Project SIXCFJ (Robot Móvil)
`https://github.com/ferqui/SIXCFJ`

Juan Francisco Cabrera Sánchez
Carlos Gallardo Polanco
Fernando Manuel Quintana Velázquez

7 de febrero de 2018

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

INTRODUCCIÓN

1.1. Objetivo

El objetivo de esta práctica es la resolución de un laberinto de 5x5 casillas, en el que la posición de las paredes, se desconoce a priori y en el que las casillas tienen un tamaño de 20x20 cm. Además de resolver el laberinto, es decir, llegar a una casilla de salida cuya posición se desconoce. Una vez que se haya encontrado la casilla de salida, volver a la casilla de entrada por el camino más corto posible.

- Movimientos: Desplazamiento hacia delante, hacia atrás, giro a la izquierda y giro a la derecha
- Detección: Detección de paredes mediante el uso de sensores Sharp y ultrasonidos, y detección de color mediante el uso de sensores CNY70.
- Recogida de información: Se recoge información desde los sensores Sharp, CNY70 y el ultrasonidos.
- Algoritmo de resolución del laberinto: Se tiene una jerarquía de movimientos, estando en primer lugar, moverse a la derecha, luego al frente y por último a la izquierda. Todos estos movimientos se van guardando en una pila para que según se vaya avanzando en el laberinto en la pila sólo queden los movimientos que permiten alcanzar la casilla final, para que haciendo los movimientos inversos a los que están en la pila, el robot pueda volver a llegar a la casilla inicial por el camino más corto.
- Monitorización de información: La monitorización de la información se realiza a través de la aplicación móvil, que muestra la posición del robot en el laberinto junto con el nivel de batería restante, entre otras cosas.

1.2. Hardware empleado

En el diseño final se han utilizado los siguientes componentes:

- CNY70: Sensor que mide la intensidad de la luz, permitiendo distinguir colores.
- Ultrasonidos: Sensor que permite calcular distancias mediante la producción de sonidos y su rebote.
- Sharp: Sensor que permite calcular las distancias mediante el uso de un haz de luz.
- HC-06 (Bluetooth): Módulo que permite leer y escribir datos mediante el puerto serie.
- Motores DC: Elemento mecánico que permite el movimiento.
- LED: Elemento electrónico que se ilumina con el paso de la corriente.
- Batería: Proporciona una fuente de alimentación externa.

1.2.1. Arduino Leonardo

Para ello, se utilizará una placa Arduino Leonardo, un ordenador personal y un móvil. A través del ordenador, se le descargará en la placa el programa para resolver el laberinto, una vez realizada la descarga del programa, se enlaza mediante Bluetooth la placa y un dispositivo móvil, donde se podrá ver la posición del robot en el laberinto y otros datos interesantes como el porcentaje de batería restante.

1.2.2. Sensores

- CNY70: Su uso es necesario para comprobar el paso de una casilla a otra así como para comprobar si se ha llegado a la casilla de salida o no.
- Sharp: Se usa para detectar la existencia de paredes laterales.
- Ultrasonidos: Se usa para detectar la existencia de paredes frontales.

1.2.3. Actuadores

- Interruptor: El interruptor es necesario para encender la placa.

1.2.4. Elementos de comunicación

- HC-06: El conector Bluetooth se usa para mandar información a la app móvil para que en ésta se vean el nivel batería y la posición en el laberinto.

1.2.5. Alimentación

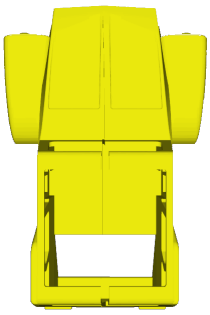
- Batería de 6 pilas AA: Sirve para alimentar la placa.

1.2.6. Justificación

Se consideran que estos elementos son los mínimos imprescindibles para una correcta resolución del laberinto, ya que, desde un primer momento, el hecho de buscar un diseño similar al de un coche, teníamos claro que el ultrasonidos iría colocado en el lugar de los faros delanteros y por lo tanto, el uso de los servos resulta totalmente prescindible. Por otra parte, tampoco se usa el conector Bluetooth para el PC por estar utilizando una aplicación móvil.

1.2.7. Resumen de Pines

PIN	ELEMENTO
A5	CNY Trasero
A0	CNY Izquierdo
A1	CNY Derecho
A8	Sharp Izquierdo
A4	Sharp Derecho
RX, TX	Bluetooth
A3	Ultrasonidos
12	LED1
13	LED2
11	LED3
A6	% Batería



Cuadro 1.1: Resumen de los elementos y sus respectivos pines

1.3. Software empleado

1.3.1. Arduino IDE

Resulta imprescindible el uso de este software por motivos obvios, gracias a él se le mandan los programas a la placa de Arduino. En nuestro caso, era también necesario para importar la librería externa que contiene la implementación de las estructuras de datos de la STL de C++ en Arduino, aunque ésto se detallará más adelante.

1.3.2. React-Native

React-Native es una librería de Javascript que permite la creación de aplicaciones que tengan una apariencia nativa en el correspondiente sistema operativo móvil, en este caso en Android y iOS, usando un mismo código. ¿Por qué usar un móvil y no un PC? La respuesta viene dada con dos componentes, el primero la portabilidad y el segundo, la potencia.

Comenzando por la portabilidad, es evidente que es muchísimo más fácil transportar un móvil que un ordenador portátil, de esta forma, ante la posibilidad de tener que hacer presentaciones, sólo haría falta el robot y el móvil, ya que la placa de Arduino poseer una memoria flash, llevaría cargado el programa previamente. En cuanto a la potencia, la realidad es que en los últimos años la tecnología móvil ha avanzado lo suficiente como para igualar en ciertos aspectos al ordenador portátil, por tanto, si tuviesemos que mandar a realizar unos cálculos ya sea al ordenador o al móvil, como por ejemplo, calcular el camino de vuelta más corto usando el algoritmo de Dijkstra no habría tanta diferencia entre que lo haga un móvil o un ordenador.

Y, por otro lado, está el factor didáctico. *React-Native* es una librería con un gran empuje en los últimos años y, de cara a aumentar nuestra formación en otras tecnologías, resultaba bastante interesante aprender lo más actual en el mundo empresarial.

1.3.3. LaTeX

El hecho de utilizar LaTeX era obligatorio en nuestra opinión para dotar a esta memoria de un aspecto académico, gracias en gran medida al paquete TikZ que permite la creación de gráficos vectoriales en LaTeX con una gran versatilidad. Este paquete permite, por ejemplo, la creación de diagramas de Gantt, de diagramas de casos de uso o de diagramas de flujo.

