IME

CONTROLADOR DIFUSO PARA SIMULACIÓN DE UN SISTEMA DE CABINA DE PINTURA

MERINO MARTINEZ IVÁN

Universidad Nacional Autónoma de México
Facultad de Estudios Superiores Cuautitlán

Profesor: David Tinoco Varela

RESUMEN

Los procesos de control clásico utilizados en la industria, y en cualquier otra aplicación, pueden involucrar una variedad de problemáticas como retrasos y falta de respuesta. Los sistemas de control convencionales basados en modelos matemáticos, debido a su poca capacidad de respuesta ante eventos no esperados, pueden no proporcionar sistemas satisfactorios. Para evitar este tipo de problemas, se busca el desarrollo de otro tipo de controladores, entre ellos, aquellos que utilizan inteligencia artificial para su funcionamiento, tales como los sistemas de control basados en lógica difusa, los cuales han venido demostrando que son una opción apropiada para hacerle frente a las complejidades de los procesos en la vida real, ya que son responsivos y "autosuficientes".

El proyecto descrito en este archivo tiene como objetivo mejorar el rendimiento de una cabina de pintura, utilizando un controlador difuso para tal fin. Para la realización de este proyecto se utilizó la paquetería *Fuzzy logic toolbox* de Matlab, así mismo, se realizó la conexión entre Matlab y la tarjeta de desarrollo Arduino UNO. De esta forma, se ingresan los valores de los sensores por medio de Arduino, y Matlab procesa tales valores por medio de un sistema difuso diseñado en tal paquetería de *software*.

1 INTRODUCCIÓN

Las grandes industrias estructuralmente están compuestas por múltiples equipos de operaciones unitarias e integrados de manera sistemática y racional. Estas industrias deben satisfacer un vasto número de requerimientos tanto de seguridad, especificaciones, regulaciones, monitoreo de mercado, operacionales y de demanda. Dichos requerimientos deben de ser monitoreados y controlados con el fin de garantizar los objetivos definidos de cada una de ellas. Obviamente, los procesos de control juegan un papel primordial, ya que son ellos los que van a permitir que todas las máquinas y dispositivos trabajen adecuadamente, y bajo los estándares requeridos.

A medida que transcurre el tiempo, las estrategias de control se han ido diversificando, y se han ido también especializando en diferentes tareas de diferentes tipos. Dentro de esta diversificación, se encuentran aquellas estrategias basadas en inteligencia artificial (IA), tales como lo controladores basados en lógica difusa, los cuales se han ido consolidando como una herramienta esencial en diversas tareas, debido a que cumplen con las mismas funciones de los controladores convencionales, pero son ideales para sistemas de control complejos y no lineales. Tal ha sido su importancia, que incluso algunos PLC, ya la integran dentro de sus funcionalidades básicas. La lógica difusa busca describir aproximaciones matemáticas a partir de datos poco precisos para resolver problemas de forma similar al razonamiento humano.

En este trabajo se plantea el desarrollo de un controlador difuso capaz de recolectar y medir las condiciones del entorno de temperatura y humedad de una cabina de pintura para decidir la acción que debería realizarse, regulando con dos salidas las condiciones. Esto a través de un sensor DHT11 que tiene la opción de tomar temperatura y humedad y dos salidas: un ventilador y una resistencia o foco de halógeno. Las gestiones de las salidas son llevadas a cabo mediante el controlador difuso implementado en Matlab, con la variación que este se diseñará para trabajar directamente en la placa de desarrollo Arduino UNO, lo que nos dará la opción de no estar conectado directamente al sistema de cómputo. Se espera con este trabajo brindar alternativas para proyectos similares, que se logren optimizar y tengan la opción de controlarlo mediante los microcontroladores sin necesidad de la conexión directa.

1.1 OBJETIVOS

- **Objetivo general:**
  Armar un sistema de control difuso para simulación de un control de cabina de pintura, utilizando los softwares de Matlab y Arduino. Obtener esquemas de conexiones y códigos de programación a nivel de software. Construir físicamente con recursos al alcance, el circuito de control, con el objetivo de modelar un sistema de control de cabina de pintura para ofrecer condiciones ideales.

- **Objetivos particulares:**
  Realizar el trabajo teórico-práctico acerca de los conceptos del modelo de control difuso para su difusión, así como alternativa de modificado para proyectos semejantes.

2 MARCO TEÓRICO

2.1 ANTECEDENTES DE LÓGICA DIFUSA

La lógica difusa tiene sus bases cuando *Lofti Zadeh*, profesor de la Universidad de California, en el año de 1964, planteó la teoría de conjuntos difusos, dentro del artículo "*Fuzzy sets*" en "*Informations and control*". A partir de ese momento se han realizado diferentes proyectos, aplicaciones, desarrollos, etc. Que lo han ido posicionando como una estrategia sumamente útil y de vanguardia.

2.2 LÓGICA DIFUSA

La lógica difusa, es una lógica que permite llegar a conclusiones "razonadas" a partir de información ambigua o imprecisa. Este concepto fue desarrollado por el profesor Lofti A. Zadeh (1965), trabajo retomado en Zadeh (1996), quien la presentó como una forma de interpretar información basada en la pertenencia parcial de tal información con respecto a un determinado tipo de conjuntos, definidos como conjuntos difusos.

Uno de los grandes aportes de esta lógica, es que permite modelar situaciones o comportamientos que son vagos en sí mismos, es decir, se adapta mejor a la realidad que una lógica clásica, en dónde solo existen dos valores a decidir. Por ejemplo, si consideramos un clima frío a 10° C y un clima caliente a 30° C, bajo una perspectiva clásica, sólo se tendría el valor frío y caliente, sin embargo, entre estos valores existen muchas otras temperaturas que tienen ambigüedad con respecto a los valores definidos, es decir bajo una lógica clásica, ¿20°C, es una temperatura fría o caliente?, bajo esta perspectiva, no se tiene certeza de a qué conjunto pertenece 20° C, pero si esta situación se analiza bajo una perspectiva difusa, es posible definir la pertenencia de 20 grados con respecto al conjunto de las temperaturas frías y al de las temperaturas calientes. Esta situación, puede observarse en la figura 1, en la cual tenemos por un lado, la lógica clásica, se puede observar que si se evalúan los valores X y Y, sólo pueden pertenecer a un conjunto lógico cada uno, sin embargo, en la lógica difusa, estos mismos valores pueden tener diferentes rangos de pertenencia con respecto a los conjuntos difusos, si se evalúa el valor X, este tiene una pertenencia del 10% con respecto al conjunto caliente y una pertenencia del 30% con respecto al conjunto frío, es decir, el valor X es un poco más frío que caliente; por otro lado, si se evalúa el valor Y, es posible ver que este tiene una pertenencia del 80% con respecto al conjunto caliente.

En esta misma figura, podemos ver que desde el inicio de los conjuntos hasta su punto máximo, el 100%, se generan pendientes de pertenencia, está pendiente indica la vaguedad de un dato con respecto a un conjunto. Existe un conjunto caliente y uno frío, pero las temperaturas no son solo

completamente calientes o completamente frías, tienen rangos de pertenencia.

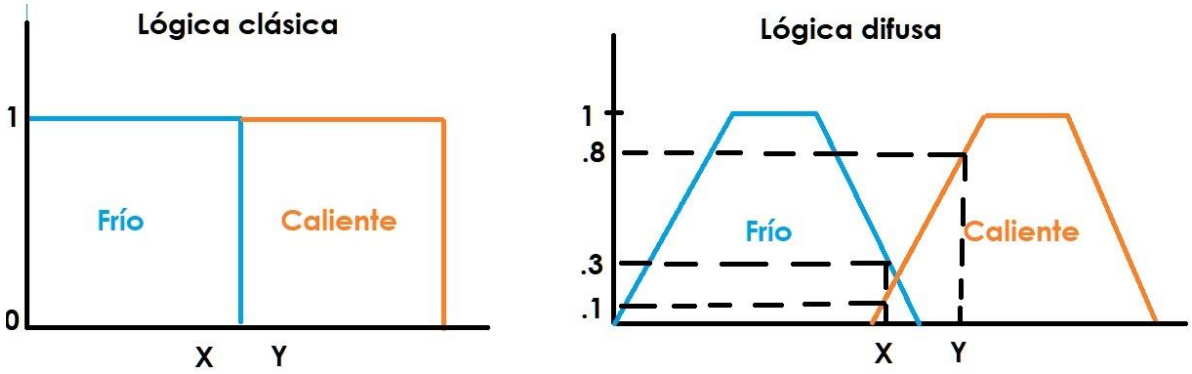

Figura 1. Comparativa entre conjuntos de lógica clásica y lógica difusa.

Es importante notar que el razonamiento humano no reacciona en una manera de lógica clásica, sino que por el contrario, evalúa el entorno y en función de las ponderaciones de cada una de las variables toma una decisión, por lo que la lógica difusa es más adecuada para tratar de emular tal comportamiento mental.

Algo relevante de la lógica difusa, es que ella ha servido para el desarrollo de una cantidad incontable de aplicaciones de todo tipo tales como medicina y bioinformática (Torres & Nieto, 2006), aplicaciones industriales (Larsen, 1980), robots autónomos (Peri & Simon, 2005), controladores de motores de inducción (Uddin et al., 2002), entre muchas otras.

Este concepto matemático ha generado una herramienta altamente utilizada y utilizable en sistemas tecnológicos: los Controladores difusos. Un sistema de control estándar basado en lógica difusa se puede representar de acuerdo a la figura 2.

