```
/* 29/10/2017 Created by Andrea Petrella */
/* Programma per la gestione di un robot capace di uscire automaticamente da un labirinto aciclico.*/

/* SCHEMA DI FUNZIONAMENTO (SCHEMA A BLOCCHI con dettaglio):

  ARDUINO
  -> DC MOTORS controllati con modulo "L298N"
  --> 4 MOTORI PER IL MOVIMENTO DELLE RUOTE CONTROLLATI A 2 A 2 PER LATO
  -> MICRO SERVO MOTOR MODEL "SG90"
    --> 1 MOTORE PER MUOVERE DI 180° IL SENSORE AD ULTRASUONI
  -> ULTRASOUND SENSOR MODEL "HC-SR04"
    --> SENSORE PER VERIFICARE LA PRESENZA DI UN OSTACOLO
  -> IR LINE FIELDS SENSOR
    --> 3 SENSORI PER IDENTIFICARE LA FINE DEL PERCORSO
  -> BATTERY VOLTAGE SENSOR
    --> INPUT ANALOGICO

*/
/* SCHEMA DI CONNESSIONE

  INPUT
  -> ULTRASOUND SENSOR ECHO     - pin  8
  -> ULTRASOUND SENSOR TRIGGER  - pin  9
  -> IR SENSOR 1                - pin A3
  -> IR SENSOR 2                - pin A4
  -> IR SENSOR 3                - pin A5
  -> BATTERY VOLTAGE SENSOR     - pin A2
  OUTPUT
  -> DC DX MOTORS PW            - pin 11
  -> DC DX MOTORS EN1           - pin 12
  -> DC DX MOTORS EN2           - pin 13
  -> DC SX MOTORS PW            - pin  6
  -> DC SX MOTORS EN3           - pin  4
  -> DC SX MOTORS EN4           - pin  5
  -> SERVO MOTOR                - pin 10

*/

/** PROBLEMA DI RICERCA **
 *
 *  DEFINIZIONE = "TROVARE L'USCITA DEL LABIRINTO"
 *  STATO INIZIALE = "START"
 *  GOAL = "EXIT"
 *  INSIEME DEGLI STATI = {ogni punto P tale che: "P è un incrocio" ^ "P è un punto del labirinto"} U {"START"; "EXIT"}
 *  FUNZIONE DI COSTO = N.D.
 *
 *  AZIONI = {"ANDARE AVANTI"; "SVOLTARE A DESTRA"; "RIGIRARSI"}
 *  AZIONI_ESTESO = AZIONI U {"SVOLTARE A SINISTRA"}
 *
 *  PROPRIETA' DEL DOMINIO DEI LABIRINTI:
 *
 *  - LABIRINTO ACICLICO
 *  - ESISTENZA DI ALMENO UNA SOLUZIONE
 *
 *  ************************************
 *
 * * RISOLUZIONE DEL PROBLEMA DI RICERCA **
 *
 *  E' possibile astrarre ogni labirinto proposto all'Agente Razionale tramite un Albero di Ricerca.
 *
 *  L'Automa ripropone con i suoi movimenti una visita Forward di Profondità DFS dell'Albero di Ricerca con uso di Backtracking.
 *  L'Agente Razionale utilizza una Ricerca Online in quanto non conosce a priori i nodi dell'Albero di ricerca
 *  ma li scopre al momento dell'esplorazione del labirinto.
 *
 *  La scelta dell'algoritmo DFS Backtracking è giustificata dalla scarsa capacità di memoria del sistema di elaborazione.
 *
 */

/* ALGORITMO DI FUNZIONAMENTO

  Concretizzazione della visita dell'Albero di Ricerca a livello fisico/umano

  Il robot può uscire da un labirinto aciclico applicando il seguente algoritmo:

  Inizio
  - Se la parete a destra è libera
  --  Svoltare a destra
  - Altrimenti
  --  Se la parete frontale è libera
  ---   Proseguire in avanti
  --  Altrimenti
  ---   Rigirarsi
  - Se si è raggiunta l'uscita
  --  Fine
  - Altrimenti
  --  Ripetere la verifica


*/

/** ARCHITETTURA ROBOTICA **
 *
```

```
 *  UTILIZZO DELLO SCHEMA REATTIVO (S-A) CON UNA uPIANIFICAZIONE
 *
 *  SENSE:  ACQUISIZIONE DEI VALORI DELL'AMBIENTE
 *
 *  uPLANE: SCELTA DELLA VIA DI FUGA
 *
 *  ACT:    SPOSTAMENTO VERSO LA VIA DI FUGA SELEZIONATA
 *
 */

//Inclusione delle librerie
#include <NewPing.h>
#include <L298NDRIVER.h> //Libreria creata da me. Serve a gestire i motori DC con la scheda integrata L298N
#include <Servo.h>

//Definizione dei PIN
#define ultraS_echo 8
#define ultraS_trigger 9
#define ir_1 A3
#define ir_2 A4
#define ir_3 A5
#define battery_v A2
#define motorSX_pw 11
#define motorSX_en1 12
#define motorSX_en2 13
#define motorDX_pw 6
#define motorDX_en1 4
#define motorDX_en2 5
#define servoMotor 10

/*** COSTANTI ***/

//Per la scansione dell'ambiente
#define MAX_DISTANCE 250 // Distanza massima da scansionare (in centimetri). Il sensore copre distanze massime attorno ai 500cm.
