vu6w_IInE&list=PL-Z_5Zx_UY59NFn0tXmD-8x4Mgz_DTYgh).
- Kolda, T., Lewis, R. M., y Torczon, V. (2003). Optimization By Direct Search: New perspective on some clasical and modern methods. *SIAM Review*, 45(3), 385-482.
- Mamdani, E. (1974). *Aplication of a fuzzy algorithms for control a simple dynamic plant* (Vol. vol. D-121). Proc. Inst. Elec. Eng.
- Mamdani, E., y Assilian, S. (1975). *An experiment in linguistic syntesis with a fuzzy logic controller* (Vol. 7). International Journal Man Machine Studies.
- Moezzi, R., Tamre, M., y Minh, V. T. (s.f.). Fuzzy Logic Control fora Ball and

- Beam System, publisher = International Journal Innovative Technology and Interdisciplinary Sciences, volume = Vol. 1, pages = Issue 1, 39-48, year = 2018.
- Passino, K., y Yurkovich, S. (1998). *Fuzzy Control*. Addison Wesley.
- Sameera, R., Arun, S., y Mathew. (2017). Position tracking of ball and beam system using fractional order controller. , *Vol. 4*, Special Issue 6.
- Simon, D. (2002). *Sum Normal Optimazation of fuzzy membership functions*. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, 10(04):363-384.
- Tremante, P. (2001). *Control de Procesos con Lógica Difusa*. Trabajo de Maestría en Ingeniería Eléctrica, Escuela de Ingeniería Eléctrica, Universidad Central de Venezuela.
- Tremante, P. (2006). *Introducción al Control Difuso* (1ra ed.). Trabajo de Ascenso a la Categoría de Agregado. Caracas: Escuela de Ingeniería Eléctrica, Universidad Central de Venezuela.
- Tremante, P. (2019). *Introducción al Control Difuso*. Apuntes de Postgrado, Sem 01-2019, Escuela de Ingeniería Eléctrica, Universidad Central de Venezuela.
- Warner, J. (2019). *Open source community, scikit-fuzzy, visitado 15-05-2020*. <https://github.com/scikit-fuzzy/scikit-fuzzy>.
- Yasunobo, S., y Miyamoto, S. (1983). Fuzzy Control for Automatic Train Operation System, in Proc. 4th IFAC/IFIP/IFORS Int. Congress on Control Transportation System.
- Zadeh, L. (1965). Fuzzy Sets. *Information and Control*, *Vol: 8*, pag: 338-353.

## Apéndice I

### CÓDIGO EN MATLAB PARA GRÁFICA

```
1 function dataAdc = plotData(x, y, s)
2 Ts = 0.012;
3 ts = 0.012:Ts:7;
4 bandera = false;
5 flushinput(s);
6
7 if (y<0)
8     y = y + 256;
9 end
10
11 if (strcmp(x, 'difuso'))
12     fwrite(s,0);
13     bandera=true;
14 end
15 if (strcmp(x, 'difusoOpti'))
16     fwrite(s,1);
17     bandera=true;
18 end
19 if (strcmp(x, 'pd'))
20     fwrite(s,2);
21     bandera=true;
22 end
23
24 if (bandera)
25     fwrite(s,y);
26     datos = fread(s,583, 'uint16');
```

```
27     flushinput(s)
28
29     for i=1:583
30         dataAdc(i) = (((datos(i)*3.3/4095.0) * 0.13468) - 0.2222)
31         * 100;
32
33     end
34
35     stepinfo(dataAdc,ts,y)
36
37     plot(ts,dataAdc)
38
39 end
```

Código 1.1. plot.m.

## Apéndice II

### CÓDIGO EN MATLAB PARA AJUSTE DE FUNCIONES DE PERTENENCIA

```
1 %La funcion devuelve los parametros del optitank.fis , x y p. A la
2 %funcion se le debe indicar:
3 %dtol:(tolerancia de convergencia)
4 %d0:( longitud del paso inicial).
5 %simetria: variable booleana en la cual se indica si se desea aplicar
6 %simetría o no.
7
8 function [ optitank x p tiempo] = fuzzypendule12(dtol,d0,simetria,time
9
10 if nargin ~= 4
11     error('Se requieren tres argumentos: dtol , d0 y simetría');
12 end
13
14 if dtol > d0 ,
15     error('Condicion no permitida: dtol > d0');
16 end
17
18 if (d0 >= 1) || (d0<=-1),
19     error('Condicion no permitida: d0 e (-1,1)');
20 end
21 if (simetria ~= 1)&&(simetria ~=0)&&(simetria ~=true)&&(simetria ~=
22     false),
23     error('"simetría" debe ser "true" o "false"');
```

```

23 end
24 if (time >= 100) || (time<=0),
25     error ('Condicion no permitida: time>0');
26 end
27
28 global pendulo; %declarar %global tanque en command window
29
30
31 %Lectura inicial del .fis y Simulación Inicial
32 pendulo=readfis ('C:\Users\brand\OneDrive\Escritorio\tesis\simulacion\
33             simu2\versionesControlPos\posicion8imple');
34 [ optitank x p tiempo]=optimizarrapida(dtol,d0,simetria,time);
35 end

```

Código 2.1. fuzzypendule12.m.

```

1 function [ optitank xm p tiempo] = optimizar12(dtol,d0,simetria,time)
2 %ESTA FUNCIÓN SE ENCARGA DE REALIZAR LA OPTIMIZACIÓN DEL SISTEMA
3 % DIFUSO
4 % POR EL MÉTODO DE BÚSQUEDA DIRECTA.
5 %VARIABLES DE ENTRADS:
6 %dtol1: criterio de parada del método de búsqueda directo.
7 %d01: es el paso inicial de búsqueda.
8 tic;
9 global pendulo;
10
11 %SE PREPARA EL VECTOR X0 PARA EL ALGORITMO DE OPTIMIZACIÓN, A PARTIR
12 % DE LOS VÉRTICES ACTUALES:
13 x0=vertices12(pendulo,simetria);
14
15 v0=indice12(pendulo,simetria);
16
17 %SE NORMALIZA Y ESCALA EL VECTOR PARA QUE TODA FP TRABAJE EN EL RANGO
18 % [-1,1]
19 [x0]=normalizar12(pendulo,x0,simetria);

```

```

17
18 %SE CALCULA EL ORDEN DEL SISTEMA:
19 dim=length(v0);
20
21 %SE INICIALIZAN LOS PARÁMETROS PARA EL MÉTODO DE BÚSQUE DIRECTA.
22 xk=x0'; %Vector que contiene los vértices de las F.P.
23 vk=v0'; %Es el vector inciial que contiene los índices (nT, n1 y n2)
            de las F.P involucradas en la optimización.
24 dk=d0;
25 D=[eye(dim) -1*eye(dim)]; %Matriz base para el dominio de búsqueda.
26 p=0;
27 vk1=zeros(dim,2*dim); %Se inicializa el vector vk1 según las
            dimensiones que corresponden.
28 fk1=zeros(2*dim,1); %Se inicializa el vector fk1 según las
            dimensiones que corresponden.
29
30 %Primera simulación:
31 xk=actualizar12(pendulo,xk,vk,simetria); %Se actualizan los vértices
            en función de los índices nT, n1 y n2 iniciales.
32 [ valido ,Px]=penalizar12(pendulo,xk',vk',simetria); %Se aplica la
            función de penalización
33
34 if valido
            xd=desnormalizar12(pendulo,xk(:,1),simetria); %Se
            desnormaliza
            pendulo=parametros12(pendulo,xd,vk,simetria); %Se
            asignan los vértices al archivo .fis
            sim('C:\Users\brand\OneDrive\Escritorio\tesis\simulacion
            \simu2\fuzzyImpleAjust',time);
35
36 end
37 fk=fobjetivo12(u,ess,salida,valido,Px,time); %Se evalua la simulación
            en la Función objetivo
38
39
40
41

```

```

42 while (dk>dtol) %MIENTRAS QUE EL PASO SEA MAYOR A LA TOLERANCIA,
43 ENTONCES:
44 aux=0;
45 p=p+1; %Puntero que indica el número de iteraciones.
46
47 %%Se calculan todos los vk+1:
48 for i=1:2*dim
49     for j=1:dim
50         vk1(j,i)=vk(j,1)+dk*D(j,i);
51     end
52 end
53
54 %%SE APLICAN LAS RESTRICCIONES Y SE EVALUAN LOS xk+1 PARA
55 OBTENER f(xk+1):
56 for i=1:2*dim
57
58     xk1=actualizar12(pendulo,xk,vk1(:,i),simetria); %Para
59     cada posible solución en el cambio de índices (nT,n1 y n2), se
60     actualizan los vértices.
61     [ valido ,Px]=penalizar12(pendulo,xk1',vk1(:,i)',simetria);
62     %Se aplica la penalización.
63
64     %Se simula para obtener la salida del sistema para cada
65     fk1:
66
67         if valido
68             xd=desnormalizar12(pendulo,xk1,simetria); %Llamado a
69             la función par desnormalizar las componentes de xk1
70
71             %Se asigna el valor desnormalizado al archivo .fis ,
72             aplicando simetría , a los efectos de obtener la simulación:
73             pendulo=parametros12(pendulo,xd,vk1(:,i),simetria);
74             sim('C:\Users\brand\OneDrive\Escritorio\tesis\
75             simulacion\simu2\fuzzyImpleAjust',time);
76
77         end
78
79         %Se plica la función objetivo para cada fk+1:
80         fk1(i,1)=fobjetivo12(u,ess,salida,valido,Px,time);

```

```

67 %disp (fk1 (i ,1)) ;
68
69 if fk1 (i ,1)<fk
70 fk=fk1 (i ,1) ;
71 vk=vk1 (: ,i); %Se asigna a vk el nuevo valor que
minimiza a la función objetivo .
72 xk=actualizar12 (pendulo ,xk ,vk ,simetria); %Se
actualiza en definitivo los vértices en función del Vk que
minimiza la F.O
73 aux=1;
74 break
75 end ;
76
77 end
78
79 if (aux==0)
80 dk=1/2*dk;
81 end
82
83 if p>100 %criterio de parada en caso de que la función no presente
mínimo (evitar loop)
84 break
85 end
86
87 end
88 xm=desnormalizar12 (pendulo ,xk ,simetria); %El vector de salida es xk
desnormalizado .
89 pendulo=parametros12 (pendulo ,xm ,vk ,simetria);
90 %Se simula para obtener la salida del sistema con el valor óptimo
91 %alcanzado :
92 sim ( 'C:\ Users\brand\OneDrive\Escritorio\tesis\simulacion\simu2\
fuzzyImpleAjust' ,time);
93
94 optitank=pendulo; %Se asigna a la salida el archivo .fis optimizado

```

```

    por esta función.

95 tiempo=toc;
96
97 end

```

Código 2.2. optimizarrapida.m.

```

1 function [ x0 ] = vertices12( pendulo , simetria )
2 %ESTA FUNCIÓN FORMA EL VECTOR X0 PARA EL ALGORITMO DE OPTIMIZACIÓN, A
3 %PARTIR DE LOS VÉRTICES ACTUALES DE LAS FUNCIONES DE PERTENENCIA
4 N_IN=length( pendulo . input );
5 x0=[];
6 for i=1:N_IN
7     if ( simetria )
8         if rem( length( pendulo . input ( i ) . mf) ,2)==0 % Si el número de
las FP son pares:
9             N_FP=length( pendulo . input ( i ) . mf ) /2;
10            else %Si el número de F.P es impar:
11                N_FP=floor( length( pendulo . input ( i ) . mf ) /2)+rem( length(
pendulo . input ( i ) . mf ) ,2);
12            end
13        else
14            N_FP=length( pendulo . input ( i ) . mf );
15        end
16        for j=1:N_FP
17            if ( j==1)
18                x0=horzcat (x0 , pendulo . input ( i ) . mf(j) . params (3:4)); %Se
considera la primera F.P trapezoidal abierta y se toman los
vértices "c" y "d" únicamente.
19            else
20                if ( j==N_FP)
21                    x0=horzcat (x0 , pendulo . input ( i ) . mf(j) . params (1:2)); %
Se considera la última F.P trapezoidal abierta y se toman los
vértices "a" y "b" únicamente.
22            else
23                x0=horzcat (x0 , pendulo . input ( i ) . mf(j) . params (1:4)); %

```

```

Se incluyen todos los vértices pra cualquier otra F.P
24         end
25     end
26 end
27
28
29 end
30
31 end

```

Código 2.3. vertices12.m.

```

1 function [ v0 ] = indice12( pendulo , simetria )
2 %ESTA FUNCIÓN FORMA EL VECTOR V0 PARA EL ALGORITMO DE OPTIMIZACIÓN, A
3 %PARTIR DE LOS INDICES ACTUALES DE LAS FUNCIONES DE PERTENENCIA
4 N_IN=length( pendulo . input );
5 v0=[];
6 for i=1:N_IN
7     if ( simetria )
8         if rem( length( pendulo . input ( i ) . mf ) ,2 )==0 % Si el número de
las FP son pares :
9             N_FP=length( pendulo . input ( i ) . mf ) /2;
10            else %Si el número de F.P es impar :
11                N_FP=floor( length( pendulo . input ( i ) . mf ) /2)+rem( length(
pendulo . input ( i ) . mf ) ,2 );
12            end
13        else
14            N_FP=length( pendulo . input ( i ) . mf );
15        end
16    for j=1:N_FP
17        v0=horzcat ( v0 , pendulo . input ( i ) . mf(j) . params(5:7) ); %Se
toman los índices (nT, n1 y n2) de las F.P
18    end
19
20
21 end

