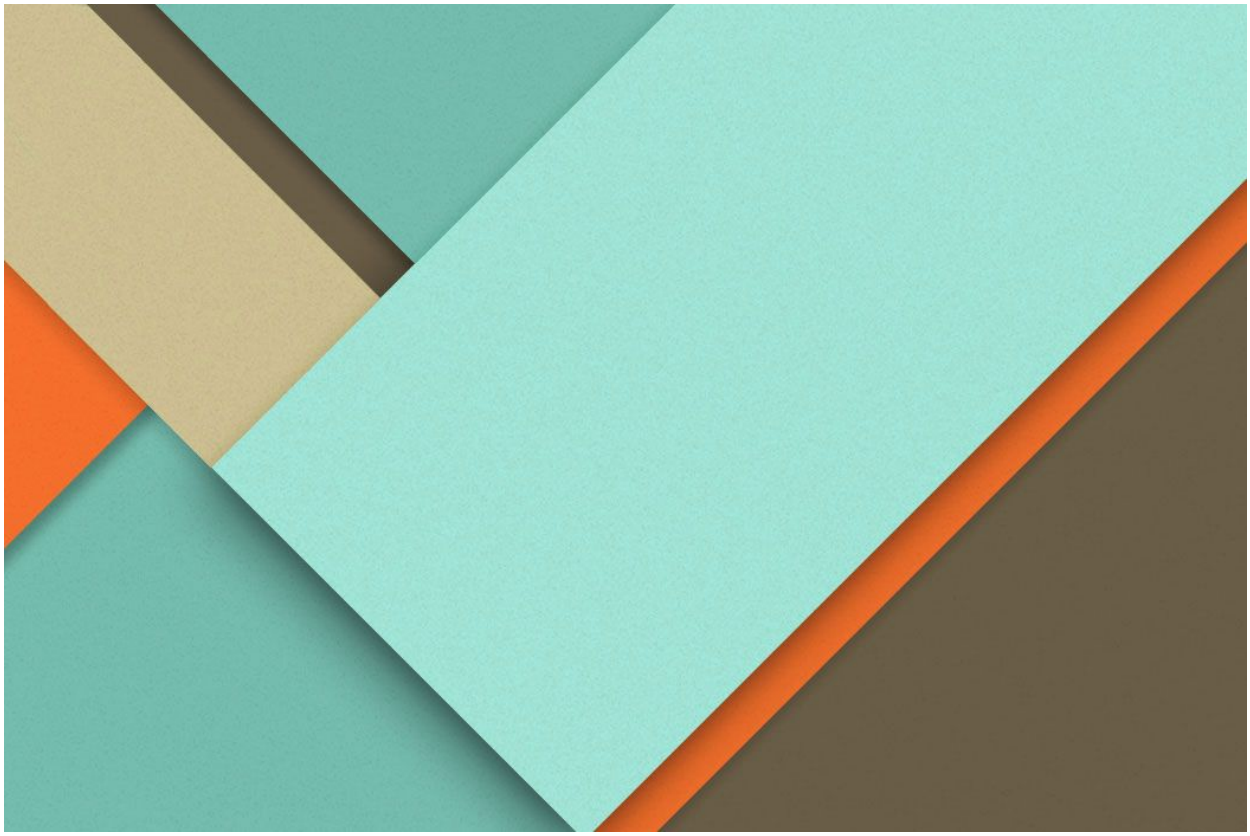


Memoria DCE

2019

Aarón Arias Pérez

Francisco Alejandro Naranjo Castañeda

Carlos Guzmán Cabrera

Contenido

1. Introducción

1.1 Objetivo

Robot móvil capaz de recorrer un laberinto de 5x5 celdas tratando de encontrar la salida. Las acciones llevadas a cabo por el robot para conseguir su objetivo son:

- Movimientos

  El robot puede moverse hacia delante, girar a la derecha, girar a la izquierda y girar hacia atrás.

- Detección

  El robot es capaz de detectar las paredes del laberinto mediante un sensor ultrasonido, el cual es utilizado junto a un servo. De esta forma, el robot es capaz de utilizar un solo sensor de ultrasonidos para detectar obstáculos tanto delante, como a ambos lados del robot.

  También dispone de 3 sensores CY70 que están ubicados en la parte inferior del robot, los cuales son utilizados para detectar las líneas de separación entre las distintas casillas del laberinto.

- Recogida de información

  El robot recoge información mediante los sensores mencionados en el apartado anterior, y es procesada por el propio robot, de cara a la resolución del laberinto. Esta información, también es enviada mediante bluetooth a un ordenador, el cual mediante una aplicación desarrollada en Python, visualiza información del robot y de la trayectoria que está realizando en el laberinto.

