ROBOT RESUELVE LABERINTOS CON ALGORITMO FLOOD FILL

Arcos Cerón Jorge Alberto, Arcos Araujo Brayan

*Resumen*—El presente informe tiene como fin exponer el análisis, diseño, simulación y correspondiente Implementación de un robot resuelve laberintos, orientado por un Arreglo de sensores QTR-8RC, controlado por micro-controlador Atmega328P incluido en la tarjeta de desarrollo Arduino Uno, codificado a partir del Algoritmo conocido como FLOOD FILL. Además de ello y con la ayuda del Software Processing se desarrollara la interfaz que permitirá la visualización del proceso de solución del laberinto. para lo que se aplicara los conocimientos adquiridos en el curso en lo que al tema respecta, para así diseñar y verificar de manera correcta, la implementación, el funcionamiento y el análisis de dicho sistema.

***Abstract -* This report aims to present the analysis, design, simulation and corresponding implementation of a robot solves mazes , guided by an array of QTR - 8RC sensors , controlled by micro - controller ATMEGA328P included in the development board Arduino Uno , coded from algorithm known as FLOOD FILL . Moreover , with the help of Software Processing interface will allow viewing of the maze solving process is developed. for which the knowledge acquired in the course as regards the issue , so design and verify correctly, implementation , operation and analysis of the system was applied.**

1. **INTRODUCCION**

El robot resuelve laberintos es comúnmente conocido como Micromouse,

El Micromouse es un robot móvil aoutonomo que integra las necesidades y características mecánicas de una plataforma móvil (velocidad, giro sincronizado, entre otras), la capacidad sensorial para el reconocimiento del entorno y la habilidad de interactuar con éste por medio de información digitalizada y codificada. El robot usa esa características para cumplir con el objetivo de solucionar un laberinto.

La solución de laberintos ha sido estudiada por muchos años en el campo de las matemáticas, por lo que existen varios algoritmos que pueden ser aplicados.

Un algoritmo es una serie de instrucciones que instruyen paso a paso a una máquina para la solución de un problema específico, la solución en nuestro caso es la ruta que le toma al robot navegar desde el inicio hasta la meta, para lo que requiere conocer información sobre la presencia de las paredes.

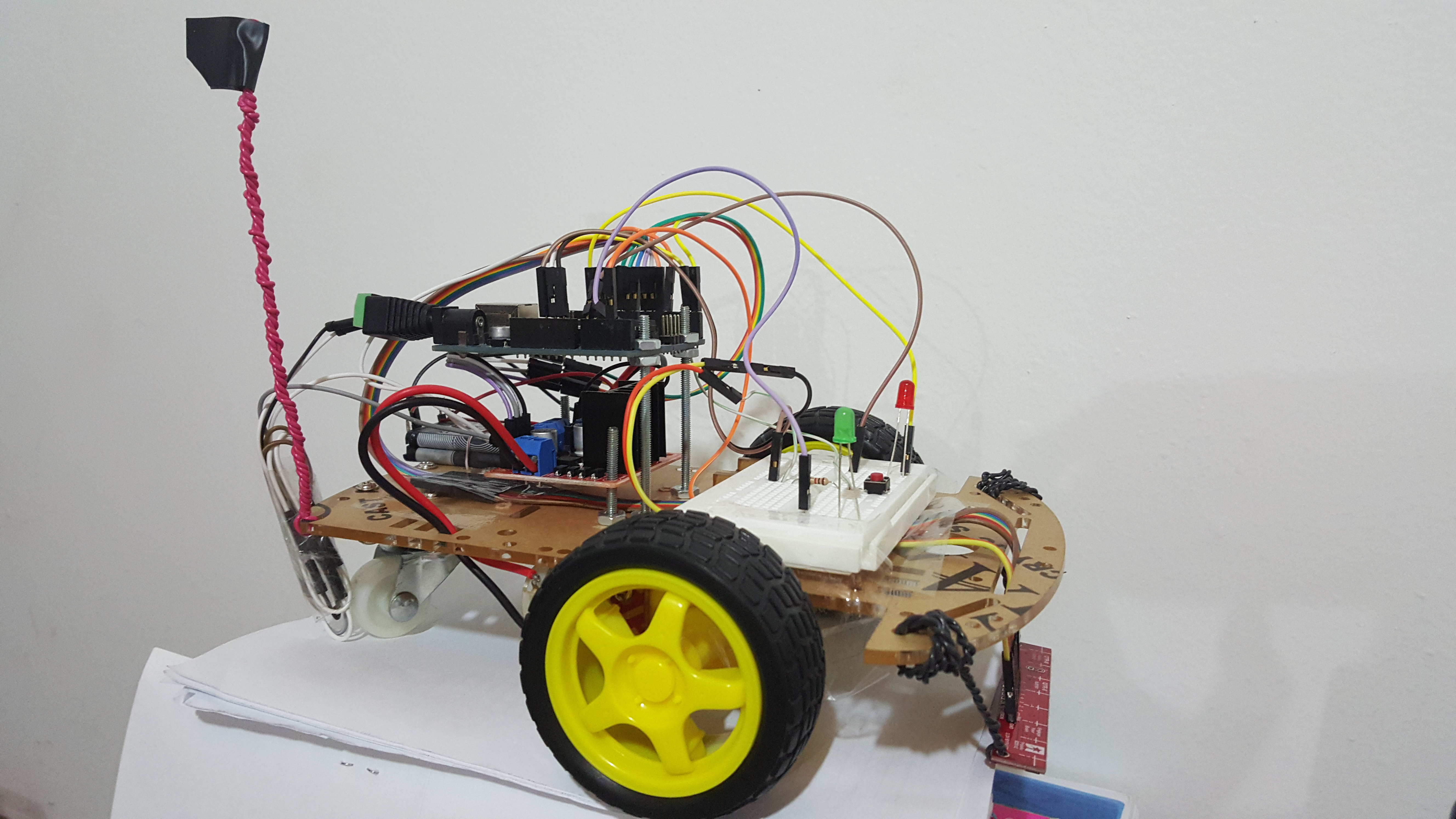
Para este proyecto se implementara el algoritmo FLOOD FILL también conocido como Algoritmo de inundación.