```

```
22  
23 end
```

Código 2.4. indice12.m.

```
1 function [xout] = normalizar12(pendulo,xin,simetria)  
2  
3 xout=xin; %El vector de vértices de salida inicialmente es igual al  
4 %de entrada.  
5 min=-1; %El mínimo del rango deseado para los efectos de trabajo del  
6 %algoritmo.  
7 max=1; %El máximo del rango deseado para los efectos de trabajo del  
8 %algoritmo.  
9 N_IN=length(pendulo.input); %Número de entradas del sistema.  
10  
11 offset=0; %Se inicializa el puntero del número de vértices.  
12 for i=1:N_IN  
13     N_FP=length(pendulo.input(i).mf); %Se obtiene el número de F.P de  
14     %la entrada (i).  
15     if (simetria==true)  
16         if rem(N_FP,2)==0 %Si el número de FP es par:  
17             N_V_FP=N_FP/2*4-2;  
18         else  
19             N_V_FP=(floor(N_FP/2)+rem(N_FP,2))*4-4;  
20         end  
21     else  
22         N_V_FP=N_FP*4-4;  
23     end  
24  
25     if (pendulo.input(i).range(1)~=min)|| (pendulo.input(i).range(2)~=max)  
26         for j=offset+1:offset+N_V_FP  
27             xout(j)=(xout(j)-pendulo.input(i).range(1))/(pendulo.  
28             input(i).range(2)-pendulo.input(i).range(1)); %Se mapea al rango  
29             %[0,1] (normalización)calar las componen
```

```

25         xout(j)=xout(j)*(max-min)+min; % Se escala al rango
26         [-1,1] (escalamiento).
27     end
28     offset=offset+N_V_FP;
29
30 end
31
32 end

```

Código 2.5. normalizar12.m.

```

1 function [ xout ] = actualizar12 ( pendulo ,x ,v , simetria )
2
3 v=v';
4 x=x';
5 dim=length(x);
6 xout=zeros(1, dim);
7
8 N_IN=length(pendulo . input );
9 offset=1;
10 k=1;
11 for i=1:N_IN
12     N_FP=length(pendulo . input ( i ) . mf );
13     if ( simetria )
14         if rem(N_FP,2)==0 %Si es par:
15             N_FP=N_FP/2;
16         else %Si es impar:
17             N_FP=( floor(N_FP/2)+rem(N_FP,2) );
18         end
19     end
20
21     for j=0:N_FP-1
22         if j==0
23             xout ( offset )=x ( offset )+v(k)*(x ( offset +1)-x ( offset )) ; %c=
24             +nT*(d-c)

```

```

24         xout ( offset+1)=x( offset )+(x( offset+1)-x( offset )) / v(k+1);
25         %d=c+(d-c) / n1
26         offset=offset +2;
27         k=k+3;
28     else
29         if j==N_FP-1
30             xout ( offset )=x( offset+1)-(x( offset+1)-x( offset )) / v(k
31             +1); %a=b-(b-a) / n1
32             xout ( offset+1)=x( offset+1)-v(k)*(x( offset+1)-x( offset
33             )) ; %b=b-nT(b-a)
34             offset=offset +2;
35             k=k+3;
36         else
37             xout ( offset )=x( offset+1)-(x( offset+1)-x( offset )) / v(k
38             +1); %a=b-(b-a) / n1
39             xout ( offset+1)=x( offset+1)-v(k)*(x( offset+1)-x( offset
40             )) ; %b=b-nT(b-a)
41             xout ( offset+2)=x( offset+2)+v(k)*(x( offset+3)-x( offset
42             +2)); %c=c+nT*(d-c)
43             xout ( offset+3)=x( offset+2)+(x( offset+3)-x( offset+2)) /
44             v(k+1); %d=c+(d-c) / n1
45             offset=offset +4;
46             k=k+3;
47     end
48 end
49

```

Código 2.6. actualizar12.m.

```

1 function [ valido ,Px] = penalizar12(pendulo,x,v,simetria)
2 %Se aplican las restricciones inherentes a las F.P y solapamientos.
3 valido=true;
4
5 %Se calcula el número de entradas:
6 N_IN=length(pendulo.input);
7
8 %Se calcula la dimensión de la matriz de restricciones
9 dim=0;
10 for i=1:N_IN
11 dim=max(dim,length(pendulo.input(i).mf));
12 end
13 if (simetria)
14     if rem(dim,2)==0 %Si es par:
15         dim=dim/2*4-2;
16     else %Si es impar:
17         dim=(floor(dim/2)+rem(dim,2))*4-4;
18     end
19 else %Si no se aplica simetria:
20     dim=dim*4-4;
21 end
22
23 %Se inicializa la matriz "m":
24 m=ones(N_IN,dim);
25
26 %Se arma la matriz de restricciones:
27 k=1;
28 for i=1:N_IN
29     N_FP=length(pendulo.input(i).mf); %Número totales de FP de la
      entrada "i".
30     if (simetria)
31         if rem(N_FP,2)==0 %Si es par:
32             N_V_FP=N_FP/2*4-2;
33         else %Si es impar:

```

```

34     N_V_FP=(floor (N_FP/2)+rem (N_FP,2) ) *4-4;
35
36 else %Si no se aplica simetria:
37     N_V_FP=N_FP*4-4; %Número totales de vértices de las FP de la
38     %entrada "i".
39
40 for j=1:N_V_FP
41     m(i,j)=x(k);
42     k=k+1;
43
44 end
45
46
47 %

```

---

```

48 %REGLA # 1 RESTRICCIONES PROPIAS DE LA FP TRAPEZOIDAL:
49
50 %Caso 1: b>a
51 for i=1:N_IN
52     N_FP=length (pendulo . input (i) . mf);
53     if (simetria)
54         if rem(N_FP,2)==0 %Si es par:
55             N_FP=N_FP/2;
56         else %Si es impar:
57             N_FP=(floor (N_FP/2)+rem (N_FP,2));
58
59     end
60
61 for j=0:(N_FP-2)
62     if m(i,3+4*j)>=m(i,4+4*j) %Si a>=b, entonces:
63         valido=false;
64

```

```

65     end
66 end
67
68 %Caso 2: c>=b
69
70 for i=1:(N_IN)
71     N_FP=length(pendulo.input(i).mf);
72     if (simetria)
73         if rem(N_FP,2)==0 %Si es par:
74             N_FP=N_FP/2+1;
75         else %Si es impar:
76             N_FP=(floor(N_FP/2)+rem(N_FP,2));
77         end
78     end
79
80     for j=0:(N_FP-3)
81         if m(i,4+4*j)>m(i,5+4*j) %Si b>c, entonces:
82             valido=false;
83         end
84     end
85 end
86
87 %Caso 3: d>c
88 for i=1:(N_IN)
89     N_FP=length(pendulo.input(i).mf);
90     if (simetria)
91         if rem(N_FP,2)==0 %Si es par:
92             N_FP=N_FP/2+1;
93         else %Si es impar:
94             N_FP=(floor(N_FP/2)+rem(N_FP,2));
95         end
96     end
97
98     for j=0:(N_FP-2)

```

```

99         if m(i,1+4*j)>=m(i,2+4*j) %Si c>=d, entonces:
100             valido=false;
101         end
102     end
103 end
104
105 P1=0;
106 if valido==false
107     P1=P1+100;
108 end
109
110 %

```

---

```

111 %REGLA # 2 GARANTIZAR SOLAPAMIENTO ENTRE FP CONTIGUAS DE LA ENTRADA "i";
112 %Vértice "d" de la F.P "j" > "a" de la F.P (j+1) son iguales,
113
114 P2=0;
115 for i=1:N_IN
116     N_FP=length(pendulo.input(i).mf);
117     if (simetria)
118         if rem(N_FP,2)==0 %Si es par:
119             N_FP=N_FP/2;
120         else %Si es impar:
121             N_FP=(floor(N_FP/2)+rem(N_FP,2));
122         end
123     end
124
125     for j=0:(N_FP-2)
126         a=m(i,3+4*j);
127         d=m(i,2+4*j);
128         P2=P2+100*max(a-d,0); %Función de penalización 1
129     end

```

```

130
131 end
132
133
134 %
-----%
135 %REGLA # 3 EVITAR PERTENENCIA TOTAL:
136 %Vértice "c" de la F.P "j" < vértice "b" de la F.P (j+1)
137
138 P3=0;
139 for i =1:N_IN
140     N_FP=length(pendulo.input(i).mf);
141     if (simetria)
142         if rem(N_FP,2)==0 %Si es par:
143             N_FP=N_FP/2;
144         else %Si es impar:
145             N_FP=(floor(N_FP/2)+rem(N_FP,2));
146         end
147     end
148
149     for j =0:(N_FP-2)
150         c=m(i,1+4*j);
151         b=m(i,4+4*j);
152         P3=P3+100*max(c-b,0); %Función de penalización 2
153     end
154
155 end
156
157 %
-----%
158 %REGLA # 4 EVITAR MÚLTIPLE SOLAPAMIENTO.
159 %Vértice "d" de la F.P "j" < vértice "a" de la F.P (j+2)

```

```

160
161 P4=0;
162 for i=1:N_IN
163     N_FP=length(pendulo.input(i).mf);
164     if (simetria)
165         if rem(N_FP,2)==0 %Si es par:
166             N_FP=N_FP/2;
167         else %Si es impar:
168             N_FP=(floor(N_FP/2)+rem(N_FP,2));
169         end
170     end
171
172     for j=0:(N_FP-3)
173         a=m(i,7+4*j);
174         d=m(i,2+4*j);
175         P4=P4+100*max(d-a,0); %Función de penalización 2
176     end
177 end
178 %
179 %



---


180 %REGLA # 5 RESTRICCIÓN DE LOS EXTREMOS
181 %El vértice "c" de la 1ra F.P >=-1 y el vértice "b" de la última F.P
182 >=1
183 P5=0;
184 for i=1:N_IN
185     c=m(i,1);
186     P5=P5+100*max(-(1+c),0);
187
188     N_FP=length(pendulo.input(i).mf);
189     if (simetria)
190         if rem(N_FP,2)==0 %Si es par:
191             N_V_FP=N_FP/2*4-2;

```

```

191         c=m(i,N_V_FP-1);
192         P5=P5+100*max(c,0);
193         d=m(i,N_V_FP);
194         P5=P5+100*max(d-1,0);
195     else %Si es impar:
196         N_V_FP=(floor(N_FP/2)+rem(N_FP,2))*4-4;
197         d=m(i,N_V_FP-2);
198         P5=P5+10*max(d,0);
199         a=m(i,N_V_FP-1);
200         P5=P5+100*max(-(1+a),0);
201         b=m(i,N_V_FP);
202         P5=P5+100*max(b,0);
203
204     end
205 else %Si no se aplica simetria:
206     N_V_FP=N_FP*4-4; %Número totales de vértices de las FP de la
207     %entrada "i".
208     b=m(i,N_V_FP);
209     P5=P5+100*max(b-1,0);
210
211
212
213 end
214 if P5~=0
215     valido=false;
216 end
217 %


---


218 %REGLA # 6 RESTRICCIONES DE LOS ÍNDICES
219 offset=0;
220 for i=1:(N_IN)
221     N_FP=length(pendulo.input(i).mf);

```

```

222 if ( simetria )
223     if rem(N_FP,2)==0 %Si es par:
224         N_FP=N_FP/2;
225     else %Si es impar:
226         N_FP=(floor(N_FP/2)+rem(N_FP,2));
227     end
228 end
229
230 for j=0:N_FP-1
231
232     if v(offset+1+3*j)>=1/v(offset+2+3*j) %T<1/n1
233         valido=false;
234     end
235     if (v(offset+1+3*j)<0 || v(offset+1+3*j)>=1) %0<=nT<1
236         valido=false;
237     end
238     if v(offset+2+3*j)<=0 %1>0
239         valido=false;
240     end
241     if v(offset+3+3*j)<=0 %2>0
242         valido=false;
243     end
244
245 end
246 offset=offset+N_FP*3;
247
248 end
249
250 %


---


251 if valido==false
252     P6=100;
253 else

```

```

254     P6=0;
255 end
256 %
257 Px=P1+P2+P3+P4+P5+P6; %Función de penalización
258 end

```

Código 2.7. penalizar12.m.

```

1 function [ xout ] = desnormalizar12( pendulo ,xin ,simetria )
2
3 xout=xin ;
4 min=-1;
5 max=1;
6 N_IN=length( pendulo . input );
7 offset=0;
8
9 for i=1:N_IN
10    N_FP=length( pendulo . input ( i ) . mf ); %Se obtiene el número de F.P de
11    la entrada ( i ) .
12    if ( simetria==true )
13        if rem(N_FP,2)==0 %Si el número de FP es par:
14            N_V_FP=N_FP/2*4-2;
15        else %Si el número de FP es impar:
16            N_V_FP=( floor( N_FP/2 )+rem( N_FP,2 ) )*4-4;
17        end
18    else
19        N_V_FP=N_FP*4-4;
20    end
21    if ( pendulo . input ( i ) . range(1)~=min ) || ( pendulo . input ( i ) . range(2)~=max )
22        for j=offset+1:offset+N_V_FP
23            xout( j )=(xout( j )-min)/(max-min); %Se normaliza
            xout( j )=xout( j )*(pendulo . input ( i ) . range(2)-pendulo . input (
i ) . range(1 ))+pendulo . input ( i ) . range(1); %Se escala al rango real

```

```

    indicado en la matriz de normalización.

24     end
25
26     offset=offset+N_V_FP;
27 end
28
29 xout=xout';
30
31 end

```

Código 2.8. desnormalizar12.m.

```

1 function [ pendulo_out ] = parametros12(pendulo,x,v,simetria)
2 %Asigna al archivo.fis los valores de los vértices del contenidos en
3 %vector "x"
4
5 pendulo_out=pendulo;
6
7 x=x';
8 v=v';
9 N_IN=length(pendulo_out.input);
10 offset=1;
11 k=1;
12 if simetria
13     for i=1:N_IN
14         N_FP=length(pendulo.input(i).mf);
15         if rem(N_FP,2)==0 % Si el número de las FP son pares:
16             N_FP_S=N_FP/2;
17         else %Si el número de F.P es impar:
18             N_FP_S=floor(N_FP/2)+rem(N_FP,2);
19         end
20         for j=1:N_FP_S
21             if (j==1)
22                 pendulo_out.input(i).mf(j).params(3:4)=x(offset:

