UNIVERSITATEA TITU MAIORESCU

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

IA - *Testul de evaluare nr.* 3

Vehicul Terestru fără Pilot (UGV) – Line Follower

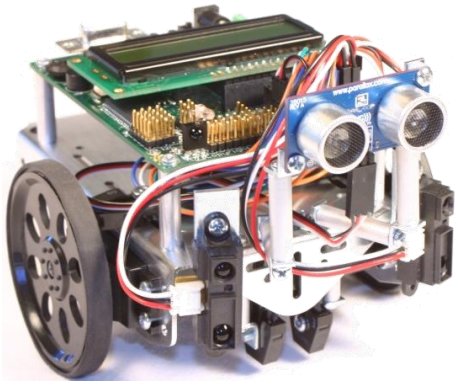
Grupa	Numele și prenumele	Semnătură student	Notă evaluare

Data: ____/____/_____

Conf.dr.ing.
Lucian Ștefăniță GRIGORE

Conf.dr.ing.
Iustin PRIESCU

Ș.L.dr.ing.
Dan-Laurențiu GRECU



Cuprins

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);
}
```

2. SOFTWARE ARDUINO UNO

Software Arduino UNO motor pentru PWM pe pinul 11:

http://arduino.cc/en/Reference/Libraries

```
*/

#include <SoftPWMServo.h>

//Setting up the Hardware pins
// First the line following (IR) sensors
const int irLeft = 2; //Left line sensor is on pin A2
const int irRight = 3; //Right line sensor is on pin A3

//Setting up the Arduino Motor Shield
const int leftDIR = 12;
const int rightDIR = 13;
const int leftPWM = 3;
const int rightPWM = 11;
const int leftBrake = 9;
const int rightBrake = 8;
const char bothSpeed = 100; //sets how fast the motors will spin (0 to 255)

//Here we set up variable that will hold the ADC value representing the line sensor values
int leftSees = 0; //A2 ADC value (0 to 1023)
int rightSees = 0; //A3 ADC value (0 to 1023)

void setup()
{

  //Make sure to set all of our control signal pins as output
  pinMode(leftDIR, OUTPUT);
  pinMode(rightDIR, OUTPUT);
  pinMode(leftBrake, OUTPUT);
  pinMode(rightBrake, OUTPUT);

  //Next we make sure our brake signals are set LOW
  digitalWrite(leftBrake, LOW);
  digitalWrite(rightBrake, LOW);
}

void loop()
{
    //Start by reading the left sensor on A2
  int leftEye = analogRead(irLeft);
  //delay a little bit
  delay(5);
  //next read the right sensor connected A3
  int rightEye = analogRead(irRight);
  //Next, we run the motors based on the sensor reading
  //If both sensors see black (ADC value greater than 1000), then back up
  if ((leftEye >= 1000)&&(rightEye >= 1000)) reverse();
  //Otherwise, if only the left sensor sees black, then turn off the left motor
  //so the robot veer to the left
  else if ((leftEye >= 1000)&&(rightEye < 1000)) turnLeft();
  //Otherwise, if only the right sensor sees black, then turn off the right motor
  //so the robot veer to the right
  else if ((leftEye < 1000)&&(rightEye >= 1000)) turnRight();
  //Otherwise, move forward
  else forward();
}
```

```
//Turn right by turning off the right motor
//i.e disable the PWM to that wheel
void turnRight(void)
{
  digitalWrite(leftDIR, HIGH);
  digitalWrite(rightDIR, HIGH);
  SoftPWMServoPWMWrite(leftPWM, bothSpeed);
  SoftPWMServoPWMWrite(rightPWM, 0);
}
//Turn left by turning off the left motor
//i.e disable the PWM to that wheel
void turnLeft(void)
{
  digitalWrite(leftDIR, HIGH);
  digitalWrite(rightDIR, HIGH);
  SoftPWMServoPWMWrite(leftPWM, 0);
  SoftPWMServoPWMWrite(rightPWM, bothSpeed);
}
//Move forward by enabling both wheels
void forward(void)
{
  digitalWrite(leftDIR, HIGH);
  digitalWrite(rightDIR, HIGH);
  SoftPWMServoPWMWrite(leftPWM, bothSpeed);
  SoftPWMServoPWMWrite(rightPWM, bothSpeed);
}

//Reverse by enabling both wheels
void reverse(void)
{
  digitalWrite(leftDIR, LOW);
  digitalWrite(rightDIR, LOW);
  SoftPWMServoPWMWrite(leftPWM, bothSpeed);
  SoftPWMServoPWMWrite(rightPWM, bothSpeed);
}
```

Un exemplu foarte răspândit este acela care utilizează cip-ul Atmel AT89C51 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.

