# UNIVERSITATEA TITU MAIORESCU

FACULTATEA: INFORMATICĂ
DEPARTAMENT: INFORMATICĂ
Programa de studii: INFORMATICĂ
DISCIPLINA: INTELIGENȚĂ ARTIFICIALĂ

# IA - *Testul de evaluare nr.* 6

## Robot Terestru pentru Stingerea Incendiilor – FIRE FIGHTING ROBOT

**EXAMEN**

| Nr. crt. | Grupa | Numele și prenumele | Semnătură student | Notă evaluare |
|---|---|---|---|---|
| 1 | | | | |
| 2 | | | | |
| 3 | | | | |
| 4 | | | | |
| 5 | | | | |
| 6 | | | | |

Data: __/__/____

CS I dr.ing.
Lucian Ștefăniță GRIGORE

Conf.dr.ing.
Iustin PRIESCU

Ș.I. dr.ing.
Dan-Laurențiu GRECU

## Cuprins

# 1. SENZORI

## 1.1 Senzori navigație terestră

### *1.1.1 SRF04 Ultra-Sonic Sensor*

Instrucțiunea de programare a SRF04

```
#define ECHOPIN 2                                 // Pin to receive echo pulse
#define TRIGPIN 3                                 // Pin to send trigger pulse

void setup(){
  Serial.begin(9600);
  pinMode(ECHOPIN, INPUT);
  pinMode(TRIGPIN, OUTPUT);
}

void loop(){
  digitalWrite(TRIGPIN, LOW);                     // Set the trigger pin to low for 2uS
  delayMicroseconds(2);
  digitalWrite(TRIGPIN, HIGH);                    // Send a 10uS high to trigger ranging
  delayMicroseconds(10);
  digitalWrite(TRIGPIN, LOW);                     // Send pin low again
  int distance = pulseIn(ECHOPIN, HIGH);          // Read in times pulse
  distance= distance/58;                          // Calculate distance from time of pulse
  Serial.println(distance);
  delay(50);                                      // Wait 50mS before next ranging
}
```

### *1.1.2 Senzor de distanță Sharp GP2Y0A02YK0F*

Instrucțiunea program pentru Sharp GP2Y0A02YK0F

```
void setup() {
  Serial.begin(9600);
}

void loop() {
  int valoareSenzor = readDistanceMediata(10, 0);
  Serial.print("Valoare senzor:");
  Serial.println(valoareSenzor);
  delay(100);
}

int readDistanceMediata(int count, int pin) {
  int sum = 0;
  for (int i = 0; i<count; i++) {
    float volts = analogRead(pin) * ((float) 5 / 1024);
    float distance = 65 * pow(volts, -1.10);
    sum = sum + distance;
    delay(5);
  }
  return (int) (sum/count);
}
```

### *1.1.3 Senzor proximitate VCNL400*

<u>Instrucțiunea program pentru Vishay VCNL400</u>

```
#include <Wire.h>
#define VCNL4000_ADDRESS 0x13  //I2C Address of the board
void setup(){
  Serial.begin(9600);  // Serial's used to debug and print data
  Wire.begin();  // initialize I2C stuff
  initVCNL4000(); //initialize and setup the board
}

void loop(){
  unsigned int ambientValue = readAmbient(); //can a tiny bit slow
  unsigned int proximityValue = readProximity();

  Serial.print(ambientValue);
  Serial.print(" | ");
  Serial.println(proximityValue);
  delay(100);  //Just here to slow down the printing
  //note that the readings take about 100ms to execute
}

void initVCNL4000(){
  byte temp = readVCNLByte(0x81);

  if (temp != 0x11){  // Product ID Should be 0x11
    Serial.print("initVCNL4000 failed to initialize");
    Serial.println(temp, HEX);
  }else{
    Serial.println("VNCL4000 Online...");
  }

  /*VNCL400 init params
   Feel free to play with any of these values, but check the datasheet first!*/
  writeVCNLByte(0x84, 0x0F);  // Configures ambient light measures - Single conversion mode, 128
averages
  writeVCNLByte(0x83, 15);  // sets IR current in steps of 10mA 0-200mA --> 200mA
  writeVCNLByte(0x89, 2);  // Proximity IR test signal freq, 0-3 - 781.25 kHz
  writeVCNLByte(0x8A, 0x81);  // proximity modulator timing - 129, recommended by Vishay
}

unsigned int readProximity(){
  // readProximity() returns a 16-bit value from the VCNL4000's proximity data registers
  byte temp = readVCNLByte(0x80);
  writeVCNLByte(0x80, temp | 0x08);  // command the sensor to perform a proximity measure

  while(!(readVCNLByte(0x80)&0x20));  // Wait for the proximity data ready bit to be set
  unsigned int data = readVCNLByte(0x87) << 8;
  data |= readVCNLByte(0x88);

  return data;
}
```

```
unsigned int readAmbient(){
  // readAmbient() returns a 16-bit value from the VCNL4000's ambient light data registers
  byte temp = readVCNLByte(0x80);
  writeVCNLByte(0x80, temp | 0x10);  // command the sensor to perform ambient measure

  while(!(readVCNLByte(0x80)&0x40));  // wait for the proximity data ready bit to be set
  unsigned int data = readVCNLByte(0x85) << 8;
  data |= readVCNLByte(0x86);

  return data;
}

void writeVCNLByte(byte address, byte data){
  // writeVCNLByte(address, data) writes a single byte of data to address
  Wire.beginTransmission(VCNL4000_ADDRESS);
  Wire.write(address);
  Wire.write(data);
  Wire.endTransmission();
}

byte readVCNLByte(byte address){
  // readByte(address) reads a single byte of data from address
  Wire.beginTransmission(VCNL4000_ADDRESS);
  Wire.write(address);
  Wire.endTransmission();
  Wire.requestFrom(VCNL4000_ADDRESS, 1);
  while(!Wire.available());
  byte data = Wire.read();

  return data;
}
```