```

offset+1); %Si es la primera F.P se considera trapezoidal abierta

```

23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
.
pendulo_out . input ( i ) . mf( N_FP+1-j ) . params ( 1:2 )
=(-1)*fliplr (x( offset : offset +1)) ; %Si es la primera F.P se
considera trapezoidal abierta .
pendulo_out . input ( i ) . mf( j ) . params ( 5:7 )=v( k:k+2 );
%Se cargan los indices (nT, n1 y n2) de cada F.P
pendulo_out . input ( i ) . mf( N_FP+1-j ) . params ( 5:7 )=v( k
:k+2 );
k=k+3;
offset=offset+2;
else
if ( j==N_FP_S )
pendulo_out . input ( i ) . mf( j ) . params ( 1:2 )=x(
offset : offset +1); %Si es la ultima F.P se considera trapezoidal
abierta .
pendulo_out . input ( i ) . mf( N_FP+1-j ) . params ( 3:4 )
=(-1)*fliplr (x( offset : offset +1)); %Si es la primera F.P se
considera trapezoidal abierta .
pendulo_out . input ( i ) . mf( j ) . params ( 5:7 )=v( k:k
+2 ); %Se cargan los indices (nT, n1 y n2) de cada F.P
pendulo_out . input ( i ) . mf( N_FP+1-j ) . params ( 5:7 )
=v( k:k+2 );
k=k+3;
offset=offset+2;
else
pendulo_out . input ( i ) . mf( j ) . params=x( offset :
offset +3); %Se consideran todos los verties en cualquier otra F.P
pendulo_out . input ( i ) . mf( N_FP+1-j ) . params=(-1)
*fliplr (x( offset : offset +3)); %Se consideran todos los verties en
cualquier otra F.P
pendulo_out . input ( i ) . mf( j ) . params ( 5:7 )=v( k:k
+2 );
pendulo_out . input ( i ) . mf( N_FP+1-j ) . params ( 5:7 )
=v( k:k+2 );

```

```

41           k=k+3;
42           offset=offset+4;
43       end
44   end
45 end
46
47
48
49
50 else % Si no hay simetria:
51     for i=1:N_IN
52         N_FP=length(tanque.input(i).mf);
53         for j=1:N_FP
54             if (j==1)
55                 tanque_out.input(i).mf(j).params(3:4)=x(offset:
56 offset+1); %Si es la primera F.P se considera trapezoidal abierta
57
58             tanque_out.input(1).mf(j).params(5:7)=v(k:k+2);
59             offset=offset+2;
60             k=k+3;
61         else
62             if (j==N_FP)
63                 tanque_out.input(i).mf(j).params(1:2)=x(
64 offset:offset+1); %Si es la última F.P se considera trapezoidal
65                 abierta.
66
67             offset=offset+2;
68             tanque_out.input(1).mf(j).params(5:7)=v(k:k
69 +2); %Se cargan los indices (nT, n1 y n2) de cada F.P
70             k=k+3;
71         else
72             tanque_out.input(i).mf(j).params(1:4)=x(
73 offset:offset+3); %Se consideran todos los vérties en cualquier
74                 otra F.P
75
76             tanque_out.input(1).mf(j).params(5:7)=v(k:k

```

```

+2); %Se cargan los indices (nT, n1 y n2) de cada F.P
68                               offset=offset+4;
69                               k=k+3;
70           end
71       end
72   end
73
74   end
75 end
76 end

```

Código 2.9. parametros12.m.

```

1 function [ fx ] = fobjetivo12(u, ess, salida, valido, Px, time)
2
3 if valido
4
5     Up=max(u);
6     Ud=min(u);
7 %SE CALCULA LA FUNCIÓN OBJETIVO (ERROR CUADRÁTICO, ERROR EN ESTADO
8 %ESTACIONARIO Y SOBREPICO)
9 %


---


10 %Error cuadrático:
11 f1=sumsqr(ess)/length(ess);
12 %
13 %


---


14 %Error en estado estacionario;
15
16 f2=abs(Up-salida.signals.values(round((length(salida.signals.
17 values)/2*3/4))))+abs(Ud-salida.signals.values(round((length(
18 salida.signals.values)*3/4))))) ;

```

```

17 %
18 %

19 %Sobrepico:
20
21     %f max( salida.signals.values)>Up %Se comprueba si existe un
22     %sobrepico.
23     %    sobrepico_up=max( salida.signals.values); %Se determina el
24     %sobre pico.
25     %else
26     %    sobrepico_up=Up; %Si no hay sobrepico, se iguala a la
27     %referencia para anular el cálculo en la función objetivo.
28     %end
29     if min( salida.signals.values( find( salida.signals.values<Ud) ))<Ud
30 %Sobrepico en el vaciado.
31         sobrepico_ud=min( salida.signals.values( find( salida.signals.
32         values<Ud) ));
33     else
34         sobrepico_ud=Ud;
35     end
36     %3=abs(sobrepico_up-Up)+abs(sobrepico_ud-Ud);
37     f3=abs(sobrepico_ud-Ud);
38 %

39 %Tiempo de establecimiento:
40     w=2; %variable auxiliar para el cálculo del ts.
41     if max( salida.signals.values)<Ud %sistema de 2do orden o con
42     %sobrepico
43         z=find( ( salida.signals.values<Ud+0.05)&(salida.signals.
44         values>Ud-0.05)); %Se busca en "salida" los valores que estén
45         %entorno al 5% de la referencia.
46         for i=length(z):-1:2 %Se busca el la estabilidad entorno al

```

```

+/-5% de la referencia (Se ignoran los cruces por la referencia
que generen sobrepicos por encima del +/-5%
39 if z(i) ~= z(i-1)+1
40     w=i;
41 end
42 end
43 ts=salida.time(z(w)); %Se determina el tiempo de
establecimiento.
44 else
45     z=find(salida.signals.values<0.05);
46     if isempty(z)
47         ts=time; %Si no alcanza el criterio del 5%, se penaliza
la funcion asignando el maximo valor el tiempo de simulacion.
48     else
49         ts=salida.time(min(z));
50     end
51 end
52 f4=ts/time; %normalizado
53 %

```

---

```

54 %Tiempo de alza:
55 if max(salida.signals.values)<Ud
56     ta=salida.time(find(salida.signals.values<Ud, 1));
57     f5=ta/time; %normalizado
58 else
59     f5=0;
60 end
61
62 fx=0.5*f1+0.1*f2+0.2*f3+0.15*f4+0.05*f5+Px;
63 else
64     fx=Px;
65 end

```

66 end

Código 2.10. fobjetivo12.m.

## Apéndice III

### CÓDIGO EN EL ESP32

```
1  /*
2   *      Universidad Central de Venezuela
3   *      Controlador Difuso para el Sistema Ball and Beam de Quanser
4   *      Autor: Br. Brandon Jimenez
5   */
6
7 #include "uart1.h"
8
9 //Funcion de inicio
10 extern "C" void app_main()
11 {
12     //Instancia de clase Timer para inicio del timer para el muestreo
13     Timer timer1(TIMER_GROUP_0, TIMER_0, TIMER_ALARM_EN, 1, 0.012);
14
15     //Funcion configuracion del UART
16     uart_config();
17 }
```

Código 3.1. main.cpp.

```
1
2 #ifndef UART1_H_
3 #define UART1_H_
4
5 #include <string.h>
6 #include "interrupcion.h"
7
```

```

8 //Tamano del buffer del UART
9 #define RD_BUF_SIZE (BUF_SIZE)
10 #define BUF_SIZE (1024)

11
12 //Pines del UART
13 #define Tx1 4
14 #define Rx1 5

15
16 //LOG UART
17 static const char *TAG = "uart_events";
18
19 //Set point
20 static int8_t dato_set_point;

21
22 //Manejo de cola de recepcion del UART
23 static QueueHandle_t uart0_queue;

24
25 //Funcion de configuracion del UART
26 void uart_config();
27
28 #endif

```

Código 3.2. uart1.h.

```

1
2 #include "uart1.h"
3
4 // Variable externa Set point
5 extern float set_point;
6 // Variable externa booleana para grafica
7 extern bool plot;
8 // Variable para el tipo de controlador
9 extern uint8_t tipoControl;
10
11 // Variable para Maquina de estados del sistema
12 enum estado{

```

```

13     tipo=0,
14     controlador
15 }estado;
16
17 // Drivers para control del UART
18 #include "driver/uart.h"
19
20 //Cola de envio de datos para la grafica
21 extern QueueHandle_t dataplot;
22
23 //Instancia de clase Timer para inicio del timer para l toma de datos
24 //para la grafica
24 Timer timer2(TIMER_GROUP_0, TIMER_1, TIMER_ALARM_EN, 0, 7.0);
25
26
27 //Eventos del UART
28 static void uart_event_task(void *pvParameters)
29 {
30     uart_event_t event;
31     size_t buffered_size;
32     uint8_t * dtmp = (uint8_t *) malloc(RD_BUF_SIZE);
33     while(1) {
34         //Se espera ocurra un evento en el UART
35         if(xQueueReceive(uart0_queue, (void *) &event, (portTickType)
36             portMAX_DELAY)) {
37             bzero(dtmp, RD_BUF_SIZE);
38             ESP_LOGI(TAG, "uart[%d] event:", UART_NUM_1);
39             switch(event.type) {
40                 case UART_DATA:
41                     uart_read_bytes(UART_NUM_1, dtmp, event.size,
42                         portMAX_DELAY);
42                     //Se identifica el estado
43             }
44         }
45     }
46 }
```

```

44         switch(estado){
45
46             case tipo:
47                 switch (*dtmp){
48                     // Se establece el tipo de controlador
49                     case 0:
50                         tipoControl = 0;
51                         ESP_LOGI("FUZZY", "Control Difuso en uso");
52                         estado = controlador;
53                         break;
54
55                     case 1:
56                         tipoControl = 1;
57                         ESP_LOGI("FUZZY", "Control Difuso Ajustado en uso");
58                         estado = controlador;
59                         break;
60
61                     case 2:
62                         tipoControl = 2;
63                         ESP_LOGI("PD", "Control PD en uso");
64                         estado = controlador;
65                         break;
66
67                     default:
68                         ESP_LOGI(TAG, "Funcion no valida");
69                         break;
70                     }
71                     break;
72
73             case controlador:
74                 // Se establece la consigna
75                 dato_set_point = (int8_t)*dtmp;
76
77             if (dato_set_point >= -15 && dato_set_point <= 15) {

```

```

78
79         ESP_LOGI(TAG, "[UART DATA] Length: %d", event.size);
80         set_point = (float)dato_set_point / 100;
81         ESP_LOGI(TAG, "[UART DATA]: %f", set_point);
82         plot = true;
83         timer2.timer_inicio();
84         uart_flush_input(UART_NUM_1);
85
86     } else {
87         ESP_LOGI(TAG, "Fuera de rango");
88     }
89     estado = tipo;
90
91     break;
92
93     default:
94     break;
95
96 }
97
98
99
100    break;
101 case UART_FIFO_OVF:
102     ESP_LOGI(TAG, "hw fifo overflow");
103     uart_flush_input(UART_NUM_1);
104     xQueueReset(uart0_queue);
105     break;
106 case UART_BUFFER_FULL:
107     ESP_LOGI(TAG, "ring buffer full");
108     uart_flush_input(UART_NUM_1);
109     xQueueReset(uart0_queue);
110     break;
111 case UART_BREAK:

```

```

112         ESP_LOGI(TAG, "uart rx break");
113         break;
114     case UART_PARITY_ERR:
115         ESP_LOGI(TAG, "uart parity error");
116         break;
117     case UART_FRAME_ERR:
118         ESP_LOGI(TAG, "uart frame error");
119         break;
120     default:
121         ESP_LOGI(TAG, "uart event type: %d", event.type);
122         break;
123     }
124 }
125 }
126 free(dtmp);
127 dtmp = NULL;
128 vTaskDelete(NULL);
129 }
130
131 //Funcion para envio de datos para grafica
132 void plot_task(void *params){
133     uint16_t hola[1] = {0};
134     uint16_t *buffer = hola;
135     while(1){
136         //Se recibe datos en la cola
137         xQueueReceive(dataplot, buffer, portMAX_DELAY);
138         //Se envian un uint16 con el valor del ADC
139         uart_write_bytes(UART_NUM_1, (const char*)hola, 2);
140     }
141     vTaskDelete(NULL);
142 }
143
144 //Funcion para configuracion del UART
145 void uart_config(){

```

```

146
147     uart_config_t uart_config = {
148         .baud_rate = 115200,
149         .data_bits = UART_DATA_8_BITS,
150         .parity = UART_PARITY_DISABLE,
151         .stop_bits = UART_STOP_BITS_1,
152         .flow_ctrl = UART_HW_FLOWCTRL_DISABLE
153     };
154     uart_param_config(UART_NUM_1, &uart_config);
155
156     uart_set_pin(UART_NUM_1, Tx1, Rx1, UART_PIN_NO_CHANGE,
157                  UART_PIN_NO_CHANGE);
158     uart_driver_install(UART_NUM_1, BUF_SIZE * 2, BUF_SIZE * 2, 20, &
159                         uart0_queue, 0);
160
161     //Se crean tareas para el Eventos del UART y la tarea para envio de
162     //datos para la grafica
163     xTaskCreatePinnedToCore(uart_event_task, "uart_event_task", 2048,
164                            NULL, 12, NULL, 0);
165     xTaskCreatePinnedToCore(plot_task, "plot", 2048*2, NULL, 5, NULL, 0)
166     ;
167
168 }

```

Código 3.3. uart1.cpp.

```

1
2 #ifndef INTERRUPCION_H_
3 #define INTERRUPCION_H_
4
5 #include "controlador.h"
6 // Drivers para control de Timers
7 #include "driver/periph_ctrl.h"
8 #include "driver/timer.h"
9
10 // Variables para control de los Timers

