[[1]](#footnote-1)

Proyecto Inteligencia Artificial carrito seguidor de línea resuelve laberintos por lógica difusa

Suarez L. Juan Diego1

Juan.suarezl@usantoto.edu.co

*Abstrac: Along of the human being history had been multiple tries for emulate the nature in devices (electric or mechanical), in an effort for improve them and in consequence improve our life style. One of them is our brain behavior that provides a good approach of how to make devices with learn capacity, in other words, solve a lot of different kind of tasks with base in experience method.*

*Key word. Fuzzy logic, Artificial intelligence, membership function, Arduino, MATLAB.*

# INTRODUCTION

E

N Antaño al mencionar la palabra rotica, llegaba a la mente una imagen de seres antropomórficos o con taxonomía animal impulsados por electricidad construidos por el hombre, los cuales, imitaban el comportamiento de su análogo en la naturaleza para realizar diferentes tareas. Estos seres llegaban al tal punto de poder adquirir una capacidad de razonamiento y conciencia de sí mismos. Hoy en día, esta forma de percibir la robótica no es más una simple cuestión de ciencia ficción. Se han realizado investigaciones profundas sobre el aprendizaje de los robots para que estos realicen tareas de una manera más “intuitiva”, imitando así la forma en que un humano aprende a desenvolverse en diferentes tareas, pero con una velocidad de adaptación muchísimo mejor. El presente informe se basa en la elaboración de un carrito seguidor de línea para la asignatura Inteligencia Artificial, el cual tendrá por objetivo resolver un laberinto mediante lógica difusa.

# marco teórico

### Lógica difusa:

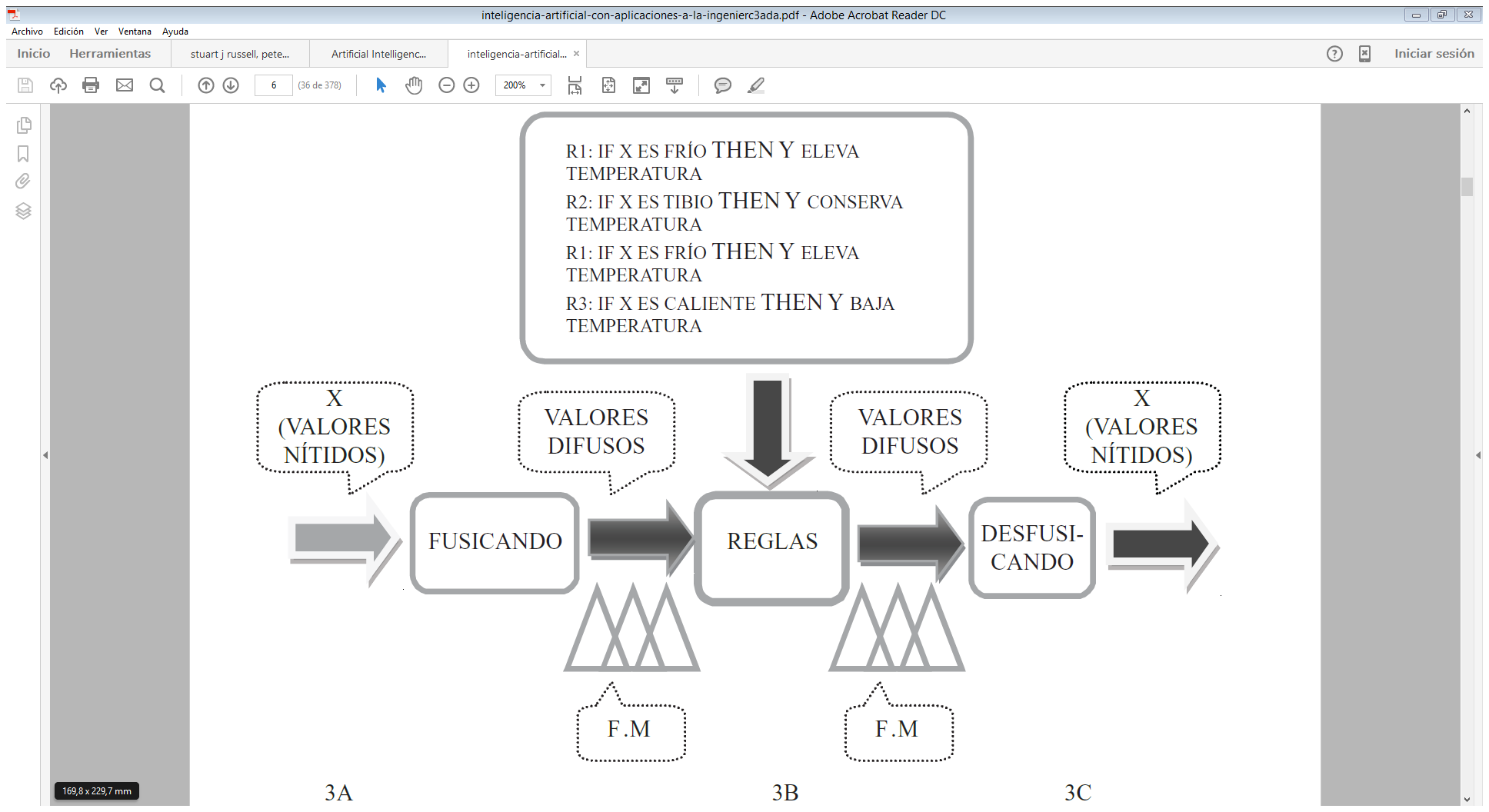
### La lógica difusa es una rama de la IA que le permite a una computadora analizar información del mundo real en una escala entre lo falso y verdadero. Esta metodología proporciona una manera simple de obtener una conclusión a partir de información de entrada, la cual puede ser vaga, ambigua, imprecisa, con ruido o incompleta. Esta se basa en como una persona toma una decisión basada en información con las anteriores características. Esta tecnología tiene la

### ventaja de poder ser implementada tanto en hardware como en software.

La lógica difusa consta de tres etapas para obtener el resultado deseado. La primera etapa se basa en un proceso donde las variables tienen un grado de incertidumbre metalingüístico. Por lo tanto, el rango de valores (universo de discurso) de cada variable puede clasificarse por conjuntos difusos, por ejemplo, baja, media, alta. Cuando los sensores miden las variables, los valores pasan a un proceso de fusificación que consiste en pasar dichos valores a un rango de pertenencia entre cero (0) y uno (1). Se busca determinar en qué grado el valor que se está adquiriendo pertenece a un conjunto difuso. Los conjuntos difusos son caracterizados mediante funciones de membresía, las cuales están sintonizadas al punto de operación adecuado para el funcionamiento del sistema.

En la segunda etapa se proponen reglas lingüísticas (inferencia) que servirán de guía para que el sistema se comporte de manera más adecuada, idónea o deseada según el modelo de referencia o los objetivos del usuario. El grado de pertenencia de cada una de las variables se evalúa en un conjunto de reglas de inferencia. Dichas reglas de inferencia fueron determinadas con ayuda de un experto. El conjunto de reglas de inferencia determina una consecuencia, es decir, asigna un grado de pertenencia a un conjunto difuso que caracteriza a las salidas.

Una vez obtenidas las consecuencias, la tercera etapa es un proceso para determinar los valores óptimos de salida, conocido como desfusificación, y que consiste en pasar el grado de pertenencia, proveniente de la consecuencia de la regla de inferencia, a un valor nítido o real. Para hacer eso, previamente se sintonizaron funciones de membresía de cada una de las salidas con el fin de obtener un valor cuantificable. La Figura 1 muestra el diagrama esquemático del controlador difuso.



**Figura 1.** El diagrama de bloques para desarrollar la metodología de lógica difusa muestra las tres etapas que constituyen el control. Tomando como entradas X (3A), se lleva a cabo el proceso conforme al método (3B), obteniendo una respuesta de salidas Y (3C).

