**Proyecto 3**

**Robot micro mouse maze solving**

Pérez C. Carlos Felipe, Marmolejo V. Alejandro, Foronda M. José

***Resumen*—En este proyecto 3, se diseñará e implementará un robot micro mouse que resolverá un laberinto, mediante, el algoritmo o la técnica de solución de laberintos flood fill, que a su vez conectará los datos mediante comunicación serial, a una interfaz de processing que actualizará los valores de la posición del seguidor, en la tarjeta de desarrollo ATMEGA 2650.**

*Palabras claves*—flood fill, ATMEGA 2560, robot micromouse, motores, llantas,mano derecha.

# Introducción

El algoritmo flood fill, o de inundación, es un algoritmo que se usa para resolver laberintos, basándose en la búsqueda informada, de la posición en la que se encuentran las paredes y de esa manera asignar valores a la posición en la que se encuentre, actualizando sus datos para de manera ordenada llegar así al cero o el centro de la matriz que se define para este tipo de algoritmos. Básicamente este divide un laberinto en celdas cada una con un valor entero, estos valores dependen de la solución del laberinto, es decir de la forma que se lograra llegar al centro del mismo o al cero que es el punto de llegada. Para este proyecto se implementará un robot micro mouse que resolverá un laberinto, mediante el uso del algoritmo flood fill, el cual será controlado con la tarjeta de desarrollo ATMEGA 2650, la cual será el encargado del movimiento y manejo del micro mouse, para esto usaremos , el arreglo de sensores QTR-RC POLOLU , los cuales se encargaran de censar la línea negra, y determinar la posición del robot a medida, que con el algoritmo flood fill se asigna la posición a las paredes y el camino correcto para llegar al punto de solución del laberinto. Para esto además se hará uso de la plataforma de programación Processing, la cual nos facilita la interfaz gráfica de nuestra posición del micro mouse, ya que esta facilita la conexión de datos con la plataforma de arduino.

1. OBJETIVOS
   1. *GENERAL*

Implementar un robot que resuelva un laberinto usando el algoritmo flood fill y a su vez enviar datos al computador usando la interfaz serial para ser graficados en la plataforma Proccesing, se resalta además que se combinara el algoritmo de flood fill con el algoritmo de la mano derecha.

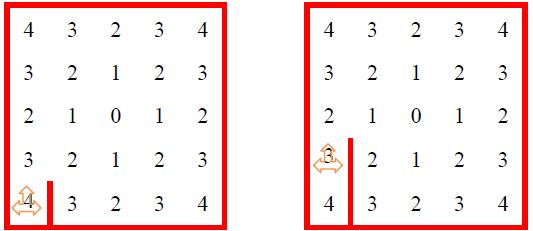
* 1. *ESPECÍFICOS*
* Acondicionar la tarjeta de desarrollo Arduino MEGA 2560 al vehículo de forma que pueda ejercer la ejecución o intercambio de señales con el mismo, de acuerdo a valores censados con el arreglo de sensores QTR8C, ejecutar acciones, que se vean traducidas de acuerdo al software implementado en movimientos lógicos que contribuyan a la resolución del laberinto propuesto.
* Implementar un algoritmo flood fill, con características del método de la mano izquierda, que permitan simplemente con la regleta de sensores QTR8C llevar a cabo el proceso de solución deseado.
* Diseñar e implementar un laberinto físico con paredes y caminos con líneas negras, este con 4 entradas diferentes que dirige a un mismo centro.