```
// 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)
  {
            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;
            }
 }
}
```

Mai departe este prezentat modulul de programare al PID-ului:

```
...
unsigned char MaxSpeed;                    // Hold Motor Maximum Speed
unsigned int Kp,Ki,Kd;                     // PID Parameters
int prev_res=0, prev_err_1=0, prev_err_2=0;     // PID Control Variables

int motor_res,err_func;
float KP,KI,KD,cont_res;
// Get the Error Function
err_func=sensor_val - TARGET_VAL;
// Divide all the PID parameters for decimal value
KP=Kp * 0.1;
KI=Ki * 0.01;
KD=Kd * 0.01;
// Calculate the Motor Response using PID Control Equation
cont_res=(float)(prev_res + KP * (err_func - prev_err_1) + KI * (err_func + prev_err_1)/2.0
          + KD * (err_func - 2.0 * prev_err_1 + prev_err_2));

// Now we have to Limit the control response to the Maximum of our motor PWM Motor Value
motor_res=(int)cont_res;
if (motor_res > MaxSpeed)
  motor_res = MaxSpeed;
if (motor_res < -MaxSpeed)
  motor_res = -MaxSpeed;
// Save the Motor Response and Error Function Result
prev_res=motor_res;
prev_err_2=prev_err_1;
prev_err_1=err_func;
```

3. SISTEMUL SENZORIAL

Senzorii IR (model QRE1113) funcționează cel mai bine atunci când sunt aproape de suprafața de contact, distanța optimă fiind de aproximativ 3,175 [mm].

```
#include <RedBot.h>
RedBotSensor IRSensor1 = RedBotSensor(A3); // initialize a sensor object on A3
RedBotSensor IRSensor2 = RedBotSensor(A6); // initialize a sensor object on A6
RedBotSensor IRSensor3 = RedBotSensor(A7); // initialize a sensor object on A7

void setup()
{
  Serial.begin(9600);
  Serial.println("Welcome to experiment 6!");
  Serial.println("------------------------");
}

void loop()
{
  Serial.print("IR Sensor Readings: ");
  Serial.print(IRSensor1.read());
  Serial.print("\t");  // tab character
  Serial.print(IRSensor2.read());
  Serial.print("\t");  // tab character
  Serial.print(IRSensor3.read());
  Serial.println();
  delay(100);
}
```

Senzorul infraroșu QRB1114

```
// Bot builder BoxBot line follower code for the Photoreflector sensor bar calibration.
int LeftOpto = 0; // Create a variable called LeftOpto on analog pin 0 so we can read the value of the Left Photosensor.
int RightOpto = 1; // Create a variable called RightOpto on analog Pin 1 so we can read the value of the Right Photosensor.
int lval = 0; // Initialise a variable called lval, this is used to store the Left sensors value, set it to 0.
int rval = 0; // Initialise a variable called rval, this is used to store the Left sensors value, set it to 0.
void setup() {
                Serial.begin(9600); // Start the serial service @ 9600 baud rate.
}

void loop() {
  lval = analogRead(LeftOpto); // Read the value incoming from the photoreflector on pin 0 and assign it to the variable lval.
                Serial.print("Left");
                Serial.println(lval); // Print to the serial port the lval value so you can see what it is.
  rval = analogRead(RightOpto); // Read the value incoming from the photoreflector on pin 1 and assign it to the variable rval.
                Serial.print("Right:");
                Serial.println(rval); // Print to the serial port the rval value so you can see what it is.
   }
delay(1000); // Delay 1 second on each read so you have a chance to see the value.
}
int LeftOpto = 0; // Create a variable called LeftOpto on analog pin 0 so we can read the value of the Left Photosensor.
int RightOpto = 1; // Create a variable called RightOpto on analog Pin 1 so we can read the value of the Right Photosensor.
int TurnLED = 13; // We need to declare that we are using pin 13 for our turn indicator LED.
int lval = 0; // Initialise a variable called lval, this is used to store the Left sensors value, set it to 0.
int rval = 0; // Initialise a variable called rval, this is used to store the Left sensors value, set it to 0.
#include <Servo.h> // Include the servo library, so we can easily use servos.
Servo LeftServo; // Tell the library that we want to create a servo called LeftServo.
Servo RightServo; // Tell the library that we want to create a servo called RightServo.
void setup() {
  pinMode(TurnLED, OUTPUT); // Set the turn LED pin to be an output (so we can turn it on and off).
  LeftServo.attach(9); // Tell the servo object to attach the LeftServo to Pin 9.
  RightServo.attach(10); // Tell the servo object to attach the LeftServo to Pin 10.
}
void loop() {
  lval = analogRead(LeftOpto); // Read the value incoming from the photoreflector on pin 0 and assign it to the variable lval.
    if (lval > 550) { // If lval is greater that the threshold of what we determined the black line is - ie. we are on white.
     digitalWrite(TurnLED, HIGH); // Turn on the Turn LED on pin 13.
     LeftServo.write(90); // Stop the left servo by centering it.
   } else {
```