Al final el control entregará valores nítidos o reales, consecuencia de las reglas lingüísticas previamente estructuradas, con lo cual este sistema interpretará las órdenes y realizará las acciones pertinentes.

Para entender la lógica difusa primero es indispensable establecer cierta nomenclatura y terminología. Cuando se habla de conjuntos nítidos, la variable típica a usar es la X. En conjuntos difusos la función de pertenencia que se utiliza es la . Ésta toma los valores entre cero (0) y uno (1); como se mencionó, la forma de representación de los conjuntos difusos puede ser de dos maneras: de forma continua o discreta, como se presenta a continuación.

Un conjunto difuso se escribe con una tilde arriba del nombre del conjunto:

Ésta se utiliza para diferenciarlos de los conjuntos nítidos. En la lógica difusa los conjuntos se pueden presentar en forma continua o discreta.

* 1. Conjunto difuso discreto

En este punto es importante recordar que el signo (+) no indica suma sino unión. Dicha forma de representación es muy empleada en los sistemas digitales como los microcontroladores, computadoras, etcétera.

* 1. Conjunto difuso continúo

Un conjunto convencional se define por una función característica, que se conoce también como función de pertenencia. El símbolo de integral denota ∫ unión de elementos del conjunto.

* 1. Lógica simbólica

La lógica difusa tiene sus bases en la lógica simbólica. La lógica simbólica permite el establecimiento de un lenguaje artificial empleando símbolos para de esta forma representar argumentos lógicos complicados. Partiendo de proposiciones, es decir, de oraciones verdaderas o falsas, es posible traducirlas a un lenguaje de símbolos y representaciones, para posteriormente simplificar y ejecutar operaciones, e incluso traducir nuevamente hacia proposiciones de lenguaje ordinario.

Una proposición puede ser simple, con valor de Verdadero o Falso, o compuesta, dependiendo de los valores de verdad de componentes simples conectados a partir de operadores como: y, o, no, entre otros. El operador y se denomina conjunción, y se simboliza con ∧. El operador o se denomina disyunción, y se simboliza con ∨. En la Tabla 1 se muestran las definiciones de la conjunción y disyunción para dos proposiciones simples p y q.

**Tabla 1** Tabla de verdad de la conjunción y la disyunción

|  |  |  |  |
| --- | --- | --- | --- |
| p | q | P **∧** q | P **∨** q |
| V | V | V | V |
| V | F | F | V |
| F | V | F | V |
| F | F | F | F |

La negativa de una proposición se denomina negación.

Ejemplo:

“Tengo un lápiz y una pluma” es una conjunción.

“Tengo un lápiz o una pluma” es una disyunción.

“No tengo un lápiz” es una negación.

Existen proposiciones condicionales de la forma “si p, entonces q”; p es el antecedente o hipótesis, y q es el consecuente o la conclusión. Una forma de simbolizar las proposiciones condicionales es: p → q, como se muestra en la Tabla 2.

**Tabla 2** Tabla de verdad de una proposición condicional.

|  |  |  |
| --- | --- | --- |
| p | q | P ***→*** q |
| V | V | V |
| V | F | F |
| F | V | V |
| F | F | V |

Ejemplo:

“Si 1+1=2, entonces 4>0” es una proposición condicional verdadera.

“Si 1+1=3, entonces 4>0” es una proposición condicional verdadera.

“Si 1+1=3, entonces 4<0” es una proposición condicional verdadera.

“Si 1+1=2, entonces 4<0” es una proposición condicional falsa.

* 1. **Tautologías y quasi-tautologías**

Las proposiciones que siempre son verdaderas se denominan tautologías. Para probar que una proposición es una tautología se construye su tabla de verdad y se verifica que todos los casos sean verdaderos. Si una proposición condicional es una tautología se denomina implicación.

En la lógica convencional existen ocho tautologías: trivial, ley de la doble negación, ley del medio excluido, razonamiento directo, razonamiento indirecto, ley de transitividad, ley de la contra recíproca y silogismo disyuntivo.

En la lógica difusa no existen tautologías pero sí quasi-tautologías, ya que las proposiciones no adquieren valores de 0 o 1 como en la lógica booleana, sino que contemplan valores de pertenencia entre 0 y 1.

Ejemplo:

Es muy claro que las tablas de verdad de elementos difusos se encuentran determinadas por las operaciones básicas de dichos elementos, como son la unión, intersección y complemento. Este ejemplo se puede utilizar para denotar la Tabla 3 de implicación difusa que parte de valores entre cero y uno.

Determinar A → B

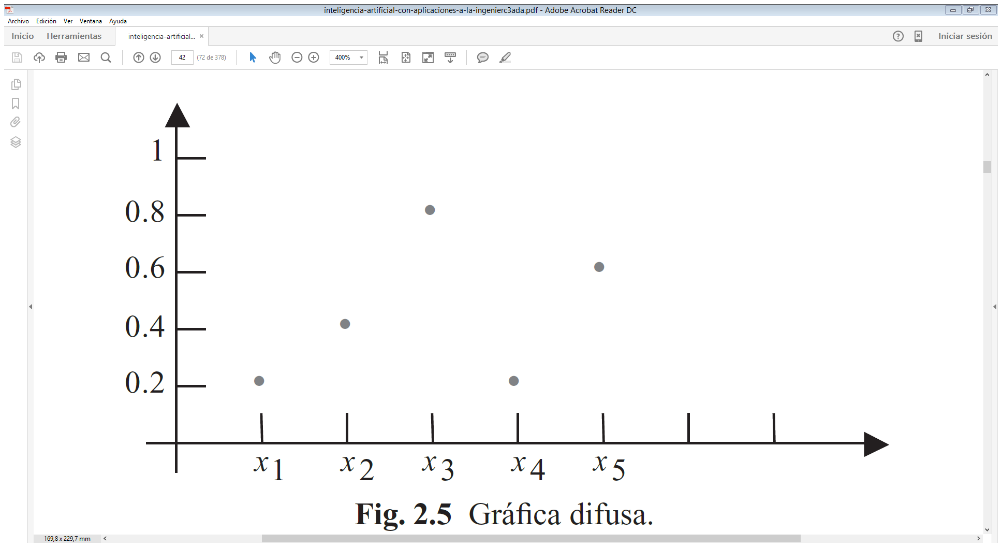
**Tabla 3** Tabla de verdad de una quasi-tautología.

|  |  |  |
| --- | --- | --- |
| A | B | A → B=máx () |
| 0.3 | 0.2 | 0.7 |
| 0.3 | 0.8 | 0.8 |
| 0.7 | 0.2 | 0.3 |
| 0.7 | 0.8 | 0.8 |

Después de revisar las operaciones básicas de unión, intersección y complemento que se presentan más adelante, se recomienda realizar la tabla de verdad de la implicación.

* 1. **Representación de conjuntos difusos discretos**

Se puede expresar en forma gráfica un conjunto difuso discreto como la unión de sus elementos; el número de elementos depende del problema que se va a resolver. Es muy común que a través de una interpolación de cada uno de los elementos se pueda construir una trayectoria con cada uno de ellos, la cual es la base de una función de membresía o pertenencia. Tomemos como ejemplo el conjunto difuso que se muestra en la Figura 2.



**Figura 2.** Gráfica difusa.

El conjunto difuso se define a partir de la gráfica en la Figura 2. Como se puede observar, los elementos de este conjunto están formados por fracciones, en las cuales el denominador es el elemento y el numerador es el grado de pertenencia del elemento, pero con una cierta función de membresía.