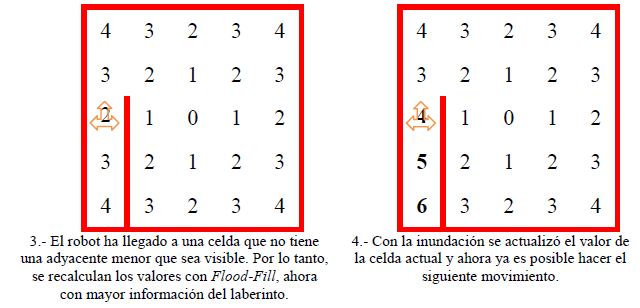
# MARCO TEORICO

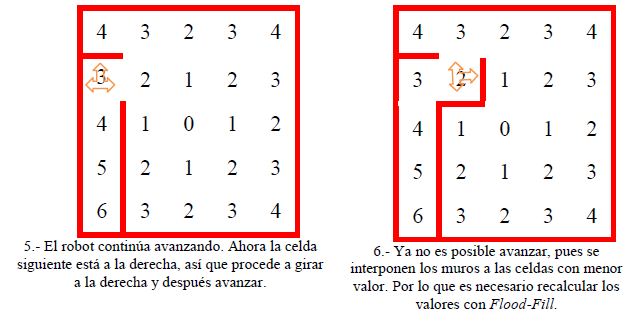
El algoritmo de flood fill, se basa en dividir el laberinto en celdas, a cada una se le da un valor que va de 4 a cero. estos valores se calculan a partir de la solución del laberinto. Es decir, la solución o celda objetivo del laberinto tendrá un valor de 0, lo que quiere decir que se requieren 0 pasos para llegar a ella.

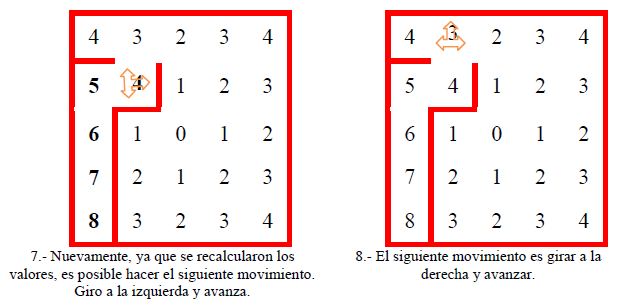
Posteriormente, las celdas adyacentes al objetivo, ya sean de manera vertical u horizontal, tendrán un valor de 1 (primer nivel); las celdas adyacentes a las del primer nivel tendrán un valor de 2 (segundo nivel); las celdas adyacentes a éstas últimas tendrán valor de 3 (tercer nivel), y así sucesivamente. Sin embargo, si existe un muro entre dos celdas, éstas no se considerarán adyacentes visibles.

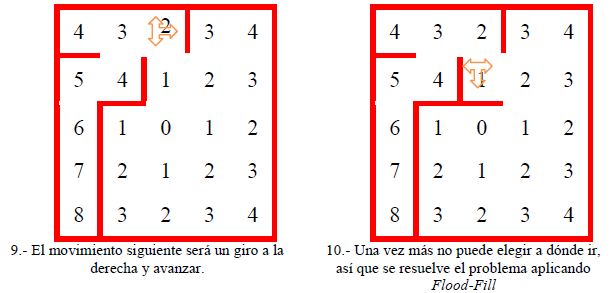
Si bien este algoritmo se le llama “por inundación”, porque este inunda, el laberinto de valores, los cuales determinan el camino a llegar , se resalta que el robot, no conoce el laberinto , el primero lo reconocerá y establecerá donde se encuentran las paredes, y así reasignara valores de acuerdo al camino adecuado a seguir.

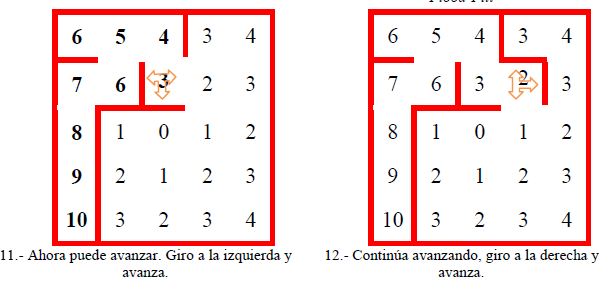












El método de la mano derecha, es un método para resolver algoritmos, el cual se basa en recorrer el laberinto, hasta llegar al final, partiendo de la analogía de poner la mano izquierda en la pared, estableciendo unos parámetros de giro, para entender mejor este, se puede simplificar dando unos cuantos pasos, los cuales son:

Como prioridad tener en cuenta que donde pueda girar a la izquierda, girara a la izquierda.

En este orden de ideas el segundo concepto en importancia es que, si no puede tomar un giro a la izquierda en determinada parte del camino, tenga como segunda opción seguir un camino recto.

Si en una intersección no puede girar a la izquierda, ni seguir recto en el camino, tomará en cuenta como siguiente opción de giro la dirección a la derecha, y si tampoco puede tomar un giro a la derecha, esto ya significa que está en un callejón sin salida por lo que deberá girar 180°.