```

```

11 #define TIMER_DIVIDER           16 // Hardware timer clock divider
12 #define TIMER_SCALE            (TIMER_BASE_CLK / TIMER_DIVIDER) // convert counter value to seconds
13 #define TEST_WITHOUT_RELOAD     0          // testing will be done without auto reload
14 #define TEST_WITH_RELOAD        1          // testing will be done with auto reload
15
16 //Log Timer
17 static char* TIMER_TAG = "TIMERS";
18
19 //Clase Timer
20 class Timer{
21
22 public:
23     Timer(timer_group_t timer_group, timer_idx_t timer_idx, bool
24         interrupcion, bool start, double timer_interval_sec);
25     timer_idx_t timer_idx;
26     timer_group_t timer_group;
27     double timer_interval_sec;
28     esp_err_t pause();
29     esp_err_t timer_inicio();
30
31 private:
32     bool interrupcion;
33     bool inicio;
34     void timer_config();
35     typedef struct {
36         int type; // the type of timer's event
37         int timer_group = this->timer_group;
38         int timer_idx = this->timer_idx;
39         uint64_t timer_counter_value;
40     } timer_event_t;

```

```

41     protected :
42         intr_handle_t manejador_timer;
43         static void start_timer_group0_isr(void *para);
44         void timer_group0_isr();
45
46     };
47
48 #endif

```

Código 3.4. interrupcion.h.

```

1
2 #include "interrupcion.h"
3
4 //Semaforo para interrupciones de Timers
5 SemaphoreHandle_t test = NULL;
6 SemaphoreHandle_t muestrea = NULL;
7
8 //Constructor de clase Timer
9 Timer :: Timer(timer_group_t timer_group , timer_idx_t timer_idx , bool
10               interrupcion , bool inicio , double timer_interval_sec){
11     //Se establecen parametros del Timer
12     this->timer_idx = timer_idx;
13     this->timer_group = timer_group;
14     this->timer_interval_sec = timer_interval_sec;
15     this->interrupcion = interrupcion;
16     this->timer_interval_sec = timer_interval_sec;
17     this->inicio = inicio;
18     timer_config();
19 }
20 //Se configura Timer
21 void Timer::timer_config(){
22
23     timer_config_t config;
24     config.divider = TIMER_DIVIDER;

```

```

25 config.counter_dir = TIMER_COUNT_UP;
26 config.counter_en = TIMER_PAUSE;
27 config.alarm_en = this->interrupcion;
28 config.intr_type = TIMER_INTR_LEVEL;
29 config.auto_reload = 1;
30
31 timer_init(this->timer_group, this->timer_idx, &config);
32
33 timer_set_counter_value(this->timer_group, this->timer_idx, 0
34 x00000000ULL);
35
36 if(this->interrupcion){
37     if(this->timer_idx == 0){
38         muestrea = xSemaphoreCreateBinary();
39         //Se crea Tarea de interrupcion para muestrear
40         xTaskCreatePinnedToCore(control, "interrupcion", 3072*3, NULL,
41         5, NULL, 1);
42     } else{
43         test = xSemaphoreCreateBinary();
44         //Se crea Tarea que activa bandera de envio de datos
45         xTaskCreatePinnedToCore(send_plot, "plot", 1024*2, NULL, 5,
46         NULL, 0);
47     }
48     timer_set_alarm_value(this->timer_group, this->timer_idx, this->
49     timer_interval_sec * TIMER_SCALE);
50     timer_enable_intr(this->timer_group, this->timer_idx);
51     timer_isr_register(this->timer_group, this->timer_idx, &(this->
52     start_timer_group0_isr), this, 0, &manejador_timer);
53 }
```

```

54
55     ESP_LOGI(TIMER_TAG, "Se inicializo el Timer , Grupo %d, Numero: %d,
56     Tiempo Interrupcion: %f s", this->timer_group, this->timer_idx,
57     this->timer_interval_sec);
58
59 //Pausa timer
60 esp_err_t Timer::pause(void){
61
62     esp_err_t error;
63
64     error = timer_pause(this->timer_group, this->timer_idx);
65
66     return error;
67 }
68
69 //Inicia Timer
70 esp_err_t Timer::timer_inicio(void){
71
72     esp_err_t error;
73
74     error = timer_start(this->timer_group, this->timer_idx);
75
76     return error;
77 }
78
79 //Manejo de interrupcion del timer
80 void IRAM_ATTR Timer::start_timer_group0_isr(void* arg)
81 {
82     reinterpret_cast<Timer*>(arg)->timer_group0_isr();
83 }
84
85 void Timer :: timer_group0_isr(){

```

```

86
87     int timer_idx = this->timer_idx;
88
89     uint32_t intr_status = TIMERG0.int_st_timers.val;
90
91     TIMERG0.hw_timer[timer_idx].update = 1;
92
93     uint64_t timer_counter_value =
94         ((uint64_t) TIMERG0.hw_timer[timer_idx].cnt_high) << 32
95         | TIMERG0.hw_timer[timer_idx].cnt_low;
96
97     timer_event_t evt;
98     evt.timer_group = this->timer_group;
99     evt.timer_idx = this->timer_idx;
100    evt.timer_counter_value = timer_counter_value;
101
102    if ((intr_status & BIT(timer_idx)) && timer_idx == TIMER_0) {
103        evt.type = TEST_WITHOUT_RELOAD;
104        TIMERG0.int_clr_timers.t0 = 1;
105        timer_counter_value += (uint64_t) (this->timer_interval_sec *
106            TIMER_SCALE);
107        TIMERG0.hw_timer[timer_idx].alarm_high = (uint32_t) (
108            timer_counter_value >> 32);
109        TIMERG0.hw_timer[timer_idx].alarm_low = (uint32_t)
110            timer_counter_value;
111    } else if ((intr_status & BIT(timer_idx)) && timer_idx == TIMER_1
112    ) {
113        evt.type = TEST_WITH_RELOAD;
114        TIMERG0.int_clr_timers.t1 = 1;
115    } else {
116        evt.type = -1;
117    }
118
119    TIMERG0.hw_timer[timer_idx].config.alarm_en = TIMER_ALARM_EN;

```

```

116
117     if (this->timer_idx == 1){
118         this->pause();
119         //Se activa semaforo para recoleccion de datos de grafica
120         xSemaphoreGiveFromISR(test, NULL);
121     } else{
122         //Se activa semaforo para muestrear
123         xSemaphoreGiveFromISR(muestrea, NULL);
124     }
125 }
```

Código 3.5. interrupcion.cpp.

```

1
2 #ifndef CONTROLADOR_H_
3 #define CONTROLADOR_H_
4
5 //Manejo de control convencional
6 #include "pid.h"
7 //Manejo de control difuso
8 #include "controlDifuso.h"
9 //Libreria para manejo del RTOS
10 #include "freertos/FreeRTOS.h"
11 #include "freertos/task.h"
12 #include "freertos/queue.h"
13 #include "freertos/semp.h"
14
15 //Variables para configurar ADC y DAC
16 static const adc_channel_t input_angulo = ADC_CHANNEL_4;
17 static const adc_channel_t input_posicion = ADC_CHANNEL_6;
18 static const dac_channel_t output_motor = DAC_CHANNEL_1;
19 static const dac_channel_t output_debug = DAC_CHANNEL_2;
20
21 //Log Control
22 static const char *CONTROL = "CONTROL TASK";
23
```

```

24 //Bandera para debug tiempo de muestreo
25 static bool bandera_debug = true;
26
27 //Funcion para debug del tiempo de muestreo
28 void control_debug();
29
30 //Tarea de control del sistema
31 void control(void *param);
32
33 //Tarea para activar bandera para envio de datos
34 void send_plot(void *param);
35
36 #endif

```

Código 3.6. controlador.h.

```

1
2 #include "controlador.h"
3
4 //Manejador de cola de datos para grafica
5 QueueHandle_t dataplot;
6
7 //Variable para control convencional
8 float Kp_1 = 7.0, Kp_2 = 12, Kd = 7;
9
10 //Set point
11 float set_point = 0.0;
12
13 //Variable para tipo de control
14 uint8_t tipoControl = 0;
15
16 //Semaforos externos accionados por Timers
17 extern SemaphoreHandle_t muestrea;
18 extern SemaphoreHandle_t test;
19
20 //instancia de clase ControlPD

```

```

21 ControladorPD ctrlPD (Kp_2, Kd, Kp_1, set_point) ;
22
23 //instancia de clase Control Difuso sin ajustar
24 Difuso controlDifuso (0) ;
25
26 //instancia de clase Control Difuso ajustado
27 Difuso controlDifusoOpt (1) ;
28
29 //instancia de clase Adc1 para medir angulo
30 Adc1 angulo_adc( input_angulo , ADC_WIDTH_BIT_12, ADC_ATTEN_DB_11,
   NO_OF_SAMPLES) ;
31
32 //instancia de clase Dac para salida a motor
33 Dac motor_dac(output_motor) ;
34
35 //instancia de clase Adc1 para medir posicion
36 Adc1 posicion_adc(input_posicion , ADC_WIDTH_BIT_12, ADC_ATTEN_DB_11,
   NO_OF_SAMPLES) ;
37
38 //instancia de clase Dac para debug de timepo de muestreo
39 Dac debug_dac(output_debug) ;
40
41 //Variable booleana para envio de datos para grafica
42 bool plot = false ;
43
44 //Tarea para activar bandera para envio de datos
45 void send_plot(void *param){
46
47   while(1){
48
49     if( xSemaphoreTake( test , portMAX_DELAY) == pdTRUE ){
50       //Se limpia bandera
51       plot = ! plot ;
52       ESP_LOGI("PLOT","Datos enviados");

```

```

53     }
54
55   }
56   vTaskDelete(NULL);
57
58 }
59
60 //Funcion para debug del tiempo de muestreo
61 void control_debug(){
62
63   //Genera funcion cuadrada
64   if(bandera_debug){
65     debug_dac.dac_output(127);
66   }else{
67     debug_dac.dac_output(0);
68   }
69   bandera_debug = !bandera_debug;
70
71 }
72
73 //Tarea de control del sistema
74 void control(void *parama){
75   ESP_LOGI(CONTROL, "Inicio tarea de Control");
76   float posicion, angulo;
77   uint16_t posicion_value, angulo_value;
78   uint8_t salida = 0;
79   dataplot = xQueueCreate(5,2);
80   uint16_t count = 0;
81   while(1){
82     if( xSemaphoreTake( muestrea , portMAX_DELAY) == pdTRUE ){
83
84       //Muestra en 4095-0
85       posicion_value = (uint16_t) posicion_adc.get_muestra();
86       angulo_value = (uint16_t) angulo_adc.get_muestra();

```

```

87
88     // Escalar en un rango 3.3-0 V
89     posicion = posicion_value*3.3/4095;
90     angulo = angulo_value*3.3/4095;
91
92     // Elige tipo de controlador
93     switch(tipoControl){
94
95         case 2:
96             // Asigna a salida controlador PD
97             salida = (uint8_t)ctrlPD.controlPD(posicion,angulo, set_point);
98             break;
99
100        case 0:
101            // Asigna a salida controlador difuso sin ajustar
102            salida = (uint8_t)controlDifuso.output(posicion,angulo, set_point);
103            break;
104
105        case 1:
106            // Asigna a salida controlador difuso ajustado
107            salida = (uint8_t)controlDifusoOpt.output(posicion,angulo, set_point);
108            break;
109
110        default:
111            salida = 127;
112            break;
113
114    }
115
116    if(plot){
117        // Envio de datos para grafica

```

```

118     xQueueSendToFront ( dataplot ,& posicion_value ,pdMS_TO_TICKS
119     (1000)) ;
120
121     // Asigna salida
122     motor_dac.dac_output ( salida ) ;
123     // Debug tiempo de muestreo
124     control_debug () ;
125
126 }
127 vTaskDelete (NULL) ;
128 }
```

Código 3.7. controlador.cpp.

```

1
2 #ifndef CONTROLADOR_PID_H_
3 #define CONTROLADOR_PID_H_
4
5 // Se define tiempo de muestreo para calculo de componente derivativa
6 static float sample_time = 0.012;
7
8 // Log PID
9 static const char *PID = "PID";
10
11 // Clase control PD
12 class ControladorPD{
13
14 public:
15
16     ControladorPD ( int Kp_p, int Kd_p, int Kp_a, float set_point );
17     int Kp_p, Kd_p, Kp_a;
18     void set_Kp_p ( int valor );
19     void set_Kd_p ( int valor );
20     void set_Kp_a ( int valor );
21     float controlPD ( float posicion, float angulo, float set_point );
```

```

22
23 private:
24     float error_posicion[2], output[2], dac_value, error_angulo,
25     posicion_value, angulo_value, set_angulo;
26 };
27
28 #endif

```

Código 3.8. pid.h.

```

1
2 #include "pid.h"
3
4 // Constructor para clase Controlador PD
5 ControladorPD::ControladorPD(int Kp_p, int Kd_p, int Kp_a, float
6     consigna){
7
8     this->Kp_p = Kp_p;
9     this->Kd_p = Kd_p;
10    this->Kp_a = Kp_a;
11    this->error_posicion[0] = 0;
12    this->error_posicion[1] = 0;
13    this->output[0] = 0;
14    this->output[1] = 0;
15    this->dac_value = 0;
16    this->error_angulo = 0;
17    this->posicion_value = 0;
18    this->angulo_value = 0;
19    this->set_angulo = 0;
20
21
22
23 }
24

```

```

25 //Set de variable Kp para control de posicion
26 void ControladorPD::set_Kp_p(int valor){
27
28     this->Kp_p = valor;
29     ESP_LOGI(PID, "Cambio Kp_p %d", this->Kp_p);
30
31 }
32
33 //Set de variable Kd para control de posicion
34 void ControladorPD::set_Kd_p(int valor){
35
36     this->Kd_p = valor;
37     ESP_LOGI(PID, "Cambio Kd_p %d", this->Kd_p);
38
39 }
40
41 //Set de variable Kp para control de angulo
42 void ControladorPD::set_Kp_a(int valor){
43
44     this->Kp_a = valor;
45     ESP_LOGI(PID, "Cambio Kp_a %d", this->Kp_a);
46
47 }
48
49 //Funcion de control
50 float ControladorPD::controlPD(float posicion, float angulo, float
51 set_point){
52
53     this->error_posicion[0] = this->error_posicion[1];
54
55     posicion = posicion * (0.13468) - 0.2222;
56
57     this->error_posicion[1] = set_point - posicion;