```
    digitalWrite(TurnLED, LOW); // Turn off the turn LED on pin 13.
    LeftServo.write(120); // Drive the left servo forward.
  }
 rval = analogRead(RightOpto); // Read the value incoming from the photoreflector on pin 1 and assign it to the variable rval.
 if (rval > 550) { // If rval is greater that the threshold of what we determined the black line is - ie. we are on white.
   digitalWrite(TurnLED, HIGH); // Turn on the Turn LED on pin 13.
   RightServo.write(90); // Stop the right servo by centering it.
  } else {
   digitalWrite(TurnLED, LOW); // Turn off the turn LED on pin 13.
   RightServo.write(50); // Drive the right servo in reverse (makes the bot go forward, as the servos are inverted).
  }
delay(200); // Delay 200 microseconds, just to smooth out any jerky movements.
}
```

Giroscop triaxial

```
//Arduino 1.0+ only

#include <Wire.h>

#define CTRL_REG1 0x20
#define CTRL_REG2 0x21
#define CTRL_REG3 0x22
#define CTRL_REG4 0x23
#define CTRL_REG5 0x24

int L3G4200D_Address = 105; //I2C address of the L3G4200D

int x;
int y;
int z;

void setup(){

  Wire.begin();
  Serial.begin(9600);

  Serial.println("starting up L3G4200D");
  setupL3G4200D(2000); // Configure L3G4200  - 250, 500 or 2000 deg/sec

  delay(1500); //wait for the sensor to be ready
}

void loop(){
   getGyroValues();  // This will update x, y, and z with new values

  Serial.print("X:");
  Serial.print(x);

  Serial.print(" Y:");
  Serial.print(y);

  Serial.print(" Z:");
  Serial.println(z);

  delay(100); //Just here to slow down the serial to make it more readable
}

void getGyroValues(){

  byte xMSB = readRegister(L3G4200D_Address, 0x29);
```

```
  byte xLSB = readRegister(L3G4200D_Address, 0x28);
  x = ((xMSB << 8) | xLSB);

  byte yMSB = readRegister(L3G4200D_Address, 0x2B);
  byte yLSB = readRegister(L3G4200D_Address, 0x2A);
  y = ((yMSB << 8) | yLSB);

  byte zMSB = readRegister(L3G4200D_Address, 0x2D);
  byte zLSB = readRegister(L3G4200D_Address, 0x2C);
  z = ((zMSB << 8) | zLSB);
}

int setupL3G4200D(int scale){
  //From  Jim Lindblom of Sparkfun's code

  // Enable x, y, z and turn off power down:
  writeRegister(L3G4200D_Address, CTRL_REG1, 0b00001111);

  // If you'd like to adjust/use the HPF, you can edit the line below to configure CTRL_REG2:
  writeRegister(L3G4200D_Address, 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(L3G4200D_Address, CTRL_REG3, 0b00001000);

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

  if(scale == 250){
    writeRegister(L3G4200D_Address, CTRL_REG4, 0b00000000);
  }else if(scale == 500){
    writeRegister(L3G4200D_Address, CTRL_REG4, 0b00010000);
  }else{
    writeRegister(L3G4200D_Address, CTRL_REG4, 0b00110000);
  }

  // CTRL_REG5 controls high-pass filtering of outputs, use it
  // if you'd like:
  writeRegister(L3G4200D_Address, CTRL_REG5, 0b00000000);
}

void writeRegister(int deviceAddress, byte address, byte val) {
    Wire.beginTransmission(deviceAddress); // start transmission to device
    Wire.write(address);       // send register address
    Wire.write(val);         // send value to write
    Wire.endTransmission();     // end transmission
}

int readRegister(int deviceAddress, byte address){

    int v;
    Wire.beginTransmission(deviceAddress);
    Wire.write(address); // register to read
    Wire.endTransmission();

    Wire.requestFrom(deviceAddress, 1); // read a byte

    while(!Wire.available()) {
        // waiting
```

```
    }

    v = Wire.read();
    return v;
}
```

4. LINE FOLLOWING cu SENZOR IR

```
#include <RedBot.h>
RedBotSensor left = RedBotSensor(A3);   // initialize a left sensor object on A3
RedBotSensor center = RedBotSensor(A6); // initialize a center sensor object on A6
RedBotSensor right = RedBotSensor(A7);  // initialize a right sensor object on A7
// constants that are used in the code. LINETHRESHOLD is the level to detect
// if the sensor is on the line or not. If the sensor value is greater than this
// the sensor is above a DARK line.
//
// SPEED sets the nominal speed

#define LINETHRESHOLD 800
#define SPEED 60  // sets the nominal speed. Set to any number from 0 - 255.