### *1.1.4 Accelerometru 3 axe ADXL377*

Instrucțiunea program pentru ADXL377

```
int scale = 3;
boolean micro_is_5V = true;
void setup()
{
  // Initialize serial communication at 115200 baud
  Serial.begin(115200);
}
void loop()
{
  // Get raw accelerometer data for each axis
  int rawX = analogRead(A0);
  int rawY = analogRead(A1);
  int rawZ = analogRead(A2);
float scaledX, scaledY, scaledZ; // Scaled values for each axis
  if (micro_is_5V) // microcontroller runs off 5V
  {
    scaledX = mapf(rawX, 0, 675, -scale, scale); // 3.3/5 * 1023 =~ 675
  }
  else // microcontroller runs off 3.3V
  {
    scaledX = mapf(rawX, 0, 1023, -scale, scale);
  }
  // Print out raw X,Y,Z accelerometer readings
  Serial.print("X: "); Serial.println(rawX);

  // Print out scaled X,Y,Z accelerometer readings
  Serial.print("X: "); Serial.print(scaledX); Serial.println(" g");
delay(2000);
```

### *1.1.5 Giroscop 3 axe L3G4200D*

Instrucțiunea program pentru MEMS L3G4200D

```
*/
#include <SPI.h>
#include "L3G4200D.h"
// pin definitions
const int int2pin = 6;
const int int1pin = 7;
const int chipSelect = 10;
// gyro readings
int x, y, z;
void setup()
{
  Serial.begin(9600);
  Serial.println("L3G4200D Basic Example\r\n");
 // Start the SPI library:
  SPI.begin();
  SPI.setDataMode(SPI_MODE3);
  SPI.setClockDivider(SPI_CLOCK_DIV8);
  pinMode(int1pin, INPUT);
  pinMode(int2pin, INPUT);
  pinMode(chipSelect, OUTPUT);
  digitalWrite(chipSelect, HIGH);
  delay(100);
  setupL3G4200D(2);  // Configure L3G4200 with selectabe full scale range
  // 0: 250 dps
  // 1: 500 dps
  // 2: 2000 dps
    //Serial.println("int L3G4200D()\r\n");
  //int L3G4200D(); //Test and intialize the l3g4200
 // Serial.println("PASS int L3G4200D()\r\n");
}
void loop()
{
  // Don't read gyro values until the gyro says it's ready
  while(!digitalRead(int2pin))
    ;
  getGyroValues();  // This will update x, y, and z with new values
  Serial.print(x, DEC);
  Serial.print("\t");
  Serial.print(y, DEC);
  Serial.print("\t");
  Serial.print(z, DEC);
  Serial.print("\t");
  Serial.println();
  //delay(100); // may want to stick this in for readability
}
int readRegister(byte address)
{
  int toRead;
  address |= 0x80;  // This tells the L3G4200D we're reading;
  digitalWrite(chipSelect, LOW);
  SPI.transfer(address);
  toRead = SPI.transfer(0x00);
  digitalWrite(chipSelect, HIGH);

  return toRead;

}

void writeRegister(byte address, byte data)

{

  address &= 0x7F;  // This to tell the L3G4200D we're writing

  digitalWrite(chipSelect, LOW);

  SPI.transfer(address);

  SPI.transfer(data);

  digitalWrite(chipSelect, HIGH);

}

int setupL3G4200D(byte fullScale)

{

  // Let's first check that we're communicating properly

  // The WHO_AM_I register should read 0xD3

  if(readRegister(WHO_AM_I)!=0xD3)

    return -1;

  // Enable x, y, z and turn off power down:

  writeRegister(CTRL_REG1, 0b00001111);

  // If you'd like to adjust/use the HPF, you can edit the line below to configure CTRL_REG2:

  writeRegister(CTRL_REG2, 0b00000000);

  // Configure CTRL_REG3 to generate data ready interrupt on INT2

  // No interrupts used on INT1, if you'd like to configure INT1

  // or INT2 otherwise, consult the datasheet:

  writeRegister(CTRL_REG3, 0b00001000);

  // CTRL_REG4 controls the full-scale range, among other things:

  fullScale &= 0x03;
  writeRegister(CTRL_REG4, fullScale<<4);
  // CTRL_REG5 controls high-pass filtering of outputs, use it
  // if you'd like:
  writeRegister(CTRL_REG5, 0b00000000);
}
void getGyroValues()
{
  x = (readRegister(0x29)&0xFF)<<8;
  x |= (readRegister(0x28)&0xFF);
  y = (readRegister(0x2B)&0xFF)<<8;
  y |= (readRegister(0x2A)&0xFF);
  z = (readRegister(0x2D)&0xFF)<<8;
  z |= (readRegister(0x2C)&0xFF);
}
```