```

```

58     this->output[0] = this->output[1];
59
60     // Filtro para derivada
61     this->output[1] = 0.314 * this->error_posicion[0] + this->output[0]
62     * 0.686;
63
64     if (this->output[1] < -0.76){
65         this->output[1] = -0.76;
66     }
67     if (this->output[1] > 0.76){
68         this->output[1] = 0.76;
69     }
70
71
72
73     angulo = (angulo * (-0.6058) + 1.0904);
74
75     this->error_angulo = this->set_angulo - angulo;
76
77     // Angulo de offset
78     if (this->error_angulo < 0){
79         this->error_angulo -= 0.1745;
80     } else{
81         this->error_angulo += 0.0175;
82     }
83
84     if (this->error_angulo < -0.76){
85         this->error_angulo = -0.76;
86     }
87     if (this->error_angulo > 0.76){
88         this->error_angulo = 0.76;
89     }
90

```

```

91     this->dac_value = this->Kp_a*this->error_angulo;
92
93     if (this->dac_value < -5.0) {
94         this->dac_value = -5.0;
95     }
96     if (this->dac_value > 5.0) {
97         this->dac_value = 5.0;
98     }
99
100    this->dac_value = -(this->dac_value / 10.0) + 0.5;
101    this->dac_value = this->dac_value * 255.0;
102
103    return this->dac_value;
104 }
```

Código 3.9. pid.cpp.

```

1
2 #ifndef DIFUSO_DEF
3 #define DIFUSO_DEF
4 #include "adc.h"
5 #include <Fuzzy.h>
6
7 // Clase Difuso
8 class Difuso{
9     public:
10         Difuso(bool opti);
11         float output(float posicion, float angulo, float set_point);
12
13
14     private:
15
16         float error_pos[2];
17         float error_dev_pos[2];
18         float error_ang[2];
19         float error_dev_ang[2];
```

```

20
21      //se crean dos sistemas difuso para el control de posicion y
22      //angulo
23
24      Fuzzy *controlPos = new Fuzzy();
25      Fuzzy *controlAngulo = new Fuzzy();
26
27      //Funciones de pertenencia para posicion
28
29      FuzzySet *MDR = new FuzzySet(-0.2, -0.2, -0.15, -0.1, 0, 1,
30      1);
31      FuzzySet *DR = new FuzzySet(-0.15, -0.1, -0.05, -0.03, 0, 1,
32      1);
33      FuzzySet *pdr = new FuzzySet(-0.05, -0.03, -0.01, 0, 0, 1, 1)
34      ;
35      FuzzySet *pdro = new FuzzySet(-0.05, -0.03, -0.01, 0, 0, 1,
36      2);
37      FuzzySet *cp = new FuzzySet(-0.0099, 0, 0, 0.0099, 0, 1, 1);
38      FuzzySet *cpo = new FuzzySet(-0.01, 0, 0, 0.01, 0, 1, 1);
39      FuzzySet *piz = new FuzzySet(0, 0.01, 0.03, 0.05, 0, 1, 1);
40      FuzzySet *pizo = new FuzzySet(0, 0.01, 0.03, 0.05, 0, 1, 2);
41      FuzzySet *IZ = new FuzzySet(0.03, 0.05, 0.10, 0.15, 0, 1, 1);
42      FuzzySet *MIZ = new FuzzySet(0.1, 0.15, 0.2, 0.2, 0, 1, 1);
43
44      FuzzySet *PRP = new FuzzySet(-0.2, -0.2, -0.12, -0.04, 0, 1,
1);
45      FuzzySet *plp = new FuzzySet(-0.08, -0.04, -0.04, 0, 0, 1, 1)
46      ;
47      FuzzySet *plpo = new FuzzySet(-0.08, -0.04124, -0.03876, 0,
48      0.003906, 1, 1.031);
49      FuzzySet *CDP = new FuzzySet(-0.04, 0, 0, 0.04, 0, 1, 1);
50      FuzzySet *nlp = new FuzzySet( 0, 0.03876, 0.04124, 0.08,
51      0.003906, 1, 1.031);
52      FuzzySet *nlpo = new FuzzySet( 0, 0.04, 0.04, 0.08, 0, 1, 1);

```

```

45     FuzzySet *NRP = new FuzzySet( 0.04, 0.12, 0.2, 0.2, 0, 1, 1);
46
47     FuzzySet *MHP = new FuzzySet(-1.125, -0.7875, -0.7875, -0.45,
48                                     0, 1, 1);
49     FuzzySet *HP = new FuzzySet( -0.7875, -0.45, -0.45, -0.2, 0,
50                                   1, 1);
51     FuzzySet *PHP = new FuzzySet( -0.4, -0.2, -0.2, 0, 0, 1, 1);
52     FuzzySet *CAP = new FuzzySet(-0.2, 0, 0, 0.2, 0, 1, 1);
53     FuzzySet *PAP = new FuzzySet( 0, 0.2, 0.2, 0.4, 0, 1, 1);
54     FuzzySet *AP = new FuzzySet(0.2, 0.45, 0.45, 0.7875, 0, 1, 1)
55     ;
56
57 //Funciones de pertenencia para angulo
58
59     FuzzySet *muyAbajo = new FuzzySet
60     (-0.785,-0.785,-0.5236,-0.1745, 0, 1, 1);
61     FuzzySet *abajo = new FuzzySet(-0.349, -0.1745, -0.1745,
62                                     -0.01745, 0, 1, 1);
63     FuzzySet *cero_ang = new FuzzySet(-0.08, 0, 0, 0.08, 0, 1, 1)
64     ;
65     FuzzySet *arriba = new FuzzySet(0.01745, 0.1745, 0.1745,
66                                     0.349, 0, 1, 1);
67     FuzzySet *muyArriba = new FuzzySet(0.1745, 0.5235, 0.785,
68                                     0.785, 0, 1, 1);
69
70     FuzzySet *posRapido = new FuzzySet(-3, -3, -1.8, -1.2, 0, 1,
71                                       1);
72     FuzzySet *posLento = new FuzzySet(-1.8, -0.9, -0.9, 0, 0, 1,
73                                       1);
74     FuzzySet *cero_dev_ang = new FuzzySet(-0.3, 0, 0, 0.3, 0, 1,
75                                           1);
76     FuzzySet *negLento = new FuzzySet(0, 0.9, 0.9, 1.8, 0, 1, 1);

```

```

67     FuzzySet *negRapido = new FuzzySet(1.2, 1.8, 3, 3, 0, 1, 1);
68
69     FuzzySet *horaRapido = new FuzzySet(-6.2, -4.96, -4.96,
70 -3.72, 0, 1, 1);
71     FuzzySet *horario = new FuzzySet(-3.707, -2.467, -2.467,
72 -1.227, 0, 1, 1);
73     FuzzySet *cero_pwm = new FuzzySet(-0.62, 0, 0, 0.62, 0, 1, 1)
74 ;
75     FuzzySet *antiH = new FuzzySet(1.24, 2.48, 2.48, 3.72, 0, 1,
76 1);
77     FuzzySet *antiHRapido = new FuzzySet(3.72, 4.96, 4.96, 6.2,
78 0, 1, 1);
79
80     void init_pos(FuzzySet *PDR, FuzzySet *CP, FuzzySet *PIZ,
81 FuzzySet *PLP, FuzzySet *NLP);
82
83     void init_ang();
84 };
85
86
87 #endif

```

Código 3.10. controlDifuso.h.

```

1
2 #include "include/controlDifuso.h"
3
4 //Contructor de clase Difuso
5 Difuso::Difuso(bool opti){
6
7     if(opti){
8         //Se crea el control difuso ajustado
9         ESP_LOGI("FUZZY", "Inicio Control Difuso Ajustado");
10        init_pos(this->pdro, this->cpo, this->pizo, this->plpo, this->nupo);
11
12    } else{
13

```

```

14     init_pos(this->pdr, this->cp, this->piz, this->plp, this->nlp);
15     ESP_LOGI("FUZZY", "Inicio Control Difuso");
16
17 }
18
19 init_ang();
20
21
22 }
23
24 // inicia control difuso para la posicion
25 void Difuso::init_pos(FuzzySet *PDR, FuzzySet *CP, FuzzySet *PIZ,
26                       FuzzySet *PLP, FuzzySet *NLP){
27
28     /******Control de Posicion******/
29
30     FuzzyInput *EP = new FuzzyInput(1);
31
32     EP->addFuzzySet(MIZ);
33     EP->addFuzzySet(IZ);
34     EP->addFuzzySet(PIZ);
35     EP->addFuzzySet(CP);
36     EP->addFuzzySet(PDR);
37     EP->addFuzzySet(DR);
38     EP->addFuzzySet(MDR);
39     controlPos->addFuzzyInput(EP);
40
41     FuzzyInput *EDP = new FuzzyInput(2);
42
43     EDP->addFuzzySet(NRP);
44     EDP->addFuzzySet(NLP);
45     EDP->addFuzzySet(CDP);
46     EDP->addFuzzySet(PLP);
47     EDP->addFuzzySet(PR);

```

```

47 controlPos->addFuzzyInput (EDP) ;
48
49 FuzzyOutput *ADP = new FuzzyOutput (1) ;
50
51 ADP->addFuzzySet (MHP) ;
52 ADP->addFuzzySet (HP) ;
53 ADP->addFuzzySet (PHP) ;
54 ADP->addFuzzySet (CAP) ;
55 ADP->addFuzzySet (PAP) ;
56 ADP->addFuzzySet (AP) ;
57 ADP->addFuzzySet (MAP) ;
58 controlPos->addFuzzyOutput (ADP) ;
59
60 FuzzyRuleAntecedent *ifMuyIzquierdaAndNegRapido = new
   FuzzyRuleAntecedent () ;
61 ifMuyIzquierdaAndNegRapido->joinWithAND (MIZ, NRP) ;
62
63 FuzzyRuleAntecedent *ifMuyIzquierdaAndNegLento = new
   FuzzyRuleAntecedent () ;
64 ifMuyIzquierdaAndNegLento->joinWithAND (MIZ, NLP) ;
65
66 FuzzyRuleAntecedent *ifMuyIzquierdaAndCero_dev_pos = new
   FuzzyRuleAntecedent () ;
67 ifMuyIzquierdaAndCero_dev_pos->joinWithAND (MIZ, CDP) ;
68
69 FuzzyRuleAntecedent *ifMuyIzquierdaAndPosLento = new
   FuzzyRuleAntecedent () ;
70 ifMuyIzquierdaAndPosLento->joinWithAND (MIZ, PLP) ;
71
72 FuzzyRuleAntecedent *ifMuyIzquierdaAndPosRapido = new
   FuzzyRuleAntecedent () ;
73 ifMuyIzquierdaAndPosRapido->joinWithAND (MIZ, PRP) ;
74
75

```

```

76
77 FuzzyRuleAntecedent *ifIzquierdaAndNegRapido = new
78     FuzzyRuleAntecedent () ;
79 ifIzquierdaAndNegRapido->joinWithAND (IZ , NRP) ;
80
81 FuzzyRuleAntecedent *ifIzquierdaAndNegLento = new
82     FuzzyRuleAntecedent () ;
83 ifIzquierdaAndNegLento->joinWithAND (IZ , NLP) ;
84
85 FuzzyRuleAntecedent *ifIzquierdaAndCero_dev_pos = new
86     FuzzyRuleAntecedent () ;
87 ifIzquierdaAndCero_dev_pos->joinWithAND (IZ , CDP) ;
88
89 FuzzyRuleAntecedent *ifIzquierdaAndPosLento = new
90     FuzzyRuleAntecedent () ;
91 ifIzquierdaAndPosLento->joinWithAND (IZ , PLP) ;
92
93
94 FuzzyRuleAntecedent *ifPocoIzquierdaAndNegRapido = new
95     FuzzyRuleAntecedent () ;
96 ifPocoIzquierdaAndNegRapido->joinWithAND (PIZ , NRP) ;
97
98 FuzzyRuleAntecedent *ifPocoIzquierdaAndNegLento = new
99     FuzzyRuleAntecedent () ;
100 ifPocoIzquierdaAndNegLento->joinWithAND (PIZ , NLP) ;
101 FuzzyRuleAntecedent *ifPocoIzquierdaAndCero_dev_pos = new
102     FuzzyRuleAntecedent () ;
103 ifPocoIzquierdaAndCero_dev_pos->joinWithAND (PIZ , CDP) ;

```

```

102
103 FuzzyRuleAntecedent *ifPocoIzquierdaAndPosLento = new
104     FuzzyRuleAntecedent () ;
105 ifPocoIzquierdaAndPosLento->joinWithAND (PIZ , PLP) ;
106
107 FuzzyRuleAntecedent *ifPocoIzquierdaAndPosRapido = new
108     FuzzyRuleAntecedent () ;
109 ifPocoIzquierdaAndPosRapido->joinWithAND (PIZ , PRP) ;
110
111
112 FuzzyRuleAntecedent *ifCero_posAndNegRapido = new
113     FuzzyRuleAntecedent () ;
114 ifCero_posAndNegRapido->joinWithAND (CP , NRP) ;
115
116 FuzzyRuleAntecedent *ifCero_posAndNegLento = new
117     FuzzyRuleAntecedent () ;
118 ifCero_posAndNegLento->joinWithAND (CP , NLP) ;
119
120 FuzzyRuleAntecedent *ifCero_posAndCero_dev_pos = new
121     FuzzyRuleAntecedent () ;
122 ifCero_posAndCero_dev_pos->joinWithAND (CP , CDP) ;
123
124 FuzzyRuleAntecedent *ifCero_posAndPosLento = new
125     FuzzyRuleAntecedent () ;
126 ifCero_posAndPosLento->joinWithAND (CP , PLP) ;
127
128 FuzzyRuleAntecedent *ifPocoDerechaAndNegRapido = new