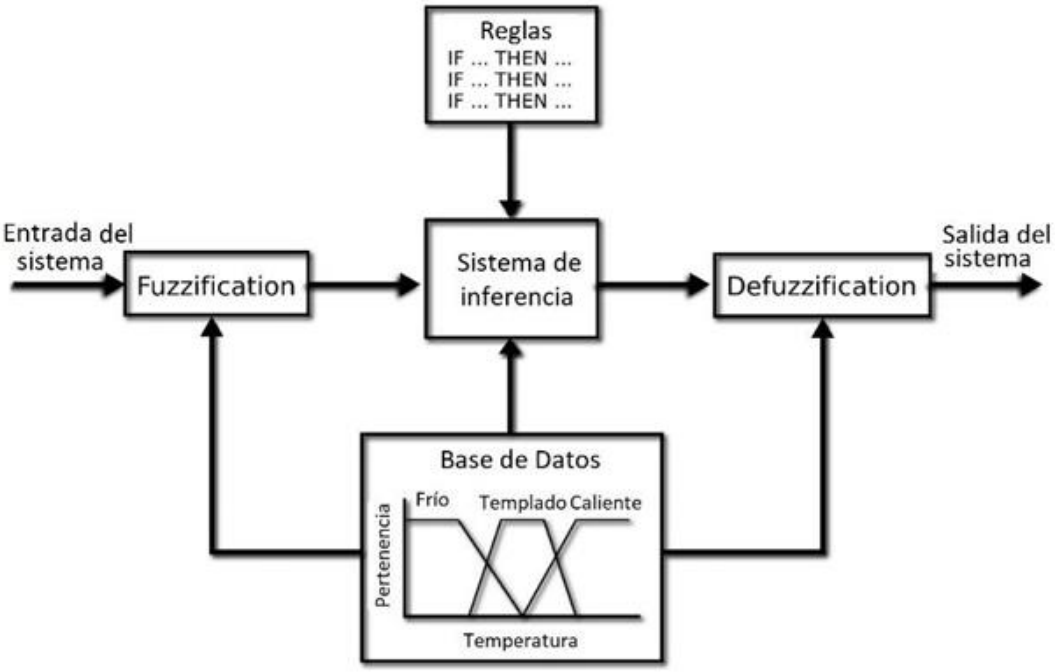

Figura 2. Esquema básico de un sistema de control difuso.

Estos sistemas que han sido eje principal de desarrollo de una gran cantidad de aplicaciones e interfaces, entre las que se pueden mencionar, el control de señales PWM para mejorar el rendimiento y la eficiencia en unidades de frecuencia variable (VFD), y así controlar el funcionamiento de los motores de inducción que generalmente son de naturaleza no lineal (Sharma et. al., 2019); el control de la estabilización de un sistema de silla de ruedas con carga útil en movimiento (Jamin, et al, 2018); control de sistemas de robots con llantas (Castillo & Aguilar, 2019); para el control de sistemas domóticos (Pau & Salerno, 2019); y en particular una aplicación interesante para los fines de este proyecto, es el desarrollo de un sistema que puede operar el proceso de cambio de la transmisión manual de un automóvil de forma automática. (Ramadhan et al., 2019). Según los autores, hay dos programas importantes dentro del sistema; el programa de cambio automático y el programa del acelerador automático.

2.3 Cabina de pintura

Sin tener en cuenta la aplicación, la selección cuidadosa del sistema y equipo juegan un papel importante en la actuación exitosa de los acabados ya que todo debe de estar bajo un equilibrio, no sólo para el proceso de preparado y pintado, sino también para un exitoso acabado final. El conocimiento de las instalaciones y el proceso de la producción es importante para escoger el equipo correcto. Existen diferentes factores o variables que se debe tomar en cuenta para que el proceso de pintado se lleve a cabo de la mejor manera, de las cuales en este proyecto nos enfocaremos en dos principales: Temperatura y humedad.

2.3.1 Temperatura controlada

Los parámetros deben ser controlados de acuerdo a los valores recomendados por el fabricante de las cabinas. La temperatura ambiental debe oscilar en un rango especificado por el fabricante de la pintura. Generalmente suele ser entre 15° C y 32°C. Por debajo de los 10°C, el tiempo de secado y curado puede extenderse, complicando la labor e inclusive afectando el recubrimiento. Por el contrario, una temperatura superior a los 32ºC puede acelerar el secado afectando la uniformidad y la adherencia.

Luego del pintado, la temperatura ambiental, también debe controlarse hasta que esté completo su curado; sobre todo en la noche ya que la temperatura puede descender al punto de congelamiento. Prestar especial cuidado si se pinta al sol, ya que la superficie se encuentra se encuentra a una temperatura superior a la del ambiente, lo que puede modificar su secado y curado, afectando la terminación final.

Figura 3. Cabina de pintura UFIZZI UF-8000.

2.3.2 Humedad relativa controlada

El aire del ambiente posee agua en forma de vapor. Cuando esta condensa sobre la superficie a pintar, se contamina con agua afectando la adherencia de las pinturas, el secado, el curado parejo y el brillo. Para evitarlo, como regla general, se suspende el pintado si la humedad relativa es mayor a 85%.

Para ser más precisos, se debe analizar la posibilidad de que se produzca condensación sobre el sustrato. Se puede precisar midiendo la temperatura del aire, de bulbo seco, la humedad relativa, con un psicrómetro de voleo y la temperatura de la superficie, con un termómetro de superficie. Luego utilizando una tabla psicométrica, se relacionan los valores, encontrando el punto de rocío, que es la temperatura a la cual la mezcla aire, vapor de agua condensa. Se recomienda suspender la aplicación cuando la diferencia entre la temperatura de la superficie del sustrato y la del punto de rocío sea inferior a 3º C, ya que es probable que se condense el agua sobre la superficie. La temperatura del sustrato siempre debe estar 3º C por encima del punto de rocío.

2.4 Arduino

Arduino es una placa de desarrollo (existen diferentes tipos y modelos) que se crea bajo la idea de que cualquier usuario pueda desarrollar un proyecto sin tener conocimientos profundos de programación, electrónica, robótica, etc. Esta placa se ha convertido en el corazón de muchos, y muy variados, proyectos tecnológicos. Es posible encontrar desde sistemas que encienden y apagan un LED, hasta sistemas que son capaces de controlar dispositivos complejos como sillas de ruedas robotizadas y exoesqueletos. A pesar de la gran cantidad de tarjetas de desarrollo que existen en el mercado, Arduino se ha posicionado rápidamente como una de las más populares (probablemente la más popular), entre estudiantes y profesionales.

Esta tarjeta de desarrollo permite interactuar con el entorno físico por medio de sensores y actuadores, los cuales son fácilmente controlables por medio de su IDE, sobra decir que su programación es en demasía sencillo.

Actualmente, y debido a su popularidad, muchos sistemas en software y hardware independientes han generado estructuras para su interacción con la mencionada tarjeta, una de ellas: Matlab, de lo cual nos apoyaremos en este proyecto.

2.5 Fritzing

Fritzing es un programa de diseño electrónico que permite al usuario generar circuitos y esquemáticos con gran aporte visual, una de sus características es que dentro de sus elementos con los que se pueden modelar los circuitos, se encuentran diferentes tipos de placas de desarrollo Arduino.

2.6 Matlab

Matlab, es una herramienta ampliamente utilizada en la ingeniería, ya que contiene una gran cantidad de *toolbox*, enfocadas a muchas y variadas ramas de ingeniería y desarrollo. Matlab es una abreviatura de *Matrix Laboratory*, y este nombre representa básicamente la forma en la que Matlab trabaja, todo por medio de la representación matricial de sus elementos.

Matlab se considera en sí mismo un entorno de desarrollo y un lenguaje de programación de alto rendimiento.

Dentro de sus toolbox, se encuentran aquellos que esta n enfocados a comunicarse con elementos físicos, tales como las tarjetas de desarrollo, y aquellos enfocados a temas concretos, tal como la IA y la lógica difusa, situación que nos permitirá desarrollar el proyecto planteado.

3 Materiales y equipo

Para el desarrollo del proyecto se utilizaron una variedad de componentes que se nombran a continuación.

3.1 Hardware

- Sensor DHT11
- Placa Arduino UNO
- Ventilador 10V
- Dos placas Protoboard
- Adaptador de pilas y pilas 12V
- Pila 9V
- Foco halógeno
- Mostfet IRL530
- Resistencia 4.7 KΩ
- 2 Resistencia 100 KΩ
- Diodo D11N4001
- DIYmall Cable USB para Arduino UNO
- Cable para conexiones
- Pantalla LCD 16X2
- Trimpot 50 KΩ
- Transistor 2N3904

3.2 Software

- Matlab R2021
- Arduino
- Fritzing

4 Funcionamiento y desarrollo del sistema

El sistema de simulación de cabina de pintura, emplea el material previamente mencionado. Contamos con un módulo de sensor DHT11, de temperatura y humedad. El sensor DHT11 se caracteriza por tener la señal digital calibrada, asegurando alta estabilidad y fiabilidad a lo largo del tiempo. El sensor integra sensores resistivos para temperatura y otro para humedad. Puede medir la humedad en un rango desde 20% hasta 90% y temperatura en el rango de 0ºC a 50ºC. este módulo de sensor nos permite tener las dos entradas para nuestro controlador.

Posteriormente, tenemos las salidas del sistema. La primera de ellas es la conexión de un ventilador con un transistor y una fuente, el ventilador va a reaccionar con la señal de salida del Arduino. Así mismo tenemos la segunda el cual es la conexión hacia un foco de halógeno que toma la referencia de un emisor de calor, el cual está conectado para amplificar la señal que sale del Arduino mediante un Mosfet y una batería. Aunado a estas conexiones, se implementó un LCD, el cual nos permite contar con una salida visual de los valores en tiempo real de los componentes de entrada y de salida.

En la figura 4 se presenta el modelo esquemático del circuito en cuestión, desarrollado mediante el programa de Fritzing:

Nota: Fritzing es una iniciativa de hardware de código abierto que hace que la electrónica sea accesible como material creativo para cualquier persona. Este programa nos permite desarrollar esquemas de los circuitos con una variedad de componentes y permitiendo agregar nuevos.