RedBotMotors motors;
int leftSpeed;   // variable used to store the leftMotor speed
int rightSpeed;  // variable used to store the rightMotor speed

void setup()
{
   Serial.begin(9600);
   Serial.println("Welcome to experiment 6.2 - Line Following");
   Serial.println("-----------------------------------------");
   delay(2000);
   Serial.println("IR Sensor Readings: ");
   delay(500);
}

void loop()
{
   Serial.print(left.read());
   Serial.print("\t");  // tab character
   Serial.print(center.read());
   Serial.print("\t");  // tab character
   Serial.print(right.read());
   Serial.println();

   // if on the line drive left and right at the same speed (left is CCW / right is CW)
   if(center.read() > LINETHRESHOLD)
   {
      leftSpeed = -SPEED;
      rightSpeed = SPEED;
   }

   // if the line is under the right sensor, adjust relative speeds to turn to the right
   else if(right.read() > LINETHRESHOLD)
   {
      leftSpeed = -(SPEED + 50);
      rightSpeed = SPEED - 50;
   }

   // if the line is under the left sensor, adjust relative speeds to turn to the left
   else if(left.read() > LINETHRESHOLD)
   {
      leftSpeed = -(SPEED - 50);
      rightSpeed = SPEED + 50;
   }
```

```
    // if all sensors are on black or up in the air, stop the motors.
    // otherwise, run motors given the control speeds above.
    if((left.read() > LINETHRESHOLD) && (center.read() > LINETHRESHOLD) && (right.read() > LINETHRESHOLD) )
    {
      motors.stop();
    }
    else
    {
      motors.leftMotor(leftSpeed);
      motors.rightMotor(rightSpeed);

    }
    delay(0);  // add a delay to decrease sensitivity.
}
```

==========

```
#include <QTRSensors.h>

void calculateIntegral();
void calculateProportional();
void readValues();

#define NUM_SENSORS 8
#define TIMEOUT 2500
#define EMITTER_PIN 0
#define avgSpeed 255

int time = 0;

int pwmA = 3;
int pwmB = 11;
int dirA = 12;
int dirB = 13;

int kp = 1;
int kd = 1;
int ki = 1;

int error = 0;
int lastError = 0;

int proportional = 0;
int derivative = 0;
int integral = 0;

QTRSensorsRC qtrrc((unsigned char[]) {2, 4, 5, 6, 7, 8, 9, 10},
  NUM_SENSORS, TIMEOUT, EMITTER_PIN);
unsigned int sensorValues[NUM_SENSORS];

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

  pinMode(pwmA, OUTPUT);
  pinMode(pwmB, OUTPUT);
  pinMode(dirA, OUTPUT);
  pinMode(dirB, OUTPUT);
```

```
  pinMode(1, OUTPUT);
  for (int i=0; i<5; i++){
     digitalWrite(1, HIGH);
     delay(50);
     digitalWrite(1, LOW);
     delay(950);
    }

  analogWrite(pwmA, avgSpeed);
  analogWrite(pwmB, avgSpeed);

}

void loop(){

  readValues();
  calculateProportional();

  derivative = error-lastError;
  integral += proportional;

  if(integral > 255){
    integral=255;
  };
  if(integral < -255){
    integral=-255;
  };

  int turn = proportional*kp + derivative*kd + integral*ki;

  if(turn>=255)
    turn=255;
  if(turn<=-255)
    turn=-255;

  int speedA=0;
  int speedB=0;

  if(turn>=0){
    speedA=avgSpeed;
    speedB=avgSpeed-turn;
  }
  else{
    speedA=avgSpeed+turn;
    speedB=avgSpeed;
  }

  Serial.print("P=");
  Serial.print(proportional);
  Serial.print('\t');
  Serial.print("I=");
  Serial.print(integral);
  Serial.print('\t');
  Serial.print("D=");
  Serial.print(derivative);
  Serial.print('\t');
  Serial.print("Turn=");
  Serial.print(turn);
  Serial.print('\t');


Serial.print("speedA=");
```

```
  Serial.print(speedA);
  Serial.print('\t');
  Serial.print("speedB=");
  Serial.print(speedB);
  Serial.print('\t');

  analogWrite(pwmA, speedA);
  analogWrite(pwmB, speedB);

  lastError=error;

  Serial.println();
}

void readValues(){
  qtrrc.read(sensorValues);
  for (int i=0; i<NUM_SENSORS; i++){
//    Serial.print(sensorValues[i]);
//    Serial.print('\t');
    if(sensorValues[i]>400)
      sensorValues[i]=1;
      else
      sensorValues[i]=0;
  }
}

void calculateProportional(){
   int sum = 0;
   int posLeft = 10;
   int posRight = 10;

   for (int i=0; i<NUM_SENSORS/2; i++){
     sum=sum+sensorValues[i];
     if(sensorValues[i]==1){
       posRight=i-3;
     }
   }
   for (int i=NUM_SENSORS-1; i>=NUM_SENSORS/2; i--){
     sum=sum+sensorValues[i];
     if(sensorValues[i]==1){
       posLeft=i-4;
     }
   }

   if(sum>=3){
     sum=2;
   }

   if(sum==0){
     if(lastError<0){
       error=-8;
     }
     else{
       error=8;
     }
   }
   else if((posLeft!=10)&&(posRight!=10)){
     error=0;
   }
     else if(posLeft!=10){
       error=posLeft*2+sum;
     }
```

```
        else if(posRight!=10){
          error=posRight*2-sum;
        }

    proportional = error;