Capítulo 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 90^{o} a derecha
RF03	Movimiento	El robot debe ser capaz de pivotar 90^{o} a 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 es 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 recorrida
RF19	Información	El robot envía al PC información sobre el tiempo transcurrido desde la entrada al laberinto
RF20	Información	El robot envía al PC 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

Cuadro 2.1: Requisitos Funcionales

2.2. Requisitos no funcionales

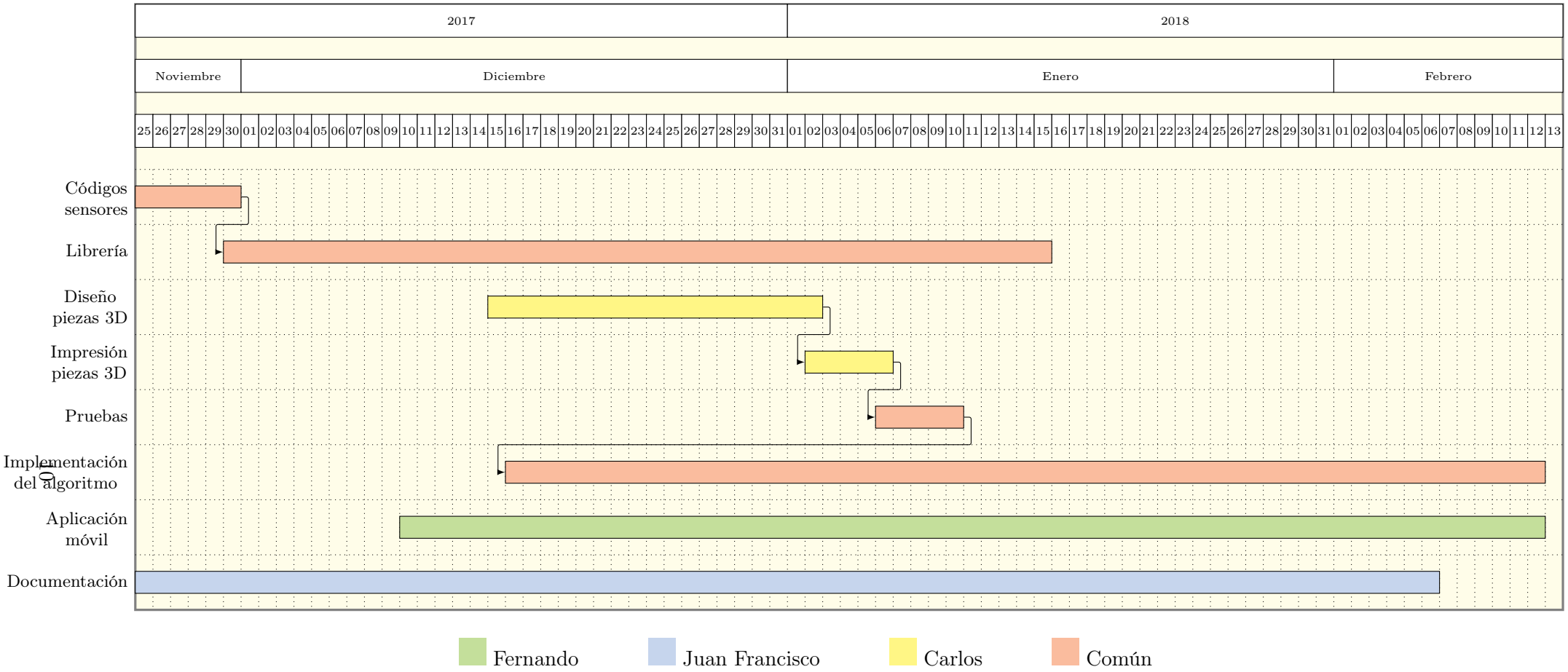
ID	CATEGORÍA	DESCRIPCIÓN	ACCIÓN
RNF01	Tamaño	Condicionado por las dimensiones del laberinto <Casilla de 20cm x 20cm; Laberinto de 1m x 1m>	
RNF02	Consumo	Condicionado por la batería disponible <6 pilas AA >	
RNF03	Errores	El robot choca contra una pared	-
RNF04	Errores	Batería a punto de agotarse	Sustituir batería
RNF05	Errores	El robot gira continuamente	No se produce
RNF06	Errores	El robot sobrepasa 20 minutos sin conseguir salir	-

Cuadro 2.2: Requisitos no Funcionales

Capítulo 3

PLANIFICACIÓN

En la primera reunión se decidió que todos colaboraríamos en la codificación del algoritmo del robot y que, además, cada uno de los integrantes del grupo, además de esta tarea, abarcaría otra tarea individual. Esto queda claro en el sistema de colores empleado en diagrama de Gantt que podrá ver en la siguiente página.


2017
2018
Noviembre
Diciembre
Enero
Febrero
25 26 27 28 29 30 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 01 02 03 04 05 06 07 08 09 10 11 12 13
Códigos sensores
Librería
Diseño piezas 3D
Impresión piezas 3D
Pruebas
Implementación del algoritmo
Aplicación móvil
Documentación
Fernando
Juan Francisco
Carlos
Común

Capítulo 4

PRESUPUESTO

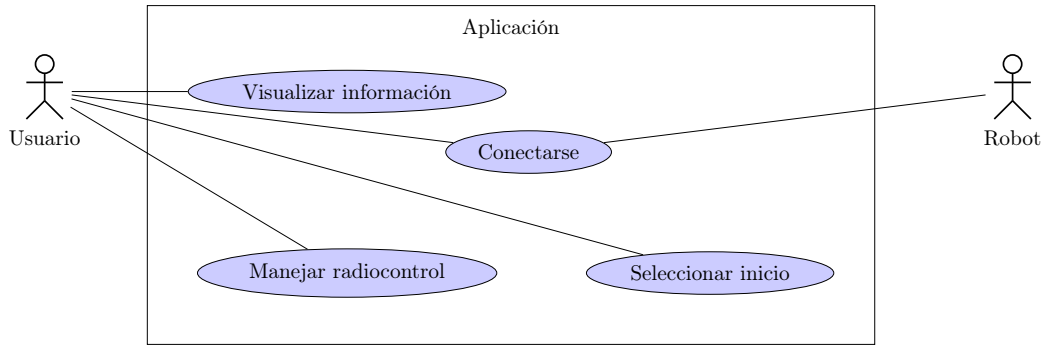
Producto	Nº de unidades	Precio/Unidad	Precio Total
Arduino Leonardo	1	18€	18€
Shield	1	30€	30€
CNY70	3	1.15€	3.45€
HC-06	1	3.50€	3.50€
Sharp	2	3.30€	6.60€
Ultrasonidos	1	1€	1€
Batería	1	1.65€	1.65€
Bovina PLA	150g	150g * 0.02€	3€
Rendimiento de la impresora 3D	0.150kW/h	0.150kW/h* 24h * 0.145€kW/h	0.52€
Mano de Obra	100h	17€	1700€
			Total: 1767.72€