1. **OBJETIVOS**

* Analizar Diseñar e implementar un robot resuelve laberintos, que mediante el seguimiento de una línea negra resuelva un laberinto usando el Algoritmo FLOOD FILL.
* Implementar la Codificación necesaria para lograr la visualización del proceso de recorrido y solución del laberinto en la PC, por medio de una comunicación serial y con ayuda del software Processing.

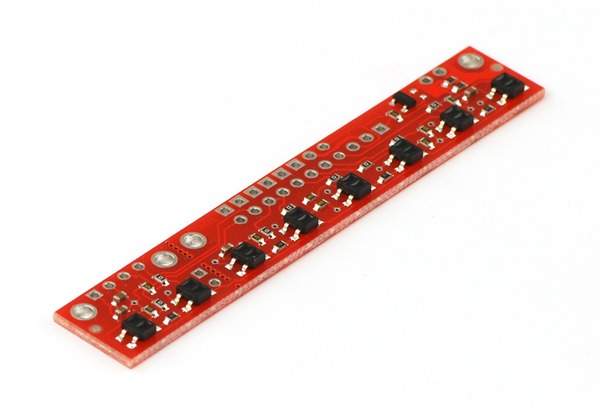
1. **MATERIALES**

El proyecto actual está conectado directamente con el proyecto del Carro Seguidor de Línea, por lo que este último será la base del Robot resuelve Laberintos.



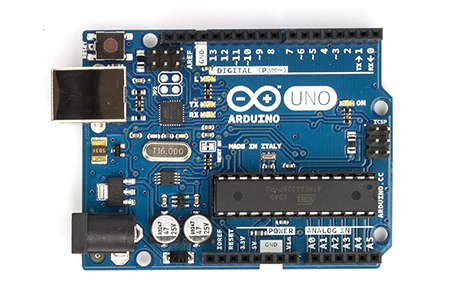
Recordando Los componentes principales del carro seguidor de Linea, que ahora serán de gran impacto en este Nuevo proyecto son:

**ARREGLO DE SENSORES QTR-8RC**



El arreglo Digital de 8 sensores, permitirá al Micro Mouse la navegación adecuada por el laberinto, pues este estará delineado por una línea negra centrada extendida por todas las calles del laberinto.

**PLACA ARDUINO UNO**



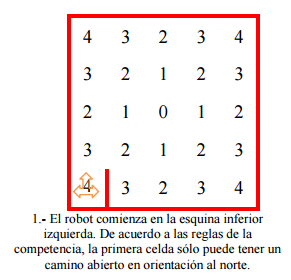
El Arduino UNO es una placa de desarrollo que integra un micro controlador ATmega328P, que será el cerebro de nuestro robot, pues será quien tome las decisiones para la dirección y velocidad de los motores de acuerdo a la lectura de los sensores que este mismo haga.