- Algoritmo de resolución del laberinto

  El algoritmo de resolución del laberinto se basa en seguir siempre la pared derecha, ya que este laberinto no contiene islas, por lo tanto si se sigue siempre la pared por la derecha o por la izquierda, siempre acabará encontrando la salida.

- Monitorización de información

  La monitorización de la información se realiza en una aplicación desarrollada con Python, con los datos obtenidos mediante la conexión bluetooth con el robot.

En dicha aplicación, se muestran el porcentaje de batería, los centímetros recorridos, el número de casillas recorridas, la velocidad actual y el tiempo que ha transcurrido desde que comenzó a resolver el laberinto, en forma de texto.

De manera gráfica, la aplicación muestra el recorrido del robot en tiempo real (se va dibujando el laberinto casilla a casilla recorrida, para que no dependa de que el laberinto sea de determinado tamaño).

1.2 Hardware empleado

a. Arduino Leonardo

Para ello, se utilizará una placa basado en microcontrolador, Arduino Leonardo, y un ordenador personal.

La placa Arduino Leonardo usada en el robot, se encuentra conectada a una placa que posee todas las conexiones y salidas preparadas para la conexión de diversos actuadores y sensores, así como la batería.

b. Sensores

Sensor ultrasonidos HC-SR04: Usado para la detección de obstáculos.

Sensor óptico reflexivo CNY70: Usado para la detección de las casillas de juego y la casilla de meta.

c. Actuadores

Dos motores DCM-0003: Usados para mover el robot.

Controlador de motor SN754410NE: mediante los cuales podremos manipular los motores.

Servo SG90: Utilizado para desplazar el sensor de ultrasonidos para poder hacer comprobaciones en diferentes puntos.

d. Elementos de comunicación

Dispositivo bluetooth HC06: Mediante este dispositivo, transmitiremos la información del robot al ordenador personal para poder procesarla mediante la aplicación Python.

e. Alimentación

La alimentación del robot es proporcionada mediante una batería, formada por 6 pilas AA, 1.5 V. Suministra la energía para que el robot pueda funcionar.

1.3 Software empleado

a. Arduino IDE

Ha sido utilizado para la carga del programa en la placa Arduino Leonardo, y para el uso de funciones básica de Arduino.

b. Python 3

Utilizando Python, se ha desarrollado una aplicación capaz de obtener los datos mediante una conexión bluetooth con el robot, para procesarla y mostrar en una interfaz gráfica, el porcentaje de batería, los centímetros recorridos, el número de casillas recorridas y el tiempo que ha transcurrido desde que comenzó a resolver el laberinto, en forma de texto así como el recorrido del robot en tiempo real.

2. Especificación de requisitos

2.1 Requisitos funcionales

ID	CATEGORÍA	DESCRIPCIÓN
RF01	Movimiento	El robot debe ser capaz de moverse
RF02	Movimiento	El robot debe ser capaz de pivotar 90º a la derecha
RF03	Movimiento	El robot debe ser capaz de pivotar 90º a la izquierda
RF04	Movimiento	El robot debe ser capaz de avanzar en línea recta
RF05	Movimiento	El robot avanza adecuadamente hasta la siguiente celda
RF06	Detección	El robot debe ser capaz de detectar una pared frontal
RF07	Detección	El robot debe ser capaz de detectar una pared lateral derecha
RF08	Detección	El robot debe ser capaz de detectar una pared lateral izquierda
RF09	Detección	El robot debe ser capaz de detectar la transición entre celdas

RF10	Detección	El robot debe ser capaz de detectar la celda de salida
RF11	Resolución	El robot almacena información sobre las celdas del laberinto
RF12	Resolución	El robot debe ser capaz de decidir el siguiente movimiento en base a la información sobre la celda
RF13	Resolución	El robot es capaz de recorrer varias celdas del laberinto siguiendo el algoritmo empleado
RF14	Resolución	El robot es capaz de salir del laberinto
RF15	Información	El robot envía al PC información sobre el número de celdas recorridas
RF16	Información	El robot envía al PC información sobre obstáculos en cada celda
RF17	Información	El robot envía al PC información sobre la velocidad de movimiento
RF18	Información	El robot envía al PC información sobre la distancia que lleva
RF19	Información	El robot envía al PC información sobre el tiempo transcurrido desde la entrada al laberinto
RF20	Información	El robot envía al PC información sobre la trayectoria ejecutada
RF21	Información	El robot envía al PC información sobre el número de celdas recorridas
RF22	Información	El PC muestra una representación gráfica del laberinto
RF23	Usuario	Interfaz gráfica en PC que recopile información
RF24	Pruebas	Incluir modo test al arranque del robot

2.2 Requisitos no funcionales

ID	CATEGORÍA	DESCRIPCIÓN	ACCIÓN
RFN01	Tamaño	Condicionado por las dimensiones del laberinto. 11 x 10.5 x 16.5	
RFN02	Consumo	Condicionado por la batería disponible, 6 pilas de 1.5V, 9V en total	
RFN03	Errores	El robot choca contra una pared	Se le corrige la posición manualmente
RFN04	Errores	Batería a punto de agotarse	Nivel de batería mostrada en interfaz. Efectuar intercambio al ser necesario
RFN05	Errores	El robot gira continuamente	Se corrige manualmente, haciendo que detecte las líneas negras para detener el giro.
RFN06	Errores	El robot sobrepasa 20 minutos sin conseguir salir	No se ha contemplado esta situación.