Cuadro 4.1: Presupuesto del Robot Móvil

Capítulo 5

ANÁLISIS

5.1. Casos de uso



5.2. Diagrama de flujo

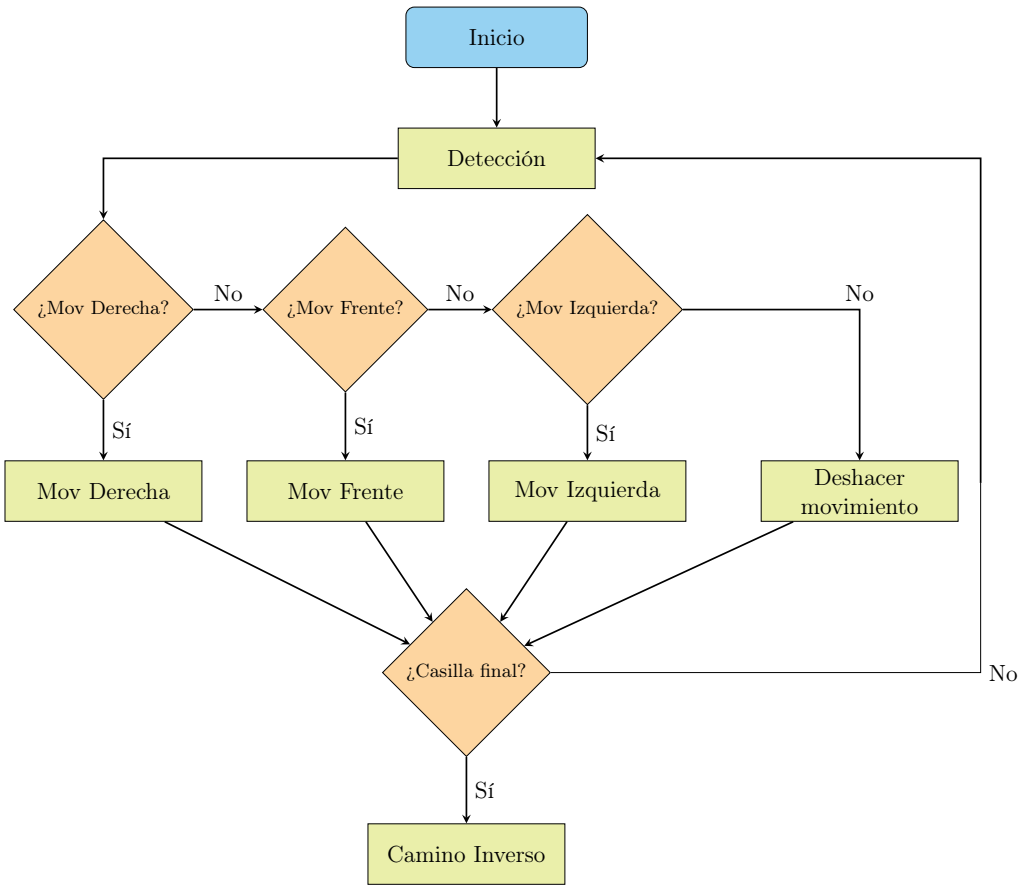


Figura 5.1: Diagrama de flujo

Capítulo 6

DISEÑO

6.1. Estructura

Seguidamente puede ver el código principal para la resolucón del laberinto:

```
#include <StandardCplusplus.h>
#include <stack>
#include "Timer.h"
#include "Robot.hpp"

const int RIGHT = 1;
const int FORWARD = 2;
const int LEFT = 4;
const int BACKWARD = 8;
const int LEFT_BACKWARD = 16;
const int RIGHT_BACKWARD = 32;

Timer t;

int lastMovement = 0;
std::stack<int> Movements;

int CNY[] = {A0,A1,A5}; // CNYLeft, CNYRight, CNYBottom
int Sharp[] = {A8,A4}; // SHarpLeft, SharpRight
int Ultrasonic = A3;
int MotorR[] = {5,6};
int MotorL[] = {9,10};
int Led[] = {12,13,11}; // Led1, Led2, Led3
int Battery = A6;

Robot::tState State = Robot::lookingforEXIT; // Must start in waiting

Robot robot(CNY,Sharp,Ultrasonic,MotorL,MotorR,Led,Battery,State,3,2);

void handle_battery(){
  //Serial1.print("bt" + String(pos[0]) + String(pos[1])+"#");
  Serial1.print("bt" + String(robot.BatteryState(),2) + "#");
}

void setup() {
  Serial1.begin(9600); // Bluetooth
  Serial.begin(9600);
  robot.init();
  int tickEvent = t.every(500, handle_battery);
}

void loop() {
  delay(10000);
  switch (State) {
    case Robot::waiting:
```

```
    break;

    case Robot::lookingforEXIT:
      lookforEXIT();
    break;

    case Robot::lookingforSTART:

    break;
  }
}

void lookforEXIT() {
  while (!isTheEnd()) {
    int allowedMovement = checkWalls();
    if (allowedMovement & RIGHT) {
      robot.Move(Robot::right,200);
      robot.Move(Robot::forward,200);
      robot.Move(Robot::stoprobot,0); testSensors(); //delay(5000);
    } else {
      if (allowedMovement & FORWARD) {
        robot.Move(Robot::forward,200);
        robot.Move(Robot::stoprobot,0); testSensors(); //delay(5000);
      } else {
        if (allowedMovement & LEFT) {
          robot.Move(Robot::left,200);
          robot.Move(Robot::forward,200);
          robot.Move(Robot::stoprobot,0); testSensors(); //delay(5000);
        } else {
          robot.Move(Robot::turn_back,200);
          robot.Move(Robot::stoprobot,0); testSensors(); //delay(5000);
        }
      }
    }
  }
  robot.Move(Robot::stoprobot,0);
}

void lookforSTART() {

}

int checkWalls() {
  int result = 0;
  if (12 < robot.ReadSharp('L') || robot.ReadSharp('L') < 0) {
    result |= LEFT;
  }
  if (12 < robot.ReadSharp('R') || robot.ReadSharp('R') < 0) {
    result |= RIGHT;
  }
  if (5 < robot.ReadUltrasonic()) {
    result |= FORWARD;
  }
  return result;
}

void radioControl(String m) {
   robot.Encoder(false);
   //if (Serial1.available() > 0 ) {
     //String m = robot.ReadBT();
     robot.MoveAbsolute(m[0],150,150);
   //}
}

bool isTheEnd() { // All CNY are Black
```

```
  return robot.ReadCNY('L')&&robot.ReadCNY('R')&&robot.ReadCNY('B');
}

void testSensors() {
  if (robot.ReadCNY('L'))
    Serial1.println("CNY L Black");
  else Serial1.println("CNY L White");
  if (robot.ReadCNY('R'))
    Serial1.println("CNY R Black");
  else Serial1.println("CNY R White");
  if (robot.ReadCNY('B'))
    Serial1.println("CNY B Black");
  else Serial1.println("CNY B White");
  Serial1.print("Ultrasonic: "); Serial1.println(robot.ReadUltrasonic());
  Serial1.print("Sharp L: "); Serial1.println(robot.ReadSharp('L'));
  Serial1.print("Sharp R: "); Serial1.println(robot.ReadSharp('R'));
}
```

