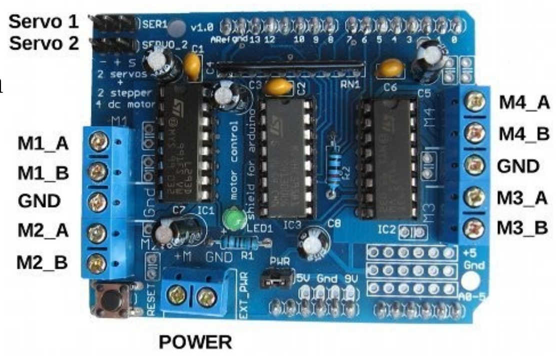
Nume proiect: IR Veh

Autori: Pirghie Dimitrie Ionut, Vlas Catalin

Echipament necesar :
1. Arduino Uno
2. IR Senzor
3. Sasiu masina (http://www.robofun.ro/kit-roboti)
4. Adafruit MotorShield V1 (sau similar)
5. Bateri

Librarii folosite :
1. Arduino-IRremote (https://github.com/z3t0/Arduino-IRremote)
   (Pentru senzorul infrarosu)
2. Adafruit-Motor-Shield-for-Arduino (https://github.com/adafruit/Adafruit-Motor-Shield-for-Arduino)
   (Pentru shield motoare)
3. ArduinoThread (https://github.com/ivanseidel/ArduinoThread)
   (Pentru o imbunatatii codul si pentru o planificare mai usoare a task-ulor)
4.

Descrierea proiectului :



Modulul infra-rosu citeste butoanele mapate de la o telecomanda pentru un modulator FM pentru masina.
Modurile in care functioneaza masinuta sunt : automat si manual. In modul manual se deschide sistemul si dispune de urmatoarele comenzi : **Buton 2** actioneaza masina sa se miste inainte, **4** inapoi, **2** la stanga si **6** dreapta. Butonul **0** face schimbarea intre modul automat si manual.
In cazul modului manual motoarele sunt actionate in directia data de butonul apasat cat timp este apasat.
De exemplu pentru butonul 2 citim de la senzor valuarea 0xFF18E7 si daca este mentinut apasat se citeste valoarea 0xFFFFFF, dar nu mereu, in bucla se poate intampla sa nu citim o valoare. Din aceasta cauza am realizat un “debouncing” pentru senzorul IR. Daca un buton este apasat se transmite motoarelor actiunea respectiva apoi starea se schimba cand, sunt NR_NO_IR_AFTER rulari alea functiei de citire sau se apasa alt buton. Cand nu se trece in starea NO_IR motoarele se opresc.
In modul automat se citeste distanta, cand se citeste o distanta mai mica decat IMPEDIMENT_LIMIT se incearca toate directiile (stanga, dreapta, spate) pana cand este o directie cu distanta mai mare de IMPEDIMENT_LIMIT, in cazul in care nu se gaseste nici o directie libera atunci masinuta se opreste si LED-ul rosu intra in starea de blink. Cand se detecteaza un obstacol si pana la gasirea directiei libere LED-ul rosu se aprinde.

Conexiuni : 9d, 0a, 1a, a6

Shield-ul de Motoare de la AdaFruit V1 utilizeaza toti pinii digitali din care **9d** si **10d** sunt pentru Servo1 si Servo2 si nu sunt sudati, deci ii punem folosi daca nu conectam un motor servo/stepper. Shield-ul este atasat la un Arduino Uno. Deoarece avem motoare mai mari de 5 V si nu este recomandat sa alimentam motoarele DC direct din Arduino am folosit alimentarea externa. Pentru