3. Planificación

4. Presupuesto

Componente:	Cantidad:	Precio (€):
Arduino Leonardo	1	19,90
Batería (pilas)	6	4,00
CNY_70	3	3,51
HC-SR04	1	3,84
SG90	1	2,00
SN754410NE	1	2,20
DCM-0003	2	26,40
Otros	-	~15

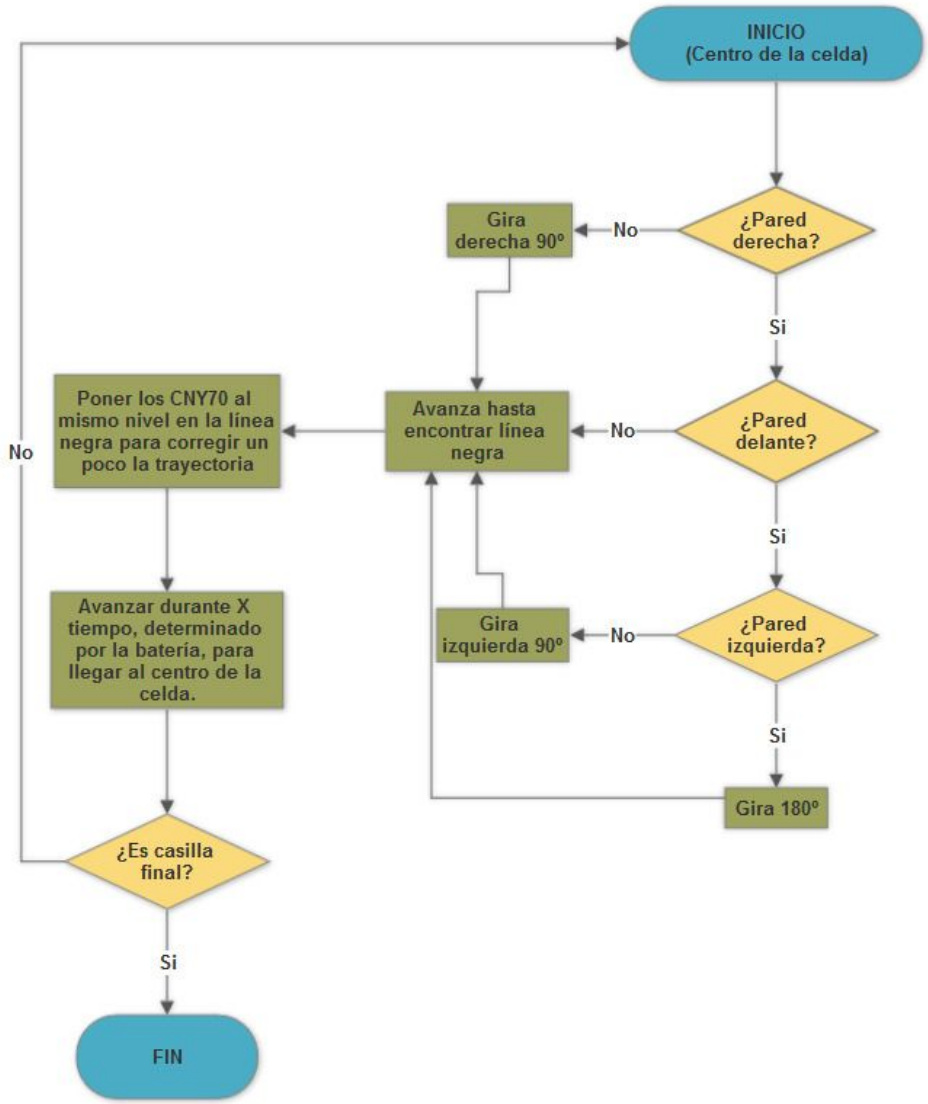
Precio total:	76,85 €

5. Análisis

5.1 Casos de uso

El usuario no tiene capacidad de interactuar con la aplicación, ésta sólo muestra los datos que recibe por el bluetooth.