## 1.2 Senzori mediu și detecție substanțe și temperatură

### *1.2.1 Weather Sensor Assembly p/n 80422*

```
***************************************************************/
#define uint unsigned int
#define ulong unsigned long
#define PIN_ANEMOMETER 2 // Digital 2
#define PIN_VANE 5 // Analog 5
// How often we want to calculate wind speed or direction
#define MSECS_CALC_WIND_SPEED 5000
#define MSECS_CALC_WIND_DIR 5000
volatile int numRevsAnemometer = 0;
// Incremented in the interrupt ulong nextCalcSpeed;
// When we next calc the wind speed  ulong nextCalcDir;
// When we next calc the direction ulong time;
// Millis() at each start of loop().
// ADC readings:
#define NUMDIRS 8
ulong adc[NUMDIRS] = {26, 45, 77, 118, 161, 196, 220, 256};
// These directions match 1-for-1 with the values in adc, but
// will have to be adjusted as noted above. Modify 'dirOffset'
// to which direction is 'away' (it's West here).
char *strVals[NUMDIRS] = {"W","NW","N","SW","NE","S","SE","E"};
byte dirOffset=0;
//=======================================================
// Initialize
//=======================================================
void setup() {
        Serial.begin(9600);
        pinMode(PIN_ANEMOMETER, INPUT);
        digitalWrite(PIN_ANEMOMETER, HIGH);
        attachInterrupt(0, countAnemometer, FALLING);
        nextCalcSpeed = millis() + MSECS_CALC_WIND_SPEED;
        nextCalcDir = millis() + MSECS_CALC_WIND_DIR;
        }
//=======================================================
// Main loop.
//=======================================================
void loop() {
        time = millis();
        if (time >= nextCalcSpeed) {
                calcWindSpeed();

                nextCalcSpeed = time + MSECS_CALC_WIND_SPEED;

                }

        if (time >= nextCalcDir) {

                calcWindDir();

                nextCalcDir = time + MSECS_CALC_WIND_DIR;

                }

        }

//=============================================

// Interrupt handler for anemometer. Called each time the reed

// switch triggers (one revolution).

//=============================================

void countAnemometer() {

        numRevsAnemometer++;

        }

//=============================================

// Find vane direction.

//=============================================

void calcWindDir() {

        int val;

        byte x, reading;

        val = analogRead(PIN_VANE);

        val >>=2;

        // Shift to 255 range reading = val;

        // Look the reading up in directions table.

        Find the first value

        // that's >= to what we got.
        for (x=0; x= reading) break;
        } //Serial.println(reading, DEC);
        x = (x + dirOffset) % 8;
        // Adjust for orientation Serial.print(" Dir: ");
        Serial.println(strVals[x]);
        }
//=============================================
// Calculate the wind speed, and display it (or log it, whatever).
// 1 rev/sec = 1.492 mph
//=============================================
void calcWindSpeed() {
        int x, iSpeed;
        // This will produce mph * 10
        // (didn't calc right when done as one statement) long speed = 14920;
        speed *= numRevsAnemometer;
        speed /= MSECS_CALC_WIND_SPEED;
        iSpeed = speed;
        // Need this for formatting below Serial.
        print("Wind speed: ");
        x = iSpeed / 10;
        Serial.print(x);
        Serial.print('.');
        x = iSpeed % 10;
        Serial.print(x);
        numRevsAnemometer = 0;
        // Reset counter
        }
```