```

```

    FuzzyRuleAntecedent () ;
129 ifPocoDerechaAndNegRapido->joinWithAND (PDR, NRP) ;

130
131 FuzzyRuleAntecedent *ifPocoDerechaAndNegLento = new
    FuzzyRuleAntecedent () ;
132 ifPocoDerechaAndNegLento->joinWithAND (PDR, NLP) ;

133
134 FuzzyRuleAntecedent *ifPocoDerechaAndCero_dev_pos = new
    FuzzyRuleAntecedent () ;
135 ifPocoDerechaAndCero_dev_pos->joinWithAND (PDR, CDP) ;

136
137 FuzzyRuleAntecedent *ifPocoDerechaAndPosLento = new
    FuzzyRuleAntecedent () ;
138 ifPocoDerechaAndPosLento->joinWithAND (PDR, PLP) ;

139
140 FuzzyRuleAntecedent *ifPocoDerechaAndPosRapido = new
    FuzzyRuleAntecedent () ;
141 ifPocoDerechaAndPosRapido->joinWithAND (PDR, PRP) ;

142
143
144
145 FuzzyRuleAntecedent *ifDerechaAndNegRapido = new
    FuzzyRuleAntecedent () ;
146 ifDerechaAndNegRapido->joinWithAND (DR, NRP) ;

147
148 FuzzyRuleAntecedent *ifDerechaAndNegLento = new FuzzyRuleAntecedent
    () ;
149 ifDerechaAndNegLento->joinWithAND (DR, NLP) ;

150
151 FuzzyRuleAntecedent *ifDerechaAndCero_dev_pos = new
    FuzzyRuleAntecedent () ;
152 ifDerechaAndCero_dev_pos->joinWithAND (DR, CDP) ;

153
154 FuzzyRuleAntecedent *ifDerechaAndPosLento = new FuzzyRuleAntecedent

```

```

() ;

155 if DerechaAndPosLento->joinWithAND (DR, PLP) ;

156

157 FuzzyRuleAntecedent *ifDerechaAndPosRapido = new
    FuzzyRuleAntecedent () ;
158 if DerechaAndPosRapido->joinWithAND (DR, PRP) ;

159

160

161

162 FuzzyRuleAntecedent *ifMuyDerechaAndNegRapido = new
    FuzzyRuleAntecedent () ;
163 if MuyDerechaAndNegRapido->joinWithAND (MDR, NRP) ;

164

165 FuzzyRuleAntecedent *ifMuyDerechaAndNegLento = new
    FuzzyRuleAntecedent () ;
166 if MuyDerechaAndNegLento->joinWithAND (MDR, NLP) ;

167

168 FuzzyRuleAntecedent *ifMuyDerechaAndCero_dev_pos = new
    FuzzyRuleAntecedent () ;
169 if MuyDerechaAndCero_dev_pos->joinWithAND (MDR, CDP) ;

170

171 FuzzyRuleAntecedent *ifMuyDerechaAndPosLento = new
    FuzzyRuleAntecedent () ;
172 if MuyDerechaAndPosLento->joinWithAND (MDR, PLP) ;

173

174 FuzzyRuleAntecedent *ifMuyDerechaAndPosRapido = new
    FuzzyRuleAntecedent () ;
175 if MuyDerechaAndPosRapido->joinWithAND (MDR, PRP) ;

176

177

178

179

180 FuzzyRuleConsequent *thenMaxHora = new FuzzyRuleConsequent () ;
181 thenMaxHora->addOutput (MHP) ;

```

```

182
183 FuzzyRuleConsequent *thenHora = new FuzzyRuleConsequent() ;
184 thenHora->addOutput (HP) ;
185
186 FuzzyRuleConsequent *thenMinHora = new FuzzyRuleConsequent() ;
187 thenMinHora->addOutput (PHP) ;
188
189 FuzzyRuleConsequent *thenCero_ang_des = new FuzzyRuleConsequent() ;
190 thenCero_ang_des->addOutput (CAP) ;
191
192 FuzzyRuleConsequent *thenMinAntiH = new FuzzyRuleConsequent() ;
193 thenMinAntiH->addOutput (PAP) ;
194
195 FuzzyRuleConsequent *thenAntiHora = new FuzzyRuleConsequent() ;
196 thenAntiHora->addOutput (AP) ;
197
198 FuzzyRuleConsequent *thenMaxAntiH = new FuzzyRuleConsequent() ;
199 thenMaxAntiH->addOutput (MAP) ;
200
201 // if muyIzquierda and negRapido_pos then maxHora
202
203 FuzzyRule *fuzzyRule1_pos = new FuzzyRule(1 ,
204     ifMuyIzquierdaAndNegRapido , thenMaxHora) ;
205 controlPos->addFuzzyRule(fuzzyRule1_pos) ;
206
207 // if muyIzquierda and negLento_pos then maxHora
208
209 FuzzyRule *fuzzyRule2_pos = new FuzzyRule(2 ,
210     ifMuyIzquierdaAndNegLento , thenMaxHora) ;
211 controlPos->addFuzzyRule(fuzzyRule2_pos) ;
212
213 // if muyIzquierda and cero_dev_pos then maxHora
214
215 FuzzyRule *fuzzyRule3_pos = new FuzzyRule(3 ,

```

```

    ifMuyIzquierdaAndCero_dev_pos , thenMaxHora) ;
214 controlPos->addFuzzyRule(fuzzyRule3_pos) ;

215
216 // if muyIzquierda and posLento_pos then Hora
217
218 FuzzyRule *fuzzyRule4_pos = new FuzzyRule(4 ,
219     ifMuyIzquierdaAndPosLento , thenMaxHora) ;
220 controlPos->addFuzzyRule(fuzzyRule4_pos) ;

221
222 // if muyIzquierda and posRapido_pos then maxHora
223
224 FuzzyRule *fuzzyRule5_pos = new FuzzyRule(5 ,
225     ifMuyIzquierdaAndPosRapido , thenHora) ;
226 controlPos->addFuzzyRule(fuzzyRule5_pos) ;

227
228 // if izquierda and negRapido_pos then maxHora
229
230 FuzzyRule *fuzzyRule6_pos = new FuzzyRule(6 ,
231     ifIzquierdaAndNegRapido , thenMaxHora) ;
232 controlPos->addFuzzyRule(fuzzyRule6_pos) ;

233
234 // if izquierda and negLento_pos then maxHora
235
236 FuzzyRule *fuzzyRule7_pos = new FuzzyRule(7 , ifIzquierdaAndNegLento
237     , thenMaxHora) ;
238 controlPos->addFuzzyRule(fuzzyRule7_pos) ;

239
240 // if izquierda and cero_dev_pos then maxHora
241
242 FuzzyRule *fuzzyRule8_pos = new FuzzyRule(8 ,
243     ifIzquierdaAndCero_dev_pos , thenHora) ;
244 controlPos->addFuzzyRule(fuzzyRule8_pos) ;

245
246 // if izquierda and posLento_pos then cero_ang_des

```

```

242
243 FuzzyRule *fuzzyRule9_pos = new FuzzyRule(9, ifIzquierdaAndPosLento
244 , thenCero_ang_des);
controlPos->addFuzzyRule(fuzzyRule9_pos);

245
246 // if izquierda and posRapido_pos then antiHora
247

248 FuzzyRule *fuzzyRule10_pos = new FuzzyRule(10,
249 ifIzquierdaAndPosRapido , thenAntiHora);
controlPos->addFuzzyRule(fuzzyRule10_pos);

250
251 // if cero_pos and negRapido_pos then maxHora
252

253 FuzzyRule *fuzzyRule11_pos = new FuzzyRule(11,
254 ifCero_posAndNegRapido , thenHora);
controlPos->addFuzzyRule(fuzzyRule11_pos);

255
256 // if cero_pos and negLento_pos then hora
257

258 FuzzyRule *fuzzyRule12_pos = new FuzzyRule(12 ,
259 ifCero_posAndNegLento , thenCero_ang_des);
controlPos->addFuzzyRule(fuzzyRule12_pos);

260
261 // if cero_pos and cero_dev_pos then cero_ang_des
262

263 FuzzyRule *fuzzyRule13_pos = new FuzzyRule(13 ,
264 ifCero_posAndCero_dev_pos , thenCero_ang_des);
controlPos->addFuzzyRule(fuzzyRule13_pos);

265
266 // if cero_pos and posLento_pos then antiHora
267

268 FuzzyRule *fuzzyRule14_pos = new FuzzyRule(14 ,
269 ifCero_posAndPosLento , thenCero_ang_des);
controlPos->addFuzzyRule(fuzzyRule14_pos);

```

```

270
271 // if cero_pos and posRapido_pos then maxAntiH
272
273 FuzzyRule *fuzzyRule15_pos = new FuzzyRule(15,
274     ifCero_posAndPosRapido, thenAntiHora);
275 controlPos->addFuzzyRule(fuzzyRule15_pos);

276 // if derecha and negRapido_pos then Hora
277
278 FuzzyRule *fuzzyRule16_pos = new FuzzyRule(16,
279     ifDerechaAndNegRapido, thenHora);
280 controlPos->addFuzzyRule(fuzzyRule16_pos);

281 // if derecha and negLento_pos then cero_ang_des
282
283 FuzzyRule *fuzzyRule17_pos = new FuzzyRule(17, ifDerechaAndNegLento
284     , thenCero_ang_des);
285 controlPos->addFuzzyRule(fuzzyRule17_pos);

286 // if derecha and cero_dev_pos then antiHora
287
288 FuzzyRule *fuzzyRule18_pos = new FuzzyRule(18,
289     ifDerechaAndCero_dev_pos, thenAntiHora);
290 controlPos->addFuzzyRule(fuzzyRule18_pos);

291 // if derecha and posLento_pos then maxAntiHora
292
293 FuzzyRule *fuzzyRule19_pos = new FuzzyRule(19, ifDerechaAndPosLento
294     , thenMaxAntiH);
295 controlPos->addFuzzyRule(fuzzyRule19_pos);

296 // if derecha and posRapido_pos then maxAntiHora
297
298 FuzzyRule *fuzzyRule20_pos = new FuzzyRule(20,

```

```

    ifDerechaAndPosRapido , thenMaxAntiH) ;
299 controlPos->addFuzzyRule(fuzzyRule20_pos) ;

300
301 // if muyDerecha and negRapido_pos then antiHora
302
303 FuzzyRule *fuzzyRule21_pos = new FuzzyRule(21 ,
304     ifMuyDerechaAndNegRapido , thenAntiHora) ;
305 controlPos->addFuzzyRule(fuzzyRule21_pos) ;

306
307 // if muyDerecha and negLento_pos then maxAntiHora
308
309 FuzzyRule *fuzzyRule22_pos = new FuzzyRule(22 ,
310     ifMuyDerechaAndNegLento , thenMaxAntiH) ;
311 controlPos->addFuzzyRule(fuzzyRule22_pos) ;

312
313 // if muyDerecha and cero_dev_pos then maxAntiHora
314
315 FuzzyRule *fuzzyRule23_pos = new FuzzyRule(23 ,
316     ifMuyDerechaAndCero_dev_pos , thenMaxAntiH) ;
317 controlPos->addFuzzyRule(fuzzyRule23_pos) ;

318
319 // if muyDerecha and posLento_pos then maxAntiHora
320
321 FuzzyRule *fuzzyRule24_pos = new FuzzyRule(24 ,
322     ifMuyDerechaAndPosLento , thenMaxAntiH) ;
323 controlPos->addFuzzyRule(fuzzyRule24_pos) ;

324
325 // if muyDerecha and posRapido_pos then maxAntiHora
326
327 FuzzyRule *fuzzyRule25_pos = new FuzzyRule(25 ,
328     ifMuyDerechaAndPosRapido , thenMaxAntiH) ;
329 controlPos->addFuzzyRule(fuzzyRule25_pos) ;

330
331 // if pocoIzquierda and negRapido_pos then maxHora

```

```

327
328 FuzzyRule *fuzzyRule26_pos = new FuzzyRule(26 ,
329     ifPocoIzquierdaAndNegRapido , thenMaxHora) ;
330 controlPos->addFuzzyRule(fuzzyRule26_pos) ;
331
332 // if pocoIzquierda and negLento_pos then maxHora
333
334 FuzzyRule *fuzzyRule27_pos = new FuzzyRule(27 ,
335     ifPocoIzquierdaAndNegLento , thenHora) ;
336 controlPos->addFuzzyRule(fuzzyRule27_pos) ;
337
338 // if pocoIzquierda and cero_dev_pos then horario
339
340 FuzzyRule *fuzzyRule28_pos = new FuzzyRule(28 ,
341     ifPocoIzquierdaAndCero_dev_pos , thenMinHora) ;
342 controlPos->addFuzzyRule(fuzzyRule28_pos) ;
343
344 // if pocoIzquierda and posLento_pos then pocoAnti
345
346 FuzzyRule *fuzzyRule29_pos = new FuzzyRule(29 ,
347     ifPocoIzquierdaAndPosLento , thenMinAntiH) ;
348 controlPos->addFuzzyRule(fuzzyRule29_pos) ;
349
350
351 // if pocoIzquierda and posRapido_pos then maxAntiH
352
353 FuzzyRule *fuzzyRule30_pos = new FuzzyRule(30 ,
354     ifPocoIzquierdaAndPosRapido , thenAntiHora) ;
355 controlPos->addFuzzyRule(fuzzyRule30_pos) ;
356
357
358 // if pocoDerecha and negRapido_pos then maxAntiH
359
360 FuzzyRule *fuzzyRule31_pos = new FuzzyRule(31 ,