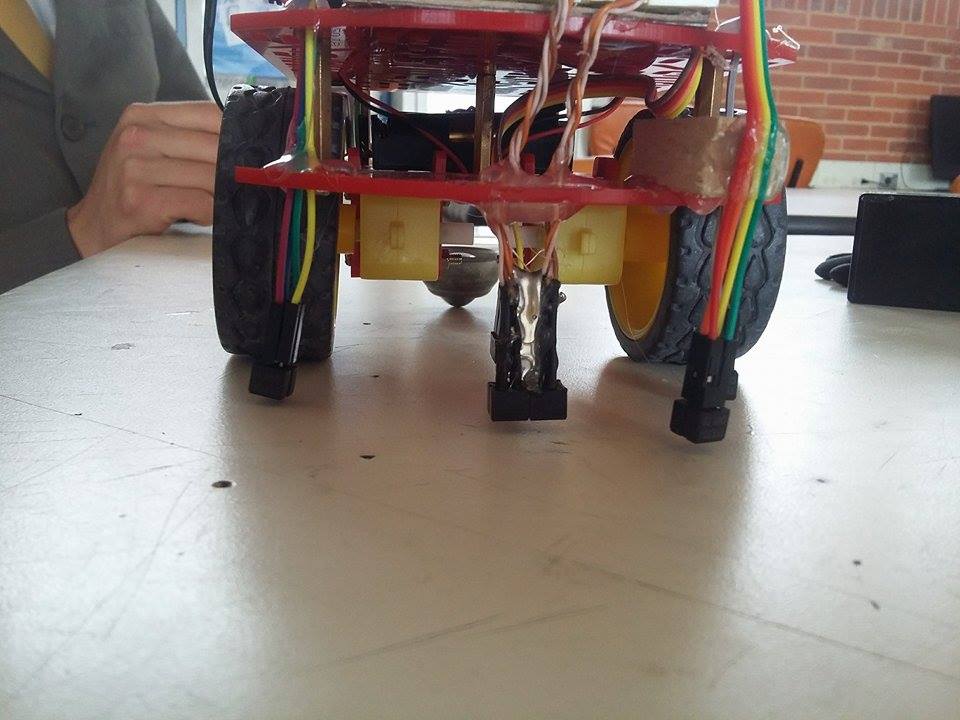
# IMPLEMENTACIÓN

Para la realización de este proyecto se implementó un carrito suministrado por la universidad con 2 ruedas de tracción delanteras y una rueda loca en la parte mitad anterior el cual se muestra en las Figura 3.

  
**Figura 3.** Carro utilizado para el proyecto

El censado se realizó con un juego de 5 emisor-receptor infrarrojos montados en la parte delantera inferior del carrito.

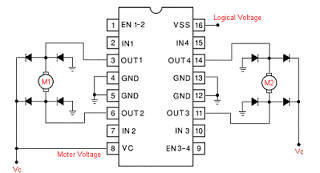
2 en el centro delanteros para seguir la línea, uno derecho e izquierdo para poder detectar los giros a sus respectivas direcciones y uno en la intersección de los delanteros e izquierdo en la parte de adelante, para poder saber si se llegó a la meta. Este arreglo en montaje se muestra en la Figura 4.



**Figura 4.** Arreglo de los sensores de línea en montaje del carrito.

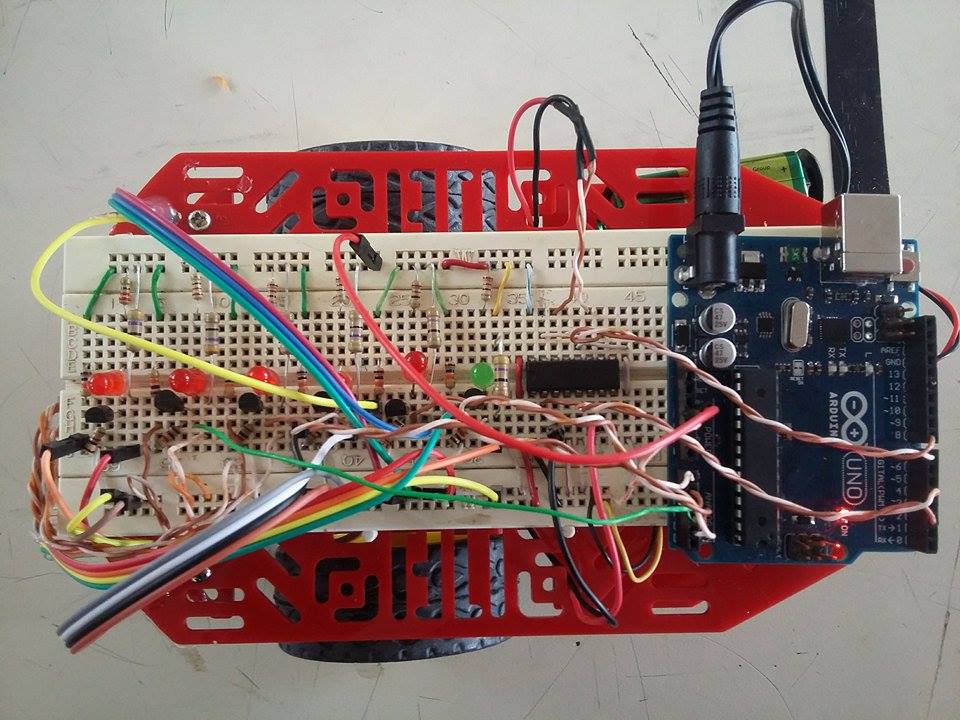
Luego de adquirir los valores análogos se llevan estos a una placa Arduino uno, el cual tendrá la logia que tomara las decisiones sobre los actuadores que serán 2 motores.

Las direcciones de motores a su vez serán controladas por el Arduino mediante un puente H l293d, Figura 5.



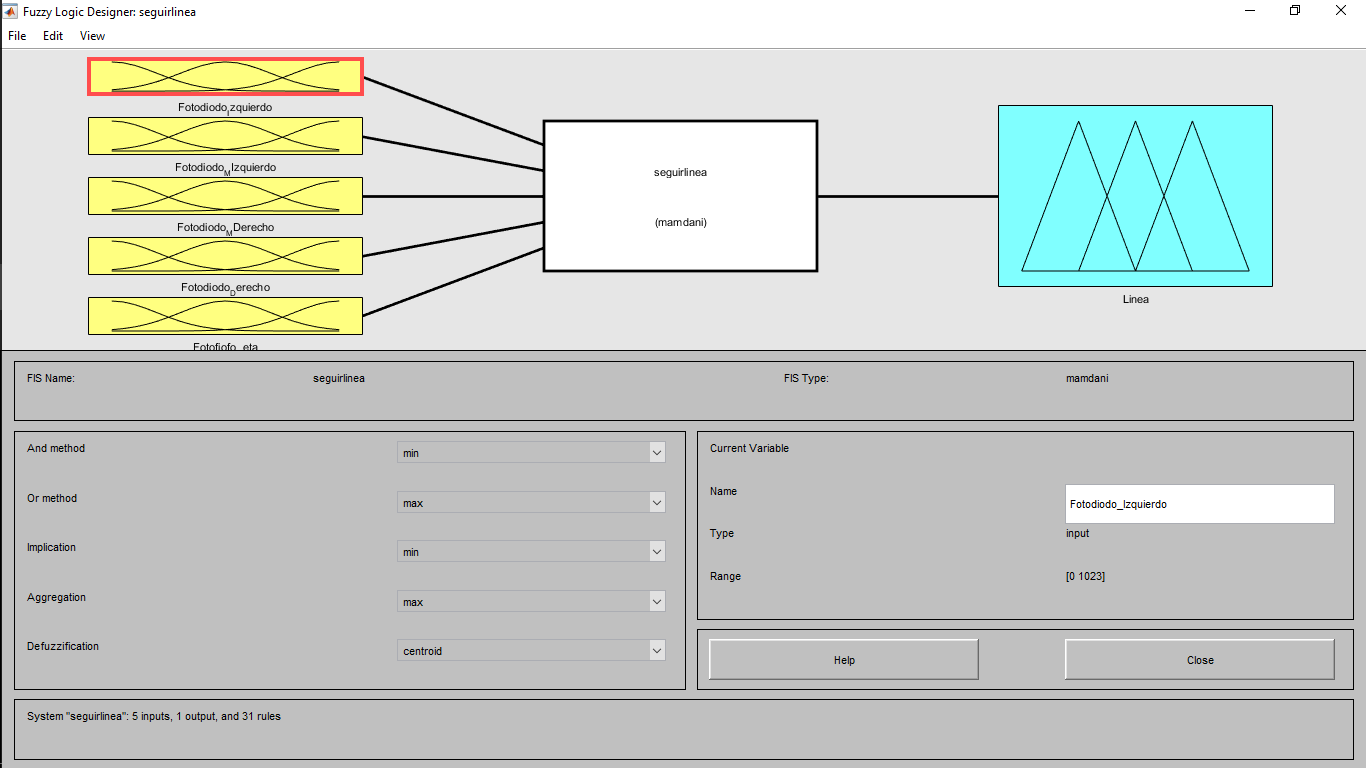
**Figura 5.** Configuración de un puente H 1293d para el control de dirección e 2 motores