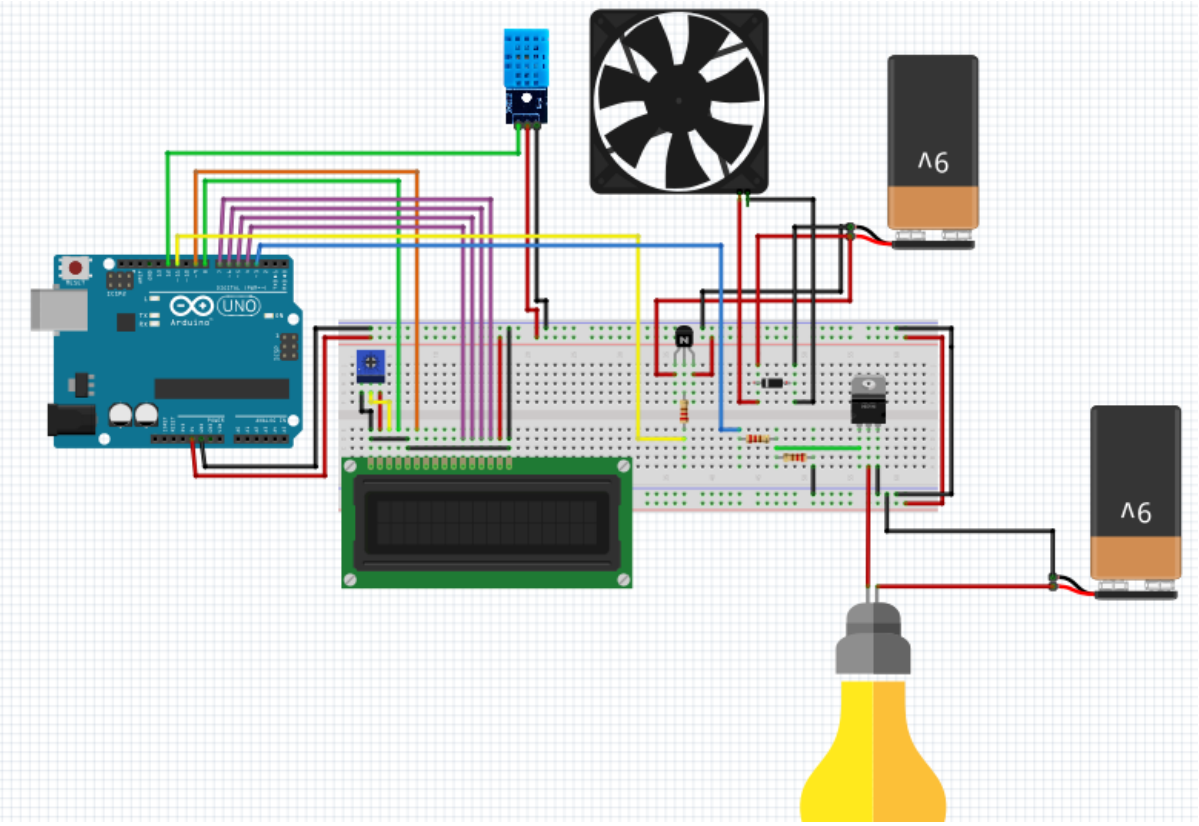

Figura 4. Modelo esquemático de las conexiones del circuito.

4.1| DESARROLLO DEL CONTROLADOR DIFUSO

En esta sección se detalla el procedimiento que se siguió para estructurar el modelo de controlador difuso, cabe mencionar que se tomaron consideraciones para la definición de rangos de acuerdo a una semejanza de una cabina de pintura.

Utilizamos el programa de Matlab 2021 para el desarrollo del controlador, mediante la función de *Fuzzy logic*. Esta función la abrimos escribiendo la palabra “fuzzy” en la ventana de comando. Nos aparece una nueva ventana como la mostrada en la figura 5:

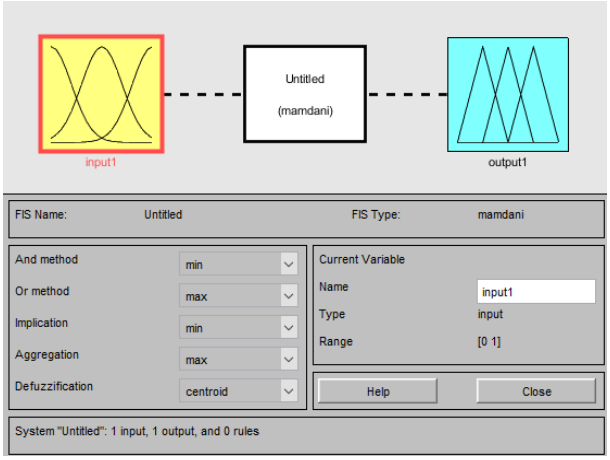

Figura 5. Ventana Matlab, función fuzzy.

4.1.1 Definición de variables

Comenzamos definiendo las variables de entrada del sistema, sobre la base de ellas se estructuran las de salida. Las salidas están dadas directamente por el planteamiento de las funciones del circuito. Para agregar las variables de entrada y las de salida, se selecciona "edit" seguido de "add variable". Al definir nuestras cuatro variables, seleccionamos en cualquiera de ellas para editarlas con las características específicas. Se abre una nueva ventana para modificar las variables y darles rangos los cuales seleccionamos de acuerdo a valores correctos que debe seguir nuestro controlador, ver figura 6.

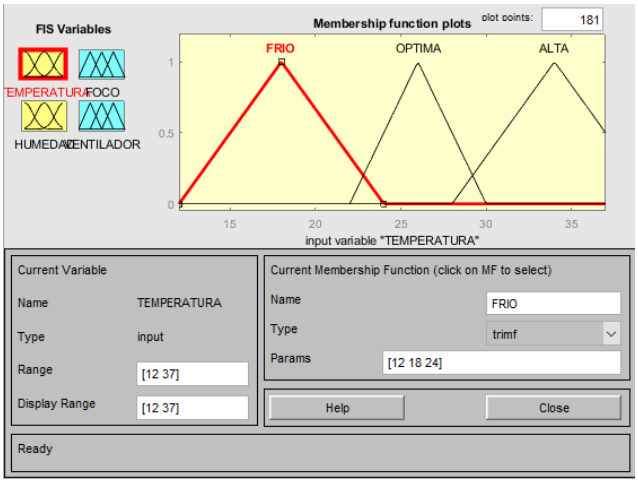

Figura 6. Definición de variables.

Para la variable de temperatura su función de membresía de tipo triangular que va a tomar lugar entre los rangos de frio de 0 a 24, optima de 22 a 30 y alta de 28 a 40, los valores de estos rangos están en grados centígrados. Para el caso de la variable humedad, su función de membresía de tipo triangular que va a tomar lugar entre los rangos deseco que va de 0 a 40, regular de 30 a 70 y húmedo que va de 60 a 100, los valores están en porcentaje de humedad.

Para las variables de salida consideramos las salidas del Arduino PWM (*modulación por ancho de pulsos*) dada en Hz. En Arduino la frecuencia de PWM es de 500Hz. Pero es un valor que puede modificarse, para nuestro caso lo dejamos de 0 a 255, que se genera con 8 bits. Para el caso de la

variable del foco, su función de membresía de tipo triangular, en este caso se definen cuatro conjuntos que la describen, que van de 0 a 10 para el estado apagado, de 0 a 100 para oscuro, de 60 a 210 para medio y 160 a 255 para el estado brillante. Para nuestra última variable de salida, ventilador, tomamos una función de membresía de tipo trapezoidal para el estado apagado que va de 0 a 80 y tres funciones más de tipo triangular, lento que va de 50 a 150, normal de 130 a 220 y rápido que va de 200 a 255, todo esto se puede ver en la figura 7.

Nota: Cualquiera de los rangos se puede modificar para que nuestro controlador funcione de manera óptima, etoesta en función del diseñador y los valores que se consideren adecuados.

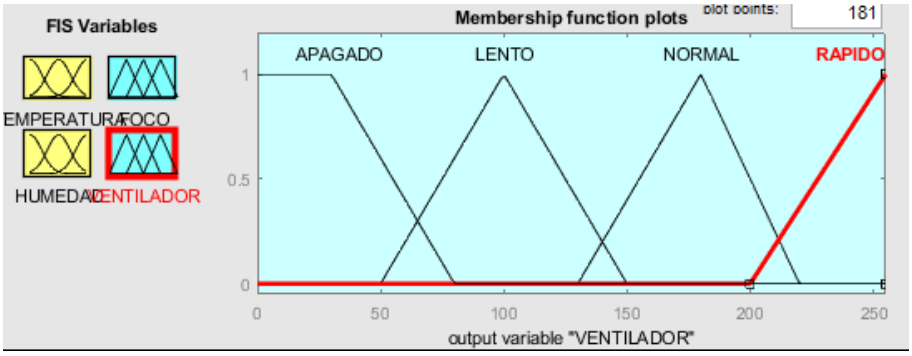

Figura 7. Estructura y rangos de funciones de membresía para salidas del controlador.

4.1.2 Base de conocimiento

Para la base de conocimientos nos referimos al punto de partida para la generación del conjunto de reglas sobre las que se rige la inferencia de nuestro controlador. Se describen los términos usados y se muestra la base en forma de matriz asociada.

Para el desarrollo de la base de conocimientos, fue necesario hacer una investigación profunda. Esta nos permitió que el controlador se base en la mejor información posible. Se desarrollaron dos bases de conocimientos distintas, ya que al tener dos salidas es necesario un control por cada una de ellas, aun cuando dependan de las mismas entradas, esto lo vemos en las tablas 1 y 2.

Tabla 1. Base de conocimientos Foco.

FOCO		TEMPERATURA		
		FRIO	OPTIMA	ALTA
HUMEDAD	SECO	BRILLANTE	MEDIO	OBSCURO
	REGULAR	MEDIO	OBSCURO	APAGADO
	HUMEDO	MEDIO	OBSCURO	APAGADO

Tabla 2. Base de conocimientos Ventilador.

VENTILADOR		TEMPERATURA		
		FRIO	OPTIMA	ALTA
HUMEDAD	SECO	APAGADO	LENTO	NORMAL
	REGULAR	LENTO	NORMAL	RAPIDO
	HUMEDO	LENTO	NORMAL	RAPIDO

De esta forma tenemos elaborado la base de reglas que son aplicadas al controlador. Para esto modificamos dentro de la función de fuzzy en la sección de "rule editor", las bases de la estructura de nuestro controlador. Cabe mencionar que estas reglas se definen en base a la lógica del funcionamiento del sistema, dictadas por el diseñador, pero reglamentadas por la situación. Las reglas de nuestro sistema se pueden observar en la figura 8.