#define LIMIT 40 // Distanza limite per fermare il robot prima che colpisca una parete laterale
#define LIMIT_FRONT 18 // Distanza limite per fermare il robot prima che colpisca la parete frontale
//Per gli spostamenti
#define cont_max 4
#define goTime_base 230
#define goDX_base 466
#define goSX_base 466
#define goBack_base 860
#define k_goTime 8.118783
#define k_goDX 8.801471
#define k_goSX 8.717647
#define k_goBack 8.950046
//Per il servo motore
#define servoZenit 82
#define servoSX 177
#define servoDX 0

/*** VARIABILI ***/

//Drivers
NewPing sonar(ultraS_trigger, ultraS_echo, MAX_DISTANCE);
L298NDRIVER motorDriver;
Servo servoDriver;

//Tensione della batteira
double valueBatteria = 0;
//Variabili per la regolazione delle rotazioni dei motori
int goTime = goTime_base;
int cont_rot = 0;
//Distanze dalle pareti
unsigned int sensor_value = 0; // in microsecondi (uS).
unsigned int distanceFront = 0;
unsigned int distanceRight = 0;
unsigned int distanceLeft = 0;


void setup() {

  //Setup del controller dei motori
  pinMode(motorSX_en1, OUTPUT); //output perche' definisce lo stato logico del pin IN1 del modulo L298N
  pinMode(motorSX_en2, OUTPUT); //output perche' definisce lo stato logico del pin IN1 del modulo L298N
  pinMode(motorDX_en1, OUTPUT); //output perche' definisce lo stato logico del pin IN2 del modulo L298N
  pinMode(motorDX_en2, OUTPUT); //output perche' definisce lo stato logico del pin IN2 del modulo L298N
  pinMode(motorSX_pw, OUTPUT);  //output perche' definisce il valore PWM del pin ENA del modulo L298N
  pinMode(motorDX_pw, OUTPUT);  //output perche' definisce il valore PWM del pin ENB del modulo L298N

  pinMode(battery_v, INPUT);  //sensore di tensione della batteria

  //Attivo il driver del servo motore e lo posiziono allo zenit
  servoDriver.attach(servoMotor);
  servoDriver.write(servoZenit);

  stopRobot(); //Fermo i motori
  delay(3000); //Aspetta prima di iniziare il programma

}

void loop() {
```

```
  while(!scanColor()){ //Controllo se il robot è posizionato a terra con i sensori IR

    stopRobot();//Fermo il robot
    delay(100);//Attesa per la stabilizzazione delle tensioni (anti rimbalzo)
    scanVoltage();//Controllo la carica della batteria
    scanWall(); //Avvia la scansione delle pareti

  }

  //Se il robot non è posizionato a terra lo fermo per 2 secondi prima di ricontrollare
  stopRobot();//Fermo il robot
  servoDriver.write(servoZenit); //Posiziono il sensore allo zenit
  delay(2000);

}

/*** METODI DRIVER PER GLI ATTUATORI ***/

//Metodo per far muovere il robot in avanti
void goRobot(){

  /* A causa delle diverse resistenze elettriche dei motori, il movimento dei motori di destra è meno agevolato
   * quindi va data più potenza per ottenere un movimento perpendicolare alle pareti */
  int potenzaDx = 212;
  int potenzaSx = 150;

  motorDriver.setForwardMove(motorDX_en1, motorDX_en2);
  motorDriver.setForwardMove(motorSX_en1, motorSX_en2);

  motorDriver.setPower(motorDX_pw, potenzaDx);
  motorDriver.setPower(motorSX_pw, potenzaSx);

}

//Metodo per fermare il robot
void stopRobot(){

