ELABORACIÓN DE UN SISTEMA DE CONTROL DE POSICIÓN LINEAL UTILIZANDO LÓGICA DIFUSA

Proyecto final

Asignatura: Control inteligente

Catedrático: Dr. Benjamín Ortiz Moctezuma

Alumno: Ángel Arturo Ramírez Suárez

IM 9-1

Fecha: 21 de agosto de 2013

Ciudad Victoria, Tamaulipas.

Contents

Abstract

In the following document we explain the steps needed for the development of a lineal position control of a mobile platform whose goal is to maintain a specified position by the use of fuzzy logic to control its behaviour..

Resumen

En el presente documento se explica los pasos necesarios para el desarrollo de un sistema de control de posición lineal de una plataforma móvil cuyo objetivo es mantener una posición especificada mediante el empleo de lógica difusa para controlar su comportamiento.

Capítulo 1

INTRODUCCIÓN.

Lógica difusa es una metodología de resolución de problemas de sistemas de control utilizada para la implementación en sistemas que varían desde pequeños microcontroladores embebidos a redes industriales de múltiples canales o estaciones de trabajo dedicadas a la adquisición y control de datos. Puede ser implementada en hardware, software o una combinación de ambos y provee una manera simple de alcanzar una conclusión definitiva utilizando información vaga, ambigua, imprecisa o ruidosa. La lógica difusa emula la toma de decisiones de un individuo al incorporar un sistema de condicionamiento de tipo IF (si - conocido como el antecedente) THEN (entonces - conocido como el consecuente).

El concepto de lógica difusa fue concebido por Lotfi Zadeh, un profesor de la Universidad de California en Berkeley y presentado no como una metodología de control sino como una forma de procesamiento de datos al permetir membresía parcial de las funciones en lugar de una pertenencia total o nula de las condiciones. Esta metodología no fue aplicada a sistemas de control hasta la época de los 70's debido a la insuficiente capacidad computacional de los dispositivos anteriores a esa época. El profesor Zadesh razonó que las personas no requieren entradas de información numérica precisa y sin embargo son capaces de reaizar un control altamente adaptativo. Concluyó que si es posible alimentar a los controles por retroalimentación para aceptar información ruidosa o imprecisa sería mucho más sencillo de implementar [1].

Capítulo 2

PROBLEMÁTICA.

El problema definido para la elaboración del proyecto consistió en la elaboración y desarrollo de un controlador difuso para un sistema de seguimiento de posición lineal en el cual se posee una plataforma móvil a la cual se le especifica como valor inicial una distancia “d” la cual debe ser mantenida por la plataforma de manera automática de tal forma que cuando la distancia de la plataforma con respecto a un sistema de referencia sea modificado, el controlador automáticamente realice la toma de decisiones necesario para corregir el error en la posición.

Capítulo 3

METODOLOGÍA.

Para la elaboración de la plataforma móvil se hizo uso de un sistema previamente desarrollado de radio control lo cual permite evitar muchos de los problemas concernientes con diferencias que pudiera tenerse en la estructura y por tanto simplificar el proceso de desarrollo para enfocarse en el controlador.

Para el control del dispositivo se utilizó una solución basada en software (aplicaciones) mediante el uso del microcontrolador Arduino Mega 2560 el cual es ampliamente empleado para el desarrollo de proyectos electrónicos debido a su bajo costo y facilidad de programación, además de tener un amplio soporte en comunidades de desarrolladores.

Al Arduino fue cargado el algoritmo difuso utilizando la librería EFLL (Embedded Fuzzy Logic Library o Librería de Lógica Difusa para Embebidos) la cual consiste en un conjunto de librerías desarrolladas en el lenguaje C++ por la Universidad de Piaui que elaboró las librerías con el objetivo de permitir un elaborar sistemas difusos de manera sencilla y utilizando la menor cantidad de recursos posibles para permitir que éstos puedan ser empleados por sistemas embebidos como microcontroladores.

La señal de salida del microcontrolador a su vez fue direccionada a una etapa de potencia desarrollada utilizando el integrado L298N, el cual es un controlador (driver) de potencial utilizado como un puente H para modificar la velocidad de giro de los motores. Dicho integrado es caracterizado por tener la capacidad de soportar potencias de hasta 2 Amperes, lo cual lo vuelve ideal para su uso en aplicaciones pequeñas con requerimientos de par torsor mayores.

Para la retroalimentación del sistema se hizo uso del sensor XL MB-1230 el cual consiste en un sistema ultrasónico que envía una onda sonora que se expande en el espacio y regresa una vez que hace contacto con un objeto sólido, con lo cual se vuelve posible localizar la posición de obstáculos con respecto al sensor.

Capítulo 4

DESARROLLO.