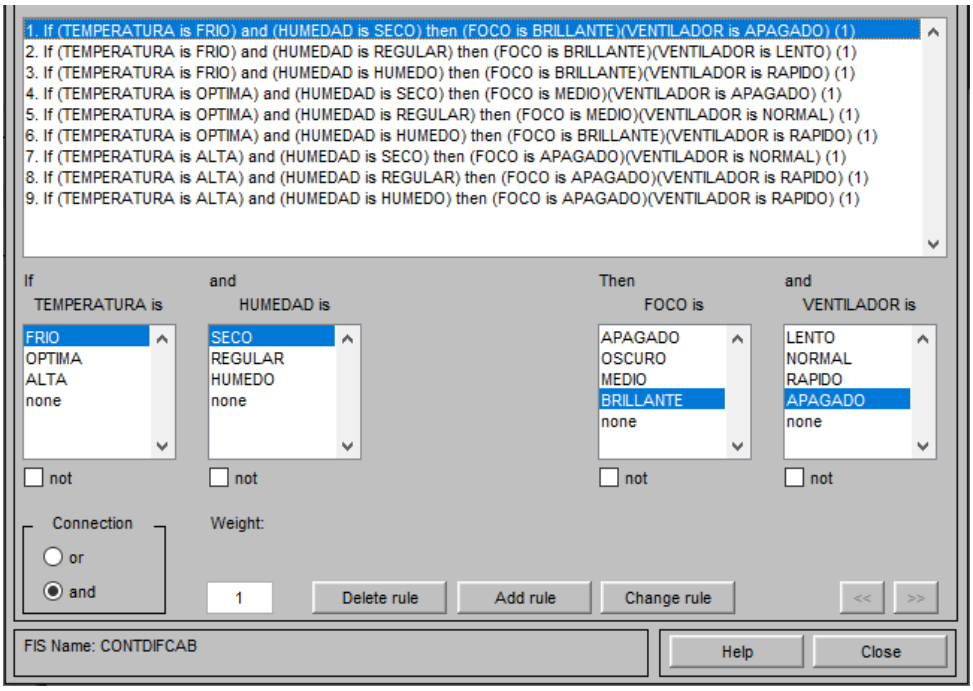

Figura 8. Reglas de controlador.

4.1.3 Proceso de inferencia

La función de fuzzy logic permite el proceso libre de inferencia utilizando el de mínimo - máximo, también conocido como Mamdani, ya que debido a sus características es de los métodos más apropiados para realizar un controlador compilado. En este método se presentan las siguientes cuestiones:

- **Valores de entrada:** la fuzificacion arroja valores de membresía para distintos subgrupos determinando parámetros representativos o cluster, en cada espacio de entrada.
- **Aplicación de regla**: las pertenencias que se dan se aplican sobre la base de reglas para saber en qué subgrupo del espacio de salida. A partir de ahí se producen ciertas combinaciones de antecedentes pertinentes a las reglas prescritas.
- **Membresías de salida:** Este valor de membresía que se agrega a los subgrupos, representa el valor mínimo de los espacios de entrada.
- **Formación del polígono:** a partir de que se aplican todas las reglas, pasamos a tener varios valores de membresía para un mismo subespacio de salida. Dado lo anterior se forman los polígonos de salida, el cual refleja el valor de las membresías a lo largo de los subgrupos. De ahí se toma siempre el valor máximo de membresía.

Lo mencionado anteriormente lo podemos observar mediante la pestaña de "view" en "rules". Aquí se nos muestra tanto las entradas con sus respectivos clusters o subgrupos como las salidas, con todos los valores que va tomando el controlador, se puede modificar el valor de las entradas

para observar el comportamiento a lo largo de todos sus valores y revelando lo que vamos a obtener en las salidas (Figura 9).

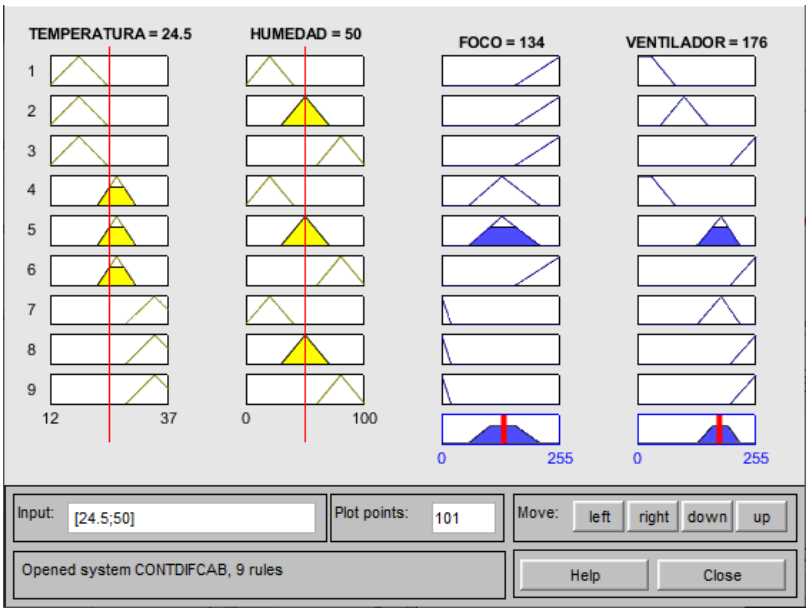

Figura 9. Vista de las variables de entrada y formación de polígono de salida.

También esto se ve reflejado a modo de superficie de control en un espacio tridimensional como se muestra en la figura 10:

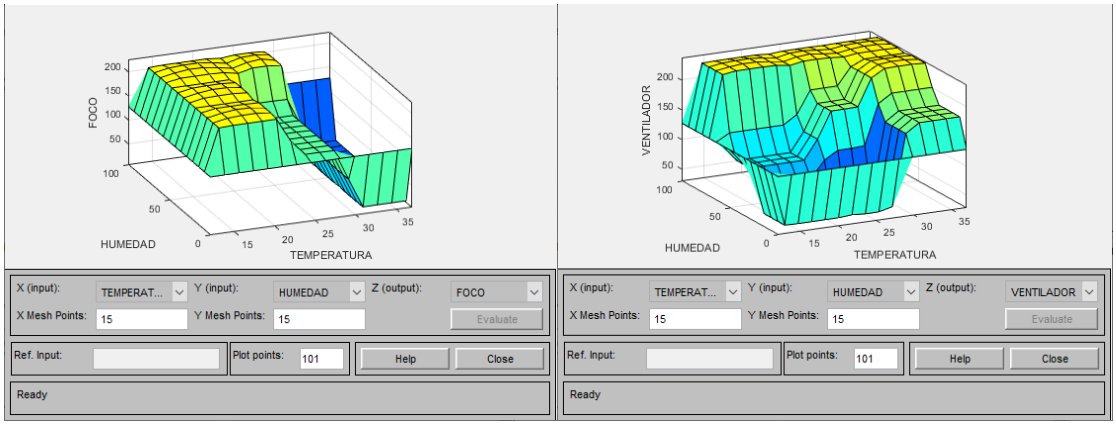

Figura 10. Vista tridimensional de las variables a partir del "Surface viewer".

De esta manera queda conformado el método de inferencia del máximo – mínimo Mamdani.

Como resultado del procedimiento descrito, se tiene conformado el controlador difuso. Dados los resultados de la simulación, se tiene seguridad que funciona de acuerdo a lo que se esperaba. Guardamos el archivo (. fis) para el procedimiento siguiente.

4.2 Transformación de archivo Matlab hacia archivo de Arduino

Existe la posibilidad de realizar la programación de cada pauta para que el controlador difuso se realice completamente mediante el programa de Matlab, pero presenta la desventaja que es necesario mantener la conexión directa del circuito de control a una computadora. Por este motivo se tomó la

decisión de hacer uso de una herramienta que está al alcance de cualquier persona. Se trata de la página web, *MakeProto*, que nos permite realizar la conversión de nuestro archivo FIS a un archivo INO, el cual se puede cargar a la placa de Arduino.

Para esto nos direccionamos en la página "MakeProto" en la sección "Arduino FIST: MATLAB Fuzzy Inference System to Arduino C Converter". Seleccionamos el archivo previamente guardado y realizamos la conversión, la página se puede ver en la figura 11.

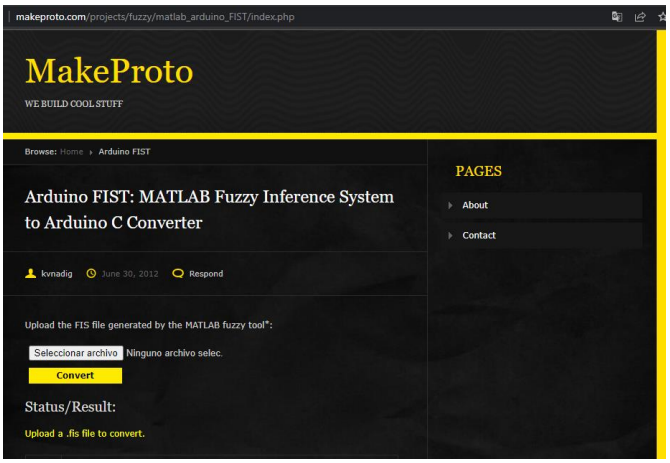

Figura 11. Página WEB para conversión de archivo.

Al terminar la conversión, automáticamente descarga una carpeta comprimida Zip con dos archivos uno (fis_header.h,), el cual es un código anexo en lenguaje C++ que contiene librerías que se debe agregar al código de Arduino para que lea correctamente lo escrito (figura 12). También un archivo fis de formato INO, el cual contiene todo el código escrito con las variables y las reglas que debe seguir el controlador difuso, tal como se ve en la figura 13.