## 2. SOFTWARE de COMANDĂ şi CONTROL SISTEM INTEGRAT

```
#include <Wire.h>

#define PIN_ANEMOMETER  2     // Digital 2
#define PIN_VANE        0     // Analog 0
#define MSECS_CALC_WIND_SPEED 500
#define MSECS_CALC_WIND_DIR   500
volatile int numRevsAnemometer = 0; // Incremented in the interrupt
unsigned long nextCalcSpeed;              // When we next calc the wind speed
unsigned long nextCalcDir;                // When we next calc the direction
unsigned long time;                       // Millis() at each start of loop().
// ADC readings:
#define NUMDIRS 8
unsigned long   adc[NUMDIRS] = {
  17, 25, 37, 57, 81, 125,160,164};
// These directions match 1-for-1 with the values in adc, but
// will have to be adjusted as noted above. Modify 'dirOffset'
// to which direction is 'away' (it's West here).
char *strVals[NUMDIRS] = {
  "S","W","SW","NW","SE","N","E","NE"};
byte dirOffset=0;
int state=0;
int tmp102Address = 0x48;
void countAnemometer() {
  numRevsAnemometer++;
}
void setup(){
  Serial.begin(9600);
  Wire.begin();
  // init motors
  pinMode(6,OUTPUT);
  pinMode(7,OUTPUT);
  pinMode(10,OUTPUT);
  pinMode(11,OUTPUT);
  //
  pinMode(9, OUTPUT);
  pinMode(12, OUTPUT);
  //init anemometru
  pinMode(PIN_ANEMOMETER, INPUT);
  digitalWrite(PIN_ANEMOMETER, HIGH);
  attachInterrupt(0, countAnemometer, FALLING);
  nextCalcSpeed = millis() + MSECS_CALC_WIND_SPEED;
  nextCalcDir   = millis() + MSECS_CALC_WIND_DIR;
}

//=======================================================
// Find vane direction.
//=======================================================
void calcWindDir() {
  int val;
  byte x, reading;
  val = analogRead(PIN_VANE);
  val >>=2;                        // Shift to 255 range
  reading = val;
  //Serial.println(reading);
  // Look the reading up in directions table. Find the first value
  // that's >= to what we got.
  for (x=0; x<NUMDIRS; x++) {
    if (adc[x] >= reading)
      break;
  }
  //Serial.println(reading, DEC);
  //x = (x + dirOffset) % 8;   // Adjust for orientation
  Serial.print("  Dir: ");
  //Serial.print(x-1);
  Serial.println(strVals[x]);
}

//=======================================================
// Calculate the wind speed, and display it (or log it, whatever).
// 1 rev/sec = 1.492 mph
//=============================================================
void calcWindSpeed() {
  int x, iSpeed;
  // This will produce mph * 10
  // (didn't calc right when done as one statement)
  long speed = 14920;
  speed *= numRevsAnemometer;
  speed /= MSECS_CALC_WIND_SPEED;
  iSpeed = speed;         // Need this for formatting below
  Serial.print(" Wind speed: ");
  x = iSpeed / 10;
  Serial.print(x);
  Serial.print('.');
  x = iSpeed % 10;
  Serial.print(x);
  numRevsAnemometer = 0;        // Reset counter
}

void calculateWindDir(){
  time = millis();
  if (time >= nextCalcSpeed) {
    calcWindSpeed();
    nextCalcSpeed = time + MSECS_CALC_WIND_SPEED;
  }

  if (time >= nextCalcDir) {
    calcWindDir();
    getTemperature();
    getSmoke();
    nextCalcDir = time + MSECS_CALC_WIND_DIR;
  }

}
```

```
void getTemperature(){
  Wire.requestFrom(tmp102Address,2);
  byte MSB = Wire.read();
  byte LSB = Wire.read();
  //it's a 12bit int, using two's compliment for negative
  int TemperatureSum = ((MSB << 8) | LSB) >> 4;
  float celsius = TemperatureSum*0.0625;
  Serial.print("Celsius: ");
  Serial.print(celsius);
  //return celsius;
}
void getSmoke(){
  int analog_reading = analogRead(2);
  int percent_reading=map(analog_reading, 0, 1023, 0, 100);
  Serial.print(" Fum: ");
  Serial.print(percent_reading);
}

void loop(){
  calculateWindDir();
  int ch1 = pulseIn(5, HIGH, 25000);
  int ch3 = pulseIn(3, HIGH, 25000);
  int ch = pulseIn(8, HIGH, 25000);
  //Serial.println(digitalRead(2));
  //Serial.print(ch1);
  //Serial.print("\t");
  //Serial.print(ch3);
  //Serial.print("\t");
  //Serial.println(ch);
  if(state!=1 && ch>1500){
    rotate(16000,0.5);
    state=1;
  }else if(state!=0 && ch<1500){
    rotate(-16000,0.5);
    state=0;
```

```
}

int sterring=map(ch1,1070,1900,-255,255);
int forward=map(ch3,1070,1900,-255,255);
int motor1;
int motor2;
if(sterring>-40 && sterring<30){
  if(forward>80){
    digitalWrite(7,LOW);
    analogWrite(6,255);
    digitalWrite(11,LOW);
    analogWrite(10,255);
  }
  else if(forward<(-80)){
    digitalWrite(7,HIGH);
    analogWrite(6,255);
    digitalWrite(11,HIGH);
    analogWrite(10,255);
  }
  else{
    digitalWrite(7,LOW);
    analogWrite(6,0);
    digitalWrite(11,LOW);
    analogWrite(10,0);
  }
}
else{
  if(sterring>0){
    digitalWrite(7,LOW);
    analogWrite(6,255);
    digitalWrite(11,LOW);
    analogWrite(10,abs(forward));
  }
  else{
    digitalWrite(7,LOW);
```

```
      analogWrite(6,abs(forward));
      digitalWrite(11,LOW);
      analogWrite(10,255);
     }
    }
    /*
    Serial.print(ch1);
     Serial.print("\t");
     Serial.print(ch3);
     Serial.print("\t");
     Serial.print(forward);