El montaje final del carro seguidor de línea con lógica difusa se muestra en la Figura 6. Los leds son utilizados para saber cuál sensor está activo.

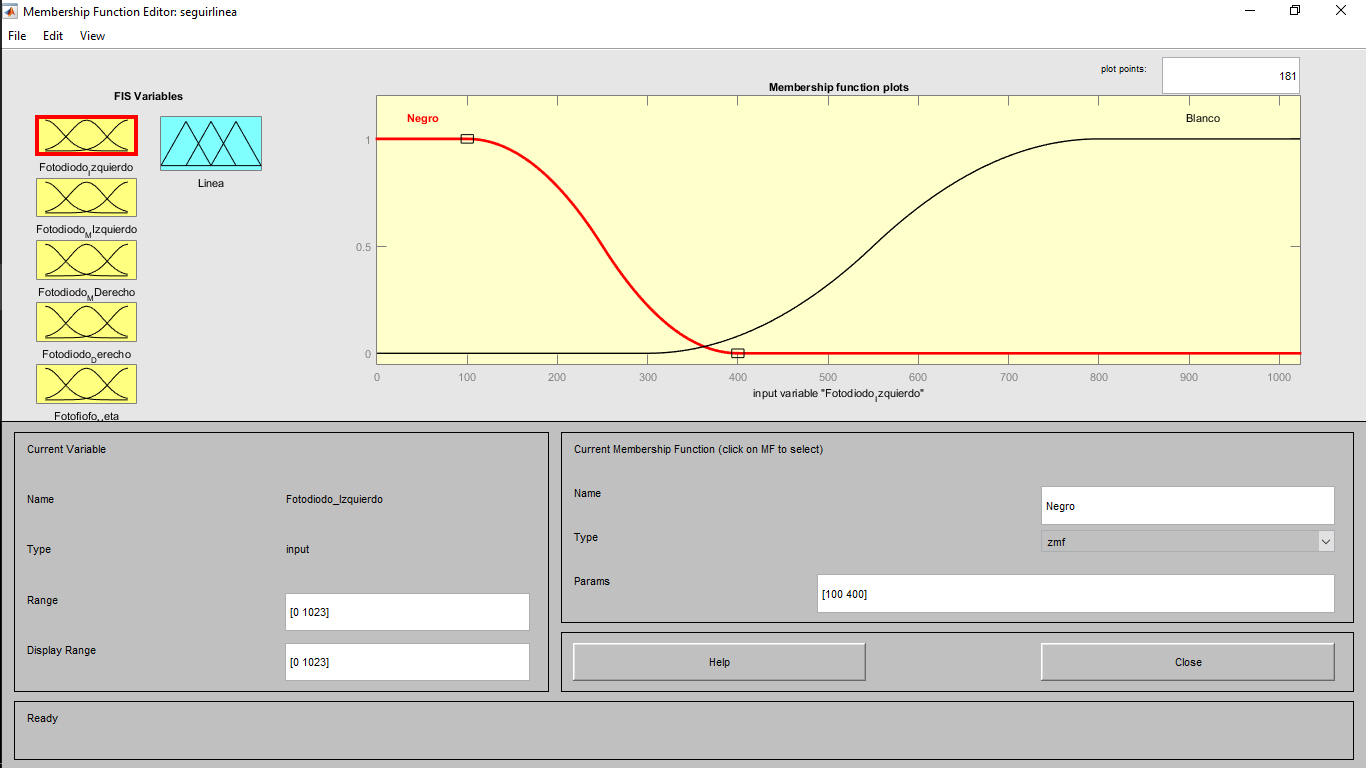
  
**Figura 6.** Montaje final del circuito del carro seguidor de línea con lógica difusa.

Primero Se parametrizaron los sensores en él Arduino, tomando resultados de alrededor 15 para valores de la cinta negra y de 800 para valores de papel blanco.

Teniendo esto en cuenta se establecieron funciones de membresía para los diferentes sensores con unas dimensiones desde 0 a 1023, lo cual es lo mínimo y máximo que admite las entradas análogas del Arduino. Hay 5 sensores, y las funciones de membresía para las 5 son las mismas, en la Figura 6 podemos ver que se usaron 5 entradas y 1 salida para la lógica.

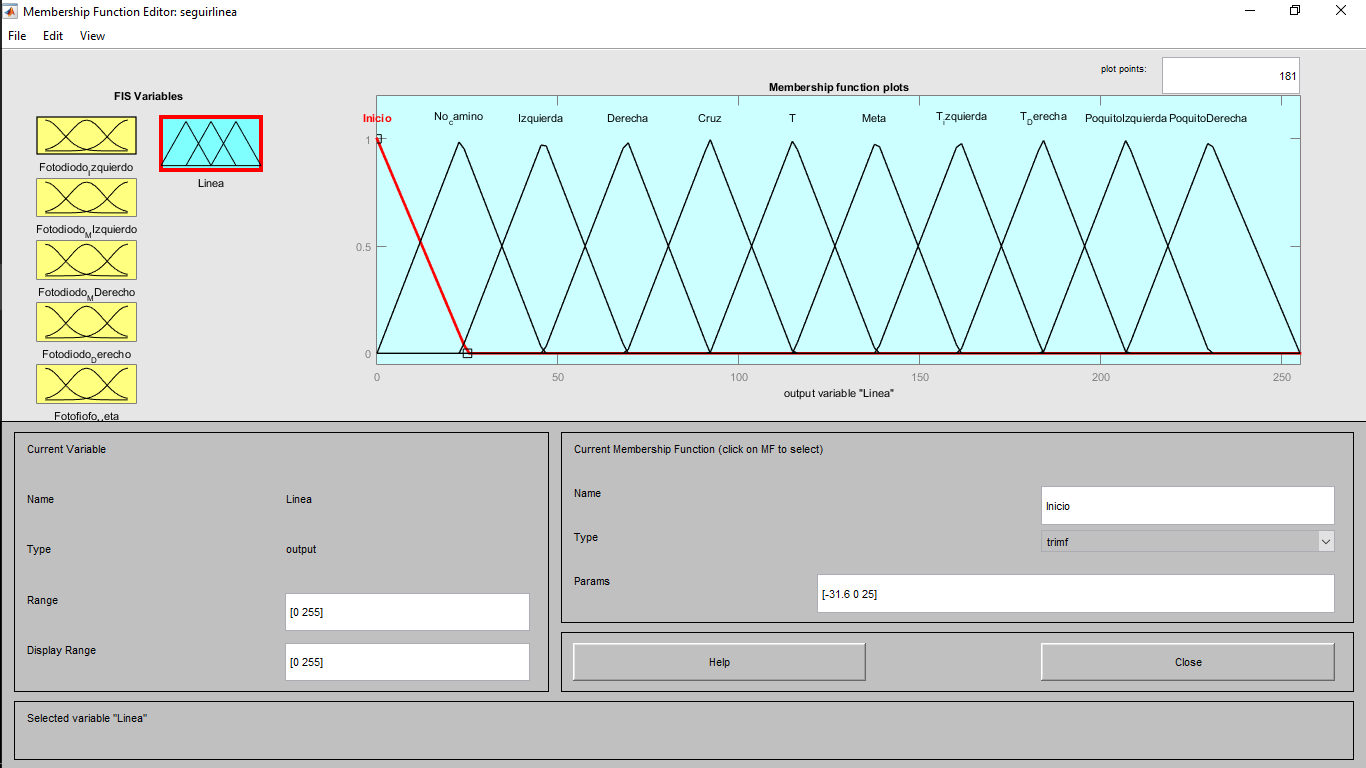
  
**Figura 6.** Lógica difusa usada para el carrito.

En la Figura 7 se muestra la lógica para las entradas de los sensores representadas en funciones de membresía.