De manera que este algoritmo define los giros o determinada dirección con letras, las cuales representan:

-L=voltea a la izquierda

-R=voltea a la derecha

- B=da la vuelta giro de 180°

-S=ir derecho luego de pasar un giro.

De manera que dependiendo de la forma del laberinto se configura su manera de llegar al final. Para entender mejor esto podremos ver una serie de gráficas.

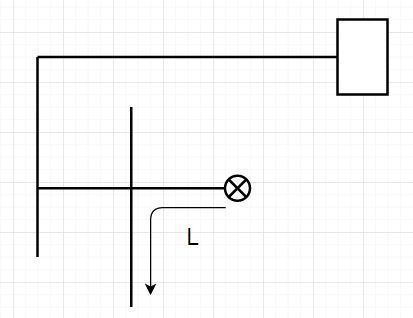


Figura1. Cambio a la izquierda

7

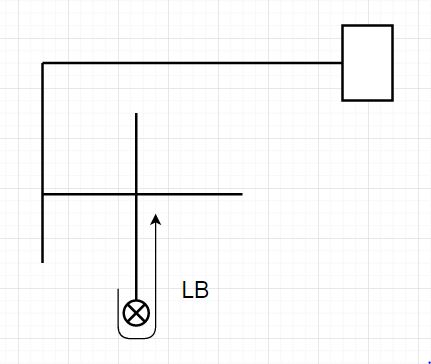


Figura 2. Da la vuelta luego de girar a la izquierda

# PROCEDIMIENTO

Para el desarrollo del chasis del carro se utilizaron los siguientes materiales:

* 2 ruedas de arrastre
* 1 rueda loca
* 1 circuito puente H
* 8 Sensores de referencia Array QTR-8RC Reflectance
* 2 moto reductores
* 1 chasis auto inteligente
* Tarjeta de desarrollo Arduino Mega 2560



Fig.2. ruedas de arrastre

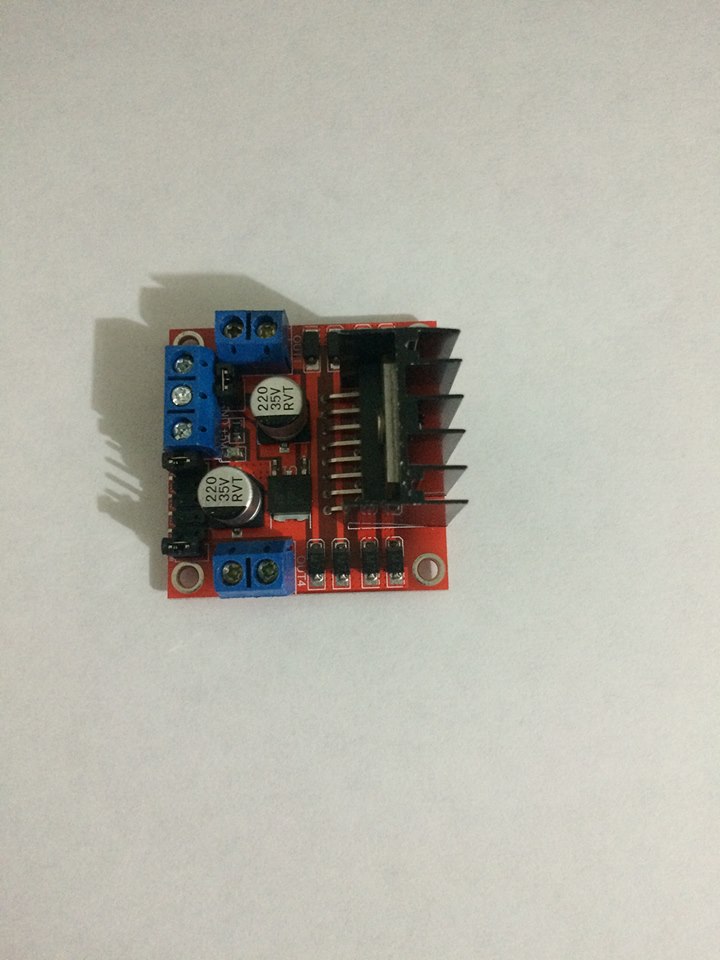


Fig.3.circuito puente H.