6.2. Plan de pruebas

Se ha implementado una pequeña función que nos permite, a partir mediante Bluetooth conocer el estado de los sensores y poder actuar en consecuencia si las mediciones son anormales.

```
void testSensors() {
  if (robot.ReadCNY('L'))
    Serial1.println("CNY L Black");
  else
    Serial1.println("CNY L White");
  if (robot.ReadCNY('R'))
    Serial1.println("CNY R Black");
  else
    Serial1.println("CNY R White");
  if (robot.ReadCNY('B'))
    Serial1.println("CNY B Black");
  else
    Serial1.println("CNY B White");
  Serial1.print("Ultrasonic: "); Serial1.println(robot.ReadUltrasonic());
  Serial1.print("Sharp L: "); Serial1.println(robot.ReadSharp('L'));
  Serial1.print("Sharp R: "); Serial1.println(robot.ReadSharp('R'));
}
```

Capítulo 7

IMPLEMENTACIÓN

7.1. Librerías

7.1.1. Librería Externa

Se intentó utilizar en un primer lugar la STL de C++, pero no se pudo utilizar tal, por tanto, buscamos alternativas y acabamos encontrando un repositorio de Github [1] en el que se habían implementado todas las estructuras de la STL en Arduino. Ésto nos permitió utilizar una de las ideas que habíamos barajado inicialmente, que era la reducción de caminos para la vuelta atrás, que, además, tenía como ventaja con respecto a utilizar uno de los algoritmos clásicos como Floyd o Dijkstra, un menor coste computacional.

7.1.2. Librería Propia

Se ha implementado una librería en la que haciendo uso del paradigma de la Programación Orientada a Objetos, se permite utilizar un objeto de la clase Robot para acceder a todas las funciones de la clase. Para ver la implementación tanto de la especificación como de la implementación, diríjase a los Apéndices A y B.

7.2. Apuntes sobre el código

Para hacer un código más legible y mantenible durante el desarrollo del TAD Robot se ha optado por separar la codificación en dos archivos, un primer archivo en el que se encuentren las cabeceras de las funciones, *Robot.hpp*, y otro segundo archivo en el que se encuentran las implementaciones del TAD.

Por otro lado, se hacen uso de las interrupciones y de los encoders para dotar de una mayor precisión a los giros, aunque existe el problema de las dimensiones del laberinto, que dificulta en ocasiones la resolución del laberinto al haber utilizado un chasis que recubre la placa.

El funcionamiento general del mismo consiste en, inicialmente hacer una detección del entorno. A partir de ésta, se decide cuál es el próximo movimiento a realizar, que se guardará posteriormente en una pila. Supongamos que el robot ha girado a la derecha y se encuentra una pared frontal. Se desharía este movimiento y como anteriormente ha realizado un giro a la derecha, ahora sólo tendría dos posibles movimientos, moverse hacia adelante o girar a la izquierda. De esta forma se consigue que el robot consiga llegar a la casilla final, con una pila que contiene el número mínimo de movimientos.

[1] https://github.com/maniacbug/StandardCplusplus

Capítulo 8

MONTAJE

Se le ofrece a continuación de estas líneas, un diagrama de bloques en el que se pueden apreciar los distintos componentes que se han utilizado y cómo se conectan entre sí.

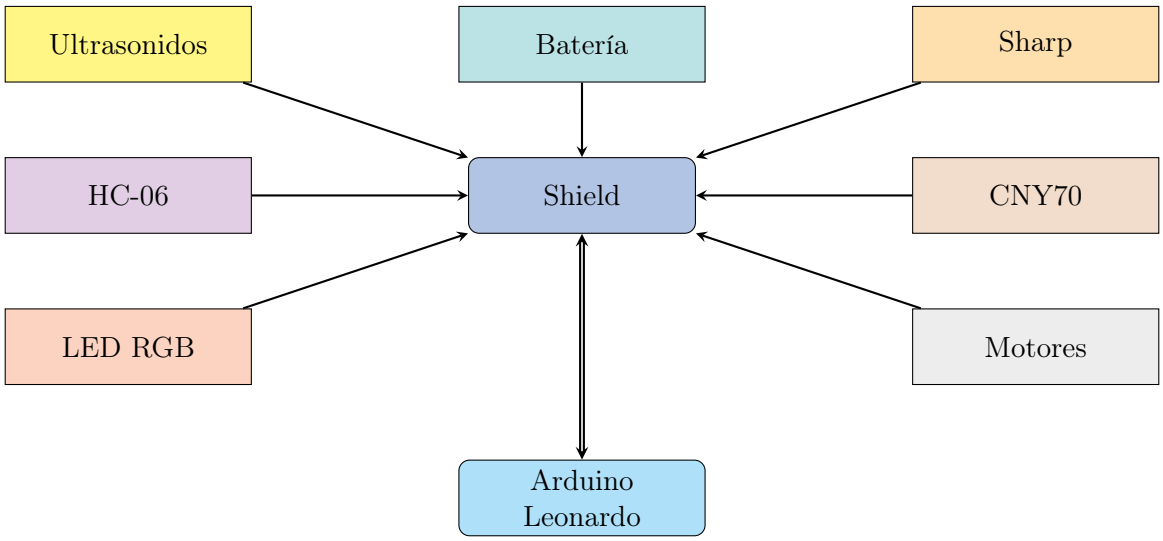


Figura 8.1: Elementos del robot

Antes de mostrar el resultado final, nos gustaría mostrar el proceso seguido hasta llegar a la versión final del robot. Se muestra en primer lugar el diseño de las piezas 3D, con la mejora del uso de soportes en formas de árbol, para que no hubiese ningún problema durante la impresión.