5.2 tDiagrama de flujo



6. Diseño

6.1 Estructura

```
#include "Robot.h"
#include <stdlib.h>
#include <string.h>

/*
pinMode(9,OUTPUT); //MOTOR IZQ
pinMode(10,OUTPUT); //MOTOR IZQ
pinMode(5,OUTPUT); //MOTOR DRC
pinMode(6,OUTPUT); //MOTOR DRC
*/

Robot::Robot(int a1,int a2, int b1, int b2, int bateria, int arriba, int abajoizq, int abajodrc){
        A1pin = a1;
        A2pin = a2;
        B1pin = b1;
        B2pin = b2;
        pinBateria = bateria;
        cnyArriba = arriba;
        cnyAbajoIzq = abajoizq;
        cnyAbajoDrc = abajodrc;



        Serial.begin(9600);
        Serial1.begin(9600); //bluetooth

        pinMode(A1pin, OUTPUT);
        pinMode(A2pin, OUTPUT);
        pinMode(B1pin, OUTPUT);
        pinMode(B2pin, OUTPUT);
```

```
        pinMode(11, OUTPUT);

}

void Robot::move_forward(){
        digitalWrite(A1pin,LOW);
digitalWrite(A2pin,HIGH);
digitalWrite(B1pin,LOW);
digitalWrite(B2pin,HIGH);

}
void Robot::move_backward(){
        digitalWrite(A1pin,HIGH);
digitalWrite(A2pin,LOW);
digitalWrite(B1pin,HIGH);
digitalWrite(B2pin,LOW);
        }
void Robot::stop(){
        digitalWrite(A1pin,LOW);
digitalWrite(A2pin,LOW);
        digitalWrite(B1pin,LOW);
digitalWrite(B2pin,LOW);
}
void Robot::move_right(){
        digitalWrite(A1pin,HIGH);
digitalWrite(A2pin,LOW);

        digitalWrite(B1pin,LOW);
digitalWrite(B2pin,HIGH);
        }
void Robot::move_left(){
        digitalWrite(A1pin,LOW);
digitalWrite(A2pin,HIGH);
```

```
        digitalWrite(B1pin,HIGH);
digitalWrite(B2pin,LOW);
}

void Robot::move_right_motor(){
        digitalWrite(A1pin,LOW);
digitalWrite(A2pin,HIGH);

        digitalWrite(B1pin,LOW);
digitalWrite(B2pin,LOW);
}

void Robot::move_left_motor(){
        digitalWrite(A1pin,LOW);
digitalWrite(A2pin,LOW);

        digitalWrite(B1pin,LOW);
digitalWrite(B2pin,HIGH);
}

void Robot::move_servo(int ang){
digitalWrite(11,LOW); //ponemos en intensidad baja
digitalWrite(11,HIGH); //comenzamos el pulso
delayMicroseconds(map(ang,0,180,544,2400));
digitalWrite(11,LOW);
delay(20);
}

float Robot::read_battery(){
        int val_tension = analogRead(pinBateria);
        return ((val_tension*0.00488)/4.15)*100;
}

/*
```

```
                Elegir cny70:
                arriba = 0
                abajo_izq = 1
                abajo_drc = 2
                si es blanco devuelve true
                si es negro devuelve false
*/
bool Robot::read_cny70(int opt){
        if(opt==0){
                float valor = analogRead(cnyArriba)*0.00488;
                if(valor <= 2.5){ //umbral: blanco o negro?
                        return true;
                }else if(valor > 2.5){
                        return false;
                }
        }else if(opt==1){
                float valor = analogRead(cnyAbajoIzq)*0.00488;
                if(valor <= 3.5){ //umbral: blanco o negro?
                        return true;
                }else if(valor > 3.5){
                        return false;
                }
        }else if(opt==2){
                float valor = analogRead(cnyAbajoDrc)*0.00488;
                if(valor <= 3){ //umbral
                        return true;
                }else if(valor > 3){
                        return false;
                }
        }else{
                Serial.println("ERROR: No se ha elegido el cny70 correcto");
        }
}
```

```
float get_bateria(int pinBateria){
	return analogRead(pinBateria)*0.00488;
}

/*
	Ecuacion de la recta para la velocidad segun la tension
	y = 5.9*x - 5.6
	y => velocidad en cm/seg
	x => tension leida de la bateria (max 4.15)
*/
float get_tiempo(float distance, int pinBateria){
	float cm_por_segundo = 5.9*get_bateria(pinBateria) - 5.6;

	return distance / cm_por_segundo;
}

void Robot::alinear_robot(){
	bool alineado = false;
	bool drc_alineado = false;
	bool izq_alineado = false;
	bool linea_encontrada = false;
	while(!alineado){
		bool es_blanco_izq = read_cny70(1);
		bool es_blanco_drc = read_cny70(2);

		if(!linea_encontrada) {
			if(es_blanco_izq && es_blanco_drc) move_forward();
			else linea_encontrada = true;

		}else{
			if(!drc_alineado){

				while(es_blanco_drc){