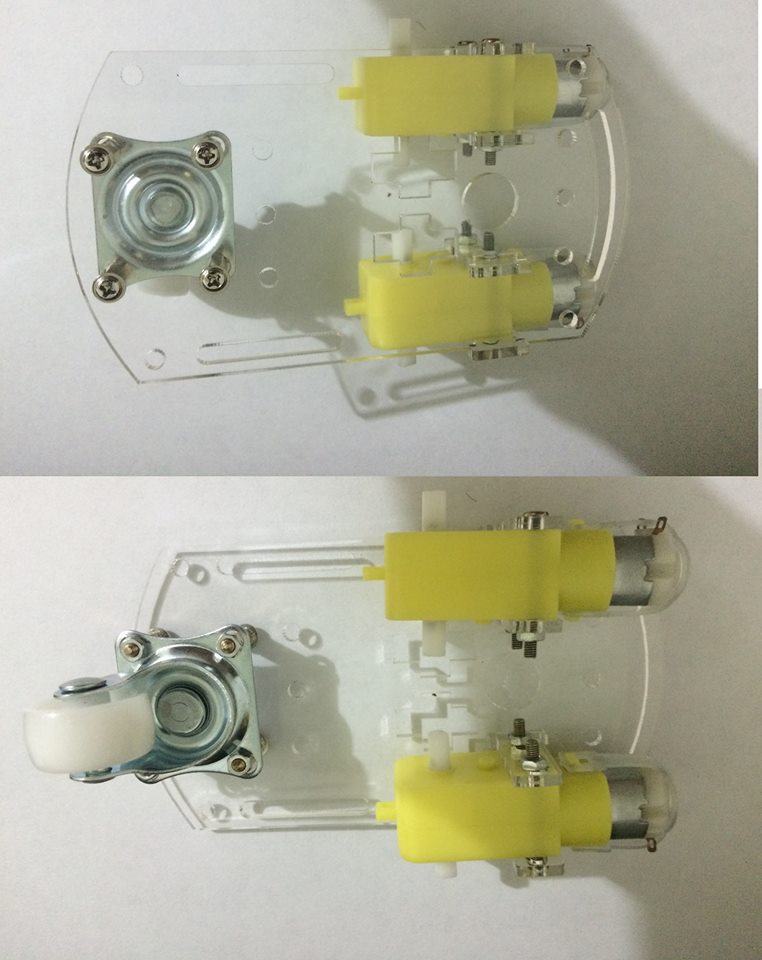


Fig.4. rueda loca, par de motores y base del chasis

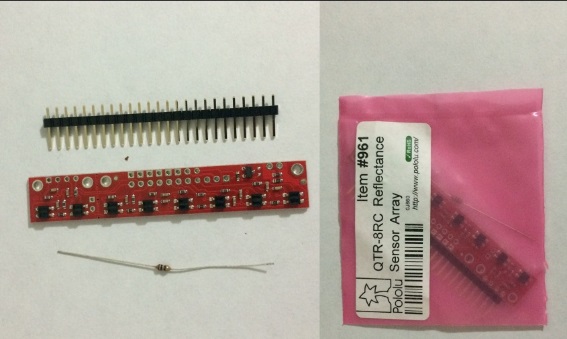


Fig.5. sensores para la detección de línea.



Fig.6. vehículo y diferentes materiales.

***Descripción del método realizado***

***Descripción del PID a emplear***

En primer lugar, definamos que es un controlador PID.

Entonces un PID es un controlador de uso genérico, el cual emplea realimentación de lazo cerrado, es un sistema al que le ingresa un error correspondiente a la diferencia entra la salida deseada (set point) y el valor de salida obtenido en la salida final del sistema. El controlador intentara en todo momento minimizar el error ajustando la entrada al sistema

El PID es un sistema al que le entra un **error** calculado a partir de la salida deseada menos la salida obtenida y su salida es utilizada como entrada en el sistema que queremos controlar. El controlador intenta minimizar el error ajustando la entrada del sistema.

Este controlador tiene implícito 3 parámetros y cada uno influye en mayor o menor medida sobre alguna característica de la salida ya sea el tiempo de establecimiento, sobre impulso, entre otros.

En este orden de ideas entonces los parámetros de los que se hablaba son

* Acción proporcional

Siendo la base de los tres parámetros, da una entrada de control proporcional con el error, usar solo este parámetro genera un error estacionario.

* Acción integral

Este da una suma de los errores previos a la entrada del sistema de control continuando hasta que el valor llegue al deseado, la mayoría de las veces el termino integral está asociado a un término proporcional, con lo cual se elimina el error estacionario.