  int potenza = 0;

  //Blocco i motori delle ruote
  motorDriver.stopMotor(motorDX_en1, motorDX_en2);
  motorDriver.stopMotor(motorSX_en1, motorSX_en2);

  motorDriver.setPower(motorDX_pw, potenza);
  motorDriver.setPower(motorSX_pw, potenza);

}

//Metodo per far muovere il robot verso destra
void goDXRobot(){

  int potenza = 230;
  int rotateDXTime = dipendentValue(goDX_base, k_goDX);

  motorDriver.setRotateDX(motorSX_en1, motorSX_en2, motorDX_en1, motorDX_en2);

  motorDriver.setPower(motorDX_pw, potenza);
  motorDriver.setPower(motorSX_pw, potenza);

  delay(rotateDXTime);

}

//Metodo per far muovere il robot verso sinistra
void goSXRobot(){

  int potenza = 230;
  int rotateSXTime = dipendentValue(goSX_base, k_goSX);

  motorDriver.setRotateSX(motorSX_en1, motorSX_en2, motorDX_en1, motorDX_en2);

  motorDriver.setPower(motorDX_pw, potenza);
  motorDriver.setPower(motorSX_pw, potenza);

  delay(rotateSXTime);

}

//Metodo per far girare su se stesso il robot
void goBackRobot(){

  int k_batterie = 60;

  int potenza = 230;
  int rotateBackTime = dipendentValue(goBack_base, k_goBack);

  motorDriver.setRotateDX(motorSX_en1, motorSX_en2, motorDX_en1, motorDX_en2);

  motorDriver.setPower(motorDX_pw, potenza);
  motorDriver.setPower(motorSX_pw, potenza);

  delay(rotateBackTime);
```

```
}

//Metodo per adattare i valori del movimento al valore di tensione delle batterie (retroazione)
double dipendentValue(int base, double k){

  double newValue = base * k /valueBatteria;

  return newValue;

}

/*** GESTIONE DEI SENSORI ***/

//Metodo per controllare il livello di tensione della batteria
void scanVoltage(){

  double offset = 0.05;

  int value = analogRead(battery_v); //Acquisisco il valore del sensore
  double volt = offset + (((value) * 2 * 4.9) / 1000); //Conversione in Volt
  valueBatteria = volt; //Salvataggio

  goTime = dipendentValue(goTime_base, k_goTime); //Retroazione

}

//Metodo per controllare il corretto posizionamento a terra del robot
boolean scanColor(){

  //Acquisisco il valore dei sensori IR
  boolean finish = digitalRead(ir_1) && digitalRead(ir_2) && digitalRead(ir_3);

  return finish;

}

/*** LOGICA PER LA RISOLUZIONE DEI LABIRINTI ***/

//Metodo per gestire la scansione delle pareti del labirinto
void scanWall(){

  //Controllo se la parete frontale non è libera
  if(scanLateral(servoZenit)){

    cont_rot = 0; //Resetto il conteggio dell'avanzamento prima di girare a destra

    //Controllo se la parete dx non è libera
    if(scanLateral(servoDX)){

      //Controlla se la parete sx non è libera
      if(scanLateral(servoSX)){

        //Vicolo cieco, faccio girare il robot su se stesso
        servoDriver.write(servoZenit);
        goBackRobot();

      }
      else{

        //"L" verso sinistra
        /* L'algoritmo originale vorrebbe che il robot si girasse su se stesso per poi ricontrollare se andare a destra o meno,
         * questo porterebbe successivamente comunque ad andare a sinistra,
         * quindi effettuo una predizione migliorando l'algoritmo andando direttamente a sinistra.