STAȚIE METEO.

*************************************************************/
    #define uint unsigned int
    #define ulong unsigned long
    #define PIN_ANEMOMETER 2 // Digital 2
    #define PIN_VANE 5 // Analog 5
    // How often we want to calculate wind speed or direction
    #define MSECS_CALC_WIND_SPEED 5000
    #define MSECS_CALC_WIND_DIR 5000
    volatile int numRevsAnemometer = 0;
    // Incremented in the interrupt ulong nextCalcSpeed;
    // When we next calc the wind speed  ulong nextCalcDir;
    // When we next calc the direction ulong time;
    // Millis() at each start of loop().
    // ADC readings:
    #define NUMDIRS 8
    ulong adc[NUMDIRS] = {26, 45, 77, 118, 161, 196, 220, 256};
    // These directions match 1-for-1 with the values in adc, but
    // will have to be adjusted as noted above. Modify 'dirOffset'
    // to which direction is 'away' (it's West here).
    char *strVals[NUMDIRS] = {"W","NW","N","SW","NE","S","SE","E"};
    byte dirOffset=0;
```

```
//=============================================
// Initialize
//=============================================
void setup() {
        Serial.begin(9600);
        pinMode(PIN_ANEMOMETER, INPUT);
        digitalWrite(PIN_ANEMOMETER, HIGH);
        attachInterrupt(0, countAnemometer, FALLING);
        nextCalcSpeed = millis() + MSECS_CALC_WIND_SPEED;
        nextCalcDir = millis() + MSECS_CALC_WIND_DIR;
        }
//=============================================
// Main loop.
//=============================================
void loop() {
        time = millis();
        if (time >= nextCalcSpeed) {
        calcWindSpeed();
        nextCalcSpeed = time + MSECS_CALC_WIND_SPEED;
        }
if (time >= nextCalcDir) {
        calcWindDir();
        nextCalcDir = time + MSECS_CALC_WIND_DIR;
        }
}
//=============================================
// Interrupt handler for anemometer. Called each time the reed
// switch triggers (one revolution).
//=============================================
void countAnemometer() {
        numRevsAnemometer++;
        }
//=============================================
// Find vane direction.
//=============================================
void calcWindDir() {
        int val;
        byte x, reading;
        val = analogRead(PIN_VANE);
        val >>=2;
        // Shift to 255 range reading = val;
        // Look the reading up in directions table.
        Find the first value
        // that's >= to what we got.
        for (x=0; x= reading) break;
        } //Serial.println(reading, DEC);
        x = (x + dirOffset) % 8;
        // Adjust for orientation Serial.print(" Dir: ");
        Serial.println(strVals[x]);
        }
//=============================================
// Calculate the wind speed, and display it (or log it, whatever).
// 1 rev/sec = 1.492 mph
//=============================================
void calcWindSpeed() {
        int x, iSpeed;
        // This will produce mph * 10
        // (didn't calc right when done as one statement) long speed = 14920;
        speed *= numRevsAnemometer;
        speed /= MSECS_CALC_WIND_SPEED;
        iSpeed = speed;
        // Need this for formatting below Serial.
        print("Wind speed: ");
        x = iSpeed / 10;
        Serial.print(x);
        Serial.print('.');
        x = iSpeed % 10;
        Serial.print(x);
        numRevsAnemometer = 0;
        // Reset counter
        }
inline void ready()
{
  unsigned long lastblock = 0;  //the last block number saved in the sd card
  unsigned long tempblock = 0;
  tempblock = EEPROM.read(0);  // remember the LSB of the last saved block
  lastblock |= tempblock;
  tempblock = EEPROM.read(1);  // remember the next LSB of the last saved block
  lastblock |= tempblock << 8;
  tempblock = EEPROM.read(2);  // remember the next LSB of the last saved block
  lastblock |= tempblock << 16;
  tempblock = EEPROM.read(3);  // remember the next MSB of the last saved block
  lastblock |= tempblock << 24;
  Serial.println("ready");  //send computer the ready to reset message
  Serial.println(lastblock);  //send computer the last saved block number
  delay(10000);  //every 10 seconds
}//end of ready
/* Card type: Ver2.00 or later Standard Capacity SD Memory Card
              1.0 and 2.0 GB cards purchased in 2009 work well.
   Usage:  Must have global variable.
              volatile unsigned char buffer[512];
           Function calls.
              unsigned char error = SDCARD.readblock(unsigned long n);
              unsigned char error = SDCARD.writeblock(unsigned long n);
            error is 0 for correct operation
            read copies the 512 bytes from sector n to buffer.
            write copies the 512 bytes from buffer to the sector n.
   References: SD Specifications. Part 1. Physical Layer Simplified Specification
               Version 2.00 September 25, 2006 SD Group.
               http://www.sdcard.org
   Code examples:  http://www.sensor-networks.org/index.php?page=0827727742