* Acción derivativa

Este da una respuesta que es proporcional a la velocidad del cambio del error, con esta acción, y adicional a las demás se mejora la velocidad del sistema y contribuye a la disminución de oscilaciones excesivas.

CODIGO ARDUINO EXPLICACION:

Para nuestro código (en ANEXOS), Este código empleado permite que el robot siga una rutina basada el algoritmo flood fill, combinado con características del procedimiento de la mano derecha, ello necesario, puesto que se empleó solo la regleta de sensores QTR8C, así que, mediante la combinación de esta técnica, en teoría se facilitaría el adecuado posicionamiento dentro de la matriz y una adecuada orientación del robot, permitiendo así movimientos acordes a lo esperado.

CODIGO PROCESSING EXPLICACION

El código empleado en la plataforma proccesing se realizó con el fin de llevar a cabo una comunicación serial inalámbrica, de forma que se pudiera comunicar al computador la posición actual del robot en la matriz, entonces lo que el código de proccesing, realiza es mostrar la matriz, emular las paredes con las que cuenta el laberinto real, y mover el micro mouse (representado por un cuadro de color negro), en este sentido se destacan las funciones empeladas en el código processing

* Setwalls: fija las paredes de acuerdo a las coordenadas que se le pasen, definidas como, NORTH, EAST, SOUTH, WEST.
* Graw: se encarga mostrar gráficamente la matriz, el micro mouse, y la numeración en la cuadricula.

# **conclusiones**

* se analizó la efectividad de la lógica en las secuencias descritas por el código flood fill combinado con l algoritmo de seguimiento de la mano izquierda.
* Se verifico el poder y alcance que pueden tener la aplicación de microcontroladores en la práctica, que en este caso, desde el punto de vista de control, representa una oportunidad importante, como se vio, la utilización de un controlador PID en una aplicación común, como el robot maze solving.
* Se comprobó como el control PID mejoro nuestra respuesta, considerablemente, pues le daba mejor control al seguidor de línea, pero este no siempre mejoraba de acuerdo a los valores establecidos en nuestras constantes su respuesta podía llegar a ser peor también, en especial si se establecía una constante de integración muy grande.
* Se verifico el comportamiento de puente H al establecer el manejo de los motores ya que este, les daba la dirección deseada y los cambios necesarios, para que el carro girara, además de sumarle algo de potencia al giro de las llantas.
* Se construyó la estructura física del vehículo seguidor de línea de acuerdo a parámetros de diseño tenidos en cuenta.
* Se acondicionó la tarjeta de desarrollo al vehículo de forma que pueda ejercer la ejecución o intercambio de señales con el mismo, y los sensores que este tiene adaptados.
* Se diseñó el código base PID que permite realizar la ejecución del movimiento de la forma más estable y acorde posible al comportamiento esperado.

# REFERENCIAS

[1] Virginia Mazzone; Controladores PID; marzo 2002

**ANEXOS**

Algoritmo en lenguaje C utilizado para la implementación.

#include <QTRSensors.h> // Pololu QTR Library

//ddefinicion sensores

#define NUM\_SENSORS 8 // sensores usados

#define TIMEOUT 2500 // 2.5 milisegundos para sensar

/// sensores 0-7 estN Conectados a pines digitales 2 hasta 10 respectivamente el pin 3 es usado por un motor

QTRSensorsRC qtrrc((unsigned char[]) {2, 4, 5, 6, 7, 8, 9,10},

NUM\_SENSORS, TIMEOUT);

unsigned int sensorValues[NUM\_SENSORS];

unsigned int line\_position=0; // 0-7000 indica valor de sensores 0-7

// motor driver

int pwm\_a = 3;

int dir\_a = 12;

int pwm\_b = 11; //PWM control for motor outputs 3 and 4 is on digital pin 11 (Right motor)

int dir\_b = 13; //direction control for motor outputs 3 and 4 is on digital pin 13 (Right motor)

//

int calSpeed = 165;

int turnSpeed = 200; //ajustar el valor de motores se ejecutarán mientras que da vuelta.

int turnSpeedSlow = 125; // tune value motors will run as they slow down from turning cycle to avoid overrun (0-255) ajustar el valor de motores se ejecutarán tal que reducir la velocidad de torneado ciclo para evitar la saturación.

int drivePastDelay = 300; // tune value in mseconds motors will run past intersection to align wheels for turn//ajustar el valor en segundos motores funcionará más allá de la intersección para alinear ruedas para girar.