```

```

356     ifPocoDerechaAndNegRapido , thenHora) ;
357     controlPos->addFuzzyRule(fuzzyRule31_pos) ;
358
359 // if pocoDerecha and negLento_pos then maxAntiH
360
361 FuzzyRule *fuzzyRule32_pos = new FuzzyRule(32 ,
362     ifPocoDerechaAndNegLento , thenMinHora) ;
363     controlPos->addFuzzyRule(fuzzyRule32_pos) ;
364
365 // if pocoDerecha and cero_dev_pos then antihora
366
367 FuzzyRule *fuzzyRule33_pos = new FuzzyRule(33 ,
368     ifPocoDerechaAndCero_dev_pos , thenMinAntiH) ;
369     controlPos->addFuzzyRule(fuzzyRule33_pos) ;
370
371 // if pocoDerecha and posLento_pos then pocoHora
372
373 FuzzyRule *fuzzyRule34_pos = new FuzzyRule(34 ,
374     ifPocoDerechaAndPosLento , thenAntiHora) ;
375     controlPos->addFuzzyRule(fuzzyRule34_pos) ;
376
377 // if pocoDerecha and posRapido_pos then maxHora
378
379 FuzzyRule *fuzzyRule35_pos = new FuzzyRule(35 ,
380     ifPocoDerechaAndPosRapido , thenMaxAntiH) ;
381     controlPos->addFuzzyRule(fuzzyRule35_pos) ;
382
383 } // inicia control difuso para el angulo
384 void Difuso::init_ang() {
385
386 /******Control de Angulo******/
387

```

```

385 FuzzyInput *err_angulo = new FuzzyInput(1);
386
387 err_angulo->addFuzzySet(muyArriba);
388 err_angulo->addFuzzySet(arriba);
389 err_angulo->addFuzzySet(cero_ang);
390 err_angulo->addFuzzySet(abajo);
391 err_angulo->addFuzzySet(muyAbajo);
392 controlAngulo->addFuzzyInput(err_angulo);
393
394 FuzzyInput *err_dev_ang = new FuzzyInput(2);
395
396 err_dev_ang->addFuzzySet(posRapido);
397 err_dev_ang->addFuzzySet(posLento);
398 err_dev_ang->addFuzzySet(cero_dev_ang);
399 err_dev_ang->addFuzzySet(negLento);
400 err_dev_ang->addFuzzySet(negRapido);
401 controlAngulo->addFuzzyInput(err_dev_ang);
402
403 FuzzyOutput *pwm_fuzzy = new FuzzyOutput(1);
404
405 pwm_fuzzy->addFuzzySet(horaRapido);
406 pwm_fuzzy->addFuzzySet(horario);
407 pwm_fuzzy->addFuzzySet(cero_pwm);
408 pwm_fuzzy->addFuzzySet(antiH);
409 pwm_fuzzy->addFuzzySet(antiHRapido);
410 controlAngulo->addFuzzyOutput(pwm_fuzzy);
411
412 FuzzyRuleAntecedent *ifMuyArribaAndPosRapido = new
413 FuzzyRuleAntecedent();
414 ifMuyArribaAndPosRapido->joinWithAND(muyArriba, posRapido);
415
416 FuzzyRuleAntecedent *ifMuyArribaAndPosLento = new
417 FuzzyRuleAntecedent();
418 ifMuyArribaAndPosLento->joinWithAND(muyArriba, posLento);

```

```

417
418 FuzzyRuleAntecedent *ifMuyArribaAndCero_dev_ang = new
419   FuzzyRuleAntecedent () ;
420 ifMuyArribaAndCero_dev_ang->joinWithAND (muyArriba , cero_dev_ang) ;
421
422 FuzzyRuleAntecedent *ifMuyArribaAndNegLento = new
423   FuzzyRuleAntecedent () ;
424 ifMuyArribaAndNegLento->joinWithAND (muyArriba , negLento) ;
425
426
427 FuzzyRuleAntecedent *ifMuyArribaAndNegRapido = new
428   FuzzyRuleAntecedent () ;
429 ifMuyArribaAndNegRapido->joinWithAND (muyArriba , negRapido) ;
430
431
432 FuzzyRuleAntecedent *ifArribaAndPosRapido = new FuzzyRuleAntecedent
433   () ;
434 ifArribaAndPosRapido->joinWithAND (arriba , posRapido) ;
435
436 FuzzyRuleAntecedent *ifArribaAndPosLento = new FuzzyRuleAntecedent
437   () ;
438 ifArribaAndPosLento->joinWithAND (arriba , posLento) ;
439
440 FuzzyRuleAntecedent *ifArribaAndCero_dev_ang = new FuzzyRuleAntecedent
441   () ;
442 ifArribaAndCero_dev_ang->joinWithAND (arriba , cero_dev_ang) ;

```

```

443
444
445 FuzzyRuleAntecedent *ifCero_angAndPosRapido = new
446     FuzzyRuleAntecedent ();
447     ifCero_angAndPosRapido->joinWithAND(cero_ang, posRapido);
448
449 FuzzyRuleAntecedent *ifCero_angAndPosLento = new
450     FuzzyRuleAntecedent ();
451     ifCero_angAndPosLento->joinWithAND(cero_ang, posLento);
452
453 FuzzyRuleAntecedent *ifCero_angAndCero_dev_ang = new
454     FuzzyRuleAntecedent ();
455     ifCero_angAndCero_dev_ang->joinWithAND(cero_ang, cero_dev_ang);
456
457 FuzzyRuleAntecedent *ifCero_angAndNegLento = new
458     FuzzyRuleAntecedent ();
459     ifCero_angAndNegLento->joinWithAND(cero_ang, negLento);
460
461 FuzzyRuleAntecedent *ifCero_angAndNegRapido = new
462     FuzzyRuleAntecedent ();
463     ifCero_angAndNegRapido->joinWithAND(cero_ang, negRapido);
464
465 FuzzyRuleAntecedent *ifAbajoAndPosRapido = new FuzzyRuleAntecedent()
466     ;
467     ifAbajoAndPosRapido->joinWithAND(abajo, posRapido);
468
469 FuzzyRuleAntecedent *ifAbajoAndPosLento = new FuzzyRuleAntecedent()
470     ;
471     ifAbajoAndPosLento->joinWithAND(abajo, posLento);
472
473 FuzzyRuleAntecedent *ifAbajoAndCero_dev_ang = new
474     FuzzyRuleAntecedent ();

```

```

469 ifAbajoAndCero_dev_ang->joinWithAND (abajo , cero_dev_ang) ;
470
471 FuzzyRuleAntecedent *ifAbajoAndNegLento = new FuzzyRuleAntecedent ()
472 ;
473 ifAbajoAndNegLento->joinWithAND (abajo , negLento) ;
474
475 FuzzyRuleAntecedent *ifAbajoAndNegRapido = new FuzzyRuleAntecedent
476 () ;
477 ifAbajoAndNegRapido->joinWithAND (abajo , negRapido) ;
478
479 FuzzyRuleAntecedent *ifMuyAbajoAndPosRapido = new
480 FuzzyRuleAntecedent () ;
481 ifMuyAbajoAndPosRapido->joinWithAND (muyAbajo , posRapido) ;
482
483 FuzzyRuleAntecedent *ifMuyAbajoAndPosLento = new
484 FuzzyRuleAntecedent () ;
485 ifMuyAbajoAndPosLento->joinWithAND (muyAbajo , posLento) ;
486
487 FuzzyRuleAntecedent *ifMuyAbajoAndCero_dev_ang = new
488 FuzzyRuleAntecedent () ;
489 ifMuyAbajoAndCero_dev_ang->joinWithAND (muyAbajo , cero_dev_ang) ;
490
491 FuzzyRuleAntecedent *ifMuyAbajoAndNegLento = new
492 FuzzyRuleAntecedent () ;
493 ifMuyAbajoAndNegLento->joinWithAND (muyAbajo , negLento) ;
494
495 FuzzyRuleConsequent *thenHoraRapido = new FuzzyRuleConsequent () ;

```

```

496     thenHoraRapido->addOutput( horaRapido ) ;

497

498     FuzzyRuleConsequent *thenHorario = new FuzzyRuleConsequent() ;
499     thenHorario->addOutput( horario ) ;

500

501     FuzzyRuleConsequent *thenCero_pwm = new FuzzyRuleConsequent() ;
502     thenCero_pwm->addOutput( cero_pwm ) ;

503

504     FuzzyRuleConsequent *thenAntiH = new FuzzyRuleConsequent() ;
505     thenAntiH->addOutput( antiH ) ;

506

507     FuzzyRuleConsequent *thenAntiHRapido = new FuzzyRuleConsequent() ;
508     thenAntiHRapido->addOutput( antiHRapido ) ;

509

510 // if muyArriba and posRapido then horaRapido

511

512     FuzzyRule *fuzzyRule1 = new FuzzyRule(1, ifMuyArribaAndPosRapido ,
513                                         thenHoraRapido) ;
514     controlAngulo->addFuzzyRule(fuzzyRule1) ;

515 // if muyArriba and posLento then horaRapido

516

517     FuzzyRule *fuzzyRule2 = new FuzzyRule(2, ifMuyArribaAndPosLento ,
518                                         thenHoraRapido) ;
519     controlAngulo->addFuzzyRule(fuzzyRule2) ;

520 // if muyArriba and cero_dev_ang then horaRapido

521

522     FuzzyRule *fuzzyRule3 = new FuzzyRule(3, ifMuyArribaAndCero_dev_ang
523                                         , thenHoraRapido) ;
524     controlAngulo->addFuzzyRule(fuzzyRule3) ;

525 // if muyArriba and negLento then horaRapido

526

```

```

527 FuzzyRule *fuzzyRule4 = new FuzzyRule(4, ifMuyArribaAndNegLento,
528     thenHoraRapido);
529 controlAngulo->addFuzzyRule(fuzzyRule4);
530
531 // if muyArriba and negRapido then horaLento
532
533 FuzzyRule *fuzzyRule5 = new FuzzyRule(5, ifMuyArribaAndNegRapido,
534     thenHoraRapido);
535 controlAngulo->addFuzzyRule(fuzzyRule5);
536
537
538 // if arriba and posRapido then horaRapido
539
540 FuzzyRule *fuzzyRule6 = new FuzzyRule(6, ifArribaAndPosRapido,
541     thenHorario);
542 controlAngulo->addFuzzyRule(fuzzyRule6);
543
544 // if arriba and posLento then horaRapido
545
546 FuzzyRule *fuzzyRule7 = new FuzzyRule(7, ifArribaAndPosLento,
547     thenHorario);
548 controlAngulo->addFuzzyRule(fuzzyRule7);
549
550 // if arriba and cero_dev_ang then horario
551
552 FuzzyRule *fuzzyRule8 = new FuzzyRule(8, ifArribaAndCero_dev_ang,
553     thenHorario);
554 controlAngulo->addFuzzyRule(fuzzyRule8);
555
556 // if arriba and negLento then horario
557
558 FuzzyRule *fuzzyRule9 = new FuzzyRule(9, ifArribaAndNegLento,

```

```

    thenHorario);

556 controlAngulo->addFuzzyRule(fuzzyRule9);

557

558 // if arriba and negRapido then cero_pwm

559

560 FuzzyRule *fuzzyRule10 = new FuzzyRule(10, ifArribaAndNegRapido,
561     thenHoraRapido);
562 controlAngulo->addFuzzyRule(fuzzyRule10);

563

564

565 // if cero_ang and posRapido then horaRapido

566

567 FuzzyRule *fuzzyRule11 = new FuzzyRule(11, ifCero_angAndPosRapido,
568     thenAntiH);
569 controlAngulo->addFuzzyRule(fuzzyRule11);

570 // if cero_ang and posLento then horario

571

572 FuzzyRule *fuzzyRule12 = new FuzzyRule(12, ifCero_angAndPosLento,
573     thenCero_pwm);
574 controlAngulo->addFuzzyRule(fuzzyRule12);

575 // if cero_ang and cero_dev_ang then cero_pwm

576

577 FuzzyRule *fuzzyRule13 = new FuzzyRule(13,
578     ifCero_angAndCero_dev_ang, thenCero_pwm);
579 controlAngulo->addFuzzyRule(fuzzyRule13);

580 // if cero_ang and negLento then antiH

581

582 FuzzyRule *fuzzyRule14 = new FuzzyRule(14, ifCero_angAndNegLento,
583     thenCero_pwm);
584 controlAngulo->addFuzzyRule(fuzzyRule14);

```

```

584
585 // if cero_ang and negRapido then antiHRapido
586
587 FuzzyRule *fuzzyRule15 = new FuzzyRule(15, ifCero_angAndNegRapido,
588 thenHorario);
controlAngulo->addFuzzyRule(fuzzyRule15);

589
590
591
592 // if abajo and posRapido then cero_pwm
593
594 FuzzyRule *fuzzyRule16 = new FuzzyRule(16, ifAbajoAndPosRapido,
595 thenAntiHRapido);
controlAngulo->addFuzzyRule(fuzzyRule16);

596
597 // if abajo and posLento then antiH
598
599 FuzzyRule *fuzzyRule17 = new FuzzyRule(17, ifAbajoAndPosLento,
600 thenAntiH);
controlAngulo->addFuzzyRule(fuzzyRule17);

601
602 // if abajo and cero_dev_ang then antiH
603
604 FuzzyRule *fuzzyRule18 = new FuzzyRule(18, ifAbajoAndCero_dev_ang,
605 thenAntiH);
controlAngulo->addFuzzyRule(fuzzyRule18);

606
607 // if abajo and negLento then antiHRapido
608
609 FuzzyRule *fuzzyRule19 = new FuzzyRule(19, ifAbajoAndNegLento,
610 thenAntiH);
controlAngulo->addFuzzyRule(fuzzyRule19);

611
612 // if abajo and negRapido then antiHRapido

```

```

613
614 FuzzyRule *fuzzyRule20 = new FuzzyRule(20, ifAbajoAndNegRapido,
615     thenAntiH);
controlAngulo->addFuzzyRule(fuzzyRule20);

616
617
618
619
620 // if muyAbajo and posRapido then antiH
621
622 FuzzyRule *fuzzyRule21 = new FuzzyRule(21, ifMuyAbajoAndPosRapido,
623     thenAntiHRapido);
controlAngulo->addFuzzyRule(fuzzyRule21);

624
625 // if muyAbajo and posLento then antiHRapido
626
627 FuzzyRule *fuzzyRule22 = new FuzzyRule(22, ifMuyAbajoAndPosLento,
628     thenAntiHRapido);
controlAngulo->addFuzzyRule(fuzzyRule22);

629
630 // if muyAbajo and cero_dev_ang then antiHRapido
631
632 FuzzyRule *fuzzyRule23 = new FuzzyRule(23,
633     ifMuyAbajoAndCero_dev_ang, thenAntiHRapido);
controlAngulo->addFuzzyRule(fuzzyRule23);

634
635 // if muyAbajo and negLento then antiHRapido
636
637 FuzzyRule *fuzzyRule24 = new FuzzyRule(24, ifMuyAbajoAndNegLento,
638     thenAntiHRapido);
controlAngulo->addFuzzyRule(fuzzyRule24);

639
640 // if muyAbajo and negRapido then antiHRapido
641