}
```

5. MODELARE PROGRAM LINE FOLLOWING ROBOT - SIMULINK

Modelarea în Simulink a comportamentului cinematic și dinamic al unui robot de linie include următoarele ecuații care descriu cinematica și dinamica robotului.

$$\begin{cases} V_L = V - \frac{\omega \cdot L}{2} \\ V_R = V + \frac{\omega \cdot L}{2} \end{cases} \quad 5.1$$

unde: V – viteza de mers înainte $[^m/_s]$; V_R – viteza de rotație a roții din dreapta $[rpm]$; V_L – viteza de rotație a roții din stânga $[rpm]$; L – axului punții motrice [mm]; ω – viteza de rotație a motoarelor $\left[^{rad}/_s\right]$.

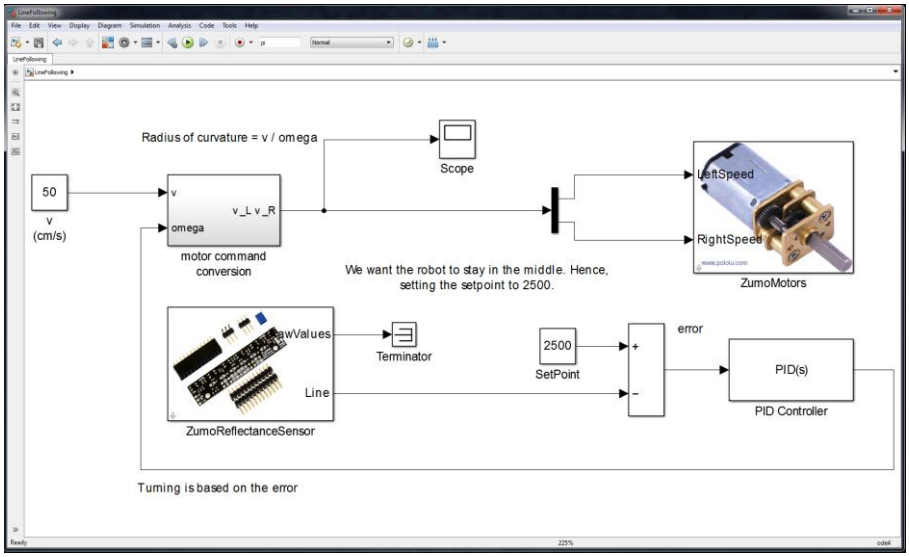


Fig. 5-1 Exemplu de folder Motor în Simulink

Modelul acceptă viteza „V“ și unghiul de rotire „ω“, ca variabile pentru cele două intrări privind roata din stânga și cea din dreapta. Subsistemul care se ocupă cu conversia celor două mărimi de intrare V și ω, implementează sistemul de relații matematice:

$$\begin{cases} v\_L = v - omega * axleLenght/2 \\ v\_R = v + omega * axleLenght/2 \end{cases} \quad 5.2$$

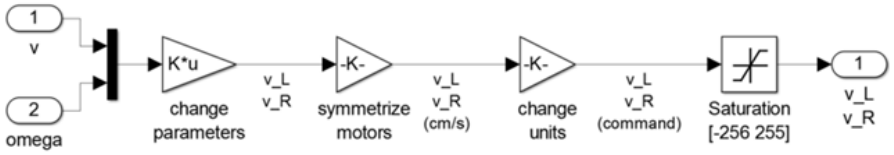


Fig. 5-2 Schema bloc – modul conversie semnale intrare

- Urmează descrierea primului bloc „change parameters” care efectuează operația de multiplicare.
- Al doilea bloc „symmetrize motors”, care multiplică turațiile celor două motoare, astfel încât, robotul să se poată deplasa rectiliniu, din construcție vitezele unghiulare ale motoarelor au fluctuații în jurul unei valori de referință.
- Al treilea bloc „change units” efectuează conversia unităților de măsură.

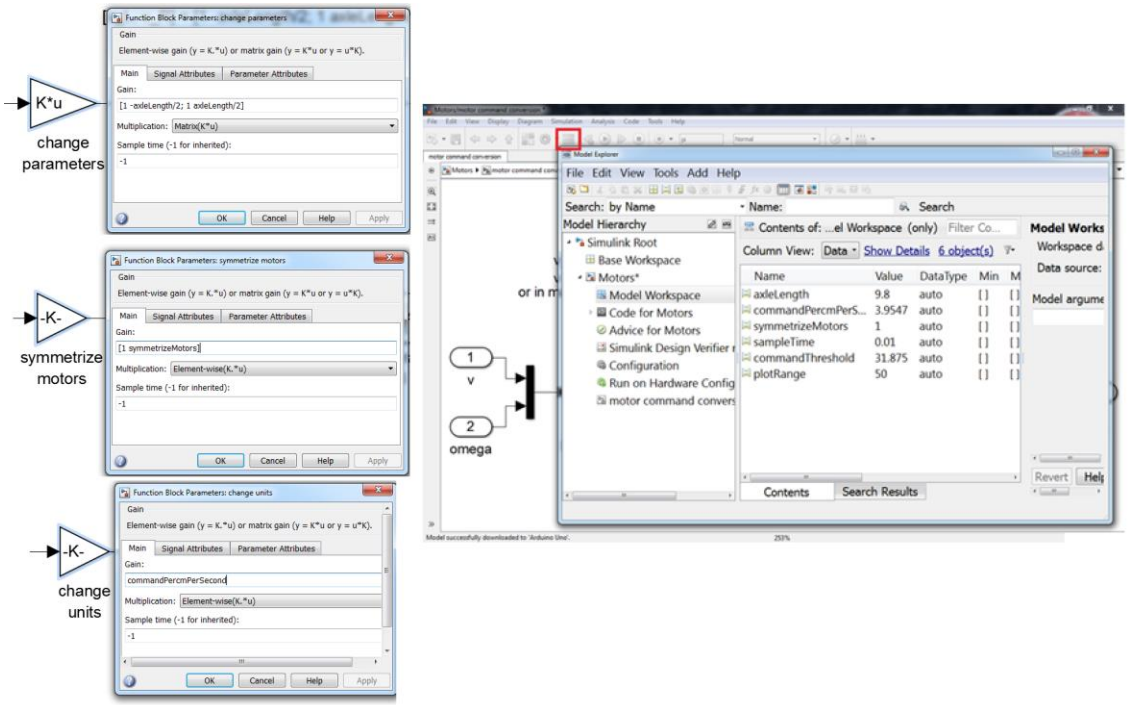


Fig. 5-3 Blocuri de comandă

Întrucât parametri descriși mai sus diferă de la Robot la Robot, valorile implicite cu care sunt livrate kit-urile de roboți de linie este bine să fie ajustate de fiecare dată.