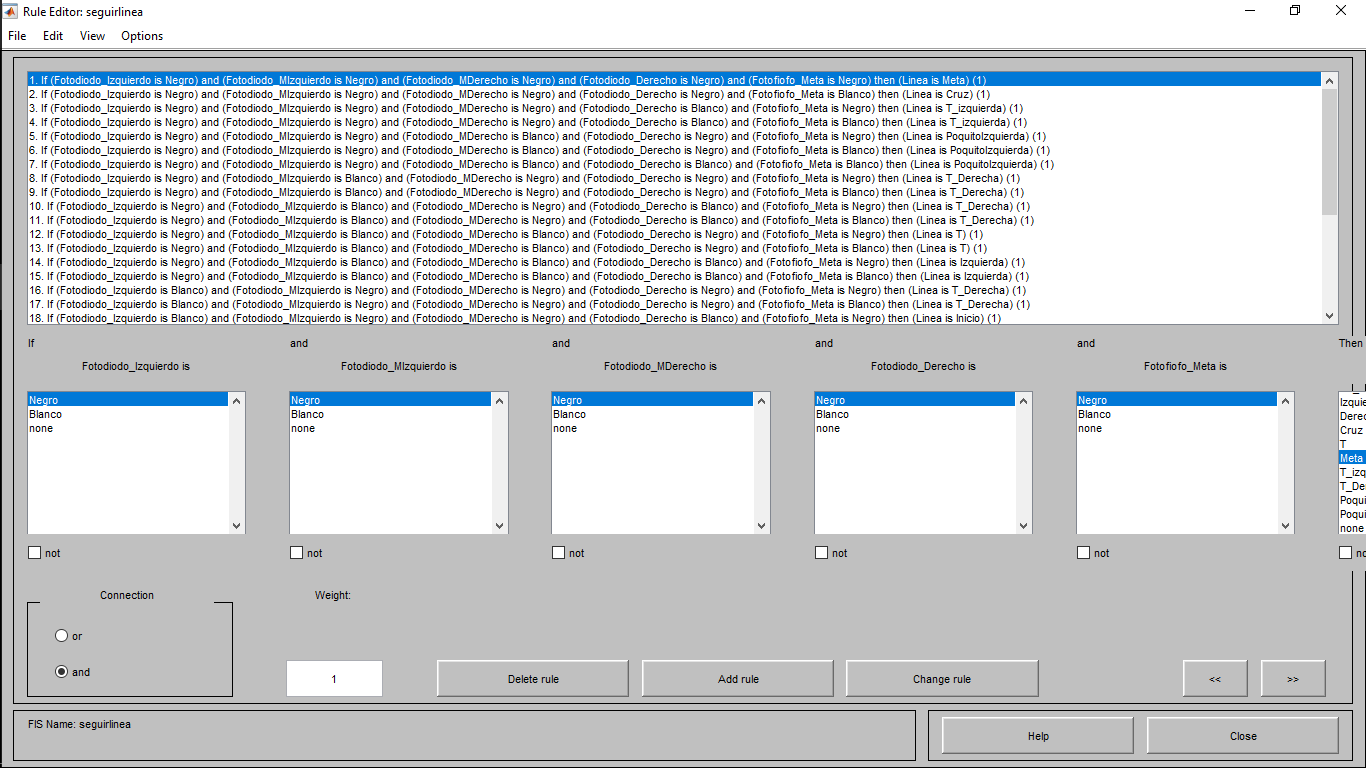
**Figura 7.** Funciones de membresía para la entrada de un sensor.

Para la salida se optaron por evaluaron diversos escenarios en los cuales los sensores pueden estar involucrados, que son inicio (hace al carro ir derecho sí los sensores delanteros están en negro), no hay camino, Izquierda, Derecha, Cruz, T, Meta, T Izquierda (T hacia la izquierda), T Derecha (T hacia la derecha), Poquita derecha (si se sale de la línea por la izquierda) y poquita izquierda (si se sale de la línea por la derecha). Esto se muestra en la Figura 8.

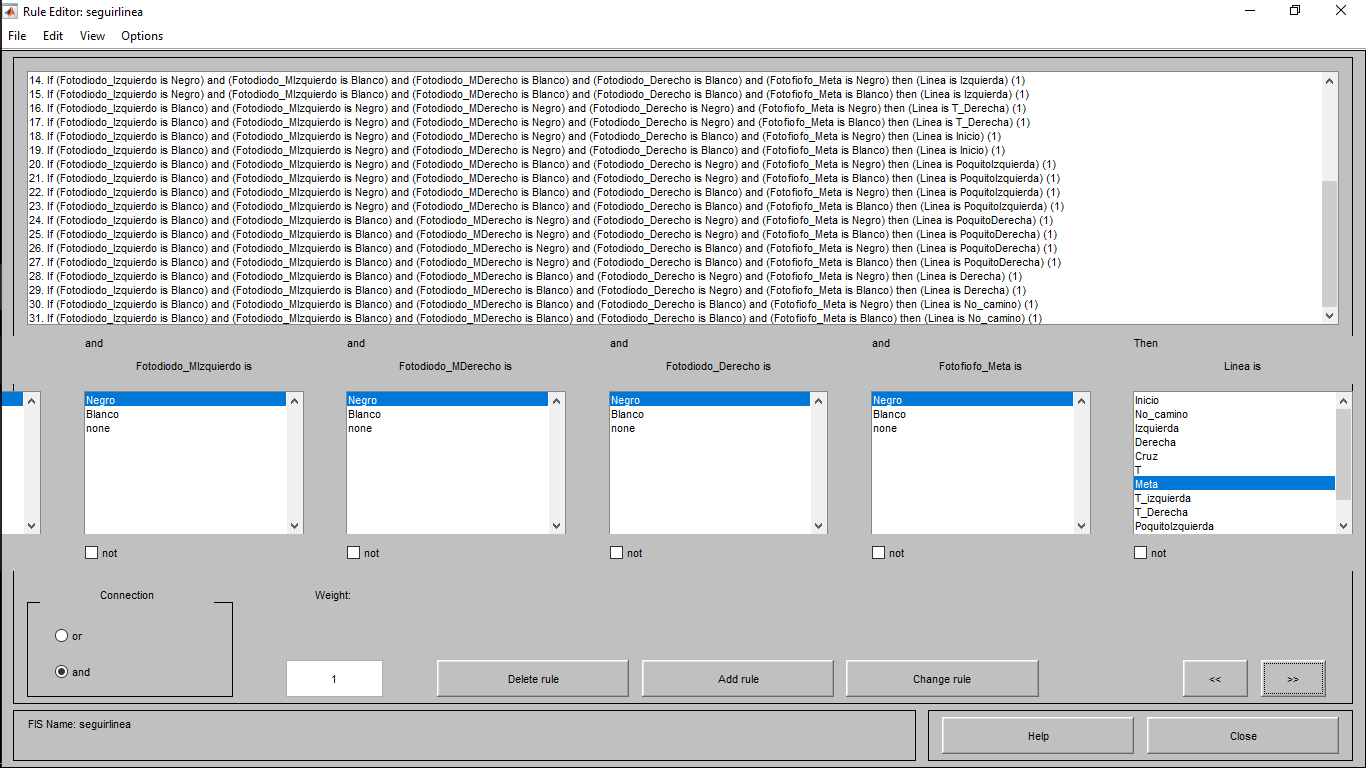


**Figura 8.** Funciones de membresía para la salida.

Luego de tener las funciones de membresía se implementaron las diferentes reglas, las cuales enlazan las entradas con las decisiones a tomar para tener una salida. Estas reglas se muestran en la Figura 9 y la Figura 10.