					es_blanco_drc = read_cny70(2);
```

```
                                        move_right();
                                }
                                drc_alineado = true;

                        }

                        if(!izq_alineado){

                                while(es_blanco_izq){
                                        es_blanco_izq = read_cny70(1);
                                        move_left();
                                }
                                izq_alineado = true;
                        }
                }

                if(izq_alineado && drc_alineado) alineado = true;
        }
}

/*
0 -> no pared derecha
1 -> no pared frente
2 -> no pared izquierda
3 -> rodeado de paredes
*/
int Robot::check_wall(){
        int dir = -1;

        delay(500);
        move_servo(180);
        float d = read_ultrasound();
        if(d < 30){
                delay(500);
```

```
                move_servo(90);
                d = read_ultrasound();
                if(d < 30){
                        delay(500);
                        move_servo(0);
                        d = read_ultrasound();
                        if(d < 30){
                                dir = 3;
                        }else{
                                dir = 2;
                        }
                }else{
                        dir = 1;
                }
        }else{
                dir = 0;
        }

        return dir;
}

void Robot::change_direction(int dir){
        if(dir==0){
                float time = get_tiempo(7.2, pinBateria);
                move_right();
                delay(time*1000);
                send_bluetooth("g0");
        }else if(dir==1){
                //SIGUE RECTO
        }else if(dir==2){
                float time = get_tiempo(7.2, pinBateria);
                move_left();
                delay(time*1000);
                send_bluetooth("g1");
```

```
        }else if(dir==3){
                float time = get_tiempo(14, pinBateria);
                move_right();
                delay(time*1000);
                send_bluetooth("g2");
        }
}

bool Robot::check_final_cell(){
        bool es_blanco_izq = read_cny70(1);
        bool es_blanco_drc = read_cny70(2);
        bool es_blanco_abajo = read_cny70(0);

        if(!es_blanco_izq && !es_blanco_abajo && !es_blanco_drc) return true; else return false;
}

float Robot::read_ultrasound(){
pinMode(A3,OUTPUT);
digitalWrite(A3,LOW);
delayMicroseconds(5);

digitalWrite(A3,HIGH);
delayMicroseconds(10);
digitalWrite(A3,LOW);

pinMode(A3, INPUT);
unsigned long time_bounce = pulseIn(A3,HIGH);

float distance = 0.017*time_bounce;

return distance;
//Serial.print("Distancia: ");
//Serial.print(distance);
//Serial.println("cm");
```

```
//delay(1000);
}

void Robot::send_bluetooth(char* str){
                        Serial1.write(str);
}

void Robot::send_battery(){
            float aux = read_battery();
            char res[10];
            dtostrf(aux,4,2, res);
            send_bluetooth("b");
send_bluetooth(res);
send_bluetooth("\n");
}

void Robot::send_velocity(){
            float velocity = 5.9*get_bateria(pinBateria) - 5.6;
            char res[10];
            dtostrf(velocity,4,2, res);
            send_bluetooth("m");
send_bluetooth(res);
send_bluetooth("\n");
}

bool Robot::algorithm(){
            // *** Empieza en el centro de la celda ***
            send_battery(); //estado de la bateria

            //checkear paredes
            int direccion = check_wall();

            //cambiar direccion
```

```
        change_direction(direccion);

        //moverse hasta alinearse con la linea negra
        alinear_robot();
                //moverse hasta el centro
        float time = get_tiempo(9, pinBateria);
        send_velocity();
        move_forward();
        delay(time*1000);
        stop();
        send_bluetooth("m0\n"); //SE PUEDE QUITAR Y USAR SOLO EL SEND(C)
        send_bluetooth("c\n");

        //checkear si se ha llegado al final
        bool algoritmo_terminado = check_final_cell();

        return algoritmo_terminado;

}
```