4.1 Etapa de control.

4.1.1 Arduino 2560.

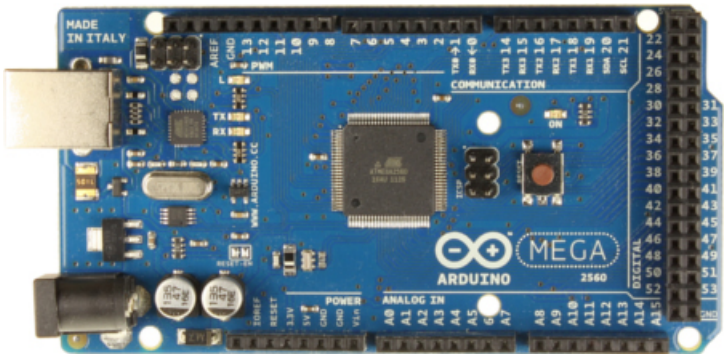


Figura 4.1: Placa del Arduino Mega 2560.

El Arduino Mega 2560 es una tarjeta microcontroladora basada en el ATMega 2560. Posee 54 pines de entrada/salida digitales con 15 que pueden ser utilizados como salida PWM, 16 entradas analógicas, 4 UARTs, un oscilador de cristal de 16MHz, conexión USB y botón de reset.

Características de operación[3]:

Voltaje de operación	5 volts
Voltaje de entrada	7 a 12 volts
Pines de entrada/salida digitales	54 con 15 de salida PWM
Pines de entrada analógica	16
Corriente directa por pin	40mA
Corriente directa para pin de 3.3 volts	50mA
Memoria flash	256kB (8 utilizados por el bootloader).
SRAM	8kB
EEPROM	4kB
Reloj	16MHz

La plataforma es ampliamente desarrollada y posee una comunidad de desarrolladores libres que permiten que la plataforma posea una amplia conectividad y acceso a librerías especializadas lo cual lo vuelven una solución de desarrollo ideal para el prototipado.

4.1.2 Sensor.

Figura 4.2: Sensor ultrasónico MB-1230.

El sensor utilizado es un XL-MaxSonar - EZTM Series, el cual consiste en un sensor ultrasónico empleado en aplicaciones de robótica y automatización que opera a través de la emisión de una onda ultrasónica que rebota contra los objetos y regresa al sensor con lo cual es posible calcular el tiempo que toma a la onda en regresar al sistema y en base a ésto poder calcular la distancia del objeto con respecto al sensor.

La distancia del sensor es de 20 a 120 centrímetros con un voltaje de operación de 5 volts.

La ventaja del sensor es que posee distintos tipos de salida, los cuales son:

1. Salida serial. Envía una cadena de 8 bits conteniendo la información de la distancia del sensor y que puede ser utilizado por sistemas que emplean interfaz tipo UART.

2. Salida analógica. Salida utilizada para el sistema difuso que envía un valor analógico que puede ser calculada por la siguiente fórmula:

$$distancia = \frac{voltajedeentrada}{1024}$$

4.2 Etapa de potencia.

4.2.1 L298N.

El integrado utilizado para el control del motor de corriente directa del navegador es el modelo L298N el cual es un integrado altamente utilizado para aplicaciones con requerimientos de corriente de hasta 2 Amperes en estado estable y 2.5 a 3 Amperes en etapa de transitorio.

Los pines del integrado poseen la siguiente configuración:

En el apéndice B se ilustran los esquemas de conexión del sistema de control con la etapa de potencia hacia los motores.

Figura 4.3: Integrado para control de motores L298N.

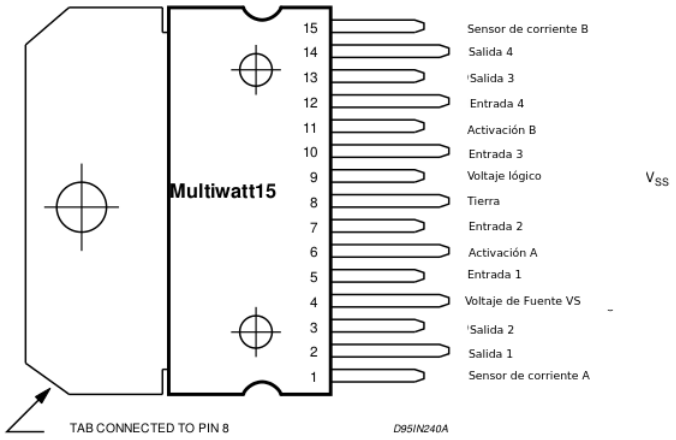


Figura 4.4: Pines del integrado L298N.

4.3 Sistema difuso.

4.3.1 Algoritmo difuso.

Para la elaboración de la base de reglas se consideraron las siguientes condiciones:

Distancia del sistema (Antecedente).

Como entrada del sistema se consideró la distancia medida del sistema con respecto a un objeto frente a éste producto de las mediciones realizadas por el sensor ultrasónico MB-1230.

Se definió lo siguiente:

- Muy alejado. Establecido en un rango mayor a 70 centímetros con respecto a la plataforma móvil.
- Poco alejado. Establecido en un rango de 60 centímetros con respecto a la plataforma móvil.
- Centrado. Establecido en un rango de 50 centímetros con respecto a la plataforma móvil.
- Poco cercano. Establecido en un rango de 40 centímetros con respecto a la plataforma móvil.
- Muy cercano. Establecido en un rango de 30 centímetros con respecto a la plataforma móvil.