         *
         * Analizzando l'angolo di rotazione, avremmo:
         * -- DX = -90°
         * -- BACK = +180°
         * -- SX = +90°
         *
         * POSIZIONE_FINALE = INIZIO + BACK + DX = INIZIO + 180° + (-90°) = INIZIO + (180° -90°) = INIZIO + 90° = INIZIO + SX
         *
         */

        servoDriver.write(servoZenit);
        goSXRobot();
        goRobot();
        delay(goTime);

      }

    }

    else{

      //Vado a destra
      servoDriver.write(servoZenit);
      goDXRobot();
      //Proseguo un po'
      goRobot();
      delay(goTime);

    }
```

```
  }

  else{

    //Controllo se la parete dx è libera
    if(!scanLateral(servoDX)){

      //Vado a destra
      servoDriver.write(servoZenit);
      //Faccio girare immediatamente solo se sono andato un po' avanti prima
      if((cont_rot >= cont_max)){

        goDXRobot();
        cont_rot = 0;
        //Proseguo un po'
        goRobot();
        delay(goTime);

      }
      else //Altrimenti proseguo
        cont_rot++;

    }

  }

  //Vado avanti
  servoDriver.write(servoZenit);
  goRobot();
  delay(goTime);

}

//Metodo per controllare le pareti laterali
boolean scanLateral(int rotation){

  unsigned int values[] = {0,0,0};
  unsigned int tmp = 0;
  int array_dimens = 3;
  int minimo = 0;
  int massimo = 0;
  int index_minimo = 0;
  int index_massimo = 0;

  //Ruota la testa del robot
  servoDriver.write(rotation);
  delay(1000); //Da il tempo alla testa di girare

  //Scansiona 3 volte e per trovare la mediana
  for(int i = 0; i < array_dimens; i++){

    delay(100);
    values[i] = sonar.ping();

    if(i == 0){
      minimo = values[i];
      massimo = values[i];
    }
    else{
      if(values[i] < minimo){
        minimo = values[i];
        index_minimo = i;
      }
      if(values[i] > massimo){
        massimo = values[i];
        index_massimo = i;
      }
    }

  }

  //Ordina
  if(index_minimo == 0){
    if(index_massimo == 1){ // {m,M,x} -> {m,x,M}
      tmp = values[2];
      values[2] = massimo;
      values[1] = tmp;
    }
    //else {m,x,M} ok
  }
  else if(index_minimo == 1){
    if(index_massimo == 2){ // {x,m,M} -> {m,x,M}
      tmp = values[0];
      values[0] = minimo;
      values[1] = tmp;
    }
    else{ // {M,m,x} -> {M,x,m}
      tmp = values[2];
      values[2] = minimo;
      values[1] = tmp;
    }
  }
  else if(index_minimo == 2){
```

```
    if(index_massimo == 1){ // {x,M,m}->{M,x,m}
      tmp = values[0];
      values[0] = massimo;
      values[1] = tmp;
    }
    //else {M,x,m} ok
  }

  /*Calcolo la mediana
    (dalla statistica: la mediana tra n valori, con n dispari, è l'elemento centrale della collezzione ordinata degli n valori)
    */
  sensor_value = values[1];

  unsigned int distance = sensor_value / US_ROUNDTRIP_CM;

  if(rotation == servoZenit)
    return (distance <= LIMIT_FRONT);
  else
    return (distance <= LIMIT  && distance > 0);

}



/*** DEPRECATED ***

//Metodo per controllare le pareti laterali
boolean scanLateral(boolean side, int rotation){

  //Ruota la testa del robot
  servoDriver.write(rotation);
  delay(1000); //Da il tempo alla testa di girare
  //Scansiona
  sensor_value = sonar.ping();
  unsigned int distance = sensor_value / US_ROUNDTRIP_CM;
  if(side)
    distanceRight = distance;
  else
    distanceLeft = distance;
  delay(100); //Attende per elaborare la risposta del sonar

  return (distance <= LIMIT  && distance > 0);

}

//Metodo per inizializzare la comunicazione Seriale
void startSerial(){

  Serial.begin(9600);
  delay(1000);
  Serial.println("*** START ***");

}

//Metodo per stampare a video sul monitor seriale il valore dela tensione delle pile
void printVolt(){

  scanVoltage();

  Serial.println();
  Serial.println("**********************");
  Serial.println();
  Serial.print("Volt: ");
  Serial.print(valueBatteria);
  Serial.println("V");
  Serial.println();
  Serial.println("**********************");
  Serial.println();

}

void printValues(unsigned int a[], int l){

  Serial.println();
  Serial.println("**********************");
  Serial.println();
  Serial.print("[");
  for(int i = 0; i < l; i++){
    Serial.print(a[i]);
    if(i != l-1)
      Serial.print(";");
  }
  Serial.println("]");
  Serial.println();
  Serial.println("**********************");
  Serial.println();

}

//Metodo per la regolazione automatica della traiettoria
void regolaSterzo(int side){

  int c = 5;
```

```
  int k = 12;

  switch(side){

    //Vado a DX
    case 0:
      offsetDx = -c;
      offsetSx = c;
    //Vado a SX
    case 1:
      offsetDx = c*k;
      offsetSx = -c*k;

    default:
      offsetDx = 0;
      offsetSx = 0;

  }

}

*/
```