Utilizarea contrastului dintre o linie neagră și un teren alb este benefic, întrucât lina de culoare neagră reflectă mai puțină lumină decât culoarea albă. Cei 6 senzori (în funcție de tipul de robot) permit determinarea poziției robotului față de linia neagră. Senzorii întorc o valoare cuprinsă între $\{0 \div 5000\}$.

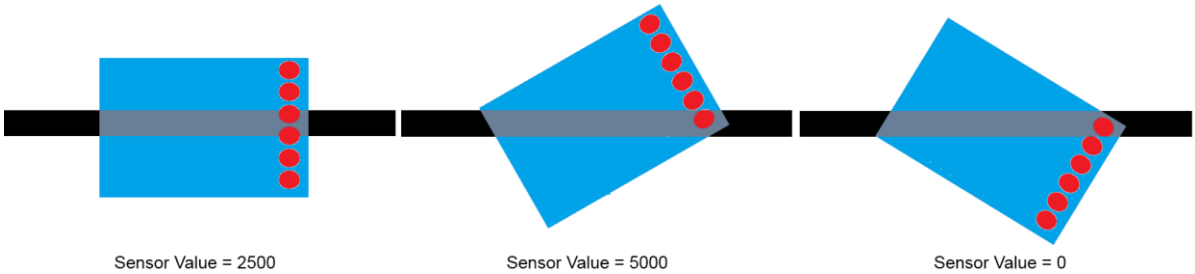


Fig. 5-4 Poziția relativă a robotului detectată de senzorii de linie, față de o linie de culoare neagră

Se consideră valoarea de 2.500 ca fiind reprezentarea poziției LFR față de linia neagră. Orice valoare mai mică sau mai mare decât 2.500 reflectă poziției LFR în stânga , respectiv dreapta liniei negre. Diferențele dintre valoarea de referință și cele reale se numesc erori, sau semnale de eroare.

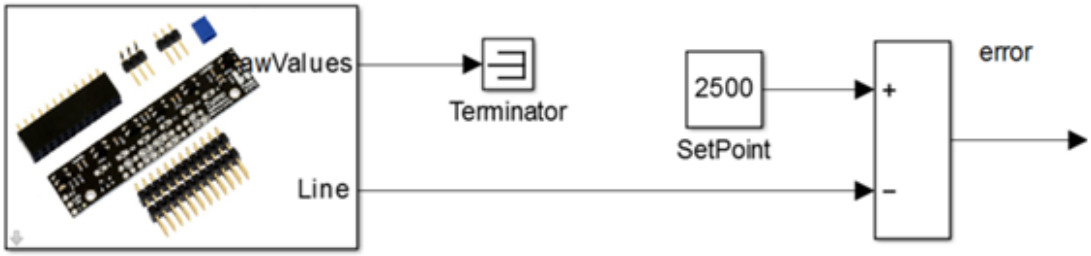


Fig. 5-5 Diagramele blocurilor responsabile de simularea determinării poziției robotului

Semnalul de eroare odată determinat (prin însumarea la valoarea de 2.500), putem trece la repoziționarea LFR.

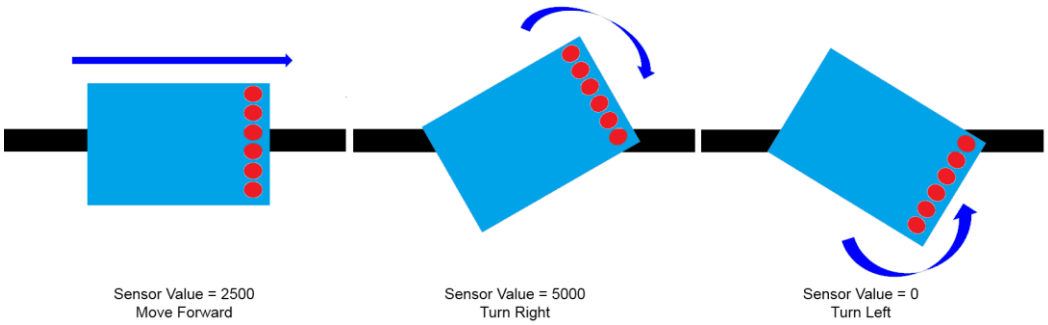


Fig. 5-6 Repoziționarea robotului în funcție de informațiile primite de la senzori.

Programul cel mai simplu presupune ca robotul să se întoarcă mereu la centrul liniei negre de pe podea. În funcție de constanta de proporționalitate, suma cu care robotul transformă pentru o anumită valoare a semnalului de eroare, va fi variabilă. Această constantă de proporționalitate „P“ este un parametru al modelului, care trebuie să fie reglat în funcție de cerințele beneficiarului.

Limitările acestui tip de PID sunt date de constrângerile de a sta pe centrul liniei negre. Acestea introduc în sistem oscilații datorate inerției. Aceasta este explicația faptului că LFR oscilează în mod continuu în jurul axului liniei negre. Componenta derivativă a PID-ului ajută la amortizarea oscilațiilor. În acest caz, vom calcula derivata instantanee a semnalului de eroare care ar servi pentru a amortiza oscilațiile. Astfel, constanta este multiplicată cu rezultatul derivării și produsul este adăugat la semnalul de control. Această constantă este cunoscută sub numele de câștig derivat și corespunde componentei „D“ a controlerului PID.