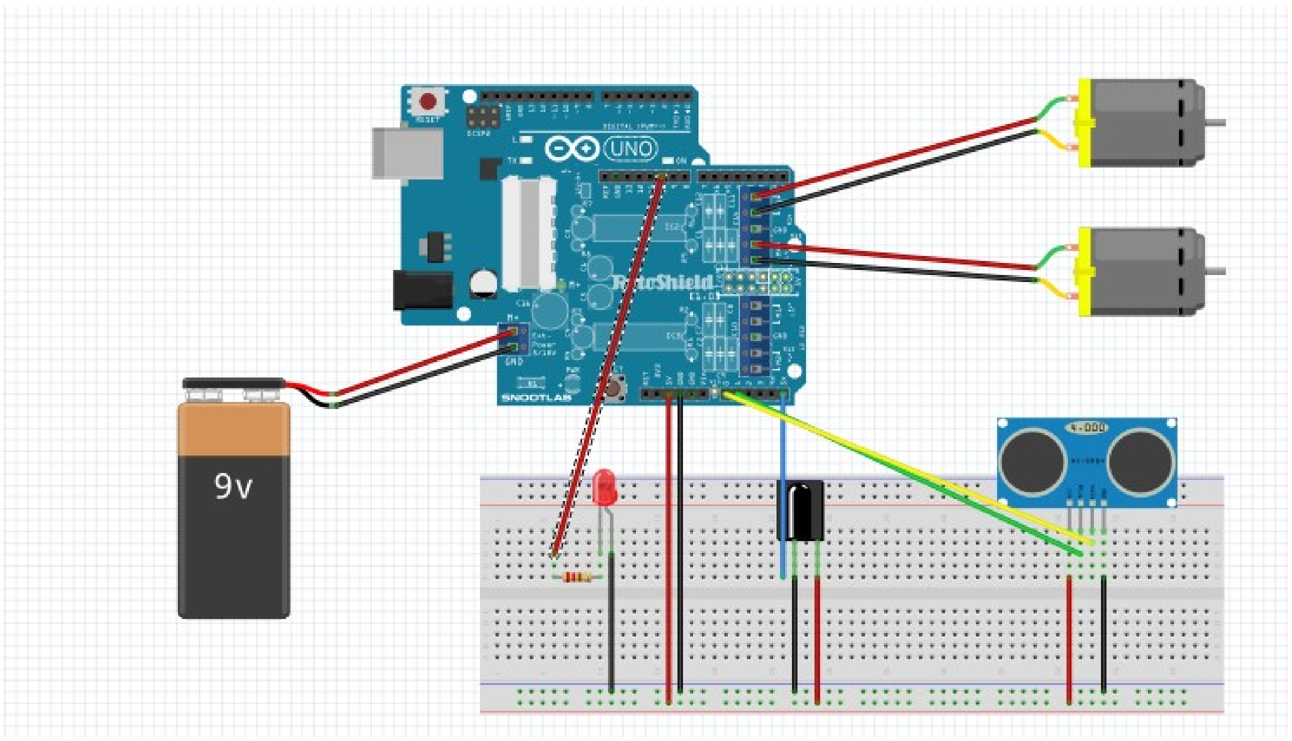
acest shield motoarele se alimenteaza de la baterie cand jumper-ul de cand legaturile de alimentare este scos.
Am preferat sa legam motoarele la M3 si M4, avand dificultati la citirea IR cand sunt legate la M1 si M2.

Pentru receptorul InfraRosu am folosit pinul **A5.**

Senzorul HC-SR04 este folosit pentru masurarea distantei, deoarece acesta functioneaza ca si un sonar are nevoie de doua conexiuni, am folosit **A0** si **A1.**

La pinul digital **10d** am conectat un led rosu.

Schema :



Cod : Pentru o mai buna programare si sincronizare a task-urilor de realizat (citirea infrarosu, actionare motoare, citire senzor distanta) am folosit libraria ArduinoThread care pentru fiecare variabila de tip Thread putem atasa o functie de callback care sa se apele la un interval de timp specificat.
Explicatii : Cand se citeste o noua valoare de la infra rosu. Pseudo-threadul se modifica valorile pentru motoare in functie de comanda data si se marcheaza faptul ca este o modificare.
In thread-ul pentru motoare se verifica daca este o modificare si daca da se aplica asupra motoarelor.
Parte de distanta este activa cand masinuta este un modul automat, starea motoarea este schimbata pentru manevre de evitarea ca si in cazul manual, iar masinuta ruleaza in fata.

```
// Header IRVeh.h contine structuri de date pentru motoare si valori pentru pini si butoane.
#define DEBUG_MSGS 1
#define IR_RECV_PIN   A5

// IR Remote buttons
#define CH_M          0xFFA25D
#define CH_S          0xFF629D
#define CH_P          0xFFE21D
#define PREV          0xFF22DD
#define NEXT          0xFF02FD
#define PLAY_PAUSE    0xFFC23D
#define VOL_M         0xFFE01F
#define VOL_P         0xFFA857
#define EQ            0xFF906F
#define B_100P        0xFF9867
#define B_200P        0xFFB04F
#define B_0           0xFF6897
#define B_1           0xFF30CF
#define B_2           0xFF18E7
#define B_3           0xFF7A85
#define B_4           0xFF10EF
#define B_5           0xFF38C7
#define B_6           0xFF5AA5
#define B_7           0xFF42BD
#define B_8           0xFF4AB5
#define B_9           0xFF52AD
#define NO_IR         0xFFAAAA
#define NR_NO_IR_AFTER 5

#define DELAY_MAV 1000
#define DEFAULT_SPEED 220
#define NO_TIME_LIMIT -1
#define IMPEDIMENT_LIMIT 28

//DISTANCE Sensor
#define TRIG_PIN A1//10
#define ECHO_PIN A0//9

#define STOP_PIN_LED 10

//DATA Structures
//Motors
struct MotorInfo {
  bool change = false;
  unsigned int motor_dir = RELEASE;
  unsigned int speed = DEFAULT_SPEED;
  unsigned long time = NO_TIME_LIMIT;
};

struct ImpledimentData{
  unsigned long frontDistance = 0;
  unsigned long leftDistance = 0;
  unsigned long righttDistance = 0;
  unsigned long backDistance = 0;
};


//IRHev.ino
#include <IRremote.h>
#include <AFMotor.h>
#include "Thread.h"
#include "ThreadController.h"
```