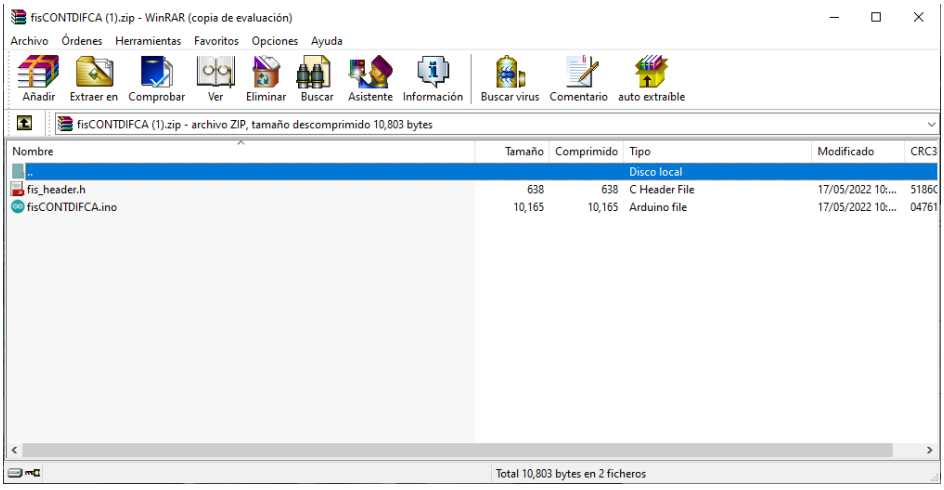

Figura 12. Carpeta de archivos .h y archivo .ino.

4.2.1| Modificación código Arduino

Debido a que, por obvias razones, el archivo se realizó a partir de una conversión, existen varios inconvenientes que se deben tomar en cuenta para que funcione de manera correcta.

fisCONTDIFCA564

```
//*********************************************************************************
// Matlab .fis to arduino C converter v2.0.1.25122016
// - Karthik Nadig, USA
// Please report bugs to:
// https://github.com/karthiknadig/ArduinoFIS/issues
// If you don't have a GitHub account mail to karthiknadig@gmail.com
//*********************************************************************************

#include "fis_header.h"

// Number of inputs to the fuzzy inference system
const int fis_gcI = 2;
// Number of outputs to the fuzzy inference system
const int fis_gcO = 2;
// Number of rules to the fuzzy inference system
const int fis_gcR = 9;

FIS_TYPE g_fisInput[fis_gcI];
FIS_TYPE g_fisOutput[fis_gcO];

// Setup routine runs once when you press reset:
void setup()
{
    // initialize the Analog pins for input.
    // Pin mode for Input: input1
    pinMode(0 , INPUT);
    // Pin mode for Input: input2
```

Figura 13. Programa convertido, muestra MakeProto.

Como se puede observar en la figura 13 el código incluye ciertos parámetros que deben ser modificados. Para comenzar es necesario agregar las líneas que vienen en el archivo (.h) continuo de descarga. Posterior a esto se agregan las librerías para el sensor de temperatura DHT.h y en nuestro caso debido a la LCD, es necesario la librería LiquidCrystal.h. Para que la librería del sensor de temperatura y humedad pueda identificarlo se define el tipo de sensor, para este caso que es un módulo DHT11, se agrega DHTPIN12 Y DHTTYPE DHT11.

Cabe resaltar que hay una pequeña alteración que impide que corra correctamente el programa y que es necesario modificar esta se encuentra en el *loop* que corre el programa donde se encuentran las señales de entrada. Es necesario modificar el analogRead por dth.readTemperature y para el caso de la humedad dht.readHumidity.

Nota: Es necesario que en el programa se encuentren descargadas las dos librerías que necesitamos de lo contrario, buscarlas en la sección de “programa” en el apartado incluir librería, o en su defecto realizar la instalación en la página de Arduino.

A continuación, definimos las constantes de entrada de la LCD, así como dos extensiones de las librerías DHT dht (DHTPIN, DHTTYPE) y LiquidCrystal. Ya no es necesario agregar las de entrada ni salida debido a que en la conversión ya se nos proporciona, solo el número del pin al cual se conectaran en el Arduino (figura 14).

fisCONTDIFCA

```
#define FIS_TYPE float
#define FIS_RESOLUSION 101
#define FIS_MIN -3.4028235E+38
#define FIS_MAX 3.4028235E+38
typedef FIS_TYPE(*_FIS_MF)(FIS_TYPE, FIS_TYPE*);
typedef FIS_TYPE(*_FIS_ARR_OP)(FIS_TYPE, FIS_TYPE);
typedef FIS_TYPE(*_FIS_ARR)(FIS_TYPE*, int, _FIS_ARR_OP);
#include <DHT.h>
#include <LiquidCrystal.h>

#define DHTPIN 12
#define DHTTYPE DHT11

const int rs = 8, en = 9, d4 = 4, d5 = 5, d6 = 6 , d7 = 7;

DHT dht(DHTPIN,DHTTYPE);
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

// Number of inputs to the fuzzy inference system
const int fis_gcI = 2;
// Number of outputs to the fuzzy inference system
const int fis_gcO = 2;
// Number of rules to the fuzzy inference system
const int fis_gcR = 9;

FIS_TYPE g_fisInput[fis_gcI];
```

Figura 14. Modificación de librerías y constantes.

Prosiguiendo con las modificaciones, es necesario escribir las líneas del código que van dirigidas al LCD, para que nos muestre en la pantalla, tanto los valores de entrada que va a tomar del sensor de temperatura y humedad, así como los que ira arrojando de acuerdo a nuestro controlador de las salidas que son el foco y el ventilador. Esto se puede observar en la figura 15.

Las lecturas también las podemos observar en la pestaña de herramienta en el monitor serie del programa de Arduino.

```
    lcd.clear();
    lcd.setCursor(0,0);
    lcd.print("I>");
    lcd.setCursor(3, 0);
    lcd.print("T:");
    lcd.setCursor(5, 0);
    lcd.print(g_fisInput[0]);
    lcd.setCursor(8, 0);
    lcd.print("C");
    lcd.setCursor(9, 0);
    lcd.print("H:");
    lcd.setCursor(11, 0);
    lcd.print(g_fisInput[1]);
    lcd.setCursor(16, 0);
    lcd.print("%");

    lcd.setCursor(0, 1);
    lcd.print("O<");
    lcd.setCursor(3, 1);
    lcd.print("V:");
    lcd.setCursor(5, 1);
    lcd.print(g_fisOutput[1]);
    lcd.setCursor(9, 1);
    lcd.print("F:");
    lcd.setCursor(11, 1);
    lcd.print(g_fisOutput[0]);

    Serial.println("_____");
    Serial.print("temperatura:");
    Serial.println(g_fisInput[0]);
    Serial.print("humedad:");
    Serial.println(g_fisInput[1]);
    Serial.print("Foco:");
```

Figura 15. Líneas de código para lectura de LCD.

Terminando con esto, se realiza la compilación del archivo para verificar que el texto no tenga errores gramaticales o de algún otro tipo que se puedan presentar. Se realiza la carga del programa en la tableta de Arduino para verificar físicamente el funcionamiento (figura 16).

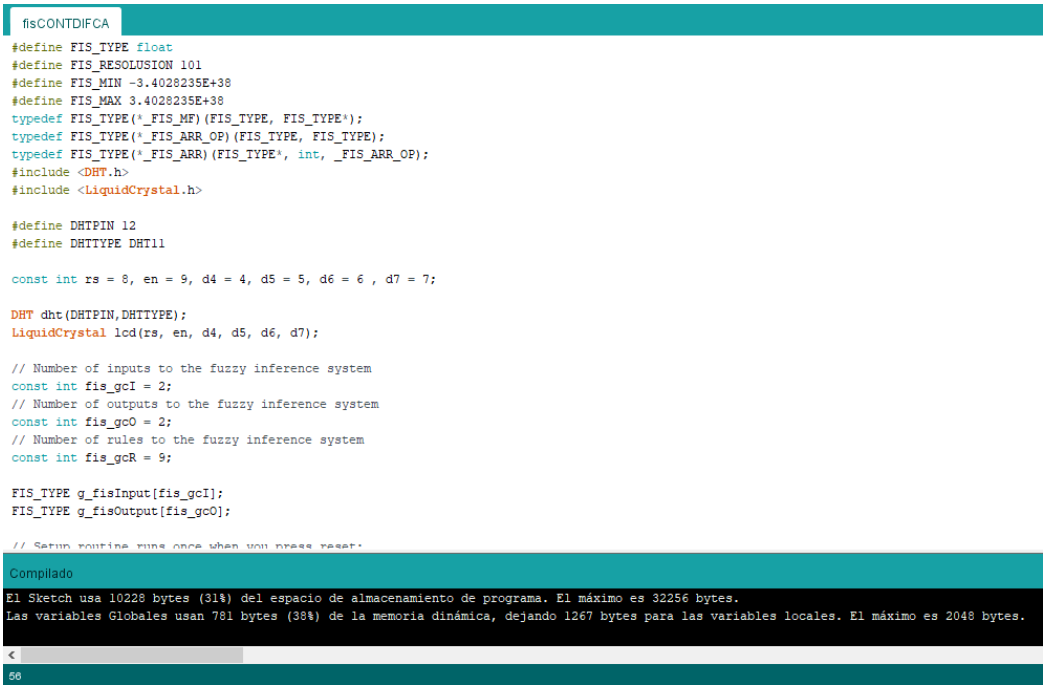

Figura 16. Archivo Arduino compilado.

5 RESULTADOS

El controlador en conjunto funciona relativamente bien, como en todo proyecto hay ciertos factores con las cuales se ve alterado el perfecto funcionamiento. Al ser señales digitales que provienen del Arduino, con un mínimo de interferencia nos modifican las salidas esperadas. Pero dentro de estas condiciones el sistema reacciona bien.

5.1 LECTURAS DEL SENSOR

Las lecturas del módulo de sensor DHT11 mostradas en el LCD, son precisas, claro como antes mencionado hay ciertas dificultades técnicas como lo son los cables, el ruido, la posición entre otras, que en ciertos momentos alteran la señal y esto tiene que modificarse para que vuelva a trabajar de forma correcta.