6.2 Plan de pruebas

-Comprobación del estado de la batería:

```
void imprimirBateria(){
val_tension = analogRead(A6);
Serial.print(val_tension*0.00488); Serial.println(" V");
}
```

-Prueba de los leds integrados:

```
void encenderLeds( ){
digitalWrite(11,LOW); //AZULITO
digitalWrite(12,LOW); //ROJO
digitalWrite(13,LOW); //VERDE
```

```
}
```

-Prueba de los motores:

```
void moverMotor(){
digitalWrite(9,LOW);
digitalWrite(10,HIGH);
delay(500);
digitalWrite(5,HIGH);
digitalWrite(6,LOW);
delay(500);
}
```

-Prueba sensores CNY70

```
void leerCNY(int pin){
//A2 NO TIENE CNY70
//A5 ES CNY DE DELANTE
//A0 ES CNY DE ATRAS DRCH
//A1 ES CNY DE ATRAS IZQ
//valor alto => negro
int cnydelante = analogRead(pin);
Serial.print("Valor cny: ");Serial.println(cnydelante);
}
```

-Prueba sensor ultrasonidos:

```
void leerUltrasonido(){
pinMode(A3,OUTPUT);
digitalWrite(A3,LOW);
delayMicroseconds(5);

digitalWrite(A3,HIGH);
delayMicroseconds(10);
```

```
digitalWrite(A3,LOW);


pinMode(A3, INPUT);
unsigned long time_bounce = pulseIn(A3,HIGH);


float distance = 0.017*time_bounce;
Serial.print("Distancia: ");
Serial.print(distance);
Serial.println("cm");


delay(1000);
}
```

-Pruebas servo:

```
void moverServo(){
for(int i=0;i<180;i++){
digitalWrite(11,LOW);
digitalWrite(11,HIGH);
delayMicroseconds(map(i,0,180,1000,2000));
digitalWrite(11,LOW);
delay(20);
}
}
```

-Pruebas sensores SHARP (no usados en nuestro robot):

```
void leerSHARP(int pin){
int lectura = analogRead(pin);
Serial.print("Valor SHARP: ");Serial.println(lectura);
}
```

7. Implementación

7.1 Bibliotecas

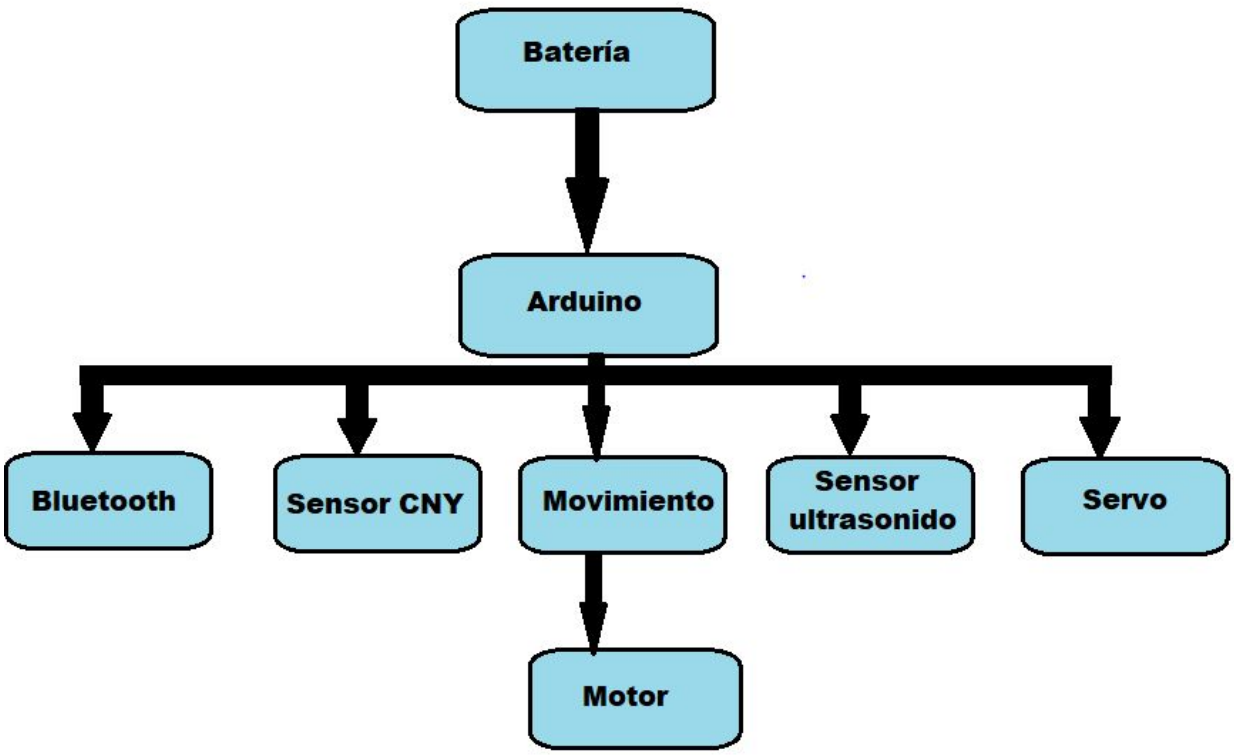
No se ha incluido ninguna biblioteca externa, solo hemos utilizado la que viene en arduino.