                   http://www.avrfreaks.net   search "sd card"
   Operation:  The code reads/writes direct to the sectors on the sd card.
               It does not use a FAT. If the card has been formatted the
```

```
             FAT at the lowest sectors and files at the higher sectors
             can be written over.
             The card is not damaged but will need to be reformatted at
             the lowest level to be used by windows/linux.
   Timing:  readblock or writeblock takes 44 msec.
   Improvement: Could initialize so that can use version 1 sd and hc sd.
              Instead of CMD1 need to use CMD8, CMD58 and CMD41.
*/
#ifndef SDCARD_h
#define SDCARD_h
#define setupSPI SPCR = 0x53; //Master mode, MSB first,
                         //SCK phase low, SCK idle low, clock/64
#define deselectSPI SPCR = 0x00;  //deselect SPI after read write block
#define clearSPI  SPSR = 0x00; // clear SPI interrupt bit
#define setupDDRB DDRB |= 0x2c;  //set SS as output for cs
#define selectSDCARD PORTB &= ~0x04;  //set the SS to 0 to select the sd card
#define deselectSDCARD PORTB |= 0x04;  //set the SS to 1 to deselect the sd card
#include "WProgram.h"
class SDCARDclass
{
public:
    unsigned char readblock(unsigned long Rstartblock);
    unsigned char writeblock(unsigned long Wstartblock);
private:
    unsigned char SD_reset(void);
    unsigned char SD_sendCommand(unsigned char cmd, unsigned long arg);
    unsigned char SPI_transmit(unsigned char data);
};//end of class SDCARDclass
extern SDCARDclass SDCARD;
#endif
inline void lastblocksave()
{
   unsigned int e = 0;  //the error code from the sd card
   e = SDCARD.writeblock(currentblock);  //save this 256 block of integer data
   while (e != 0)   //cant continue if sd card not working
   {
      Serial.println("writesderror");  //send computer sd card error
      Serial.println(e);  //send computer the error number
      digitalWrite(8, HIGH);  //turn led on to show sd card error
      delay(10000);  //every 10 seconds
   }//end of sd card not working
   currentblock +=1;  //go to the next block in sd card
   EEPROM.write(0,currentblock);  //write the LSB of saved block to EEPROM
   EEPROM.write(1,currentblock >> 8);  //write the next LSB of saved block to EEPROM
   EEPROM.write(2,currentblock >> 16);  //write the next LSB of saved block to EEPROM
   EEPROM.write(3,currentblock >> 24);  //write the MSB of saved block to EEPROM
   ramaddress = 0;   //we can now start again to save samples in RAM
}//end of sd save
Dim st As Stream = File.Open(save_sd, FileMode.Create, FileAccess.Write)
Dim bw As New BinaryWriter(st)  'to send samples to stream
For i = 0 To 255  'all old samples
    bw.Write(sd_samples(i))  'send all the samples
Next  'sector stored in file
bw.Write(date_stamp.ToString("F"))  'add date to file
date_stamp = date_stamp.Add(TimeSpan.FromSeconds(850))
bw.Close()   'sends all samples to file
st.Close()
lastblock = j 'we have uploaded one sector
upload = "uploading"
Invoke(New messdelegate(AddressOf showmessage)) 'show the progress
```

```
Private Sub ArduinoWeather_Load(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles MyBase.Load
    Dim f As New showstart
    f.ShowDialog()  'startup form to connect to arduino
    While My.Computer.FileSystem.FileExists(cd & "\" & last_sector)
        last_sector += 1 'find the next block
    End While
    If My.Computer.FileSystem.FileExists(cd & "\archive") Then
```

```
            Dim st As Stream = File.Open(cd & "\archive", FileMode.Open, FileAccess.Read)
            Dim br As New BinaryReader(st)  'to get samples from stream
            Try
                Dim n As Integer = 0  'a counter
                Do
                    archived(n) = br.ReadInt64()
                    archivedisplay.Items.Add(archived(n))
                    If archived(n) > lastblock Then lastblock = archived(n)
                    n += 1  'we get the largest archive block
                    If n = 100 Then Exit Do 'no more room
                Loop
            Catch ex As Exception  'exception of none left
            Finally
                br.Close()   'must close
                st.Close()
            End Try  'exit try when all read
        End If
        fill_buffers() 'get all the samples into thir buffers
        If overflow.Text = "" Then  'enable displays
            Try
                com8 = My.Computer.Ports.OpenSerialPort("com8", 9600)
                readthread = New Threading.Thread(AddressOf read)
                readthread.Start()  'thread runs for whole program
                                    'to get samples every 10 sec
            Catch ex As Exception
                comdisplay.Text = "no connection" & vbNewLine & _
                                  "or" & vbNewLine & "no communication"
                display_noconnect.Enabled = True  'just use a timer to display
            End Try
        End If
    End Sub
```