```

```

642 FuzzyRule *fuzzyRule25 = new FuzzyRule(25, ifMuyAbajoAndNegRapido,
643     thenAntiHRapido);
644 controlAngulo->addFuzzyRule(fuzzyRule25);
645 }
646
647 //Funcion de control difuso
648 float Difuso::output(float posicion, float angulo, float set_point){
649
650     float set_point_ang = 0, salida=0.0;
651
652     this->error_pos[0] = this->error_pos[1];
653
654     posicion = posicion*(0.13468) - 0.2222;
655
656     this->error_pos[1] = -(set_point - posicion);
657
658     if (this->error_pos[1] > 0.2) {
659         this->error_pos[1] = 0.19;
660     }
661
662     if (this->error_pos[1] < -0.2) {
663         this->error_pos[1] = -0.19;
664     }
665
666     if (this->error_pos[1] < 0.003 && this->error_pos[1] > -0.003) {
667         this->error_pos[1] = 0;
668     }
669
670     this->error_dev_pos[0] = this->error_dev_pos[1];
671
672     this->error_dev_pos[1] = (this->error_pos[1] - this->error_pos[0])
673         * 31.4 + this->error_dev_pos[0] * 0.6861;

```

```

674 if( this->error_dev_pos[1] > 0.2) {
675     this->error_dev_pos[1] = 0.19;
676 }
677
678 if( this->error_dev_pos[1] < -0.2){
679     this->error_dev_pos[1] = -0.19;
680 }
681
682 this->controlPos->setInput(1, this->error_pos[1]);
683 this->controlPos->setInput(2, this->error_dev_pos[1]);
684 this->controlPos->fuzzify();
685
686 set_point_ang = this->controlPos->defuzzify(1);
687
688 if( set_point_ang > 0.785){
689     set_point_ang = 0.77;
690 }
691
692 if( set_point_ang < -0.785){
693     set_point_ang = -0.77;
694 }
695
696 this->error_ang[0] = this->error_ang[1];
697
698 angulo = angulo * (0.6058) - 1.0904;
699
700 this->error_ang[1] = set_point_ang - angulo;
701
702 if( this->error_ang[1] > 0.785){
703     this->error_ang[1] = 0.77;
704 }
705
706 if( this->error_ang[1] < -0.785){
707     this->error_ang[1] = -0.77;

```

```

708 }
709
710     this->error_dev_ang[1] = (this->error_ang[1] - this->error_ang[0])
711     / 0.012;
712
713     if (this->error_dev_ang[1] < 0.15 && this->error_dev_ang[1] >-0.15){
714         this->error_dev_ang[1] = 0;
715     }
716
717     if (this->error_dev_ang[1] > 3){
718         this->error_dev_ang[1] = 2.9;
719     }
720
721     if (this->error_dev_ang[1] < -3){
722         this->error_dev_ang[1] = -2.9;
723     }
724
725     this->controlAngulo->setInput(1,error_ang[1]);
726     this->controlAngulo->setInput(2,error_dev_ang[1]);
727     this->controlAngulo->fuzzify();
728
729     salida = this->controlAngulo->defuzzify(1);
730
731     salida= -(salida / 10.0) + 0.5;
732     salida = salida*255.0;
733
734     return salida;
735 }
```

Código 3.11. controlDifuso.cpp.

```

1 #ifndef ADC_H_
2 #define ADC_H_
3
4 #include "esp_system.h"
5 #include "driver/adc.h"
```

```

6 #include <driver/dac.h>
7 #include "esp_adc_cal.h"
8 #include "esp_log.h"
9
10 // Variables para control de ADC
11 #define DEFAULT_VREF    1100           // Use adc1_vref_to_gpio() to
12                           obtain a better estimate
12 #define NO_OF_SAMPLES   100            // Multisampling
13
14 // Log ADC
15 static const char *ADC_MUESTRA = "ADC_MUESTRA";
16
17 // Clase Adc1
18 class Adc1{
19
20 public:
21
22 Adc1(adc_channel_t canal, adc_bits_width_t precision, adc_atten_t
23 atenuacion, uint8_t muestras);
23 int get_muestra();
24
25 private:
26
27 adc_bits_width_t precision;
28 adc1_channel_t canal;
29 adc_atten_t atenuacion;
30 uint8_t muestras;
31 bool adc_config();
32 // void print_char_val_type(esp_adc_cal_value_t val_type);
33 // static void check_efuse();
34
35
36 };
37

```

```

38 // Clase Dac
39 class Dac{
40
41 public:
42
43     Dac(dac_channel_t dac_input);
44     esp_err_t dac_output(uint8_t valor);
45
46 private:
47
48     bool dac_habilitado();
49     dac_channel_t dac_input;
50
51 };
52
53
54
55 #endif

```

Código 3.12. adc.h.

```

1 #include "adc.h"
2
3 // Contructor de clase Adc1
4 Adc1::Adc1(adc_channel_t canal, adc_bits_width_t precision,
5             adc_atten_t atenuacion, uint8_t muestras){
6
7     this->canal = (adc1_channel_t)canal;
8     this->precision = precision;
9     this->atenuacion = atenuacion;
10    this->muestras = muestras;
11    adc_config();
12 }
13
14 // Se configura el adc

```

```

15 bool Adc1 :: adc_config() {
16
17     esp_adc_cal_characteristics_t *adc_chars;
18
19     esp_err_t error;
20
21     adc1_config_width(this->precision);
22
23     error = adc1_config_channel_atten(this->canal, this->atenuacion);
24
25     if(error==ESP_OK) {
26         ESP_LOGI(ADC_MUESTRA, "GPIO %d se configuro a 12 Bits", this->
27             canal + 30);
28     } else {
29         ESP_LOGE(ADC_MUESTRA, "GPIO %d se configuro a 12 Bits", this->
30             canal + 30);
31
32         return false;
33     }
34 }
35
36 // Obtener muestra del Adc
37 int Adc1 :: get_muestra() {
38
39     int adc_reading = 0;
40
41     for (int i = 0; i < this->muestras; i++) {
42         adc_reading += adc1_get_raw((adc1_channel_t) this->canal);
43     }
44     adc_reading/=this->muestras;
45
46     if(adc_reading > 4095){

```

```

47     adc_reading = 4095;
48 }
49 if (adc_reading < 0){
50     adc_reading = 0;
51 }
52
53 return(adc_reading);
54 }
55
56 //Contructor de clase Dac
57 Dac::Dac(dac_channel_t dac_input){
58     this->dac_input = dac_input;
59     ESP_LOGI(ADC_MUESTRA, "DAC %d Inicializado", this->dac_input);
60     dac_habilitado();
61 }
62
63
64 //Habilitador del Dac
65 bool Dac::dac_habilitado(){
66     esp_err_t error;
67
68     error = dac_output_enable(this->dac_input);
69     if (error==ESP_OK){
70         ESP_LOGI(ADC_MUESTRA, "DAC %d habilitado", this->dac_input);
71         return true;
72     } else{
73         ESP_LOGE(ADC_MUESTRA, "No se inicio DAC %d Motor", this->dac_input);
74         return false;
75     }
76
77 }
78
79 //Salida del Dac

```

```
80 esp_err_t Dac::dac_output(uint8_t valor){  
81  
82     esp_err_t error;  
83  
84     error = dac_output_voltage(this->dac_input, valor);  
85  
86     return error;  
87  
88 }
```

Código 3.13. adc.cpp.

## Apéndice IV

### DISEÑO DE PCB

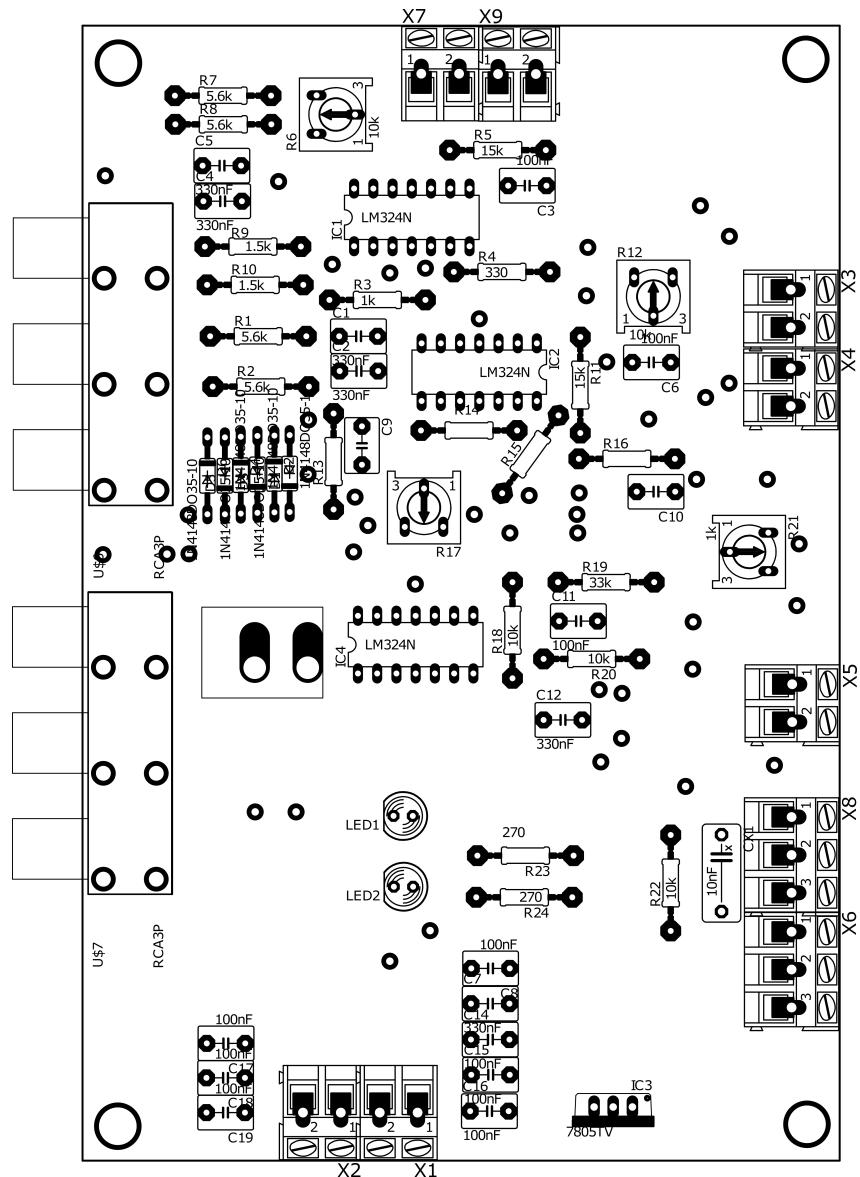
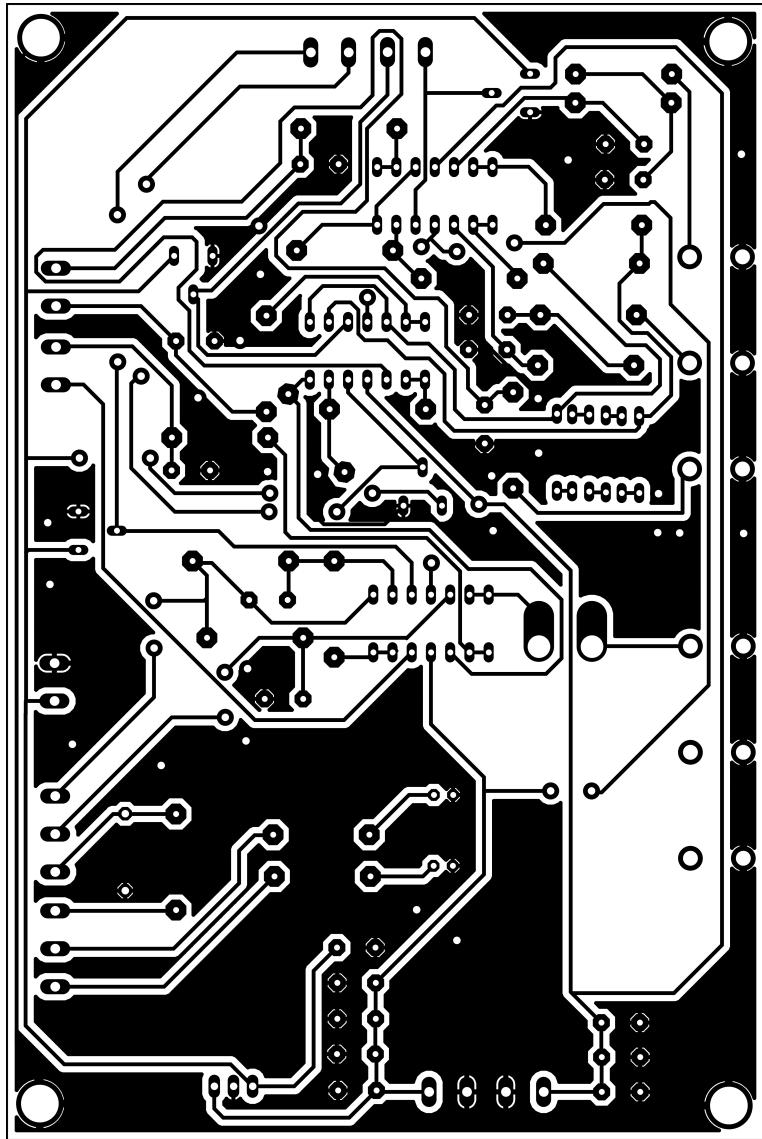


Figura 4.1. Ubicación de componentes en PCB.



**Figura 4.2.** Diseño de PCB.