// pid

float error=0;

float lastError=0;

float PV =0 ;

float kp = 0; // tune value in follow\_line() function

//float ki = 0; // ki is not currently used

float kd =0; // tune value in follow\_line() function

int m1Speed=0;

int m2Speed=0;

int motorspeed=0;

//La variable path guarda el camino que ha tomado el robot

// se almacena como una matriz de caracteres, cada uno de ellos representa el giro

//que debe hacerse en una intersección en la secuencia:

// 'L' for left

// 'R' for right

// 'S' for straight (going straight through an intersection)//va directamente a través de una intersección

// 'B' for back (U-turn)

char path[100] = "";

unsigned char path\_length = 0; //la longitud de la trayectoria

void setup()

{

pinMode(pwm\_a, OUTPUT); //Pines de control conjunto que salidas

pinMode(pwm\_b, OUTPUT);

pinMode(dir\_a, OUTPUT);

pinMode(dir\_b, OUTPUT);

int turnSpeedSlow = 125;

analogWrite(pwm\_a, 0); ///establecer dos motores para parar en (100/255 = 39) % ciclo de trabajo (lento)

analogWrite(pwm\_b, 0);

Serial.begin(9600);

delay(2000);

for (int i = 0; i <= 100; i++) // begin calibration cycle to last about 2.5 seconds (100\*25ms/call)

{

// auto calibration sweeping left/right, tune 'calSpeed' motor speed at declaration //auto calibración barrido izquierda/derecha, ajustar la velocidad del motor 'calSpeed' en la declaración

if (i==0 || i==60) //barrido lento gire a la derecha para pasar los sensores sobre la línea.

{

digitalWrite(dir\_a, LOW); //con low adelante con higt retrocede es in2

analogWrite(pwm\_a, calSpeed); //es in1

digitalWrite(dir\_b, HIGH);

analogWrite(pwm\_b, calSpeed);

}

else if (i==20 || i==100) //barrido lento gire a la izq para pasar los sensores sobre la línea.

{

digitalWrite(dir\_a, HIGH);

analogWrite(pwm\_a, calSpeed);

digitalWrite(dir\_b, LOW);

analogWrite(pwm\_b, calSpeed);

}

qtrrc.calibrate(); // lee los sensores cada 25 ms

} // termina calibracion

///leer valores de sensores calibrados y obtener una medida

//de la posición de la línea de 0 a 7000 para obtener valores de los sensores crudos

line\_position = qtrrc.readLine(sensorValues);

while (sensorValues[6] < 200) // espera estar cerca al centro de la linea

{

line\_position = qtrrc.readLine(sensorValues);

}

//disminuya la velocidad.

analogWrite(pwm\_a, turnSpeedSlow);

analogWrite(pwm\_b, turnSpeedSlow);

// find center

while (line\_position > 4350) // wait for line position to find center

{

line\_position = qtrrc.readLine(sensorValues);

}

// stop both motors

analogWrite(pwm\_b, 0); // frena

analogWrite(pwm\_a, 0);

} // end setup

void loop()

{

// read calibrated sensor values and obtain a measure of the line position from 0 to 7000

// To get raw sensor values, call:

// qtrrc.read(sensorValues); instead of unsigned int position = qtrrc.readLine(sensorValues);

unsigned int line\_position = qtrrc.readLine(sensorValues);

MazeSolve(); // comment out and run serial monitor to test sensors while manually sweeping across line//comment out and run serial monitor to test sensors while manually sweeping across line.

}

//line following subroutine

// PD Control

void follow\_line() //follow the line

{

lastError = 0;

while(1)

{

line\_position = qtrrc.readLine(sensorValues);

switch(line\_position)

{

default:

error = (float)line\_position - 3500;

kp=.5;

kd=1;

PV = kp \* error + kd \* (error - lastError);

lastError = error;

//se limita la rapidez a 55

if (PV > 55)

{

PV = 55;

}

if (PV < -55)

{

PV = -55;

}

m1Speed = 200 + PV;

m2Speed = 200 - PV;

//set motor speeds

digitalWrite(dir\_a, LOW);

analogWrite(pwm\_a, m2Speed);

digitalWrite(dir\_b, LOW);

analogWrite(pwm\_b, m1Speed);

break;