Esta placa posee un entorno de desarrollo (IDE) la cual usaremos para implementar la codificación del algoritmo FLOOD FILL que permitira al robot el desarrollo exitoso del laberinto.

1. **MARCO TEÓRICO**

**ALGORITMO FLOOD FILL O DE INUNDACIÓN**

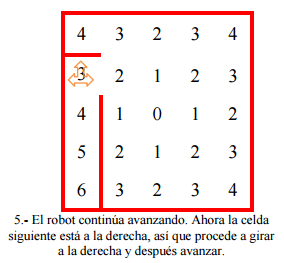
Este algoritmo es una implementación de una búsqueda informada, ya que se utilizan funciones heurísticas en cada celda del laberinto para saber la distancia a la que el robot se encuentra de la solución, que está situada en el centro del laberinto, y así poder llegar a ella. El algoritmo se basa en dividir el laberinto en celdas, cada una con un valor heurístico. Los 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. Cabe mencionar que como condiciones iniciales se tienen las dimensiones del laberinto y la posición de la celda objetivo. El término de inundación se le da a este algoritmo precisamente porque se encarga de “inundar” el laberinto con los valores, como si se vertiera agua desde la celda solución y se fuera desplazando a través del laberinto por diferentes caminos, pasando por diferentes niveles. De principio, el robot no conoce el mapa del laberinto, por lo que tendrá que navegar a través de él utilizando los actuadores (motores) y usando los sensores para detectar muros. El avance del robot será celda por celda y los movimientos son verticales u horizontales. A continuación, se muestra una serie de imágenes en las que puede observarse el comportamiento del algoritmo Flood-Fill. Para fines prácticos, en el ejemplo habrá un laberinto de orden 5x5.

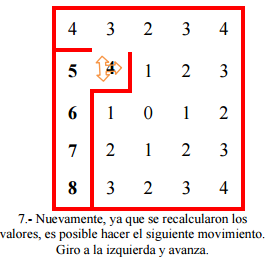
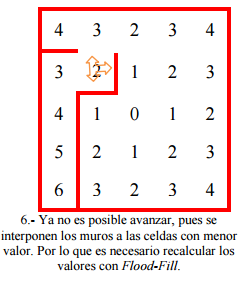


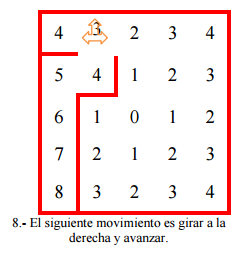


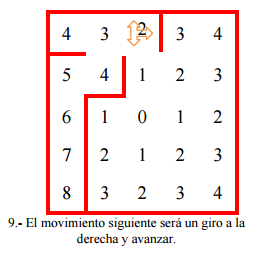


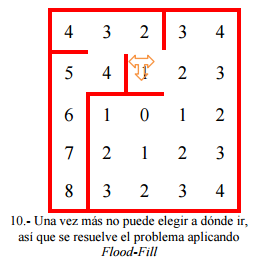


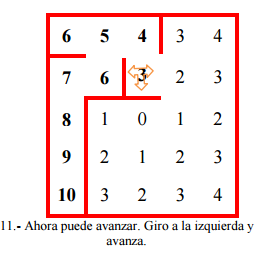


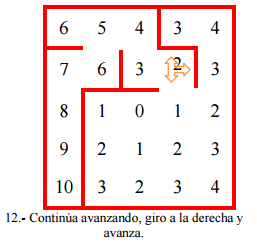


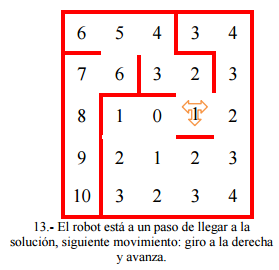


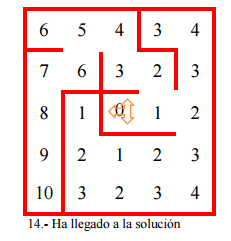












1. **FUNCIONAMIENTO**

El funcionamiento del nuestro Micro-Mouse depende esencialmente de el algoritmo que implementemos en nuestro entorno de desarrollo IDE de Arduino.