La forma de realizar el controlador y hacer la conversión del archivo nos permite una ventaja considerable con respecto a hacerlo en el programa de Matlab, debido a que al estar programado la tarjeta de Arduino, no es necesario contar con una conexión para lectura entre el programa Matlab y el programa Arduino y a su vez con el circuito. Con la el programa ya compilado y subido en la tabla de Arduino podemos verificar y comparar con el controlador de Matlab que los valores van modificando de igual manera y que funciona correctamente, en las figuras 17-23 se puede observar el circuito armado y funcional del simulador de nuestra cabina de pintura.

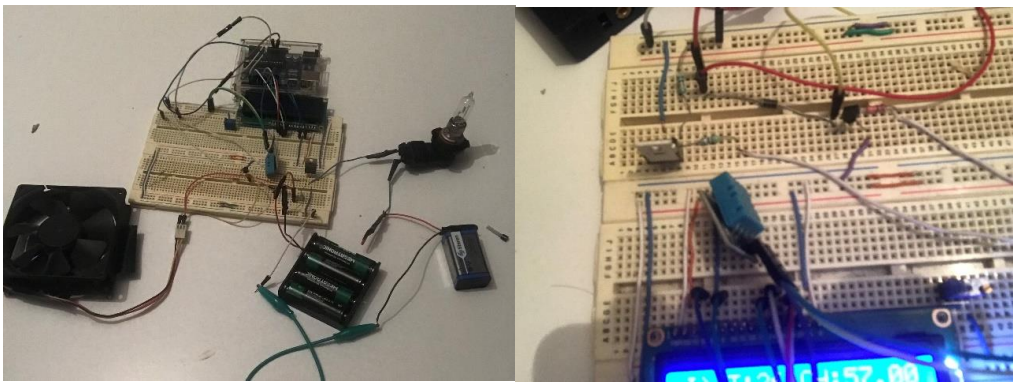

Figura 17. Circuito físico para control de simulación de cabina de pintura.

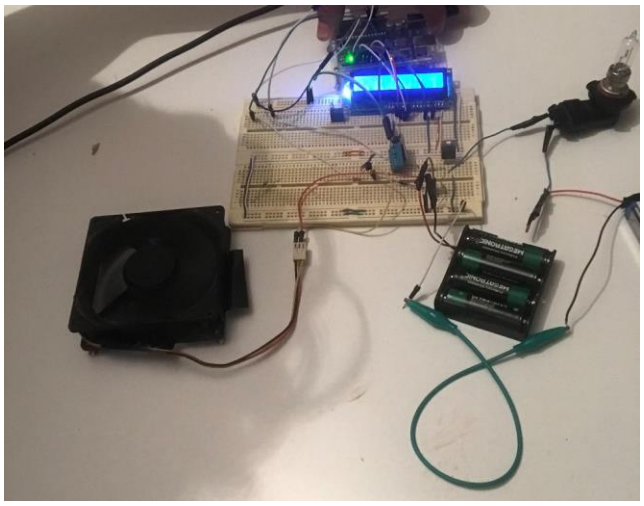

Figura 18. Circuito físico, controlador difuso C1.

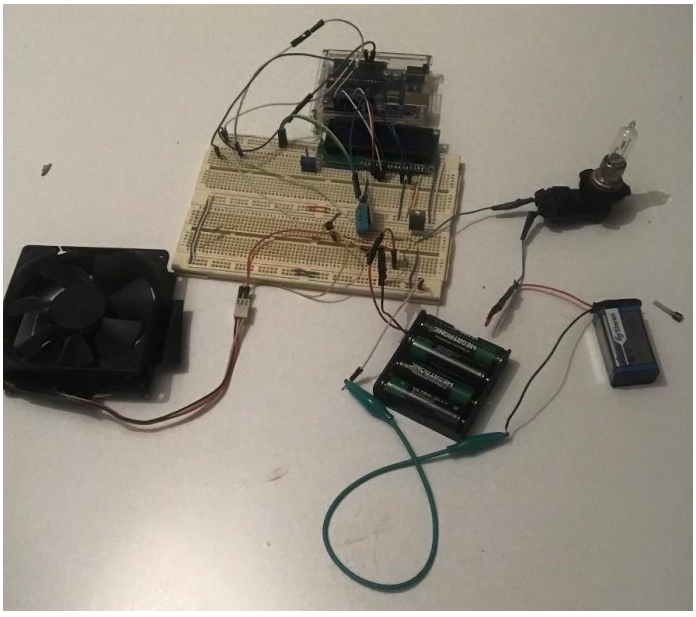

Figura 19. Circuito físico, controlador difuso C2.

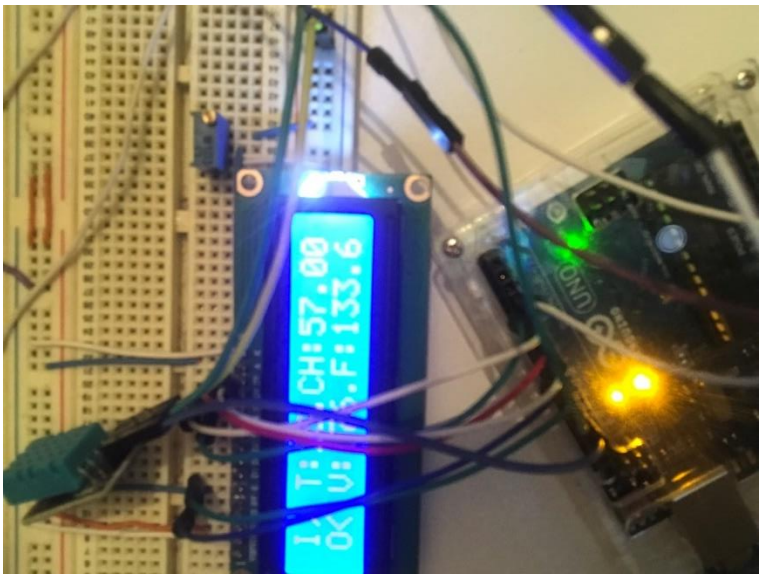

Figura 20. Circuito físico, controlador difuso C3.

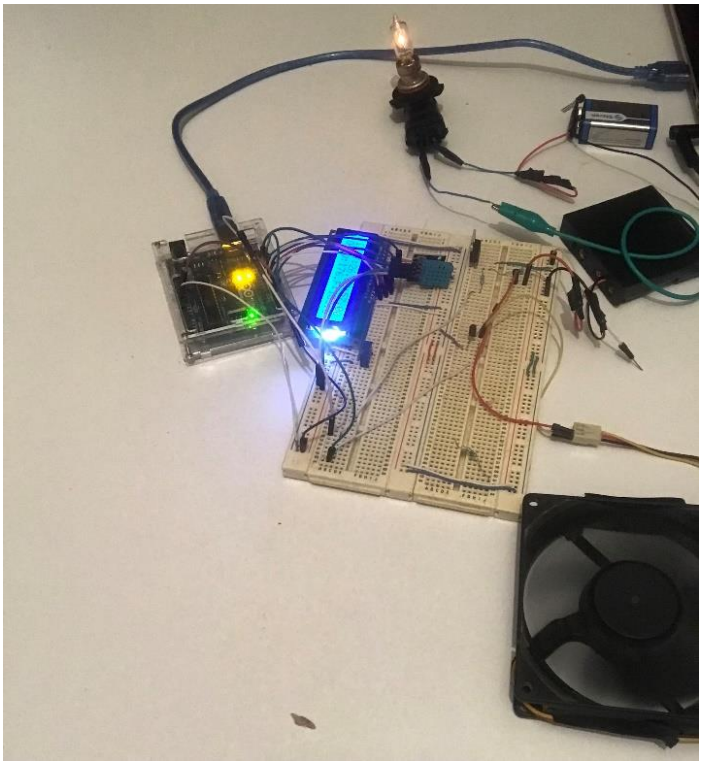

Figura 21. Circuito físico, controlador difuso C4.

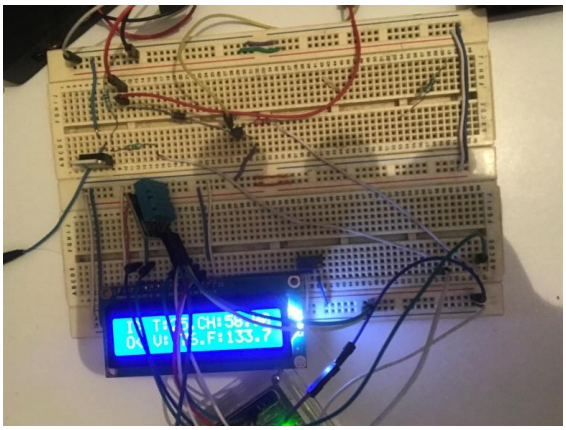

Figura 22. Circuito físico, controlador difuso C5.

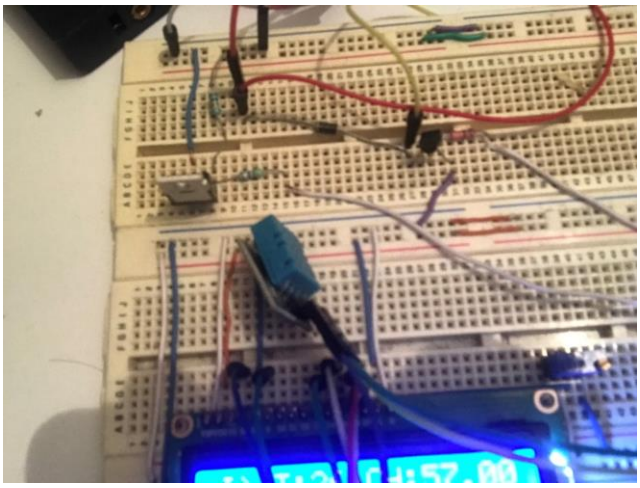

Figura 23. Circuito físico, controlador difuso C6.

6 CONCLUSIONES

A raíz de la realización de este proyecto y en base a los resultados podemos considerar que la implementación de un controlador difuso radica en cómo se plasman las normas o decisiones que se desean tomar para que un sistema como el propuesto reaccione de manera adecuada. En el sistema se deben considerar una gran cantidad de factores y variables para que tanto en las entradas, salidas y el proceso de nuestro controlador exista coherencia para logran un resultado semejante al esperado.