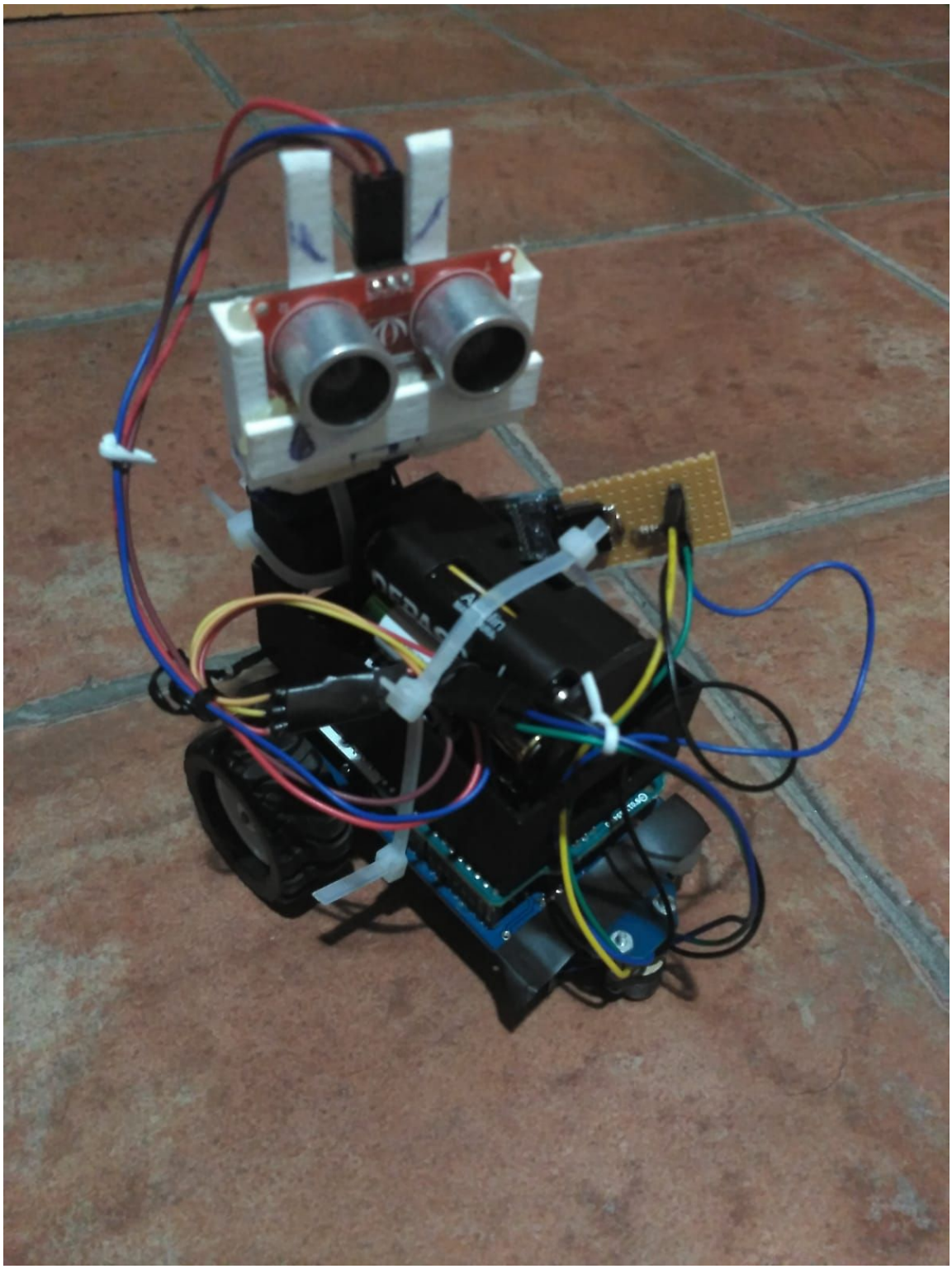
La biblioteca que se ha construido es “Robot.h” con su respectivas definiciones en “Robot.cpp”.