**Figura 9.** Reglas de la lógica difusa implementada parte 1.



**Figura 10.** Reglas de la lógica difusa implementada parte 2.

Luego de realizar la lógica difusa, esta se pasó a un lenguaje de Arduino mediante un software on-line, el cual transforma los. FIS a formato de Arduino [1].

El código resultante, implementado con l lógica para el aprendizaje del laberinto se muestra a continuación:

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Matlab .fis to arduino C converter v2.0.0.29032014

// - Karthik Nadig, USA

// Please report bugs to: karthiknadig@gmail.com

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include "fis\_header.h"

#define Izaquierdoreverza 2

#define Izquierdoadelante 3

#define Derechoreverza 4

#define Derechoadelante 7

// Number of inputs to the fuzzy inference system

const int fis\_gcI = 5;

// Number of outputs to the fuzzy inference system

const int fis\_gcO = 1;

// Number of rules to the fuzzy inference system

const int fis\_gcR = 31;

float Linea=0, LineaM[100];

int comienza=0;

bool aprendiendo=true;

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);

// initialize the Analog pins for input.

// Pin mode for Input: Fotodiodo\_Izquierdo

pinMode(0 , INPUT);

// Pin mode for Input: Fotodiodo\_MIzquierdo

pinMode(1 , INPUT);

// Pin mode for Input: Fotodiodo\_MDerecho

pinMode(2 , INPUT);

// Pin mode for Input: Fotodiodo\_Derecho

pinMode(3 , INPUT);

// Pin mode for Input: Fotofiofo\_Meta

pinMode(4 , INPUT);

// initialize the Analog pins for output.

// Pin mode for Output: Linea

pinMode(5 , OUTPUT);

}

// Loop routine runs over and over again forever:

void loop()

{

if(aprendiendo==true)

{

// Read Input: Fotodiodo\_Izquierdo

g\_fisInput[0] = analogRead(0);

// Read Input: Fotodiodo\_MIzquierdo

g\_fisInput[1] = analogRead(1);

// Read Input: Fotodiodo\_MDerecho

g\_fisInput[2] = analogRead(2);

// Read Input: Fotodiodo\_Derecho

g\_fisInput[3] = analogRead(3);

// Read Input: Fotofiofo\_Meta

g\_fisInput[4] = analogRead(4);

g\_fisOutput[0] = 0;

fis\_evaluate();

// Set output vlaue: Linea

Serial.println(g\_fisOutput[0]);

Linea=g\_fisOutput[0];

LineaM[comienza]=Linea;

delay(1000);

comienza++;

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Support functions for Fuzzy Inference System

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Z-shaped Member Function

FIS\_TYPE fis\_zmf(FIS\_TYPE x, FIS\_TYPE\* p)

{

FIS\_TYPE a = p[0], b = p[1];

FIS\_TYPE m = ((a + b) / 2.0);

FIS\_TYPE t = (b - a);

if (x <= a) return (FIS\_TYPE) 1;

if (x <= m)

{

t = (x - a) / t;

return (FIS\_TYPE) (1.0 - (2.0 \* t \* t));

}

if (x <= b)

{

t = (b - x) / t;

return (FIS\_TYPE) (1.0 - (2.0 \* t \* t));

}

return (FIS\_TYPE) 0;

}

// S-Shaped membership function

FIS\_TYPE fis\_smf(FIS\_TYPE x, FIS\_TYPE\* p)

{

FIS\_TYPE a = p[0], b = p[1];

FIS\_TYPE m = ((a + b) / 2.0);

FIS\_TYPE t = (b - a);

if (a >= b) return (FIS\_TYPE) (x >= m);

if (x <= a) return (FIS\_TYPE) 0;

if (x <= m)

{

t = (x - a) / t;

return (FIS\_TYPE) (2.0 \* t \* t);

}

if (x <= b)

{

t = (b - x) / t;

return (FIS\_TYPE) (1.0 - (2.0 \* t \* t));

}

return (FIS\_TYPE) 1;

}

// Triangular Member Function

FIS\_TYPE fis\_trimf(FIS\_TYPE x, FIS\_TYPE\* p)

{

FIS\_TYPE a = p[0], b = p[1], c = p[2];

FIS\_TYPE t1 = (x - a) / (b - a);

FIS\_TYPE t2 = (c - x) / (c - b);

if ((a == b) && (b == c)) return (FIS\_TYPE) (x == a);

if (a == b) return (FIS\_TYPE) (t2\*(b <= x)\*(x <= c));

if (b == c) return (FIS\_TYPE) (t1\*(a <= x)\*(x <= b));

t1 = min(t1, t2);

return (FIS\_TYPE) max(t1, 0);

}

FIS\_TYPE fis\_min(FIS\_TYPE a, FIS\_TYPE b)

{

return min(a, b);

}

FIS\_TYPE fis\_max(FIS\_TYPE a, FIS\_TYPE b)

{

return max(a, b);

}

FIS\_TYPE fis\_array\_operation(FIS\_TYPE \*array, int size, \_FIS\_ARR\_OP pfnOp)

{

int i;

FIS\_TYPE ret = 0;

if (size == 0) return ret;

if (size == 1) return array[0];

ret = array[0];

for (i = 1; i < size; i++)

{

ret = (\*pfnOp)(ret, array[i]);

}

return ret;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Data for Fuzzy Inference System

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Pointers to the implementations of member functions

\_FIS\_MF fis\_gMF[] =

{

fis\_zmf, fis\_smf, fis\_trimf

};

// Count of member function for each Input

int fis\_gIMFCount[] = { 2, 2, 2, 2, 2 };

// Count of member function for each Output

int fis\_gOMFCount[] = { 11 };

// Coefficients for the Input Member Functions

FIS\_TYPE fis\_gMFI0Coeff1[] = { 100, 400 };

FIS\_TYPE fis\_gMFI0Coeff2[] = { 300, 800 };

FIS\_TYPE\* fis\_gMFI0Coeff[] = { fis\_gMFI0Coeff1, fis\_gMFI0Coeff2 };

FIS\_TYPE fis\_gMFI1Coeff1[] = { 100, 400 };

FIS\_TYPE fis\_gMFI1Coeff2[] = { 300, 800 };

FIS\_TYPE\* fis\_gMFI1Coeff[] = { fis\_gMFI1Coeff1, fis\_gMFI1Coeff2 };

FIS\_TYPE fis\_gMFI2Coeff1[] = { 100, 400 };

FIS\_TYPE fis\_gMFI2Coeff2[] = { 300, 800 };

FIS\_TYPE\* fis\_gMFI2Coeff[] = { fis\_gMFI2Coeff1, fis\_gMFI2Coeff2 };

FIS\_TYPE fis\_gMFI3Coeff1[] = { 100, 400 };

FIS\_TYPE fis\_gMFI3Coeff2[] = { 300, 800 };

FIS\_TYPE\* fis\_gMFI3Coeff[] = { fis\_gMFI3Coeff1, fis\_gMFI3Coeff2 };