Para el movimiento se utilizara el código del Carro seguidor de Linea pues como se a mencionado anteriormente el laberinto esta orientado por una línea negra.

#define NUM\_SENSORS 8 //numero de sensores usados

#define TIMEOUT 2500 // tiempo de espera para dar resultado en uS

#define EMITTER\_PIN A3 //pin led on

///////////////pines arduino a utilizar/////////////////////

#define led1 A0 // led rojo

#define led2 A1 // led verde

#define mot\_i 0 // -> IN2

#define mot\_d 5 // -> IN3

#define sensores A3

#define boton\_1 2 //pulsador

#define pin\_pwm\_i 10 //-> IN1

#define pin\_pwm\_d 11 // -> IN 4

QTRSensorsRC qtrrc((unsigned char[]) {6, 7, 8, 9, 1, 4, 12, 13}

,NUM\_SENSORS, TIMEOUT, EMITTER\_PIN);

//variables para almacenar valores de sensores y posicion

unsigned int sensorValues[NUM\_SENSORS];

unsigned int position=0;

/// variables para el pid

int derivativo=0, proporcional=0, integral=0; //parametros

int salida\_pwm=0, proporcional\_pasado=0;

//\_\_\_\_\_\_\_AQUI CAMBIEREMOS LOS PARAMETROS DE NUESTRO ROBOT\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

int velocidad=100; //variable para la velocidad, el maximo es 255

float Kp=0.18;

float Kd=2;

float Ki=0.01; //constantes

//variables para el control del sensado

int linea=0; // 0 para lineas negra, 1 para lineas blancas

int flanco\_color=0; // aumenta o disminuye el valor del sensado

int en\_linea=500; //valor en el cual se considerara si el sensor esta en linea o no

int ruido=50; //valor para el cual el valor del sensor es considerado como ruido

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

void setup()

{

delay(800);

pinMode(mot\_i, OUTPUT);//pin de direccion motor izquierdo

pinMode(mot\_d, OUTPUT);//pin de direccion motor derecho

pinMode(led1, OUTPUT); //led1

pinMode(led2, OUTPUT); //led2

pinMode(boton\_1, INPUT); //boton 1 como pull up

for (int i = 0; i < 50; i++) //calibracion durante 2.5 segundos,

{ //para calibrar es necesario colocar los sensores sobre la superficie negra y luego

digitalWrite(led1, HIGH); //sobre la blanca

delay(20);

qtrrc.calibrate(); //funcion para calibrar sensores

digitalWrite(led1, LOW);

delay(20);

}

digitalWrite(led1, LOW); //apagar sensores para indicar fin

//de calibracion

delay(400);

digitalWrite(led2,HIGH); //encender led 2 para indicar la

// espera de pulsacion de boton

while(true)

{

int x=digitalRead(boton\_1); //leemos y guardamos el valor

// del boton en variable x

delay(100);

if(x==0) //si se presiona boton

{

digitalWrite(led2,LOW); //indicamos que se presiono boton

digitalWrite(led1,HIGH); //encendiendo led 1

delay(100);

break; //saltamos hacia el bucle principal

}

}

}

void loop()

{  
                    //pid(0, 120, 0.18, 4, 0.001);

  pid(linea,velocidad,Kp,Ki,Kd); //funcion para algoritmo pid  
                        //(tipo,flanco de comparacion)  
  
                    //frenos\_contorno(0,700);

  frenos\_contorno(linea,700); //funcion para frenado en curvas tipo

                          //0 para lineas negras, tipo 1 para lineas blancas

                  //flanco de comparación va desde 0 hasta 1000 , esto para ver

                 //si esta en negro o blanco

}

////////funciones para el control del robot////

//aqui esta modificado la funcion del pid para que reciba los nuevos parametros para la libreria modificada

void pid(int linea, int velocidad, float Kp, float Ki, float Kd,int flanco\_color, int en\_linea,int ruido)

{

position = qtrrc.readLine(sensorValues, QTR\_EMITTERS\_ON, linea,flanco\_color, en\_linea, ruido ); //0 para linea