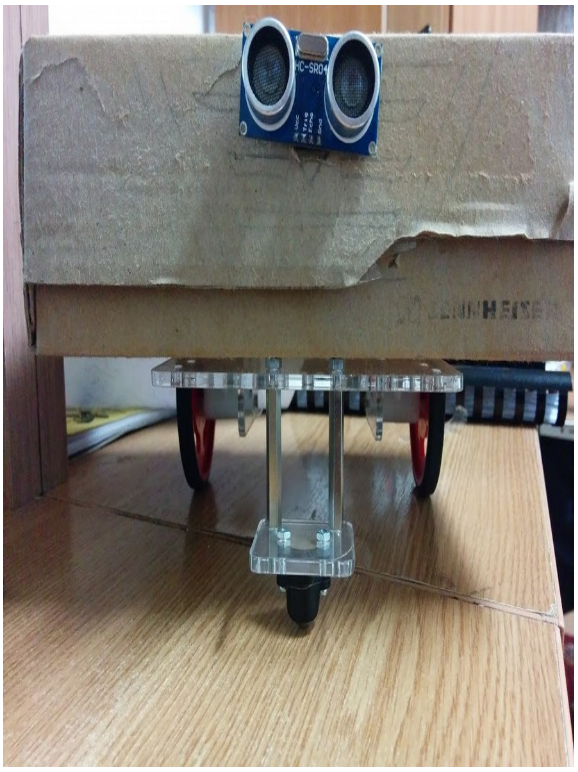
```
#include "IRVeh.h"

/////AUTOMATIC//////
bool automaticActivated = false;
bool avoidImpediment = false;
bool absoluteImpediment = false;
ImpledimentData impledimentData;

//////MOTORS/////////
AF_DCMotor motor_1(4);
AF_DCMotor motor_2(3);

MotorInfo motor_1_info;
MotorInfo motor_2_info;

/////IR RECV/////////
IRrecv irrecv(IR_RECV_PIN);
decode_results ir_result;
static unsigned long last_ir_value = NO_IR;
static unsigned long current_ir_value = NO_IR;

/////PSEUDO THREADS///
ThreadController controllThreads = ThreadController();
Thread irThread = Thread();
Thread motorsThread = Thread();
Thread distanceThread = Thread();

//////DISTANCE///////
long distance = 0;

void changeMotorDataByModeInfo(unsigned long what){

  if (true == automaticActivated){
    if(true == avoidImpediment){
      // continue
    }else if(what == NO_IR){
      return;
    }
  }

  switch(what){
    case NO_IR:
      motor_1_info.change = true;
      motor_1_info.motor_dir = RELEASE;
      motor_2_info.motor_dir = RELEASE;
    break;
    case B_2:
      motor_1_info.change = true;
      motor_1_info.motor_dir = FORWARD;
      motor_2_info.motor_dir = FORWARD;
    break;
    case B_4:
      motor_1_info.change = true;
      motor_1_info.motor_dir = FORWARD;
      motor_2_info.motor_dir = BACKWARD;
    break;
    case B_6:
      motor_1_info.change = true;
      motor_1_info.motor_dir = BACKWARD;
      motor_2_info.motor_dir = FORWARD;
    break;
    case B_8:
      motor_1_info.change = true;
      motor_1_info.motor_dir = BACKWARD;
      motor_2_info.motor_dir = BACKWARD;
    break;
    case B_0:
      if(automaticActivated == false){
#ifdef DEBUG_MSGS
        Serial.println("YES Automatic");
#endif
        automaticActivated = true;
        motor_1_info.change = true;
        motor_1_info.motor_dir = FORWARD;
        motor_2_info.motor_dir = FORWARD;
      }else{
#ifdef DEBUG_MSGS
        Serial.println("NO Automatic");
#endif
        absoluteImpediment = false;
        automaticActivated = false;
        digitalWrite(STOP_PIN_LED, LOW);

        motor_1_info.change = true;
        motor_1_info.motor_dir = RELEASE;
        motor_2_info.motor_dir = RELEASE;
      }

    break;
  }
}

void callBackIR(){
  static uint8_t no_ir_counter = 0;

  if(irrecv.decode(&ir_result)){
    irrecv.resume();
    no_ir_counter = 0; // reset no ir counter
    current_ir_value = ir_result.value;
#ifdef DEBUG_MSGS
  Serial.print("IR_MSG : 0x");
  Serial.println(ir_result.value, HEX);
#endif
    //ir received !
  }else{
    // no ir
    no_ir_counter = no_ir_counter + 1;
    if(no_ir_counter == NR_NO_IR_AFTER){
        current_ir_value = NO_IR;
        no_ir_counter = 0;
#ifdef DEBUG_MSGS
        Serial.println("IR : NO IR");
#endif
    }
  }

  if(last_ir_value != current_ir_value){
    last_ir_value = current_ir_value;
    changeMotorDataByModeInfo(last_ir_value);
#ifdef DEBUG_MSGS
    Serial.print("Last IR : 0x");
    Serial.println(last_ir_value, HEX);
#endif
  }
}
```