Velocidad lineal (Consecuente).

Se estableció como salida el control de la velocidad del motor de corriente directa de la plataforma móvil de manera que reaccione a la diferencia de distancia y corriga el error entre la distancia deseada y la distancia medida con una velocidad directamente proporcional al error. Se estableció lo siguiente:

- Retroceso rápido.
- Retroceso poco rápido.
- Cero movimiento.
- Avance poco rápido.
- Avance rápido.

Base de reglas.

Se establecieron las siguientes condiciones:

- IF distancia es muy cercano THEN velocidad es retroceso rápido.
- IF distancia es poco cercano THEN velocidad es retroceso poco rápido.
- IF distancia es centrado THEN velocidad es cero.
- IF distancia es poco alejado THEN velocidad es avance poco rápido.
- IF distancia es muy alejado THEN velocidad es avance rápido.

4.3.2 eFLL v1.0.4.[2]

Desarrollado para la elaboración de aplicaciones que emplean lógica difusa para la toma de decisiones en base a variables lingüísticas en sistema embebidos, los cuales se caracterizan por requerir la mayor economía en el uso de los recursos del sistema y memoria ya que suelen tener una baja cantidad de éste, requiriendo sistemas de mayor costo para obtener la memoria necesaria para procesar funciones más avanzadas.

La librería hace uso de los algoritmos de Máximos y Mínimos y Mamdani para interencia y composición además de Centro de área para defusificación en un universo continuo.

Instalación.

- Ingresar a la página oficial del proyecto en GitHub: `https://github.com/zerokol/eFLL`
- Clonar el proyecto utilizando Git o descargarlo utilizando el botón “Descargar como ZIP”.
- Renombrar la carpeta de “eFLL-master” a “eFLL” para facilitar reconocimiento de la misma.
- Colocar el contenido en la carpeta “Librerías” de Arduino.

Importación de la librería.

Para importar la librería solamente es necesario abrir el entorno de desarrollo Arduino y seleccionar el menú Sketch ->Librerías ->Importar librerías.

Seleccionar la librería eFLL.

Funciones de pertenencia provistas.

La librería es capaz de emplear funciones de pertenencia triangulares, rampa, trapezoidales y unitarias dependiendo la declaración del objeto de la base de reglas. En este caso:

- Función triangular.

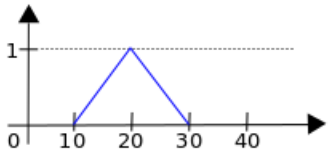


Figura 4.5: Función de tipo triangular.

```
1   FuzzySet* fs = FuzzySet(10, 20, 20, 30);
```

- Función rampa.

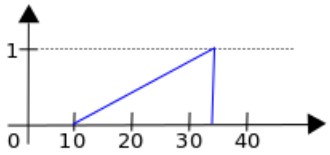


Figura 4.6: Función de tipo rampa.

```
1   FuzzySet* fs = FuzzySet(10, 33, 33, 33);
```

- Función rampa invertida.

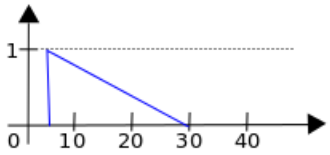


Figura 4.7: Función de tipo rampa invertida.

```
FuzzySet* fs = FuzzySet(5, 5, 5, 30);
```

- Función trapezoidal.

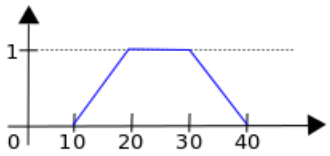


Figura 4.8: Función de tipo trapezoidal.

```
FuzzySet* fs = FuzzySet(10, 20, 30, 40);
```

- Función rampa inicial.

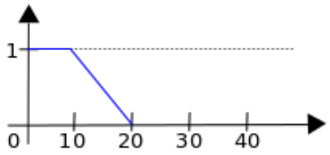


Figura 4.9: Función de tipo rampa inicial.

```
FuzzySet* fs = FuzzySet(0, 0, 10, 20);
```

- Función rampa final.

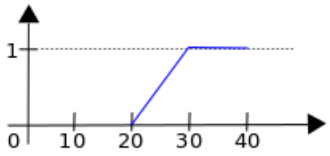


Figura 4.10: Función de tipo rampa final.

```
FuzzySet* fs = FuzzySet(20, 30, 40, 40);
```

- Función unitaria.

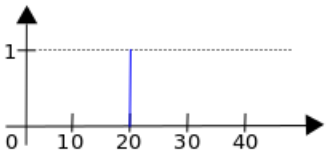


Figura 4.11: Función de tipo unitario.

```
1  FuzzySet* fs = FuzzySet(20, 20, 20, 20);
```

Capítulo 5

RESULTADOS.