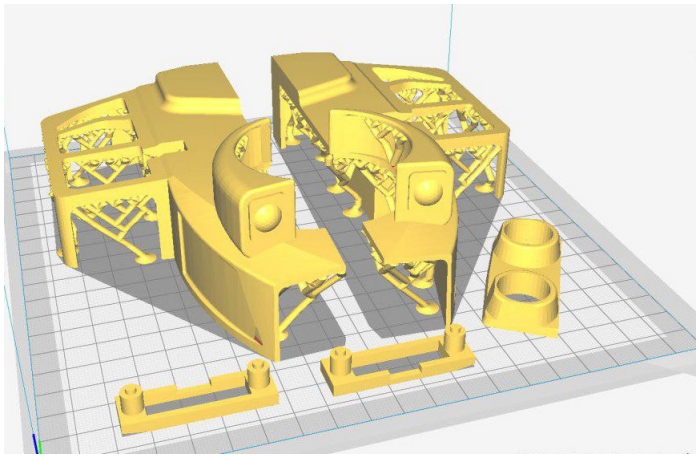


Figura 8.2: Diseño de las piezas 3D

Se observa a continuación el resultado de las piezas impresas para el ultrasonidos y los sharp.

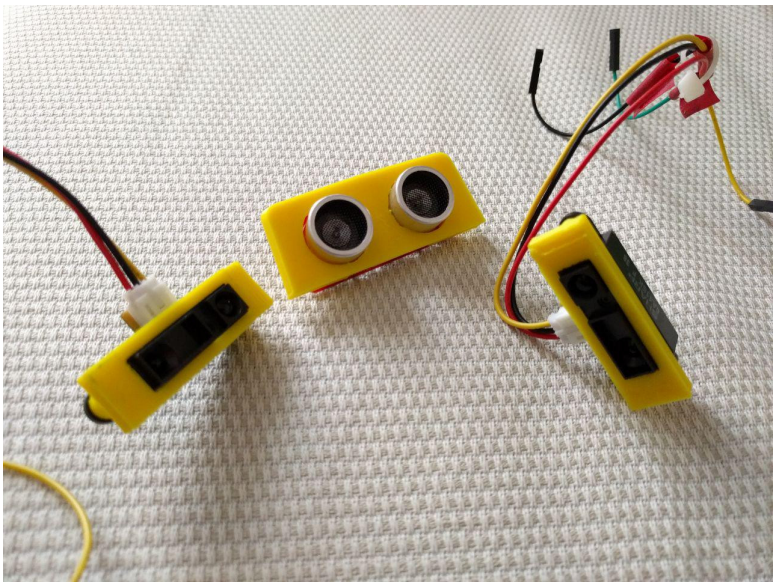


Figura 8.3: Piezas del Ultrasonidos y los Sharp

Tras unir todas las piezas, éste es el aspecto del robot:

Figura 8.4: Imagen del robot con su chásis

Se muestra además, una pequeña galería del robot en general:

Figura 8.5: Galería del robot

Finalmente se incluyen algunas capturas de la aplicación móvil desarrollada:

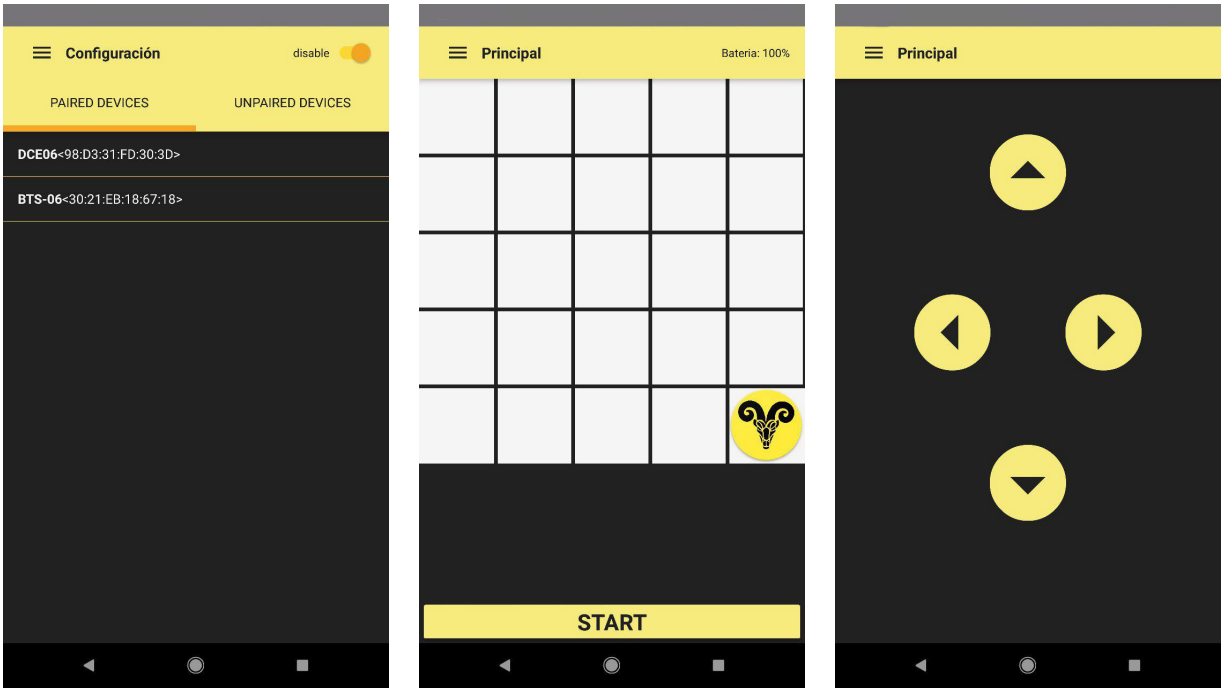


Figura 8.6: Muestra de la aplicación móvil

Capítulo 9

PRUEBAS

ID	CATEGORÍA	COMPROBACIÓN
RF01	Movimiento	✓El robot se mueve gracias al uso de los encoders.
RF02	Movimiento	✓El robot es capaz de girar 90º a la derecha con uso de encoders.
RF03	Movimiento	✓El robot es capaz de girar 90º a la izquierda con el uso de encoders.
RF04	Movimiento	✓El robot al no detectar una pared frontal, es capaz de moverse en línea recta.
RF05	Movimiento	✓Al cumplirse el requisito anterior, se cumple este.
RF06	Detección	✓Gracias al uso de ultrasonidos, se detecta la existencia de paredes frontales.
RF07	Detección	✓Gracias al uso del Sharp derecho, se detecta la existencia de paredes a la derecha.
RF08	Detección	✓Gracias al uso del Sharp izquierdo, se detecta la existencia de paredes a la derecha.
RF09	Detección	✓Gracias al uso de los CNY70 se detecta la transición entre celdas.
RF10	Detección	✓Gracias al uso de los CNY70 se detecta la casilla de salida.
RF11	Resolución	✓Se almacena información sobre las celdas ya visitadas.
RF12	Resolución	✓Debido al RF anterior, se decide cuáles son los posibles movimientos a realizar.
RF13	Resolución	✓El robot es capaz de recorrer varias celdas gracias al algoritmo implementado.
RF14	Resolución	✓El robot es capaz de resolver el laberinto.
RF15	Información	✓Se manda información a la app sobre las celdas recorridas.
RF16	Información	✓Se manda información a la app sobre sobre los obstáculos que hay en cada celda.
RF17	Información	✓Se manda información a la app sobre la velocidad que lleva el robot.
RF18	Información	✓Se manda información a la app sobre la distancia que ha recorrido el robot.
RF19	Información	✓Se manda información a la app sobre el tiempo que lleva el robot en el laberinto.
RF20	Información	✓Se manda información sobre la trayectoria realizada por el robot.
RF21	Información	✓Se manda información sobre el nº de celdas recorridas.
RF22	Información	✓La aplicación muestra el estado del laberinto, tal y como se muestra en la figura 8.4
RF23	Usuario	✓El usuario dispone de una aplicación móvil en la que puede comprobar multitud de cosas
RF24	Pruebas	✓Se dispone de una batería de pruebas que comprueba el correcto funcionamiento de todos los sensores

Cuadro 9.1: Requisitos Funcionales

Capítulo 10

MEJORAS

La principal y mayor mejora, además del diseño de las piezas 3D, es la aplicación desarrollada para las plataformas de Android e iOS. Esta aplicación permite ir más allá de la simple lectura del puerto Serial del Arduino y es que, es posible, controlar visualmente el estado del robot en el laberinto en cada momento, y además, el control de los movimientos que puede realizar el robot. Se incluyen además, algunas mejoras audiovisuales, como es el uso de la luz y de los sonidos para indicar los distintos eventos que pueden ocurrir.

Apéndices

Apéndice A

Especificacion del TAD Robot

```
#ifndef ROBOT_H
#define ROBOT_H

#include "Arduino.h"

class Robot {

public:
        enum tState {waiting, lookingforEXIT, lookingforSTART};
        enum tMove1 {forward, backward, right, left, rightbackward, leftbackward, turn_back,
        ↪  stoprobot};
        enum tMove2 {E_forward, E_backward, E_right, E_left, E_rightbackward, E_leftbackward};

        Robot(int* CNY, int* Sharp, int Ultrasonic, int* MotorL, int* MotorR,
                int* Led, int Battery, tState State, int left_encoder, int right_encoder):
                CNY{CNY}, Sharp{Sharp}, Ultrasonic{Ultrasonic}, MotorL{MotorL},
                MotorR{MotorR}, Led{Led}, Battery{Battery}, State{State},
                left_encoder{left_encoder}, right_encoder{right_encoder} {};

  void init();

        void Move(Robot::tMove1, int speed);
        void MoveEncoder(Robot::tMove2, int speedL, int speedR, float nT);
        void MoveAbsolute(char direction, int speedL, int speedR);
        void TurnOnLed(char color, int intensity);
        bool ReadCNY(char CNY);
        float ReadSharp(char sharp);
        float ReadUltrasonic();
        void WriteBT();
  String ReadBT();
  void set_State(tState s) {State=s;}
        tState get_State() {return State;}
  float BatteryState();
  void Encoder(bool state);

        ~Robot();

private :
        int* CNY; // CNYLeft, CNYRight, CNYBottom
        int* Sharp; // SHarpLeft, SharpRight
        int Ultrasonic;
        int* MotorR;
        int* MotorL;
  int* Led; // Led1, Led2, Led3
  int Battery;
  int left_encoder;
  int right_encoder;
        tState State;

};

#endif
```