//negra, 1 para linea blanca

proporcional = (position) - 3500; // set point es 3500, asi obtenemos el error

integral=integral + proporcional\_pasado; //obteniendo integral

derivativo = (proporcional - proporcional\_pasado); //obteniedo el derivativo

if (integral>1000) integral=1000; //limitamos la integral para no causar problemas

if (integral<-1000) integral=-1000;

salida\_pwm =( proporcional \* Kp ) + ( derivativo \* Kd )+(integral\*Ki);

if ( salida\_pwm > velocidad ) salida\_pwm = velocidad; //limitamos la salida de pwm

if ( salida\_pwm < -velocidad ) salida\_pwm = -velocidad;

if (salida\_pwm < 0)

{

motores(velocidad+salida\_pwm, velocidad);

}

if (salida\_pwm >0)

{

motores(velocidad, velocidad-salida\_pwm);

}

proporcional\_pasado = proporcional;

}

void motores(int motor\_izq, int motor\_der)

{

if ( motor\_izq >= 0 ) //motor izquierdo

{

digitalWrite(mot\_i,HIGH); // con high avanza

analogWrite(pin\_pwm\_i,255-motor\_izq); //se controla de manera

//inversa para mayor control

}

else

{

digitalWrite(mot\_i,LOW); //con low retrocede

motor\_izq = motor\_izq\*(-1); //cambio de signo

analogWrite(pin\_pwm\_i,motor\_izq);

}

if ( motor\_der >= 0 ) //motor derecho

{

digitalWrite(mot\_d,HIGH);

analogWrite(pin\_pwm\_d,255-motor\_der);

}

else

{

digitalWrite(mot\_d,LOW);

motor\_der= motor\_der\*(-1);

analogWrite(pin\_pwm\_d,motor\_der);

}

}

void frenos\_contorno(int tipo,int flanco\_comparacion)

{

if(tipo==0)

{

if (position<=500) //si se salio por la parte derecha de la linea

{

motores(-80,90); //debido a la inercia, el motor

//tendera a seguri girando

//por eso le damos para atras , para que frene

// lo mas rapido posible

while(true)

{

qtrrc.read(sensorValues); //lectura en bruto de sensor

if ( sensorValues[0]>flanco\_comparacion || sensorValues[1]>flanco\_comparacion )

//asegurar que esta en linea

{

break;

}

}

}

if (position>=6500) //si se salio por la parte izquierda de la linea

{

motores(90,-80);

while(true)

{

qtrrc.read(sensorValues);

if (sensorValues[7]>flanco\_comparacion || sensorValues[6]>flanco\_comparacion )

{

break;

}

}

}

}

}

Para la resolución del laberinto se a implementado inicialmente la siguiente codificación, basada en el algoritmo FOOD FILL,

void FloodFill()

{

int i,j; //Definir variables locales para esta función inicializa\_cola();

//Inicializar valores de las celdas

for(i=0;i<Orden\_L;j++)

{ celda[i][j].valor=255;

celda[i][j].calculado=false; celda[i][j].en\_cola=false;

}

//Comienza la inundación dando el valor de cero a la solución

if(!regreso)

if(Orden\_L%2==1) //En caso de que el orden del laberinto sea impar (una celda de solución)

{

celda[SolR][SolC].valor=0; celda[SolR][SolC].calculado=true; celda[SolR][SolC].en\_cola=true; almacena(SolR,SolC);

}

else //En caso de que el orden del laberinto sea par (4 celdas de solución)

{

celda[SolR][SolC].valor=0; celda[SolR][SolC].calculado=true; celda[SolR][SolC].en\_cola=true; almacena(SolR,SolC);

celda[SolR][SolC2].valor=0; celda[SolR][SolC2].calculado=true; celda[SolR][SolC2].en\_cola=true; almacena(SolR,SolC2);

celda[SolR2][SolC].valor=0; celda[SolR2][SolC].calculado=true; celda[SolR2][SolC].en\_cola=true; almacena(SolR2,SolC);

celda[SolR2][SolC2].valor=0; celda[SolR2][SolC2].calculado=true; celda[SolR2][SolC2].en\_cola=true; almacena(SolR2,SolC2);

}

else

{

celda[Ren0][Col0].valor=0; celda[Ren0][Col0].calculado=true; celda[Ren0][Col0].en\_cola=true; almacena(Ren0,Col0);

}

//Inunda las celdas adyacentes while(recupera()!=NULL)

{

Indice[0]=cola[recover-1][0]; //Renglón Indice[1]=cola[recover-1][1]; //Columna CalculaAdyacentes(Indice[0],Indice[1],celda[Indice[0]][Indice[1]].muros);

}

return;

}