}

//Utilizamos los seis sensores internos (1 al 6) para determinar si hay una línea recta

//y los sensorees 0 y 7 sensores de 0 y 7 Si el camino se vuelve.

if(sensorValues[1] < 100 && sensorValues[2] < 100 && sensorValues[3] < 100 && sensorValues[4] < 100 && sensorValues[5] < 100 && sensorValues[6] < 100)

{

//No hay ninguna línea visible por delante, y no vimos ninguna intersección. Debe ser un callejón sin salida.

return;

}

else if(sensorValues[0] > 200 || sensorValues[7] > 200)

{

// interseccion.

return;

}

}

} // fin follow\_line

void turn(char dir)

{

int turnSpeedSlow = 125;

switch(dir)

{

// gro a izquierda 90deg

case 'L':

digitalWrite(dir\_a, HIGH);

analogWrite(pwm\_a, turnSpeed);

digitalWrite(dir\_b, LOW);

analogWrite(pwm\_b, turnSpeed);

line\_position = qtrrc.readLine(sensorValues);

while (sensorValues[6] <200) // wait for outer most sensor to find the line// Espere a que sensor mas externo encuentre la línea.

{

line\_position = qtrrc.readLine(sensorValues);

}

// baja vlocidad

analogWrite(pwm\_a, turnSpeedSlow);

analogWrite(pwm\_b, turnSpeedSlow);

// find center

while (line\_position > 4350) //sintonizar - esperar a que la posición\_de\_línea de encontrar cerca del centro.

{

line\_position = qtrrc.readLine(sensorValues);

}

// stop ambos motores

analogWrite(pwm\_b, 0);

analogWrite(pwm\_a, 0);

break;

// Turn right 90deg

case 'R':

digitalWrite(dir\_a, LOW);

analogWrite(pwm\_a, turnSpeed);

digitalWrite(dir\_b, HIGH);

analogWrite(pwm\_b, turnSpeed);

line\_position = qtrrc.readLine(sensorValues);

while (sensorValues[1] <200) // essperar a q el mas externo ecuentre la linea

{

line\_position = qtrrc.readLine(sensorValues);

}

// slow down speed

analogWrite(pwm\_a, turnSpeedSlow);

analogWrite(pwm\_b, turnSpeedSlow);

// find center

while (line\_position < 3250) // tune - wait for line position to find near center

{

line\_position = qtrrc.readLine(sensorValues);

}

// stop both motors

analogWrite(pwm\_a, 0);

analogWrite(pwm\_b, 0);

break;

// Turn right 180deg to go back

case 'B':

digitalWrite(dir\_a, LOW);

analogWrite(pwm\_a, turnSpeed);

digitalWrite(dir\_b, HIGH);

analogWrite(pwm\_b, turnSpeed);

line\_position = qtrrc.readLine(sensorValues);

while (sensorValues[1] <200) // esperar a sensor mas externo se encuentre con la linea

{

line\_position = qtrrc.readLine(sensorValues);

}

// baja vel

analogWrite(pwm\_a, turnSpeedSlow);

analogWrite(pwm\_b, turnSpeedSlow);

// encuentra centro linea

while (line\_position < 3250) // esperar cercania al centro

{

line\_position = qtrrc.readLine(sensorValues);

}

// stop ambos motores

analogWrite(pwm\_a, 0);

analogWrite(pwm\_b, 0);

break;

case 'S':

//nada

break;

}

}

char select\_turn(unsigned char found\_left, unsigned char found\_straight, unsigned char found\_right)

{

if(found\_left)

return 'L';

else if(found\_straight)

return 'S';

else if(found\_right)

return 'R';

else

return 'B';

}

//Simplificación de la ruta. La estrategia es que cada vez que encontramos una secuencia xBx,

// nosotros podemos simplificar cortando el callejón sin salida. Por ejemplo, LBL-> S, porque un solo S omite el callejón sin salida representado por LBL

void simplify\_path()

{

//sólo simplificar el camino si la segunda a la última vuelta fue una 'B'

if(path\_length < 3 || path[path\_length-2] != 'B')

return;

int total\_angle = 0;

int i;

for(i=1;i<=3;i++)

{

switch(path[path\_length-i])

{

case 'R':

total\_angle += 90;

break;

case 'L':

total\_angle += 270;

break;

case 'B':

total\_angle += 180;

break;