Apéndice B

Implementacion del TAD Robot

```
#include "Robot.hpp"

volatile float nTicks = 0;

volatile unsigned long leftCount = 0;
volatile unsigned long rightCount = 0;

int* LEFT;
int* RIGHT;
bool EncoderState = true;

void Robot::Encoder(bool state){
  EncoderState = state;
}

void right_handle() {
  if (RIGHT != NULL && EncoderState) {
    if (rightCount >= nTicks) {
      analogWrite(RIGHT[0],LOW);
      analogWrite(RIGHT[1],LOW);
    }
    ++rightCount;
  }
}

void left_handle() {
  if (LEFT != NULL && EncoderState) {
    if (leftCount >= nTicks) {
      analogWrite(LEFT[0],LOW);
      analogWrite(LEFT[1],LOW);
    }
    ++leftCount;
  }
}

void Robot::init() {
  LEFT = MotorL;
  RIGHT = MotorR;
  pinMode(MotorL[0], OUTPUT);
  pinMode(MotorL[1], OUTPUT);
  pinMode(MotorR[0], OUTPUT);
  pinMode(MotorR[1], OUTPUT);
  pinMode(Led[0], OUTPUT);
  pinMode(Led[1], OUTPUT);
  pinMode(Led[2], OUTPUT);
  pinMode(left_encoder, INPUT);
  pinMode(right_encoder, INPUT);

  attachInterrupt(digitalPinToInterrupt(left_encoder), left_handle, RISING);
  attachInterrupt(digitalPinToInterrupt(right_encoder), right_handle, RISING);
}
```

```
void Robot::Move(Robot::tMove1 mov, int speed) {
  const float difference = 0.75;
  rightCount=0;
  leftCount=0;
  switch (mov) {
    case Robot::forward: // Forward
      EncoderState = false;
      while (!ReadCNY('B')) {
        int basura = ReadUltrasonic();
        if (ReadSharp('L') < 12 && ReadSharp('R') < 12) { // Two Walls
          if (ReadSharp('L')-ReadSharp('R') < -2) {
            MoveAbsolute('f',speed,speed*difference);
          } else {
            if (ReadSharp('L')-ReadSharp('R') > 2) {
              MoveAbsolute('f',speed*difference,speed);
            } else {
               MoveAbsolute('f',speed,speed);
            }
          }
        } else {
          if (ReadSharp('L') < 12 && ReadSharp('R') > 12 && ReadSharp('R') < 20) { // Left Wall
            if (ReadSharp('L') < 4) {
              MoveAbsolute('f',speed,speed*difference);
            } else {
              if (ReadSharp('L') > 8) {
                MoveAbsolute('f',speed*difference,speed);
              } else {
                MoveAbsolute('f',speed,speed);
              }
            }
          } else {
            if (ReadSharp('L') > 12 && ReadSharp('L') < 20 && ReadSharp('R') < 12) { // Right Wall
              if (ReadSharp('R') < 4) {
                MoveAbsolute('f',speed*difference,speed);
              } else {
                if (ReadSharp('R') > 8) {
                  MoveAbsolute('f',speed,speed*difference);
                } else {
                  MoveAbsolute('f',speed,speed);
                }
              }
            } else { // Now Walls
              MoveAbsolute('f',speed,speed);
            }
          }
        }
      }
      while (!ReadCNY('R') && !ReadCNY('L') && !(ReadUltrasonic()>2 && ReadUltrasonic()<3)) {
        Serial1.println(ReadUltrasonic());
        if (ReadSharp('L') < 12 && ReadSharp('R') < 12) { // Two Walls
          if (ReadSharp('L')-ReadSharp('R') < -2) {
            MoveAbsolute('f',speed,speed*difference);
          } else {
            if (ReadSharp('L')-ReadSharp('R') > 2) {
              MoveAbsolute('f',speed*difference,speed);
            } else {
               MoveAbsolute('f',speed,speed);
            }
          }
        } else {
          if (ReadSharp('L') < 12 && ReadSharp('R') > 12 && ReadSharp('R') < 20) { // Left Wall
            if (ReadSharp('L') < 4) {
              MoveAbsolute('f',speed,speed*difference);
            } else {
              if (ReadSharp('L') > 8) {
                MoveAbsolute('f',speed*difference,speed);
```

```
          } else {
            MoveAbsolute('f',speed,speed);
          }
        }
      } else {
        if (ReadSharp('L') > 12 && ReadSharp('L') < 20 && ReadSharp('R') < 12) { // Right Wall
          if (ReadSharp('R') < 4) {
            MoveAbsolute('f',speed*difference,speed);
          } else {
            if (ReadSharp('R') > 8) {
              MoveAbsolute('f',speed,speed*difference);
            } else {
              MoveAbsolute('f',speed,speed);
            }
          }
        } else { // Now Walls
          MoveAbsolute('f',speed,speed);
        }
      }
    }
  }
  EncoderState = true;
  MoveEncoder(Robot::E_backward,speed,speed,2);
break;

case Robot::backward: // Backward
  EncoderState = false;
  while (!ReadCNY('R') && !ReadCNY('L')) {
    if (ReadSharp('L') < 12 && ReadSharp('R') < 12) { // Two Walls
      if (ReadSharp('L')-ReadSharp('R') < -2) {
        MoveAbsolute('b',speed,speed*difference);
      } else {
        if (ReadSharp('L')-ReadSharp('R') > 2) {
          MoveAbsolute('b',speed*difference,speed);
        } else {
           MoveAbsolute('b',speed,speed);
        }
      }
    } else {
      if (ReadSharp('L') < 12 && ReadSharp('R') > 12 && ReadSharp('R') < 20) { // Left Wall
        if (ReadSharp('L') < 4) {
          MoveAbsolute('b',speed,speed*difference);
        } else {
          if (ReadSharp('L') > 6) {
            MoveAbsolute('b',speed*difference,speed);
          } else {
            MoveAbsolute('b',speed,speed);
          }
        }
      } else {
        if (ReadSharp('L') > 12 && ReadSharp('L') < 20 && ReadSharp('R') < 12) { // Right Wall
          if (ReadSharp('R') < 4) {
            MoveAbsolute('b',speed*difference,speed);
          } else {
            if (ReadSharp('R') > 6) {
              MoveAbsolute('b',speed,speed*difference);
            } else {
              MoveAbsolute('b',speed,speed);
            }
          }
        } else { // Now Walls
          MoveAbsolute('b',speed,speed);
        }
      }
    }
  }
```

```
  while (!ReadCNY('B')) {
    if (ReadSharp('L') < 12 && ReadSharp('R') < 12) { // Two Walls
      if (ReadSharp('L')-ReadSharp('R') < -2) {
        MoveAbsolute('b',speed,speed*difference);
      } else {
        if (ReadSharp('L')-ReadSharp('R') > 2) {
          MoveAbsolute('b',speed*difference,speed);
        } else {
           MoveAbsolute('b',speed,speed);
        }
      }
    } else {
      if (ReadSharp('L') < 12 && ReadSharp('R') > 12 && ReadSharp('R') < 20) { // Left Wall
        if (ReadSharp('L') < 4) {
          MoveAbsolute('b',speed,speed*difference);
        } else {
          if (ReadSharp('L') > 6) {
            MoveAbsolute('b',speed*difference,speed);
          } else {
            MoveAbsolute('b',speed,speed);
          }
        }
      } else {
        if (ReadSharp('L') > 12 && ReadSharp('L') < 20 && ReadSharp('R') < 12) { // Right Wall