FIS\_TYPE fis\_gMFI4Coeff1[] = { 100, 400 };

FIS\_TYPE fis\_gMFI4Coeff2[] = { 300, 800 };

FIS\_TYPE\* fis\_gMFI4Coeff[] = { fis\_gMFI4Coeff1, fis\_gMFI4Coeff2 };

FIS\_TYPE\*\* fis\_gMFICoeff[] = { fis\_gMFI0Coeff, fis\_gMFI1Coeff, fis\_gMFI2Coeff, fis\_gMFI3Coeff, fis\_gMFI4Coeff };

// Coefficients for the Input Member Functions

FIS\_TYPE fis\_gMFO0Coeff1[] = { -31.6, 0, 25 };

FIS\_TYPE fis\_gMFO0Coeff2[] = { 0, 23, 46 };

FIS\_TYPE fis\_gMFO0Coeff3[] = { 23, 46, 69 };

FIS\_TYPE fis\_gMFO0Coeff4[] = { 46, 69, 92 };

FIS\_TYPE fis\_gMFO0Coeff5[] = { 69, 92, 115 };

FIS\_TYPE fis\_gMFO0Coeff6[] = { 92, 115, 138 };

FIS\_TYPE fis\_gMFO0Coeff7[] = { 115, 138, 161 };

FIS\_TYPE fis\_gMFO0Coeff8[] = { 138, 161, 184 };

FIS\_TYPE fis\_gMFO0Coeff9[] = { 161, 184, 207 };

FIS\_TYPE fis\_gMFO0Coeff10[] = { 184, 207, 230 };

FIS\_TYPE fis\_gMFO0Coeff11[] = { 207, 230, 255 };

FIS\_TYPE\* fis\_gMFO0Coeff[] = { fis\_gMFO0Coeff1, fis\_gMFO0Coeff2, fis\_gMFO0Coeff3, fis\_gMFO0Coeff4, fis\_gMFO0Coeff5, fis\_gMFO0Coeff6, fis\_gMFO0Coeff7, fis\_gMFO0Coeff8, fis\_gMFO0Coeff9, fis\_gMFO0Coeff10, fis\_gMFO0Coeff11 };

FIS\_TYPE\*\* fis\_gMFOCoeff[] = { fis\_gMFO0Coeff };

// Input membership function set

int fis\_gMFI0[] = { 0, 1 };

int fis\_gMFI1[] = { 0, 1 };

int fis\_gMFI2[] = { 0, 1 };

int fis\_gMFI3[] = { 0, 1 };

int fis\_gMFI4[] = { 0, 1 };

int\* fis\_gMFI[] = { fis\_gMFI0, fis\_gMFI1, fis\_gMFI2, fis\_gMFI3, fis\_gMFI4};

// Output membership function set

int fis\_gMFO0[] = { 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2 };

int\* fis\_gMFO[] = { fis\_gMFO0};

// Rule Weights

FIS\_TYPE fis\_gRWeight[] = { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 };

// Rule Type

int fis\_gRType[] = { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 };

// Rule Inputs

int fis\_gRI0[] = { 1, 1, 1, 1, 1 };

int fis\_gRI1[] = { 1, 1, 1, 1, 2 };

int fis\_gRI2[] = { 1, 1, 1, 2, 1 };

int fis\_gRI3[] = { 1, 1, 1, 2, 2 };

int fis\_gRI4[] = { 1, 1, 2, 1, 1 };

int fis\_gRI5[] = { 1, 1, 2, 1, 2 };

int fis\_gRI6[] = { 1, 1, 2, 2, 2 };

int fis\_gRI7[] = { 1, 2, 1, 1, 1 };

int fis\_gRI8[] = { 1, 2, 1, 1, 2 };

int fis\_gRI9[] = { 1, 2, 1, 2, 1 };

int fis\_gRI10[] = { 1, 2, 1, 2, 2 };

int fis\_gRI11[] = { 1, 2, 2, 1, 1 };

int fis\_gRI12[] = { 1, 2, 2, 1, 2 };

int fis\_gRI13[] = { 1, 2, 2, 2, 1 };

int fis\_gRI14[] = { 1, 2, 2, 2, 2 };

int fis\_gRI15[] = { 2, 1, 1, 1, 1 };

int fis\_gRI16[] = { 2, 1, 1, 1, 2 };

int fis\_gRI17[] = { 2, 1, 1, 2, 1 };

int fis\_gRI18[] = { 2, 1, 1, 2, 2 };

int fis\_gRI19[] = { 2, 1, 2, 1, 1 };

int fis\_gRI20[] = { 2, 1, 2, 1, 2 };

int fis\_gRI21[] = { 2, 1, 2, 2, 1 };

int fis\_gRI22[] = { 2, 1, 2, 2, 2 };

int fis\_gRI23[] = { 2, 2, 1, 1, 1 };

int fis\_gRI24[] = { 2, 2, 1, 1, 2 };

int fis\_gRI25[] = { 2, 2, 1, 2, 1 };

int fis\_gRI26[] = { 2, 2, 1, 2, 2 };

int fis\_gRI27[] = { 2, 2, 2, 1, 1 };

int fis\_gRI28[] = { 2, 2, 2, 1, 2 };

int fis\_gRI29[] = { 2, 2, 2, 2, 1 };

int fis\_gRI30[] = { 2, 2, 2, 2, 2 };

int\* fis\_gRI[] = { fis\_gRI0, fis\_gRI1, fis\_gRI2, fis\_gRI3, fis\_gRI4, fis\_gRI5, fis\_gRI6, fis\_gRI7, fis\_gRI8, fis\_gRI9, fis\_gRI10, fis\_gRI11, fis\_gRI12, fis\_gRI13, fis\_gRI14, fis\_gRI15, fis\_gRI16, fis\_gRI17, fis\_gRI18, fis\_gRI19, fis\_gRI20, fis\_gRI21, fis\_gRI22, fis\_gRI23, fis\_gRI24, fis\_gRI25, fis\_gRI26, fis\_gRI27, fis\_gRI28, fis\_gRI29, fis\_gRI30 };

// Rule Outputs

int fis\_gRO0[] = { 7 };

int fis\_gRO1[] = { 5 };

int fis\_gRO2[] = { 8 };

int fis\_gRO3[] = { 8 };

int fis\_gRO4[] = { 10 };

int fis\_gRO5[] = { 10 };

int fis\_gRO6[] = { 10 };

int fis\_gRO7[] = { 9 };

int fis\_gRO8[] = { 9 };

int fis\_gRO9[] = { 9 };

int fis\_gRO10[] = { 9 };

int fis\_gRO11[] = { 6 };

int fis\_gRO12[] = { 6 };

int fis\_gRO13[] = { 3 };

int fis\_gRO14[] = { 3 };

int fis\_gRO15[] = { 9 };

int fis\_gRO16[] = { 9 };

int fis\_gRO17[] = { 1 };

int fis\_gRO18[] = { 1 };

int fis\_gRO19[] = { 10 };

int fis\_gRO20[] = { 10 };

int fis\_gRO21[] = { 10 };

int fis\_gRO22[] = { 10 };

int fis\_gRO23[] = { 11 };

int fis\_gRO24[] = { 11 };

int fis\_gRO25[] = { 11 };

int fis\_gRO26[] = { 11 };

int fis\_gRO27[] = { 4 };

int fis\_gRO28[] = { 4 };

int fis\_gRO29[] = { 2 };

int fis\_gRO30[] = { 2 };

int\* fis\_gRO[] = { fis\_gRO0, fis\_gRO1, fis\_gRO2, fis\_gRO3, fis\_gRO4, fis\_gRO5, fis\_gRO6, fis\_gRO7, fis\_gRO8, fis\_gRO9, fis\_gRO10, fis\_gRO11, fis\_gRO12, fis\_gRO13, fis\_gRO14, fis\_gRO15, fis\_gRO16, fis\_gRO17, fis\_gRO18, fis\_gRO19, fis\_gRO20, fis\_gRO21, fis\_gRO22, fis\_gRO23, fis\_gRO24, fis\_gRO25, fis\_gRO26, fis\_gRO27, fis\_gRO28, fis\_gRO29, fis\_gRO30 };