## 3. ALGORITMUL FUNCȚIONAL

Unii roboți mobili utilizează cip-ul Atmel AT89C51[1] pe 8 biți cu 4kB de memorie Flash programabilă. Acesta oferă o soluție flexibilă pentru aplicațiile de embedded control. În exemplul nostru robotul urmărește o linie neagră folosind senzori IR.

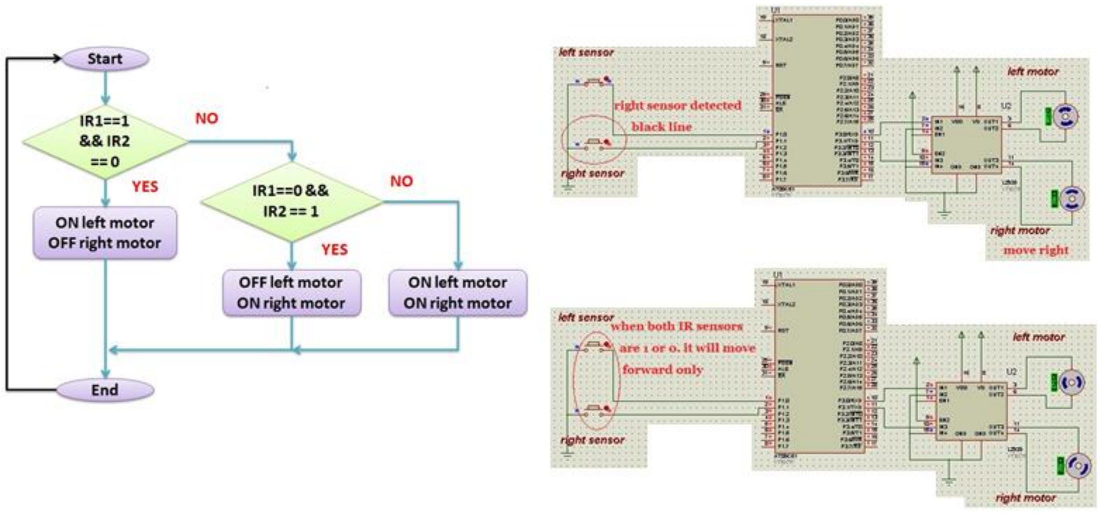
Start
IR1==1 && IR2 ==0
NO
YES
ON left motor OFF right motor
IR1==0 && IR2 == 1
NO
YES
OFF left motor ON right motor
ON left motor ON right motor
End
left sensor
right sensor detected black line
right sensor
left motor
right motor
move right
left sensor
when both IR sensors are 1 or 0, it will move forward only
right sensor
left motor
right motor

Fig. 3-1 Reprezentarea schemei logice pentru programarea în AT89C51

```
// black line follower robot
#include<reg51.h>
sbit IR_left=P1^0;
sbit IR_right=P1^1;
sbit L_motor=P3^0;
sbit R_motor=P3^1;
void main()
{
 L_motor=0;
 R_motor=0;
 //IR_left=0;
 //IR_right=0;
 while(1)
 {
  if(IR_left == 1 && IR_right == 0)
  {
```

[1] http://embedded-electronics.blogspot.ro/p/at89c51-programming.html

```
            L_motor=1;
            R_motor=0;
    }
    else if(IR_right == 1 && IR_left == 0)
    {
            L_motor=0;
            R_motor=1;
    }
    else
    {
            L_motor=1;
            R_motor=1;
            }
   }
  }
```