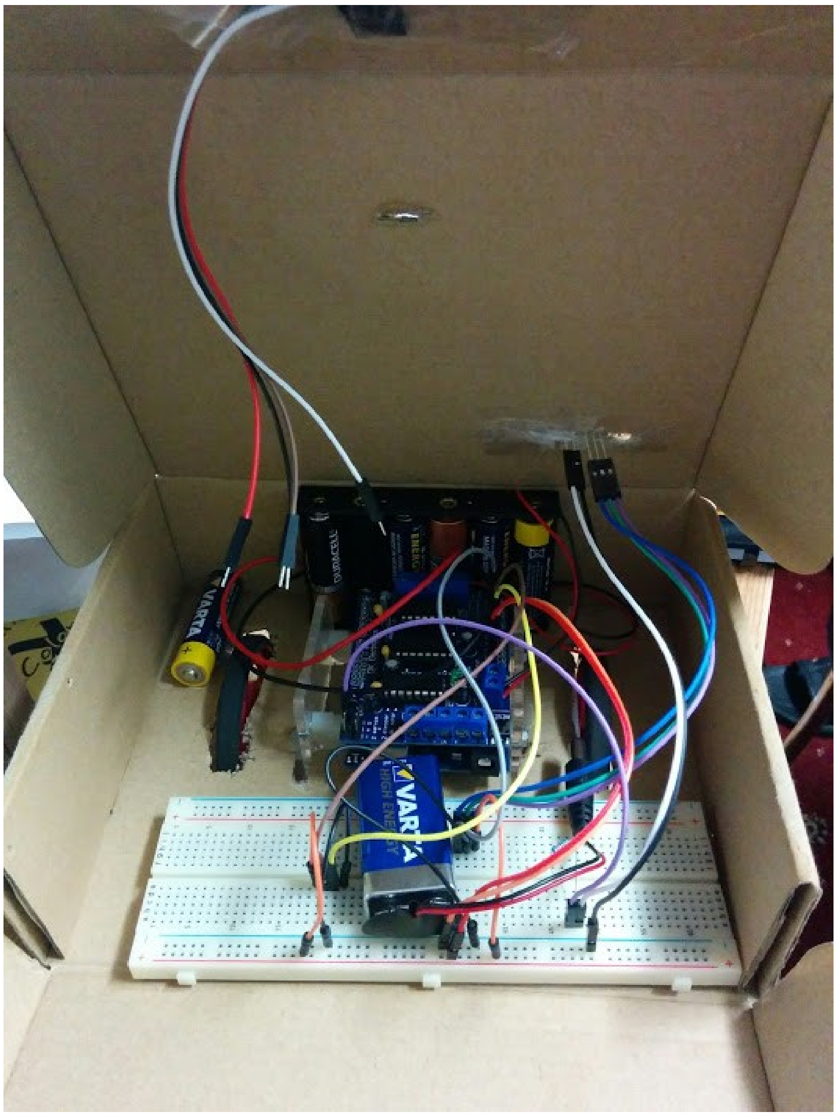
```
void callBackMotors(){

  if((false == motor_1_info.change)){
#ifdef DEBUG_MSGS
  Serial.println("MOTOR: NO CHANGE ABORT");
#endif
    return;
  }
  motor_1_info.change = false;

  motor_1.setSpeed(motor_1_info.speed);
  motor_2.setSpeed(motor_2_info.speed);

  motor_1.run(motor_1_info.motor_dir);
  motor_2.run(motor_2_info.motor_dir);

  if(motor_1_info.time != NO_TIME_LIMIT){
      delay(motor_1_info.time);
      motor_1.run(RELEASE);
      motor_2.run(RELEASE);

      motor_1_info.time = NO_TIME_LIMIT;
      delay(DELAY_MAV);
  }

#ifdef DEBUG_MSGS
  Serial.println("MOTOR UPDATE");
#endif
}

void readDistance(long &distanceInputRef){
  static long duration;

  digitalWrite(TRIG_PIN, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);
  duration = pulseIn(ECHO_PIN, HIGH);
  distanceInputRef = (duration/2)/29.1;
}

void blink_StopABS(){
  static bool ledStatus = false;
  digitalWrite(STOP_PIN_LED, (ledStatus == true ? HIGH: LOW));
  delay(1000);
  ledStatus = !ledStatus;
}

void callBackDistance(){

  if(automaticActivated == false){
      return;
  }

  if(true == absoluteImpediment){
    blink_StopABS();
    return;
  }

  readDistance(distance);
  if(distance < IMPEDIMENT_LIMIT){
#ifdef DEBUG_MSGS
```