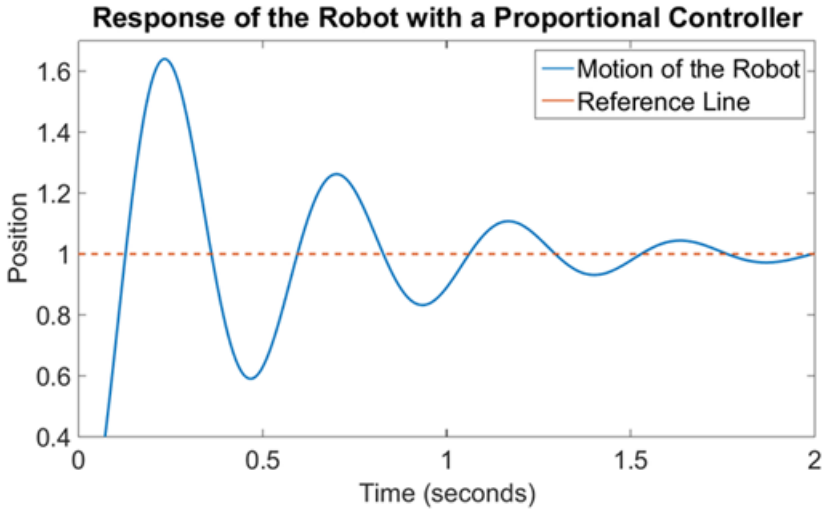


Fig. 5-7 Graficul răspunsului proporțional al PID-ului.

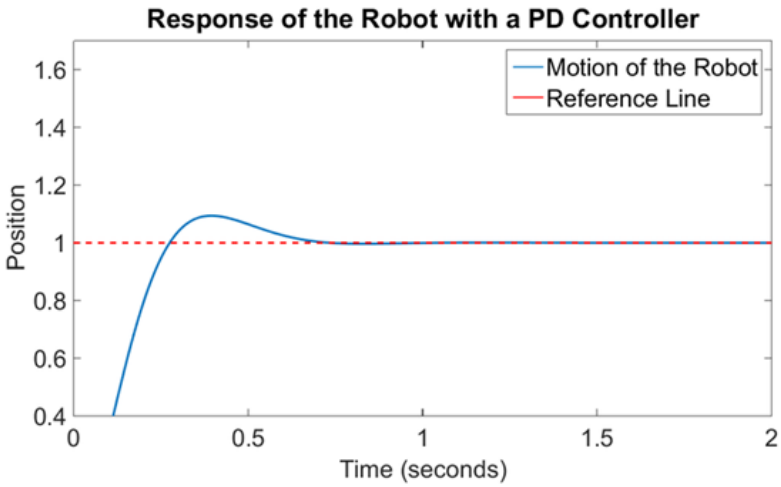


Fig. 5-8 Graficul răspunsului proporțional/derivativ al PID-ului.

În exemplul descris în lucrare componenta de Integrare „I“ nu a fost utilizată. Astfel în mod specific, acest controler este cunoscut ca un controler PD.

În Fig. 5-9, blocul etichetat „PID controller“ reprezintă controlerul descris mai sus. Se pot observa valorile pentru „P“, „I“, „D“ printr-un dublu clic pe exemplu, modelul Block. Avem valorile implicite pentru „P“ și „D“ în timp ce, pentru acest exemplu, „I=0“.

Controlerul PD se poate regla și fără a utiliza un algoritm matematic, după cum urmează:

- se reglează câștigul proporțională a controlerului până când se obține un răspuns adecvat;
- de regulă, se va seta valoarea câștigului derivatului cu 1/10 din câștigul proporțional;
- apoi se va ajusta câștigul derivat până în momentul în care se observă o schimbare adecvată a comportamentului robotului;
- creșterea datorată componentei „P“ proporțională va crește cantitatea cu care se transformă robotul
- creșterea câștigului derivat va amortiza magnitudinea oscilațiilor robotului.

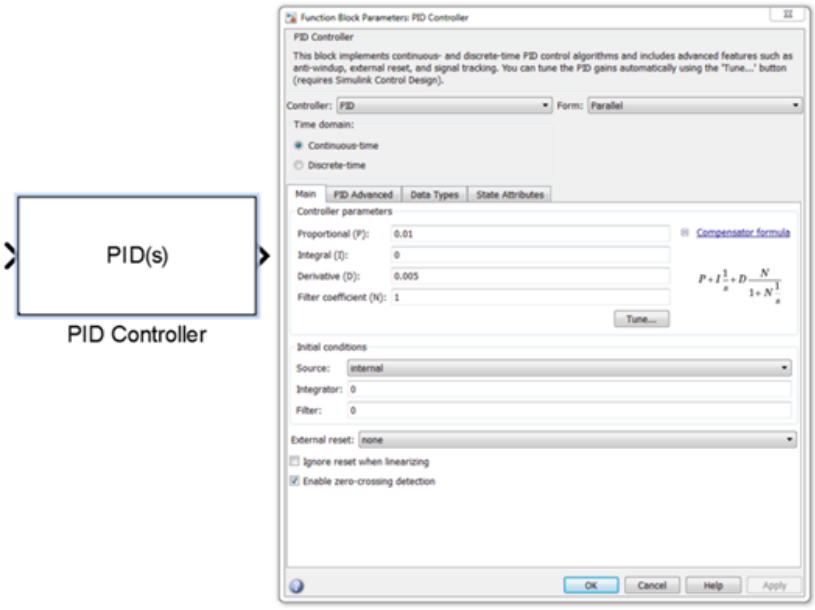


Fig. 5-9 Funcția bloc din toolbox-ul Simulink pentru un controler PID.

Decizia unui robot mobil de a se orienta spre stânga sau spre dreapta este una foarte importantă în logica de funcționare. Înainte de testarea software se verifică dacă shield-urile MEGA sunt funcționale, ce tip de rezistor SM avem. De aceea producătorii de ARDUINO au implementat un cod de la AMTEL pentru acest lucru.