}

}

// angulo entre 0-360

total\_angle = total\_angle % 360;

// reemplza secuencias encontradas

switch(total\_angle)

{

case 0:

path[path\_length - 3] = 'S';

break;

case 90:

path[path\_length - 3] = 'R';

break;

case 180:

path[path\_length - 3] = 'B';

break;

case 270:

path[path\_length - 3] = 'L';

break;

}

path\_length -= 2;

}

void MazeSolve()

{

while(1)

{

follow\_line();

//Conduzca recto un poco. Esto nos ayuda en caso de entrar en la intersección en un ángulo.

digitalWrite(dir\_a, LOW);

analogWrite(pwm\_a, 200);

digitalWrite(dir\_b, LOW);

analogWrite(pwm\_b, 200);

delay(25);

//Estas variables registran si el robot ha visto una línea a la izquierda, recto y la derecha, mientras se examina la intersección actual.

unsigned char found\_left=0;

unsigned char found\_straight=0;

unsigned char found\_right=0;

// Now read the sensors and check the intersection type.

line\_position = qtrrc.readLine(sensorValues);

// chequea sensores mas externos

if(sensorValues[0] > 200)

found\_right = 1;

if(sensorValues[7] > 200)

found\_left = 1;

//

//Conducir recto un poco más - esto es suficiente para alinear las ruedas con la intersección.

digitalWrite(dir\_a, LOW);

analogWrite(pwm\_a, 200);

digitalWrite(dir\_b, LOW);

analogWrite(pwm\_b, 200);

delay(drivePastDelay);

line\_position = qtrrc.readLine(sensorValues);

if(sensorValues[1] > 200 || sensorValues[2] > 200 || sensorValues[3] > 200 || sensorValues[4] > 200 || sensorValues[5] > 200 || sensorValues[6] > 200)

found\_straight = 1;

//Compruebe que el punto final. Si todos los sensores de mediados seis están en negro oscuro, hemos resuelto el laberinto.

if(sensorValues[1] > 600 && sensorValues[2] > 600 && sensorValues[3] > 600 && sensorValues[4] > 600 && sensorValues[5] > 600 && sensorValues[6] > 600)

break;

unsigned char dir = select\_turn(found\_left, found\_straight, found\_right);

// hacer el giro indicado por el path.

turn(dir);

// guardar camno aprendido

path[path\_length] = dir;

path\_length ++;

// simplificar camino aprendido

simplify\_path();

}

// laberinto resuelto

// ciclo infinito puede resolverlo cuantas veces quiera

while(1)

{

analogWrite(pwm\_a, 0); // par ambos motores

analogWrite(pwm\_b, 0);

line\_position = qtrrc.readLine(sensorValues);

while(sensorValues[1] > 200 && sensorValues[2] > 200 && sensorValues[3] > 200 && sensorValues[4] > 200 && sensorValues[5] > 200 && sensorValues[6] > 200)

{

line\_position = qtrrc.readLine(sensorValues);

delay(50);

}

// espera hasta que el roboo se pone de nuevo sobre la linea de inicio para hacer el recorrido

while(1)

{

line\_position = qtrrc.readLine(sensorValues);

if(sensorValues[2] > 200 || sensorValues[3] > 200 || sensorValues[4] > 200 || sensorValues[5] > 200 && sensorValues[0] < 200 && sensorValues[1] < 200 && sensorValues[6] < 200 && sensorValues[7] < 200)

break;

delay(50);

}

delay(2000);

int i;

for(i=0;i<path\_length;i++)

{

//

follow\_line();

digitalWrite(dir\_a, LOW);

analogWrite(pwm\_a, 200);

digitalWrite(dir\_b, LOW);

analogWrite(pwm\_b, 200);

delay(drivePastDelay);

turn(path[i]);

}

follow\_line();

digitalWrite(dir\_a, LOW);

analogWrite(pwm\_a, 200);

digitalWrite(dir\_b, LOW);

analogWrite(pwm\_b, 200);

delay(drivePastDelay); // tune time to allow wheels to position for correct turning

}

}

C:\Users\Alejandro Marmolejo\Desktop\descarga.png

Fig.7. Representación en bloques del sistema a emplear

C:\Users\Alejandro Marmolejo\Desktop\descarga.png

Fig 8. Diagrama de flujo del algoritmo implementado