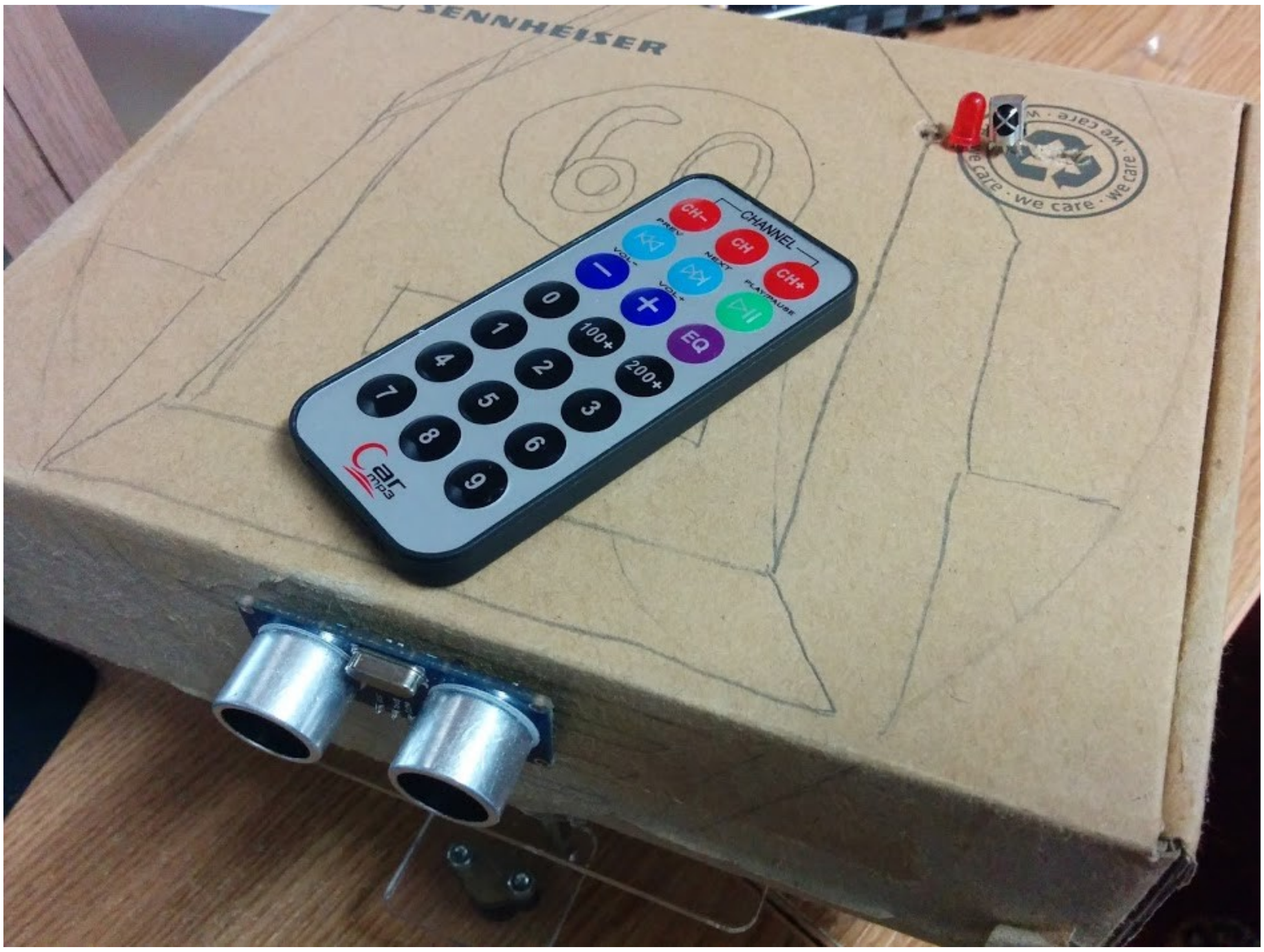
```
  Serial.print("Avoid impediment 1 : ");
  Serial.println(distance);
#endif
    avoidImpediment = true;

    // stop motors
    motor_1_info.change = true;
    motor_1_info.motor_dir = RELEASE;
    motor_2_info.motor_dir = RELEASE;
    callBackMotors();

    digitalWrite(STOP_PIN_LED, HIGH);
    delay(DELAY_MAV);

    //Move left
    motor_1_info.time = 750;
    changeMotorDataByModeInfo(B_4);
    callBackMotors();
    readDistance(distance);

    if(distance < IMPEDIMENT_LIMIT){

#ifdef DEBUG_MSGS
  Serial.print("Avoid impediment 2 : ");
  Serial.println(distance);
#endif
        //Move right
        motor_1_info.time = 1500;
        changeMotorDataByModeInfo(B_6);
        callBackMotors();
        readDistance(distance);
        if(distance < IMPEDIMENT_LIMIT){

#ifdef DEBUG_MSGS
  Serial.print("Avoid impediment 3 : ");
  Serial.println(distance);
#endif
          //Move back
          motor_1_info.time = 750;
          changeMotorDataByModeInfo(B_6);
          callBackMotors();
          readDistance(distance);

          if(distance < IMPEDIMENT_LIMIT){

#ifdef DEBUG_MSGS