```
--- TEST START ---
TEST 001 - Any stuck pins?
VoltsRead = 2.51 -- OK

TEST 002 - Digital Pins Source Current
PIN-3 PIN-4 PIN-5 PIN-6 PIN-7 PIN-8 PIN-9 PIN-10 PIN-11 PIN-12
2.49 2.48 2.49 2.48 2.48 2.48 2.48 2.49 2.49 2.48
-OK- -OK- -OK- -OK- -OK- -OK- -OK- -OK- -OK- -OK-
TEST 003 - Digital Pins Sink Current
PIN-3 PIN-4 PIN-5 PIN-6 PIN-7 PIN-8 PIN-9 PIN-10 PIN-11 PIN-12
2.48 2.48 2.48 2.48 2.48 2.49 2.49 2.48 2.48 2.48
-OK- -OK- -OK- -OK- -OK- -OK- -OK- -OK- -OK- -OK-

TEST 004 - Analog Pins Source Current
PIN-1 PIN-2 PIN-3 PIN-4 PIN-5
0.00 0.00 0.00 0.00 0.00
-OK- -OK- -OK- -OK- -OK-
TEST 005 - Analog Pins Sink Current
PIN-1 PIN-2 PIN-3 PIN-4 PIN-5
4.99   4.99   4.99   4.99   4.99
-OK- -OK- -OK- -OK- FAIL

TEST 006 - Analog Pins A0 to A5: A to D (About 2.50 V)
PIN-0 PIN-1 PIN-2 PIN-3 PIN-4 PIN-5
2.10 2.26 2.47 2.48 2.47 2.48

TEST 007 - Analog Pins A0 to A5: A to D (About 3.33 V)
PIN-0 PIN-1 PIN-2 PIN-3 PIN-4 PIN-5
2.41 2.39 3.31 3.31 3.31 3.31

TEST 008 - Analog Pins A0 to A5: A to D (About 1.66 V)
PIN-0 PIN-1 PIN-2 PIN-3 PIN-4 PIN-5
2.08 2.17 1.64 1.64 1.64 1.64
TEST COMPLETE
THERE WERE 10 FAILURES
```