          if (ReadSharp('R') < 4) {
            MoveAbsolute('b',speed*difference,speed);
          } else {
            if (ReadSharp('R') > 6) {
              MoveAbsolute('b',speed,speed*difference);
            } else {
              MoveAbsolute('b',speed,speed);
            }
          }
        } else { // Now Walls
          MoveAbsolute('b',speed,speed);
        }
      }
    }
  }
  EncoderState = true;
  rightCount=0; leftCount=0;
  MoveEncoder(Robot::E_forward,speed,speed,2);
break;

case Robot::left: // Left
  MoveEncoder(Robot::E_leftbackward,speed,speed,2);
  MoveEncoder(Robot::E_backward,speed,speed,2);
  MoveEncoder(Robot::E_rightbackward,speed,speed,2);
  MoveEncoder(Robot::E_left,speed,speed,12);
break;

case Robot::right: // Right
  MoveEncoder(Robot::E_rightbackward,speed,speed,2);
  MoveEncoder(Robot::E_backward,speed,speed,2);
  MoveEncoder(Robot::E_leftbackward,speed,speed,2);
  MoveEncoder(Robot::E_right,speed,speed,12);
break;

case Robot::leftbackward: // Left
  MoveEncoder(Robot::E_leftbackward,speed,speed,2);
  MoveEncoder(Robot::E_forward,speed,speed,3);
  MoveEncoder(Robot::E_rightbackward,speed,speed,2);
  MoveEncoder(Robot::E_leftbackward,speed,speed,12);
break;

case Robot::rightbackward: // Right
```

```
      MoveEncoder(Robot::E_rightbackward,speed,speed,2);
      MoveEncoder(Robot::E_forward,speed,speed,3);
      MoveEncoder(Robot::E_leftbackward,speed,speed,2);
      MoveEncoder(Robot::E_rightbackward,speed,speed,12);
    break;

    case Robot::turn_back:
      MoveEncoder(Robot::E_rightbackward,speed,speed,3);
      MoveEncoder(Robot::E_backward,speed,speed,4);
      MoveEncoder(Robot::E_leftbackward,speed,speed,3);
      MoveEncoder(Robot::E_forward,speed,speed,4);
      rightCount=0; leftCount=0; nTicks = 10;
      analogWrite(MotorR[0],speed);
      analogWrite(MotorR[1],LOW);
      analogWrite(MotorL[0],LOW);
      analogWrite(MotorL[1],speed);
      while((rightCount < nTicks) || (leftCount < nTicks)){}
    break;

    default: // Stop
      analogWrite(MotorR[0],LOW);
      analogWrite(MotorR[1],LOW);
      analogWrite(MotorL[0],LOW);
      analogWrite(MotorL[1],LOW);
  }
}

void Robot::MoveEncoder(Robot::tMove2 mov, int speedL, int speedR, float nT) {
  nTicks = nT;
  rightCount=0;
  leftCount=0;
  switch (mov) {
    case Robot::E_forward:
      analogWrite(MotorR[0],LOW);
      analogWrite(MotorR[1],speedR);
      analogWrite(MotorL[0],LOW);
      analogWrite(MotorL[1],speedL);
      while((rightCount < nTicks) || (leftCount < nTicks)){}
    break;

    case Robot::E_backward: // Back
      analogWrite(MotorR[0],speedR);
      analogWrite(MotorR[1],LOW);
      analogWrite(MotorL[0],speedL);
      analogWrite(MotorL[1],LOW);
      while((rightCount < nTicks) || (leftCount < nTicks)){}
    break;

    case Robot::E_right:
      analogWrite(MotorR[0],LOW);
      analogWrite(MotorR[1],LOW);
      analogWrite(MotorL[0],LOW);
      analogWrite(MotorL[1],speedL);
      while(leftCount < nTicks){}
    break;

    case Robot::E_left:
      analogWrite(MotorR[0],LOW);
      analogWrite(MotorR[1],speedR);
      analogWrite(MotorL[0],LOW);
      analogWrite(MotorL[1],LOW);
      while(rightCount < nTicks){}
    break;

    case Robot::E_rightbackward:
      analogWrite(MotorR[0],LOW);
```

```
      analogWrite(MotorR[1],LOW);
      analogWrite(MotorL[0],speedL);
      analogWrite(MotorL[1],LOW);
      while(leftCount < nTicks){}
    break;

    case Robot::E_leftbackward:
      analogWrite(MotorR[0],speedR);
      analogWrite(MotorR[1],LOW);
      analogWrite(MotorL[0],LOW);
      analogWrite(MotorL[1],LOW);
      while(rightCount < nTicks){}
    break;

    default: // Stop
      analogWrite(MotorR[0],LOW);
      analogWrite(MotorR[1],LOW);
      analogWrite(MotorL[0],LOW);
      analogWrite(MotorL[1],LOW);
  }
}


void Robot::MoveAbsolute(char position, int speedL, int speedR) {
  switch (position) {
    case 'f': // Forward
      analogWrite(MotorR[0],LOW);
      analogWrite(MotorR[1],speedR);
      analogWrite(MotorL[0],LOW);
      analogWrite(MotorL[1],speedL);
      break;

    case 'b': // Back
      analogWrite(MotorR[0],speedR);
      analogWrite(MotorR[1],LOW);
      analogWrite(MotorL[0],speedL);
      analogWrite(MotorL[1],LOW);
      break;

    case 'l': // Left
      analogWrite(MotorR[0],LOW);
      analogWrite(MotorR[1],speedR);
      analogWrite(MotorL[0],speedL);
      analogWrite(MotorL[1],LOW);
      break;

    case 'r': // Right
      analogWrite(MotorR[0],speedR);
      analogWrite(MotorR[1],LOW);
      analogWrite(MotorL[0],LOW);
      analogWrite(MotorL[1],speedL);
      break;

    default: // Stop
      analogWrite(MotorR[0],LOW);
      analogWrite(MotorR[1],LOW);
      analogWrite(MotorL[0],LOW);
      analogWrite(MotorL[1],LOW);
  }

}

void Robot::TurnOnLed(char color, int intensity) {
  switch (color) {
    case 'R' : analogWrite(Led[0],intensity); break;
    case 'G' : analogWrite(Led[1],intensity); break;
```

```
    case 'B' : analogWrite(Led[2],intensity); break;
    default:
      analogWrite(Led[0],0);
      analogWrite(Led[1],0);
      analogWrite(Led[2],0);
  }
}

bool Robot::ReadCNY(char cny) {
  int limit = 700;
  int value;
  switch (cny){
    case 'L': value = analogRead(CNY[0]); break;
    case 'R': value = analogRead(CNY[1]); break;
    case 'B': value = analogRead(CNY[2]); break;
    default: value = 0;
  }
  //Serial1.print(value);
  if (value>limit)
    return true; // Black
  else
    return false; // White
}

float Robot::ReadSharp(char sharp) {
  const float ResolutionADC = 0.00488, /*4.88mV*/
              b = 0.024, m = 11.89;
  int Value_Sharp = 0;
  switch (sharp){
    case 'L': Value_Sharp = analogRead(Sharp[0]); break;
    case 'R': Value_Sharp = analogRead(Sharp[1]); break;
    default: Value_Sharp = 0;
  }
  float Voltage = Value_Sharp*ResolutionADC;
  float x = (Voltage-b)/m;
  return (1-0.42*x)/x;
}

float Robot::ReadUltrasonic() {
  float distance;
  unsigned long time_bounce;

  pinMode(Ultrasonic,OUTPUT);
  digitalWrite(Ultrasonic,LOW);
  delayMicroseconds(5);
  digitalWrite(Ultrasonic,HIGH);
  delayMicroseconds(10);
  digitalWrite(Ultrasonic,LOW);
  pinMode(Ultrasonic,INPUT);
  time_bounce = pulseIn(Ultrasonic,HIGH);
  distance = 0.017 * time_bounce;

  return distance;
}

String Robot::ReadBT() {
  String s = Serial1.readStringUntil('#');
  Serial1.read();
  return s;
}

float Robot::BatteryState() {
  return analogRead(Battery)*(5.00/1023.00)*2+0.7;
}

Robot::~Robot(){}
```