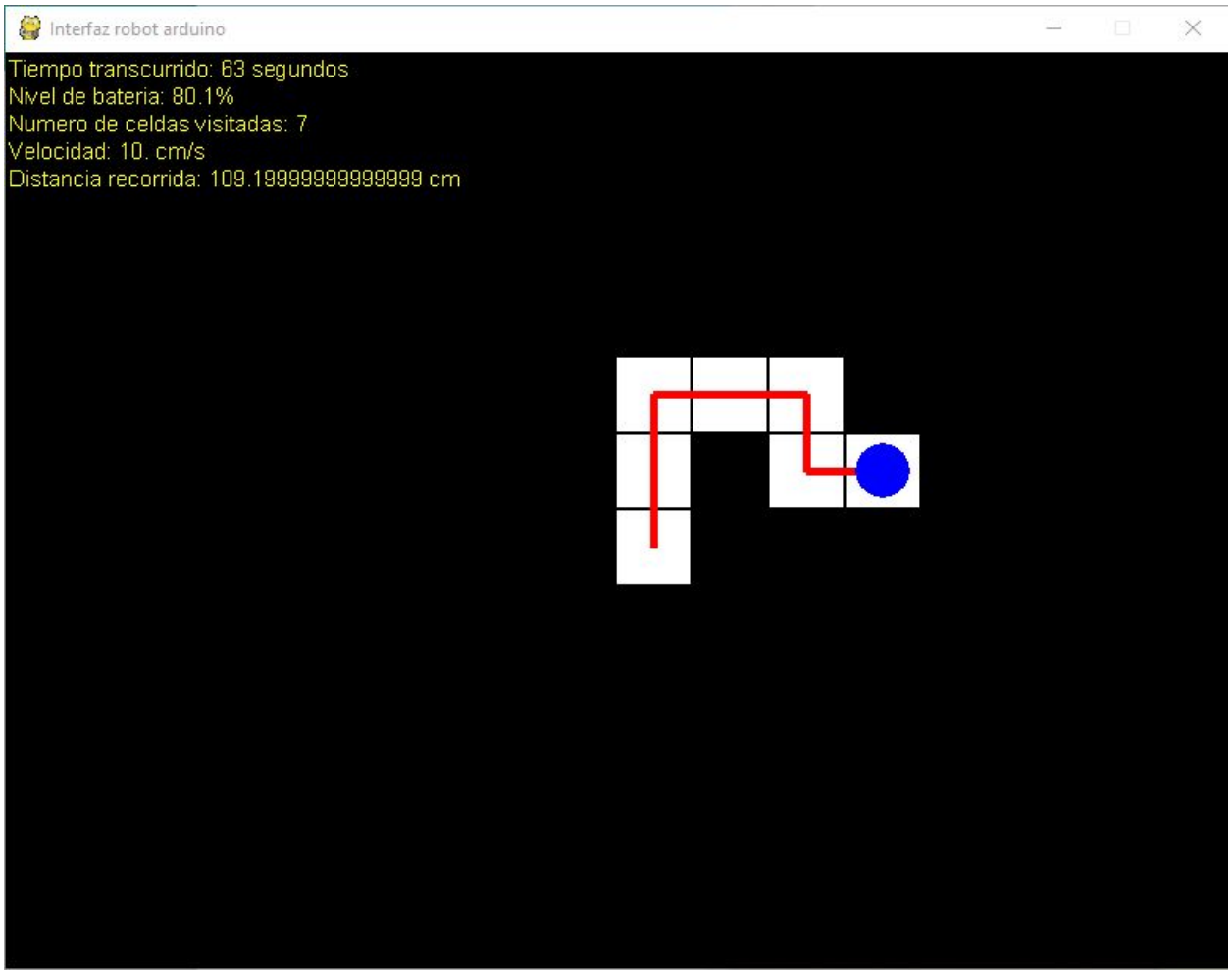
7.2 Apuntes sobre el código

Teniendo como idea el algoritmo que queríamos implementar para resolver el laberinto, simplemente lo he traducido a código, construyendo una interfaz con la que poder utilizar cómodamente los distintos sensores y actuadores del robot.

8. Montaje







9. Pruebas

Coloreados de verde aquellos que se cumplen

ID	CATEGORÍA	DESCRIPCIÓN
RF01	Movimiento	El robot debe ser capaz de moverse
RF02	Movimiento	El robot debe ser capaz de pivotar 90º a la derecha
RF03	Movimiento	El robot debe ser capaz de pivotar 90º a la izquierda
RF04	Movimiento	El robot debe ser capaz de avanzar en línea recta
RF05	Movimiento	El robot avanza adecuadamente hasta la siguiente celda

RF06	Detección	El robot debe ser capaz de detectar una pared frontal
RF07	Detección	El robot debe ser capaz de detectar una pared lateral derecha
RF08	Detección	El robot debe ser capaz de detectar una pared lateral izquierda
RF09	Detección	El robot debe ser capaz de detectar la transición entre celdas
RF10	Detección	El robot debe ser capaz de detectar la celda de salida
RF11	Resolución	El robot almacena información sobre las celdas del laberinto
RF12	Resolución	El robot debe ser capaz de decidir el siguiente movimiento en base a la información sobre la celda
RF13	Resolución	El robot es capaz de recorrer varias celdas del laberinto siguiendo el algoritmo empleado
RF14	Resolución	El robot es capaz de salir del laberinto
RF15	Información	El robot envía al PC información sobre el número de celdas recorridas
RF16	Información	El robot envía al PC información sobre obstáculos en cada celda
RF17	Información	El robot envía al PC información sobre la velocidad de movimiento
RF18	Información	El robot envía al PC información sobre la distancia que lleva
RF19	Información	El robot envía al PC información sobre el tiempo transcurrido desde la entrada al laberinto
RF20	Información	El robot envía al PC información sobre la trayectoria ejecutada
RF21	Información	El robot envía al PC información sobre el número de celdas recorridas
RF22	Información	El PC muestra una representación gráfica del laberinto
RF23	Usuario	Interfaz gráfica en PC que recopile información
RF24	Pruebas	Incluir modo test al arranque del robot

10. Mejoras

El robot sería capaz de detectar cualquier obstáculo en un rango de 180º sin tener que moverse, gracias a que hemos colocado el ultrasonido en un servomotor.