## 3.1 CODUL SURSĂ

```
/*****************************************************
Project : http://files.spogel.com/miniprojectsin-ece/p-0026--line_follower_robot.pdf
*****************************************************/

//#define debug 1
#include <mega16.h>
#include <delay.h>
#ifdef debug
#include <stdio.h>
#endif

#define
 FWD 0xAA
#define REV 0x55
#define R 0x22
#define L 0x88
```

```
#define CW 0x99
#define CCW 0x66
#define STOP 0x00
#define B 0xFF
#define RSPEED OCR1AL
#define LSPEED OCR1BL
#define SPEED0 255
#define SPEED1 0
#define SPEED2 0
#define SPEED3 0
#define MAX 3
#define HMAX 1

void move (unsigned char dir,unsigned char delay,unsigned char power);
unsigned char i,rdev,ldev,ip,delay,dir,power,dirl,history[MAX],hcount=0,rotpow;

#ifdef debug
unsigned char rep=0,prev=0;
#endif

void
 main(void)
{

// Input/Output Ports initialization
// Port A initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTA=0x00;
DDRA=0x00;

// Port B initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTB=0x00;
```

```
DDRB=0x00;

// Port C initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTC=0x00;
DDRC=0xFF;

// Port D initialization
// Func7=In Func6=In Func5=Out Func4=Out Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=0 State4=0 State3=T State2=T State1=T State0=T
PORTD=0x00;
DDRD=0x30;

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: Timer 0 Stopped
// Mode: Normal top=FFh
// OC0 output: Disconnected
TCCR0=0x00;
TCNT0=0x00;
OCR0=0x00;

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: 921.600 kHz
// Mode: Fast PWM top=00FFh
// OC1A output: Non-Inv.
// OC1B output: Non-Inv.
// Noise Canceler: Off
// Input Capture on Falling Edge
TCCR1A=0xA1;
TCCR1B=0x0A;
TCNT1H=0x00;
TCNT1L=0x00;
```

```
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0x00;
OCR1AL=0xFF;
OCR1BH=0x00;
OCR1BL=0xFF;

// Timer/Counter 2 initialization
// Clock source: System Clock
// Clock value: Timer 2 Stopped
// Mode: Normal top=FFh
// OC2 output: Disconnected
ASSR=0x00;
TCCR2=0x00;
TCNT2=0x00;
OCR2=0x00;

// External Interrupt(s) initialization
// INT0: Off
// INT1: Off
// INT2: Off
MCUCR=0x00;
MCUCSR=0x00;

#ifdef debug
// USART initialization
// Communication Parameters: 8 Data, 1 Stop, No Parity
// USART Receiver: On
// USART Transmitter: On
// USART Mode: Asynchronous
// USART Baud rate: 57600
UCSRA=0x00;
UCSRB=0x18;
UCSRC=0x86;
UBRRH=0x00;
```

```
UBRRL=0x07;
#endif

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x00;

// Analog Comparator initialization
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
ACSR=0x80;
SFIOR=0x00;

while (1){

#ifdef debug
if(rep<255)
rep++;
if(prev!=PINA) {
prev=PINA;
printf("%u\r",rep);
for(i=0;i<8;i++)
printf("%u\t",(prev>>i)&0x01);
rep=0;
}
#endif

if
(PINA!=255){
         rotpow=255;
        ldev=rdev=0;

        if(PINA.3==0)
        rdev=1;
        if(PINA.2==0)
        rdev=2;
```

```
          if(PINA.1==0)
          rdev=3;

          if(PINA.0==0)
          rdev=4;



     if(PINA.4==0)
          ldev=1;

     if(PINA.5==0)
          ldev=2;
          if(PINA.6==0)
          ldev=3;

     if(PINA.7==0)
          ldev=4;

          if(rdev>ldev)
          move(R,0,195+12*rdev);

     if(rdev<ldev)
          move(L,0,195+12*ldev);

     if(rdev==ldev)
          move(FWD,0,200);
          }

     else   {

     for(i=0,dirl=0;i<MAX;i++) {

     if(history[i]==L)
          {dirl++;}
          }
          if(rotpow<160) {rotpow=160;}

if(rotpow<255) {rotpow++;}



if(dirl>HMAX)
        {move(CW,0,rotpow);}
        else
        {move(CCW,0,rotpow);}
        }
};
}

void move (unsigned char dir,unsigned char delay,unsigned char power) {
PORTC=dir;
if(dir==L || dir==R) {
        hcount=(hcount+1)%MAX;
        history[hcount]=dir;
        }
LSPEED=RSPEED=255;
//power;
//delay_ms(delay);
}
```

## 4. CONCLUZII

Lucrarea de față și-a propus realizarea soft-ului de comandă și control al robotului cu destinație Stingerea Incendiilor și Intervenții în Caz de Situații de Urgență. Lucrarea a avut ca suport material și informațional datele și robotul inițial realizat în cadrul.

Rezultatele cercetărilor care au stat la baza prezentei lucrări au constat în:

- identificarea soluției inițiale de robot terestru telecomandat 6x6 comandat cu Arduino;
- refacerea sistemului de comandă și control;
- redimensionarea din punct de vedere al organizării și structurării platformei operaționale;
- introducerea unui sistem de acționare cu braț mobil;
- elaborarea unui cod în limbajul C++ pentru comanda celor 6 motoare brushless atât în modul de lucru simultan cât și individual;
- elaborarea secvenței de comandă pentru senzorii cu care este echipat robotul;
- refacerea codului sursă pentru stația meteo;