5.1 Experimentación.

Se realizaron 10 mediciones de la plataforma móvil estableciendo la distancia base a 60 centímetros, posición que el vehículo debe mantener al hacer uso del algoritmo difuso, obteniendo los resultados que se presentan en la tabla 5.1:

Medición	Distancia (cm).
1	21
2	20.4
3	19.9
4	20.1
5	20.1
6	22
7	19.3
8	19.9
9	20.3
10	19.4

Figura 5.1: Tabla de mediciones realizadas con la plataforma móvil.

En general se percibió un error de +- 2 centímetros y una tendencia en la plataforma a tener dificultades para alcanzar la posición inicial cuando ésta es forzada a avanzar al frente a comparación de cuando es obligada a retroceder.

Estos errores se argumenta son producto de la incapacidad del motor del dispositivo para realizar el movimiento debido a falta de par torsor ya que cuando se realizaron pruebas sin fricción por parte del sueño el dispositivo fue capaz de realizar los movimientos con un tiempo de respuesta satisfactorio.

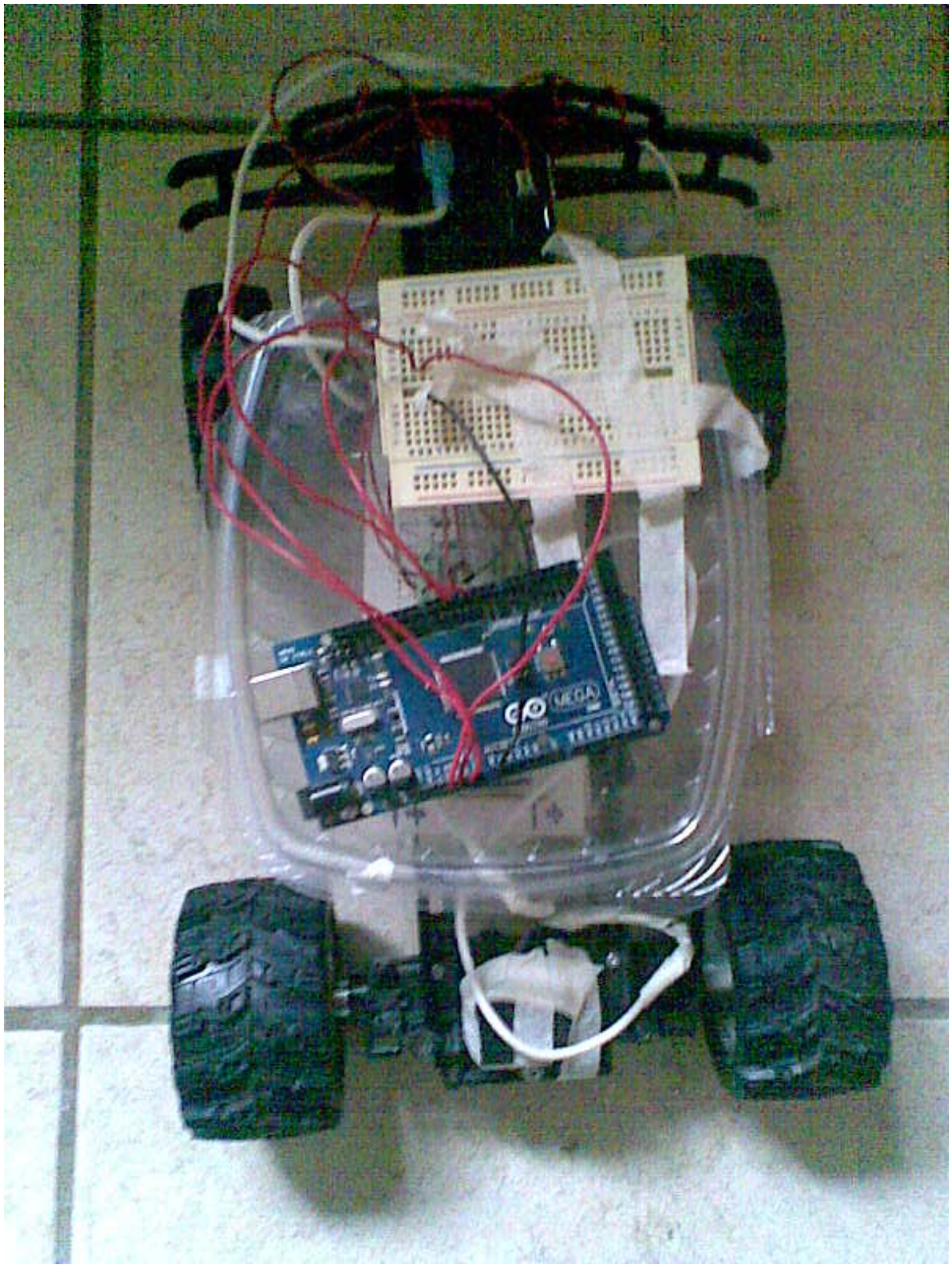


Figura 5.2: Prueba de la plataforma móvil corrigiendo el error.

Capítulo 6

CONCLUSIONES.