  Serial.print("Avoid impediment absolut : ");
  Serial.println(distance);
#endif
              //absolute impediment
              absoluteImpediment = true;
              return;
          }
        }
    }
  }
  // move forward
  changeMotorDataByModeInfo(B_2);
  callBackMotors();
  avoidImpediment = false;
  digitalWrite(STOP_PIN_LED, LOW);

#ifdef DEBUG_MSGS
```

```
  Serial.print("Distance : ");
  Serial.println(distance);
#endif
}

void setup(){
  Serial.begin(9600);
  irrecv.enableIRIn();

  //PINS
  pinMode(TRIG_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);
  pinMode(STOP_PIN_LED, OUTPUT);

  irThread.onRun(callBackIR);
  irThread.setInterval(50);

  motorsThread.onRun(callBackMotors);
  motorsThread.setInterval(50);

  distanceThread.onRun(callBackDistance);
  distanceThread.setInterval(50);

  controllThreads.add(&irThread);
  controllThreads.add(&motorsThread);
  controllThreads.add(&distanceThread);
#ifdef DEBUG_MSGS
  Serial.println("Setup done");
#endif
}

void loop() {
  controllThreads.run();
}
```

Imagini :






SENNHEISER
CH-
CHANNEL
CH
CH+
PREV
NEXT
PLAY/PAUSE
VOL-
VOL+
EQ
100+
200+
Car mp3
we care