// Input range Min

FIS\_TYPE fis\_gIMin[] = { 0, 0, 0, 0, 0 };

// Input range Max

FIS\_TYPE fis\_gIMax[] = { 1023, 1023, 1023, 1023, 1023 };

// Output range Min

FIS\_TYPE fis\_gOMin[] = { 0 };

// Output range Max

FIS\_TYPE fis\_gOMax[] = { 255 };

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Data dependent support functions for Fuzzy Inference System

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

FIS\_TYPE fis\_MF\_out(FIS\_TYPE\*\* fuzzyRuleSet, FIS\_TYPE x, int o)

{

FIS\_TYPE mfOut;

int r;

for (r = 0; r < fis\_gcR; ++r)

{

int index = fis\_gRO[r][o];

if (index > 0)

{

index = index - 1;

mfOut = (fis\_gMF[fis\_gMFO[o][index]])(x, fis\_gMFOCoeff[o][index]);

}

else if (index < 0)

{

index = -index - 1;

mfOut = 1 - (fis\_gMF[fis\_gMFO[o][index]])(x, fis\_gMFOCoeff[o][index]);

}

else

{

mfOut = 0;

}

fuzzyRuleSet[0][r] = fis\_min(mfOut, fuzzyRuleSet[1][r]);

}

return fis\_array\_operation(fuzzyRuleSet[0], fis\_gcR, fis\_max);

}

FIS\_TYPE fis\_defuzz\_centroid(FIS\_TYPE\*\* fuzzyRuleSet, int o)

{

FIS\_TYPE step = (fis\_gOMax[o] - fis\_gOMin[o]) / (FIS\_RESOLUSION - 1);

FIS\_TYPE area = 0;

FIS\_TYPE momentum = 0;

FIS\_TYPE dist, slice;

int i;

// calculate the area under the curve formed by the MF outputs

for (i = 0; i < FIS\_RESOLUSION; ++i){

dist = fis\_gOMin[o] + (step \* i);

slice = step \* fis\_MF\_out(fuzzyRuleSet, dist, o);

area += slice;

momentum += slice\*dist;

}

return ((area == 0) ? ((fis\_gOMax[o] + fis\_gOMin[o]) / 2) : (momentum / area));

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Fuzzy Inference System

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void fis\_evaluate()

{

FIS\_TYPE fuzzyInput0[] = { 0, 0 };

FIS\_TYPE fuzzyInput1[] = { 0, 0 };

FIS\_TYPE fuzzyInput2[] = { 0, 0 };

FIS\_TYPE fuzzyInput3[] = { 0, 0 };

FIS\_TYPE fuzzyInput4[] = { 0, 0 };

FIS\_TYPE\* fuzzyInput[fis\_gcI] = { fuzzyInput0, fuzzyInput1, fuzzyInput2, fuzzyInput3, fuzzyInput4, };

FIS\_TYPE fuzzyOutput0[] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };

FIS\_TYPE\* fuzzyOutput[fis\_gcO] = { fuzzyOutput0, };

FIS\_TYPE fuzzyRules[fis\_gcR] = { 0 };

FIS\_TYPE fuzzyFires[fis\_gcR] = { 0 };

FIS\_TYPE\* fuzzyRuleSet[] = { fuzzyRules, fuzzyFires };

FIS\_TYPE sW = 0;

// Transforming input to fuzzy Input

int i, j, r, o;

for (i = 0; i < fis\_gcI; ++i)

{

for (j = 0; j < fis\_gIMFCount[i]; ++j)

{

fuzzyInput[i][j] =

(fis\_gMF[fis\_gMFI[i][j]])(g\_fisInput[i], fis\_gMFICoeff[i][j]);

}

}

int index = 0;

for (r = 0; r < fis\_gcR; ++r)

{

if (fis\_gRType[r] == 1)

{

fuzzyFires[r] = FIS\_MAX;

for (i = 0; i < fis\_gcI; ++i)

{

index = fis\_gRI[r][i];

if (index > 0)

fuzzyFires[r] = fis\_min(fuzzyFires[r], fuzzyInput[i][index - 1]);

else if (index < 0)

fuzzyFires[r] = fis\_min(fuzzyFires[r], 1 - fuzzyInput[i][-index - 1]);

else

fuzzyFires[r] = fis\_min(fuzzyFires[r], 1);

}

}

else

{

fuzzyFires[r] = FIS\_MIN;

for (i = 0; i < fis\_gcI; ++i)

{

index = fis\_gRI[r][i];

if (index > 0)

fuzzyFires[r] = fis\_max(fuzzyFires[r], fuzzyInput[i][index - 1]);

else if (index < 0)

fuzzyFires[r] = fis\_max(fuzzyFires[r], 1 - fuzzyInput[i][-index - 1]);

else

fuzzyFires[r] = fis\_max(fuzzyFires[r], 0);

}

}

fuzzyFires[r] = fis\_gRWeight[r] \* fuzzyFires[r];

sW += fuzzyFires[r];

}

if (sW == 0)

{

for (o = 0; o < fis\_gcO; ++o)

{

g\_fisOutput[o] = ((fis\_gOMax[o] + fis\_gOMin[o]) / 2);

}

}

else

{

for (o = 0; o < fis\_gcO; ++o)

{

g\_fisOutput[o] = fis\_defuzz\_centroid(fuzzyRuleSet, o);

}

}

}

Este código además de la lógica difusa, implementa la parte de aprendizaje del laberinto, en el cual el carrito siempre tomara el giro a la izquierda teniendo la posibilidad, si no es el camino girara sobre si hasta regresar al camino y estando de nuevo en el cruce tomara el camino izquierdo (si se puede) o seguirá derecho si no hay otra posibilidad (sin tener en cuenta de donde ya ha pasado), el carro ira guardando el recorrido y tomara la longitud del recorrido en intervalos, esto es para poder al final tomar la menor longitud, es decir, el camino más rápido para llegar a la meta.

# CONCLUSIONES

* Tanto dimensiones del carro como la potencia de los motores, influencio en gran medida en el método de aprendizaje del laberinto, sin embargo, no afecta la lógica difusa implementada.
* Al implementar una solo función de membresía de salida extra a la salida, vario casi en el doble el consumo de memoria del Arduino implementado.
* La lógica difusa resulto un método ideal al tener sensores analógicos como lo son los fotodiodos, ya que estos presentan una enorme variación o incertidumbre.

# Referenciases

1. http://www.makeproto.com/projects/fuzzy/matlab\_arduino\_FIST/index.php
2. Ponce Cruz Pedro. “Inteligencia Artificial con Aplicaciones a la Ingeniería”. Editorial Alfaomega. 2010. Biblioteca “Francisco Mora Díaz” Universidad Santo Tomas Tunja. Cód. 621.399 P55I 1A.ED.
3. Apuntes del curso de Inteligencia Artificial.
4. Ogata, K. Discrete-Time Control Systems. Prentice Hall, 1995.
5. MatLab 7.0

<http://www.caos.tv/fuzzy.html>

# Biografía

******

***Suarez L. Juan Diego.*** Nació el 27 de enero del año 1994 en la ciudad de Tunja, hijo único de Beatriz Elvira Londoño Ruiz y Hugo Ernesto Suarez Leandro. Estudio y se graduó en el Gimnasio Campestre del Norte, Promoción 2010. Actualmente studia Ingeniería Electrónica en la Universidad Santo Tomas seccional Tunja.

1. [↑](#footnote-ref-1)