Los controladores difusos pueden ser considerados como una solución factible y robusta a problemas de control de complejidad media a elevada debido a la facilidad y rapidez con que éstos pueden ser implementados en una aplicación de control determinada. Sin embargo también pueden llegar a resultar tediosos si la cantidad de bases de reglas se incrementa debido a la complejidad del problema y por tanto requerir una capacidad de procesamiento por parte del dispositivo encargado de implementar el control mucho mayor que la que se requeriría para emplear alternativas como los controladores PID.

Sin embargo resulta altamente efectivo para aplicaciones como la presentada en este documento, en las cuales simplemente estableciendo las condiciones iniciales del problema se vuelve posible llegar a una solución práctica que permite el control del dispositivo sin problemas mayores.

También cabe destacar los beneficios de haber implementado la librería de lógica difusa eFLL ya que a través de ésta es posible colocar el algoritmo en un sistema embebido completamente independiente y con ello reducir el tamaño y coste de la aplicación desarrollada a comparación de alternativas como Matlab Simulink, otra alternativa comercial altamente utilizada que sin embargo requiere, sin la implementación de paquetes especializados, la presencia permanente de una computadora para monitoreo y control del sistema.

Se notó además que el sistema tiene una alta dependencia al entorno de operación y la capacidad del dispositivo en el cual opera ya que en el navegador, a pesar de realizar correctamente la función de pertenencia y enviar la señal al actuador, éste limitó enormemente el desempeño del mismo al carecer del par torsor necesario para desplazarlo.

Se concluye por tanto que es posible y conveniente utilizar los sistemas difusos en aplicaciones de naturaleza simple, empleando técnicas más avanzadas dependiendo la compejidad de la tarea y sugiriendo el uso conjunto de controladores PID y similares para optimizar el desempeño.

Apéndice A

CÓDIGO DEL CONTROLADOR DIFUSO.

A continuación se presenta el código utilizado para la elaboración de la base de reglas del controlador difuso para mantener la posición de la plataforma.

En la primera etapa se realiza la declaración de las librerías necesarias para el control y se definen las entradas y salidas del sensor que envía la señal al microcontrolador y las salidas por PWM que son dirigidas hacia el integrado L298N para el movimiento.

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

//Desired distance (cm).
int ddistance = 25;

int arrayIndex = 0;
// arrayIndex of the current item in the array
int total = 0;
// stores the cumlative total
int averageDistance = 0;

//Analog pin to read the values from the sensor.
int sensorPin = 0;

//PWM pins for the H bridge.
int forward = 5;
int backward = 3;

// stores the pulse in Micro Seconds
unsigned long pulseTime = 0;
// variable for storing the distance (cm)
float dist = 0;
```

```
31
32 //Declaring a new fuzzy object.
33 Fuzzy* fuzzy = new Fuzzy();
34
35 void setup(){
36   //Begin serial module.
37   Serial.begin(9600);
38
39   //Set sensor pin as INPUT.
40   pinMode(sensorPin, INPUT);
41   pinMode(forward,OUTPUT);
42   pinMode(backward,OUTPUT);
```

Posteriormente se declaran las variables de entrada y salida del sistema que en este caso son considerados distancia y velocidad. Se establecieron los antecedentes, consecuentes y las bases de reglas.

```
1
2   //Declare fuzzy input - Distance.
3   FuzzyInput* distance = new FuzzyInput(1);
4   //Stablish the rule bases that compose the fuzzy input.
5   FuzzySet* vf = new FuzzySet(0, 10, 15, 20); //Small distance.
6   distance->addFuzzySet(vf); //Adding the small distance variable.
7   FuzzySet* sf = new FuzzySet(20, 25, 25, 30); //Small distance.
8   distance->addFuzzySet(sf); //Adding the small distance variable.
9   FuzzySet* centrado = new FuzzySet(30, 40, 40, 45); //Small distance.
10  distance->addFuzzySet(centrado); //Adding the small distance variable
        .
11  FuzzySet* sc = new FuzzySet(45, 100, 110, 120); //Median distance.
12  distance->addFuzzySet(sc); //Adding the median distance variable.
13  FuzzySet* vc = new FuzzySet(130, 140, 150, 500); //Big distance.
14  distance->addFuzzySet(vc); //Adding the big distance variable.
15
16  fuzzy->addFuzzyInput(distance); //Adding the distance input to the
        fuzzy variable.
17
18  //Declare fuzzy output - Speed.
19  FuzzyOutput* velocity = new FuzzyOutput(1);
20  // Criando os FuzzySet que compoem o FuzzyOutput velocidade
21  FuzzySet* qbwd = new FuzzySet(-128, -113, -93, -78); // Velocidade
        lenta
22  velocity->addFuzzySet(qbwd); // Adicionando o FuzzySet slow em
        velocity
23  FuzzySet* sbwd = new FuzzySet(-78, -43, -63, -28); // Velocidade
        normal
24  velocity->addFuzzySet(sbwd); // Adicionando o FuzzySet average em
        velocity
25  FuzzySet* zero = new FuzzySet(-28, -10, 10, 28); // Velocidade alta
26  velocity->addFuzzySet(zero); // Adicionando o FuzzySet fast em
        velocity
```