El objetivo de este proyecto se cumplió al poder desarrollar desde el circuito físico, esquemático hasta el código para el controlador, contando con resultados favorecedores. A pesar de ser un proyecto con limitaciones es imprescindible que aun existan mejoras que se le pueden hacer, teniendo en consideración esto, el controlador sustenta las bases para posibles modificaciones de tanto el circuito como de la forma que se enfoca el control. Así mismo con la realización de este proyecto se encontró una idea para que se desarrolle el controlador sin la necesidad de conexión a una computadora, permitiendo tomar de referencia el desarrollo del mismo y modificarlo para proyectos diferentes, pero con la misma base de lógica difusa.

BIBLIOGRAFÍA

Castillo, L. T. , & Aguilar, “Fuzzy Control for Wheeled Mobile Robots,” in Type-2 Fuzzy Logic in Control of Nonsmooth Systems., 2019, pp. 85–96.

Larsen, P. M. (1980). Industrial applications of fuzzy logic control. International Journal of Man-Machine Studies, 12(1), 3-10.

Peri, V. M., & Simon, D. (2005, June). Fuzzy logic control for an autonomous robot. In NAFIPS 2005-2005 Annual Meeting of the North American Fuzzy Information Processing Society (pp. 337-342). IEEE.

Torres, A., & Nieto, J. J. (2006). Fuzzy logic in medicine and bioinformatics. journal of Biomedicine and Biotechnology, 2006.

Uddin, M. N., Radwan, T. S., & Rahman, M. A. (2002). Performances of fuzzy-logic-based indirect vector control for induction motor drive. IEEE Transactions on Industry applications, 38(5), 1219-1225.

Pau, G., & Salerno, “Wireless Sensor Networks for Smart Homes: A Fuzzy-Based Solution for an Energy-Effective Duty Cycle,” Electronics., vol. 8, no. 2, p. 131, 2019.

Ramadhan, A. Sudiarso, and M. Mahardika, “Design and Simulation of Auto-Shifting System in Manual Transmission Gearbox for Electric Race Car with Fuzzy Logic Controller on Matlab-Simulink,” J. Phys. Conf. Ser., 2019.

Sharma, S., Agrawal, K.& Bandopadhaya, S. Springer, “Fuzzy Logic Controlled Variable Frequency Drives,” Harmon. Search Nat. Inspired Optim. Algorithms, pp. 1153–1164, 2019.

Zadeh, L. A. (1965). Information and control. Fuzzy sets, 8(3), 338-353.

Zadeh, L. A. (1996). Fuzzy sets. In Fuzzy sets, fuzzy logic, and fuzzy systems: selected papers by Lotfi A Zadeh (pp. 394-432).

N. F. Jamin, N. M. A. Ghani, Z. Ibrahim, M. F. Masrom, N. A. A. Razali, and A. M. Almeshal, “Two-wheeled wheelchair stabilization using interval type-2 fuzzy logic controller,” Int. J. Simul. Syst. Sci. Technol., 2018.