```
27      FuzzySet* sfwd = new FuzzySet(28, 43, 53, 78); // Velocidade normal
28    velocity->addFuzzySet(sfwd); // Adicionando o FuzzySet average em v
29      FuzzySet* ffwd = new FuzzySet(78, 93, 113, 128); // Velocidade
          normal
30    velocity->addFuzzySet(ffwd); // Adicionando o FuzzySet average em v
31
32    fuzzy->addFuzzyOutput(velocity); // Adicionando o FuzzyOutput no
        objeto Fuzzy
33
34    //Fuzzy rules.
35    //IF distance = very close THEN speed = fast bwds
36    FuzzyRuleAntecedent* ifDistanceSmall = new FuzzyRuleAntecedent();
37    ifDistanceSmall->joinSingle(vc);
38    FuzzyRuleConsequent* thenVelocitySlow = new FuzzyRuleConsequent(); //
          Instancinado um Consequente para a expressao
39    thenVelocitySlow->addOutput(qbwd);// Adicionando o FuzzySet
        correspondente ao objeto Consequente
40    // Instanciando um objeto FuzzyRule
41    FuzzyRule* fuzzyRule01 = new FuzzyRule(1, ifDistanceSmall,
        thenVelocitySlow); // Passando o Antecedente e o Consequente da
        expressao
42    fuzzy->addFuzzyRule(fuzzyRule01); // Adicionando o FuzzyRule ao
        objeto Fuzzy
43
44    //IF distance = slighty close THEN speed = slow bwds
45    FuzzyRuleAntecedent* ifDistanceSafe = new FuzzyRuleAntecedent(); //
        Instanciando um Antecedente para a expresso
46    ifDistanceSafe->joinSingle(sc); // Adicionando o FuzzySet
        correspondente ao objeto Antecedente
47    FuzzyRuleConsequent* thenVelocityAverage = new FuzzyRuleConsequent();
          // Instancinado um Consequente para a expressao
48    thenVelocityAverage->addOutput(sbwd);// Adicionando o FuzzySet
        correspondente ao objeto Consequente
49    // Instanciando um objeto FuzzyRule
50    FuzzyRule* fuzzyRule02 = new FuzzyRule(2, ifDistanceSafe,
        thenVelocityAverage); // Passando o Antecedente e o Consequente da
          expressao
51    fuzzy->addFuzzyRule(fuzzyRule02); // Adicionando o FuzzyRule ao
        objeto Fuzzy
52
53    //IF distance = center THEN speed = zero.
54    FuzzyRuleAntecedent* ifDistanceBig = new FuzzyRuleAntecedent(); //
        Instanciando um Antecedente para a expresso
55    ifDistanceBig->joinSingle(centrado); // Adicionando o FuzzySet
        correspondente ao objeto Antecedente
56    FuzzyRuleConsequent* thenVelocityFast = new FuzzyRuleConsequent(); //
          Instancinado um Consequente para a expressao
57    thenVelocityFast->addOutput(zero);// Adicionando o FuzzySet
        correspondente ao objeto Consequente
58    // Instanciando um objeto FuzzyRule
```

```
59    FuzzyRule* fuzzyRule03 = new FuzzyRule(3, ifDistanceBig,
         thenVelocityFast); // Passando o Antecedente e o Consequente da
         expressao
60    fuzzy->addFuzzyRule(fuzzyRule03); // Adicionando o FuzzyRule ao
         objeto Fuzzy
61
62    //IF distance = slightly far THEN speed = slow fwd
63    FuzzyRuleAntecedent* ifDistanceSF = new FuzzyRuleAntecedent(); //
         Instanciando um Antecedente para a expresso
64    ifDistanceSF->joinSingle(sf); // Adicionando o FuzzySet
         correspondente ao objeto Antecedente
65    FuzzyRuleConsequent* thenVelocitySF = new FuzzyRuleConsequent(); //
         Instancinado um Consequente para a expressao
66    thenVelocitySF->addOutput(sfwd);// Adicionando o FuzzySet
         correspondente ao objeto Consequente
67    // Instanciando um objeto FuzzyRule
68    FuzzyRule* fuzzyRule04 = new FuzzyRule(4, ifDistanceSF,
         thenVelocitySF); // Passando o Antecedente e o Consequente da
         expressao
69    fuzzy->addFuzzyRule(fuzzyRule04); // Adicionando o FuzzyRule ao
         objeto Fuzzy
70
71    //IF distance = very far THEN speed = fast fwd
72    FuzzyRuleAntecedent* ifDistanceVF = new FuzzyRuleAntecedent(); //
         Instanciando um Antecedente para a expresso
73    ifDistanceVF->joinSingle(vf); // Adicionando o FuzzySet
         correspondente ao objeto Antecedente
74    FuzzyRuleConsequent* thenVelocityFF = new FuzzyRuleConsequent(); //
         Instancinado um Consequente para a expressao
75    thenVelocityFF->addOutput(ffwd);// Adicionando o FuzzySet
         correspondente ao objeto Consequente
76    // Instanciando um objeto FuzzyRule
77    FuzzyRule* fuzzyRule05 = new FuzzyRule(5, ifDistanceVF,
         thenVelocityFF); // Passando o Antecedente e o Consequente da
         expressao
78    fuzzy->addFuzzyRule(fuzzyRule05); // Adicionando o FuzzyRule ao
         objeto Fuzzy
79  }
```

Se inicializa el ciclo de ejecución y se recibe la señal del sensor ultrasónico. Se fuzzifica la entrada y se realiza la toma de decisiones para el movimiento del sistema.

Finalmente se emplea la función defuzzify para obtener la velocidad deseada y finalmente se envía la señal por medio de PWM a los pines 2 y 5 del Arduino Mega 2560.