Apéndice A: Código de Arduino

```
#define FIS_TYPE float
#define FIS_RESOLUSION 101
#define FIS_MIN -3.4028235E+38
#define FIS_MAX 3.4028235E+38
typedef FIS_TYPE(*_FIS_MF)(FIS_TYPE, FIS_TYPE*);
typedef FIS_TYPE(*_FIS_ARR_OP)(FIS_TYPE, FIS_TYPE);
typedef FIS_TYPE(*_FIS_ARR)(FIS_TYPE*, int, _FIS_ARR_OP);
#include <DHT.h>
#include <LiquidCrystal.h>
#define DHTPIN 12
#define DHTTYPE DHT11
const int rs = 8, en = 9, d4 = 4, d5 = 5, d6 = 6 , d7 = 7;
DHT dht(DHTPIN,DHTTYPE);
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);
// Number of inputs to the fuzzy inference system
const int fis_gcI = 2;
// Number of outputs to the fuzzy inference system
const int fis_gcO = 2;
// Number of rules to the fuzzy inference system
const int fis_gcR = 9;
FIS_TYPE g_fisInput[fis_gcI];
FIS_TYPE g_fisOutput[fis_gcO];
// Setup routine runs once when you press reset:
void setup()
{
Serial.begin(9600);
lcd.begin(20,4);
// initialize the Analog pins for input.
// Pin mode for Input: temperatura
// Pin mode for Input: humedad
pinMode(12 , INPUT);
//pinMode(1 , INPUT);
// initialize the Analog pins for output.
// Pin mode for Output: FOCO
pinMode(3 , OUTPUT);
// Pin mode for Output: VENTILADOR
pinMode(11 , OUTPUT);
dht.begin();
}
// Loop routine runs over and over again forever:
void loop()
{
// Read Input: temperatura
g_fisInput[0] = dht.readTemperature();
// Read Input: humedad
g_fisInput[1] = dht.readHumidity();
g_fisOutput[0] = 0;
```

```
48. g_fisOutput[1] = 0;
49. fis_evaluate();
50. // Set output vlaue: foco
51. digitalWrite(3 , g_fisOutput[0]);
52. // Set output vlaue: ventilador
53. digitalWrite(11 , g_fisOutput[1]);
54. lcd.clear();
55. lcd.setCursor(0,0);
56. lcd.print("I>");
57. lcd.setCursor(3, 0);
58. lcd.print("T:");
59. lcd.setCursor(5, 0);
60. lcd.print(g_fisInput[0]);
61. lcd.setCursor(8, 0);
62. lcd.print("C");
63. lcd.setCursor(9, 0);
64. lcd.print("H:");
65. lcd.setCursor(11, 0);
66. lcd.print(g_fisInput[1]);
67. lcd.setCursor(16, 0);
68. lcd.print("%");
69. lcd.setCursor(0, 1);
70. lcd.print("O<");
71. lcd.setCursor(3, 1);
72. lcd.print("V:");
73. lcd.setCursor(5, 1);
74. lcd.print(g_fisOutput[1]);
75. lcd.setCursor(9, 1);
76. lcd.print("F:");
77. lcd.setCursor(11, 1);
78. lcd.print(g_fisOutput[0]);
79. Serial.println("_____");
80. Serial.print("temperatura:");
81. Serial.println(g_fisInput[0]);
82. Serial.print("humedad:");
83. Serial.println(g_fisInput[1]);
84. Serial.print("Foco:");
85. Serial.println(g_fisOutput[0]);
86. Serial.print("Ventilador:");
87. Serial.println(g_fisOutput[1]);
88. }
89. //************************************************************************
90. // Support functions for Fuzzy Inference System
91. //************************************************************************
92. // Triangular Member Function
93. FIS_TYPE fis_trimf(FIS_TYPE x, FIS_TYPE* p)
94. {
95. FIS_TYPE a = p[0], b = p[1], c = p[2];
96. FIS_TYPE t1 = (x - a) / (b - a);
```

```
97. FIS_TYPE t2 = (c - x) / (c - b);
98. if ((a == b) && (b == c)) return (FIS_TYPE) (x == a);
99. if (a == b) return (FIS_TYPE) (t2*(b <= x)*(x <= c));
100.        if (b == c) return (FIS_TYPE) (t1*(a <= x)*(x <= b));
101.        t1 = min(t1, t2);
102.        return (FIS_TYPE) max(t1, 0);
103.        }
104.        // Trapezoidal Member Function
105.        FIS_TYPE fis_trapmf(FIS_TYPE x, FIS_TYPE* p)
106.        {
107.        FIS_TYPE a = p[0], b = p[1], c = p[2], d = p[3];
108.        FIS_TYPE t1 = ((x <= c) ? 1 : ((d < x) ? 0 : ((c != d) ? ((d - x) / (d - c)) : 0)));
109.        FIS_TYPE t2 = ((b <= x) ? 1 : ((x < a) ? 0 : ((a != b) ? ((x - a) / (b - a)) : 0)));
110.        return (FIS_TYPE) min(t1, t2);
111.        }
112.        FIS_TYPE fis_min(FIS_TYPE a, FIS_TYPE b)
113.        {
114.        return min(a, b);
115.        }
116.        FIS_TYPE fis_max(FIS_TYPE a, FIS_TYPE b)
117.        {
118.        return max(a, b);
119.        }
120.        FIS_TYPE fis_array_operation(FIS_TYPE *array, int size, _FIS_ARR_OP pfnOp)
121.        {
122.        int i;
123.        FIS_TYPE ret = 0;
124.        if (size == 0) return ret;
125.        if (size == 1) return array[0];
126.        ret = array[0];
127.        for (i = 1; i < size; i++)
128.        {
129.        ret = (*pfnOp)(ret, array[i]);
130.        }
131.        return ret;
132.        }
133.        //************************************************************************
    *****
134.        // Data for Fuzzy Inference System
135.        //************************************************************************
    *****
136.        // Pointers to the implementations of member functions
137.        _FIS_MF fis_gMF[] =
138.        {
139.        fis_trimf, fis_trapmf
140.        };
```

```
141.        // Count of member function for each Input
142.        int fis_gIMFCount[] = { 3, 3 };
143.        // Count of member function for each Output
144.        int fis_gOMFCount[] = { 4, 4 };
145.        // Coefficients for the Input Member Functions
146.        FIS_TYPE fis_gMFI0Coeff1[] = { 12, 18, 24 };
147.        FIS_TYPE fis_gMFI0Coeff2[] = { 22, 26, 30 };
148.        FIS_TYPE fis_gMFI0Coeff3[] = { 28, 34, 40 };
149.        FIS_TYPE* fis_gMFI0Coeff[] = { fis_gMFI0Coeff1, fis_gMFI0Coeff2,
    fis_gMFI0Coeff3 };
150.        FIS_TYPE fis_gMFI1Coeff1[] = { 0, 20, 40 };
151.        FIS_TYPE fis_gMFI1Coeff2[] = { 30, 50, 70 };
152.        FIS_TYPE fis_gMFI1Coeff3[] = { 60, 80, 100 };
153.        FIS_TYPE* fis_gMFI1Coeff[] = { fis_gMFI1Coeff1, fis_gMFI1Coeff2,
    fis_gMFI1Coeff3 };
154.        FIS_TYPE** fis_gMFICoeff[] = { fis_gMFI0Coeff, fis_gMFI1Coeff };
155.        // Coefficients for the Output Member Functions
156.        FIS_TYPE fis_gMFO0Coeff1[] = { 0, 0, 20 };
157.        FIS_TYPE fis_gMFO0Coeff2[] = { 0, 50, 100 };
158.        FIS_TYPE fis_gMFO0Coeff3[] = { 60, 130, 210 };
159.        FIS_TYPE fis_gMFO0Coeff4[] = { 160, 255, 255 };
160.        FIS_TYPE* fis_gMFO0Coeff[] = { fis_gMFO0Coeff1, fis_gMFO0Coeff2,
    fis_gMFO0Coeff3, fis_gMFO0Coeff4 };
161.        FIS_TYPE fis_gMFO1Coeff1[] = { 50, 100, 150 };
162.        FIS_TYPE fis_gMFO1Coeff2[] = { 130, 180, 220 };
163.        FIS_TYPE fis_gMFO1Coeff3[] = { 200, 255, 255 };
164.        FIS_TYPE fis_gMFO1Coeff4[] = { 0, 0, 30, 80 };
165.        FIS_TYPE* fis_gMFO1Coeff[] = { fis_gMFO1Coeff1, fis_gMFO1Coeff2,
    fis_gMFO1Coeff3, fis_gMFO1Coeff4 };
166.        FIS_TYPE** fis_gMFOCoeff[] = { fis_gMFO0Coeff, fis_gMFO1Coeff };
167.        // Input membership function set
168.        int fis_gMFI0[] = { 0, 0, 0 };
169.        int fis_gMFI1[] = { 0, 0, 0 };
170.        int* fis_gMFI[] = { fis_gMFI0, fis_gMFI1};
171.        // Output membership function set
172.        int fis_gMFO0[] = { 0, 0, 0, 0 };
173.        int fis_gMFO1[] = { 0, 0, 0, 1 };
174.        int* fis_gMFO[] = { fis_gMFO0, fis_gMFO1};
175.        // Rule Weights
176.        FIS_TYPE fis_gRWeight[] = { 1, 1, 1, 1, 1, 1, 1, 1, 1 };
177.        // Rule Type
178.        int fis_gRType[] = { 1, 1, 1, 1, 1, 1, 1, 1, 1 };
179.        // Rule Inputs
180.        int fis_gRI0[] = { 1, 1 };
181.        int fis_gRI1[] = { 1, 2 };
182.        int fis_gRI2[] = { 1, 3 };
183.        int fis_gRI3[] = { 2, 1 };
184.        int fis_gRI4[] = { 2, 2 };
185.        int fis_gRI5[] = { 2, 3 };
```

```
186.        int fis_gRI6[] = { 3, 1 };
187.        int fis_gRI7[] = { 3, 2 };
188.        int fis_gRI8[] = { 3, 3 };
189.        int* fis_gRI[] = { fis_gRI0, fis_gRI1, fis_gRI2, fis_gRI3, fis_gRI4, fis_gRI5,
    fis_gRI6, fis_gRI7, fis_gRI8 };
190.        // Rule Outputs
191.        int fis_gRO0[] = { 4, 4 };
192.        int fis_gRO1[] = { 4, 1 };
193.        int fis_gRO2[] = { 4, 3 };
194.        int fis_gRO3[] = { 3, 4 };
195.        int fis_gRO4[] = { 3, 2 };
196.        int fis_gRO5[] = { 4, 3 };
197.        int fis_gRO6[] = { 1, 2 };
198.        int fis_gRO7[] = { 1, 3 };
199.        int fis_gRO8[] = { 1, 3 };
200.        int* fis_gRO[] = { fis_gRO0, fis_gRO1, fis_gRO2, fis_gRO3, fis_gRO4, fis_gRO5,
    fis_gRO6, fis_gRO7, fis_gRO8 };
201.        // Input range Min
202.        FIS_TYPE fis_gIMin[] = { 12, 0 };
203.        // Input range Max
204.        FIS_TYPE fis_gIMax[] = { 37, 100 };
205.        // Output range Min
206.        FIS_TYPE fis_gOMin[] = { 0, 0 };
207.        // Output range Max
208.        FIS_TYPE fis_gOMax[] = { 255, 255 };
209.        //*********************************************************************
    *****
210.        // Data dependent support functions for Fuzzy Inference System
211.        //*********************************************************************
    *****
212.        FIS_TYPE fis_MF_out(FIS_TYPE** fuzzyRuleSet, FIS_TYPE x, int o)
213.        {
214.        FIS_TYPE mfOut;
215.        int r;
216.        for (r = 0; r < fis_gcR; ++r)
217.        {
218.        int index = fis_gRO[r][o];
219.        if (index > 0)
220.        {
221.        index = index - 1;
222.        mfOut = (fis_gMF[fis_gMFO[o][index]])(x, fis_gMFOCoeff[o][index]);
223.        }
224.        else if (index < 0)
225.        {
226.        index = -index - 1;
227.        mfOut = 1 - (fis_gMF[fis_gMFO[o][index]])(x, fis_gMFOCoeff[o][index]);
228.        }
229.        else
230.        {
```

```
231.        mfOut = 0;
232.        }
233.        fuzzyRuleSet[0][r] = fis_min(mfOut, fuzzyRuleSet[1][r]);
234.        }
235.        return fis_array_operation(fuzzyRuleSet[0], fis_gcR, fis_max);
236.        }
237.        FIS_TYPE fis_defuzz_centroid(FIS_TYPE** fuzzyRuleSet, int o)
238.        {
239.        FIS_TYPE step = (fis_gOMax[o] - fis_gOMin[o]) / (FIS_RESOLUSION - 1);
240.        FIS_TYPE area = 0;
241.        FIS_TYPE momentum = 0;
242.        FIS_TYPE dist, slice;
243.        int i;
244.        // calculate the area under the curve formed by the MF outputs
245.        for (i = 0; i < FIS_RESOLUSION; ++i){
246.        dist = fis_gOMin[o] + (step * i);
247.        slice = step * fis_MF_out(fuzzyRuleSet, dist, o);
248.        area += slice;
249.        momentum += slice*dist;
250.        }
251.        return ((area == 0) ? ((fis_gOMax[o] + fis_gOMin[o]) / 2) : (momentum / area));
252.        }
253.        //************************************************************************
    *****
254.        // Fuzzy Inference System
255.        //************************************************************************
    *****
256.        void fis_evaluate()
257.        {
258.        FIS_TYPE fuzzyInput0[] = { 0, 0, 0 };
259.        FIS_TYPE fuzzyInput1[] = { 0, 0, 0 };
260.        FIS_TYPE* fuzzyInput[fis_gcI] = { fuzzyInput0, fuzzyInput1, };
261.        FIS_TYPE fuzzyOutput0[] = { 0, 0, 0, 0 };
262.        FIS_TYPE fuzzyOutput1[] = { 0, 0, 0, 0 };
263.        FIS_TYPE* fuzzyOutput[fis_gcO] = { fuzzyOutput0, fuzzyOutput1, };
264.        FIS_TYPE fuzzyRules[fis_gcR] = { 0 };
265.        FIS_TYPE fuzzyFires[fis_gcR] = { 0 };
266.        FIS_TYPE* fuzzyRuleSet[] = { fuzzyRules, fuzzyFires };
267.        FIS_TYPE sW = 0;
268.        // Transforming input to fuzzy Input
269.        int i, j, r, o;
270.        for (i = 0; i < fis_gcI; ++i)
271.        {
272.        for (j = 0; j < fis_gIMFCount[i]; ++j)
273.        {
274.        fuzzyInput[i][j] =
        a.  (fis_gMF[fis_gMFI[i][j]])(g_fisInput[i], fis_gMFICoeff[i][j]);
275.        }
276.        }
```

```
277.        int index = 0;
278.        for (r = 0; r < fis_gcR; ++r)
279.        {
280.        if (fis_gRType[r] == 1)
281.        {
282.        fuzzyFires[r] = FIS_MAX;
283.        for (i = 0; i < fis_gcI; ++i)
284.        {
        a.  index = fis_gRI[r][i];
        b.  if (index > 0)
                i.  fuzzyFires[r] = fis_min(fuzzyFires[r], fuzzyInput[i][index - 1]);
        c.  else if (index < 0)
                i.  fuzzyFires[r] = fis_min(fuzzyFires[r], 1 - fuzzyInput[i][-index - 1]);
        d.  else
                i.  fuzzyFires[r] = fis_min(fuzzyFires[r], 1);
285.        }
286.        }
287.        else
288.        {
289.        fuzzyFires[r] = FIS_MIN;
290.        for (i = 0; i < fis_gcI; ++i)
291.        {
        a.  index = fis_gRI[r][i];
        b.  if (index > 0)
                i.  fuzzyFires[r] = fis_max(fuzzyFires[r], fuzzyInput[i][index - 1]);
        c.  else if (index < 0)
                i.  fuzzyFires[r] = fis_max(fuzzyFires[r], 1 - fuzzyInput[i][-index - 1]);
        d.  else
                i.  fuzzyFires[r] = fis_max(fuzzyFires[r], 0);
292.        }
293.        }
294.        fuzzyFires[r] = fis_gRWeight[r] * fuzzyFires[r];
295.        sW += fuzzyFires[r];
296.        }
297.        if (sW == 0)
298.        {
299.        for (o = 0; o < fis_gcO; ++o)
300.        {
301.        g_fisOutput[o] = ((fis_gOMax[o] + fis_gOMin[o]) / 2);
302.        }
303.        }
304.        else
305.        {
306.        for (o = 0; o < fis_gcO; ++o)
307.        {
308.        g_fisOutput[o] = fis_defuzz_centroid(fuzzyRuleSet, o);
309.        }
310.        }
311.        }
```