```
1   void loop(){
2     pulseTime = analogRead(sensorPin);
3     // Distance = pulse time / 1024 to convert to cm.
4     dist = pulseTime;
5
6     Serial.print("Distancia medida: ");
```

```
    Serial.println(dist);

    fuzzy->setInput(1, dist);

    fuzzy->fuzzify();

    float output = fuzzy->defuzzify(1);

    Serial.print("Distancia: ");
    Serial.println(averageDistance);
    Serial.print("Velocidad: ");
    Serial.println(output);

    //Send the signal to the Arduino to activate the necessary PWM output
       .
    if(output <0)
    {
      Serial.println("Moving forward...");
      analogWrite(forward,200+output);
    }
    else if(output == 0)
    {
      Serial.println("In position...");
      analogWrite(forward,0);
      analogWrite(backward,0);
    }
    else if(output > 0)
    {
      Serial.println("Moving backward...");
      analogWrite(backward,100+output);
    }
    else
    {
      Serial.println("Unknown input.");
    }

    // wait 100 milli seconds before looping again
    delay(500);
  }
```

Apéndice B

CIRCUITO DE POTENCIA.

A continuación se anexan las conexiones del circuito de control hacia la etapa de potencia utilizando el integrado L298N.

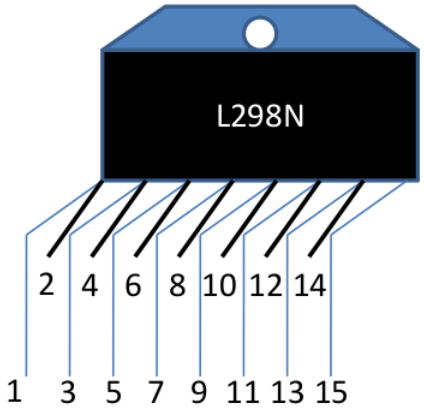


Figura B.1: Indicación de pines del integrado L298N.

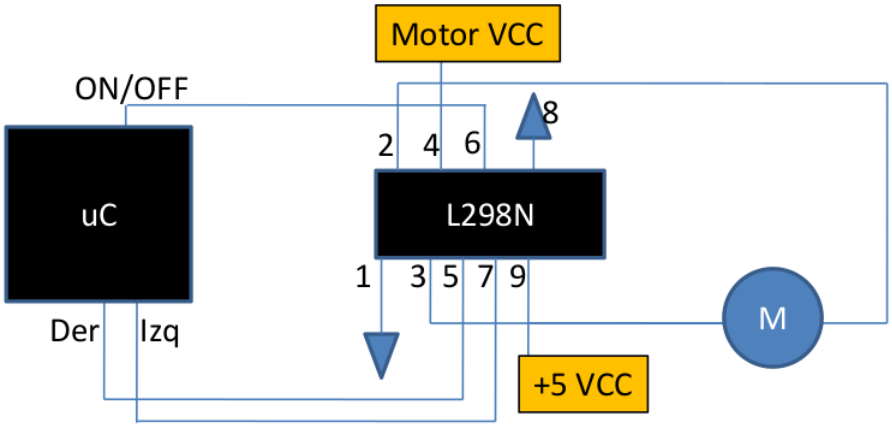
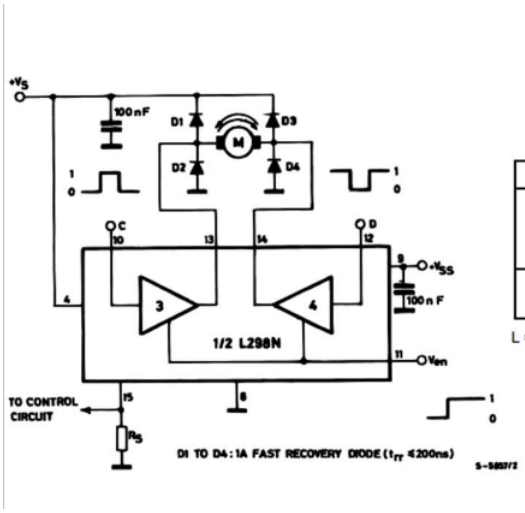


Figura B.2: Diagrama esquemático de conexión del circuito como puente H.



Inputs		Function
V_{en} = H	C = H ; D = L	Forward
	C = L ; D = H	Reverse
	C = D	Fast Motor Stop
V_{en} = L	C = X ; D = X	Free Running Motor Stop

L = Low H = High X = Don't care

Figura B.3: Esquemático obtenido de la hoja de datos del integrado para su uso como puente H.

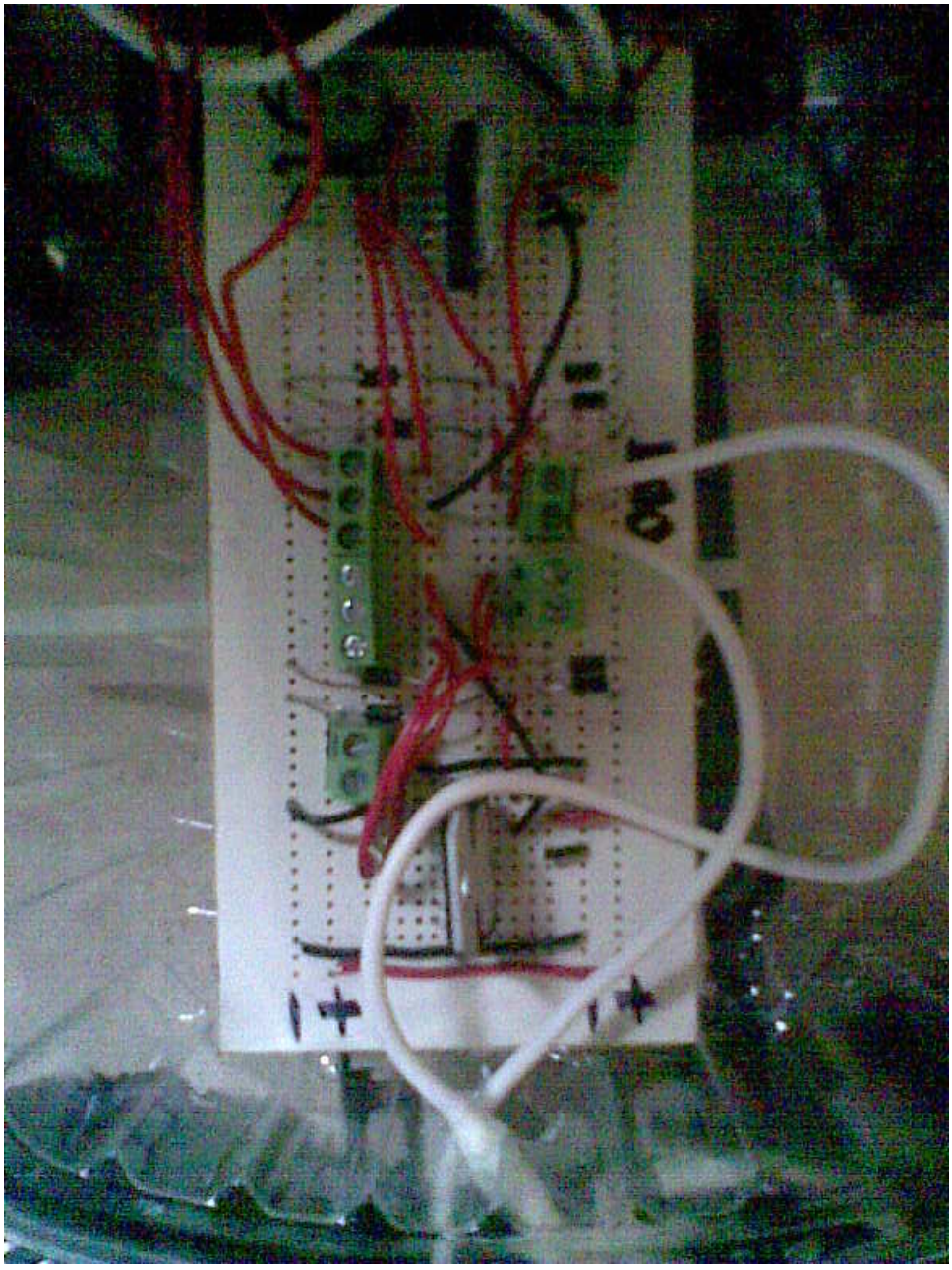


Figura B.4: Circuito de potencia elaborado en tablilla de cobre.

Bibliografía

[1] Seattle Robotics. Visto por última vez: 21/agosto/2013.
http://www.seattlerobotics.org/encoder/mar98/fuz/fl_part1.html#INTRODUCTION

[2] eFLL Zerokol. Visto por última vez: 21/agosto/2013.
http://zerokol.com/post/51e9324ee84c55a1f5000007/1/en

[3] Arduino Mega - Schematics. Visto por última vez: 21/agosto/2013.
http://arduino.cc/en/Main/arduinoBoardMega2560