Line Follower

Introducere

Ce face

Robotul meu linefollower este un dispozitiv autonom care urmărește linii trasate pe suprafețe prin intermediul unei matrice de 5 senzori infraroșu. Folosind placa Arduino UNO ca unitate centrală de procesare, robotul detectează linia de sub el și controlează două motoare DC prin intermediul driverului L298N pentru a se menține pe traseu. Construcția sa robustă, cu șasiul imprimat 3D personalizat, îi conferă durabilitate, în timp ce utilizarea chederului lipit pe roți asigură o aderență sporită la suprafață, prevenind alunecarea pe parcursul traseelor.

Care este scopul lui

Scopul principal al acestui robot este să demonstreze principiile de bază ale roboticii autonome și să ofere o platformă educațională pentru înțelegerea controlului de mișcare bazat pe feedback-ul senzorilor. Robotul este proiectat să parcurgă cu precizie trasee complexe la viteze optime, fiind capabil să navigheze prin curbe, intersecții și segmente drepte fără intervenție umană.

Care a fost ideea de la care am pornit

Ideea inițială a proiectului a pornit de la dorința mea de a crea un robot accesibil ca nivel de complexitate, dar suficient de performant pentru a fi competitiv. Am ales configurația cu 5 senzori IR pentru a avea un echilibru între precizie și simplitate, iar pentru alimentare am implementat soluția cu două baterii de 9V conectate pentru a asigura o capacitate energetică sporită și autonomie extinsă. Șasiul imprimat 3D l-am proiectat special pentru acest robot, optimizând distribuția greutății și poziționarea componentelor pentru stabilitate maximă, iar adăugarea chederului pe roți a venit ca soluție inovatoare pentru îmbunătățirea tracțiunii pe orice tip de suprafață.

De ce cred că este util pentru alții și pentru mine

Pentru mine, acest proiect reprezintă o oportunitate excelentă de a aplica practic cunoștințele de programare, electronică și design 3D într-un sistem integrat. Procesul de optimizare a codului pentru a face robotul mai eficient și mai precis mi-a dezvoltat abilitățile de rezolvare a problemelor și gândirea analitică.

Pentru alții, robotul meu oferă:

- O platformă educațională ideală pentru începători în robotică și electronică
- Un exemplu practic de implementare a sistemelor de control în buclă închisă
- O bază solidă care poate fi adaptată și extinsă pentru proiecte mai complexe
- Inspirație pentru explorarea designului 3D în aplicații practice
- Un model funcțional care demonstrează cum soluțiile simple (precum utilizarea chederului pentru aderență) pot îmbunătăți semnificativ performanța

Descriere generală

Arduino UNO

Descriere: Microcontroler bazat pe ATmega328P care servește ca unitate centrală de procesare. Interacțiune: Primește date de la senzorii IR prin conexiuni GPIO (D2-D4, D7, D12), procesează informația și transmite comenzi de direcție prin GPIO (D8-D11) și viteză prin PWM (D5, D6) către driverul de motor. Comunică cu PC-ul prin UART (D0, D1) pentru debugging. Primește alimentare de 5V de la driverul L298N.

Driver Motor L298N

Descriere: Driver dual H-bridge pentru controlul independent al celor două motoare DC. Interacțiune: Primește semnale de control logic (IN1-IN4) și PWM (ENA, ENB) de la Arduino pentru a controla direcția și viteza motoarelor. Furnizează tensiune de 5V către Arduino și transformă semnalele logice în curenți de putere pentru motoare prin ieșirile OUT1-OUT4. Este alimentat cu 12V de la baterii.

Matrice IR cu 5 Canale

Descriere: Set de 5 senzori infraroșu pentru detectarea liniei negre pe fundal deschis. Interacțiune: Trimite semnale digitale către Arduino (OUT1-OUT5) indicând poziția liniei față de robot. Primește alimentare de 5V de la Arduino.

Motoare DC (2)

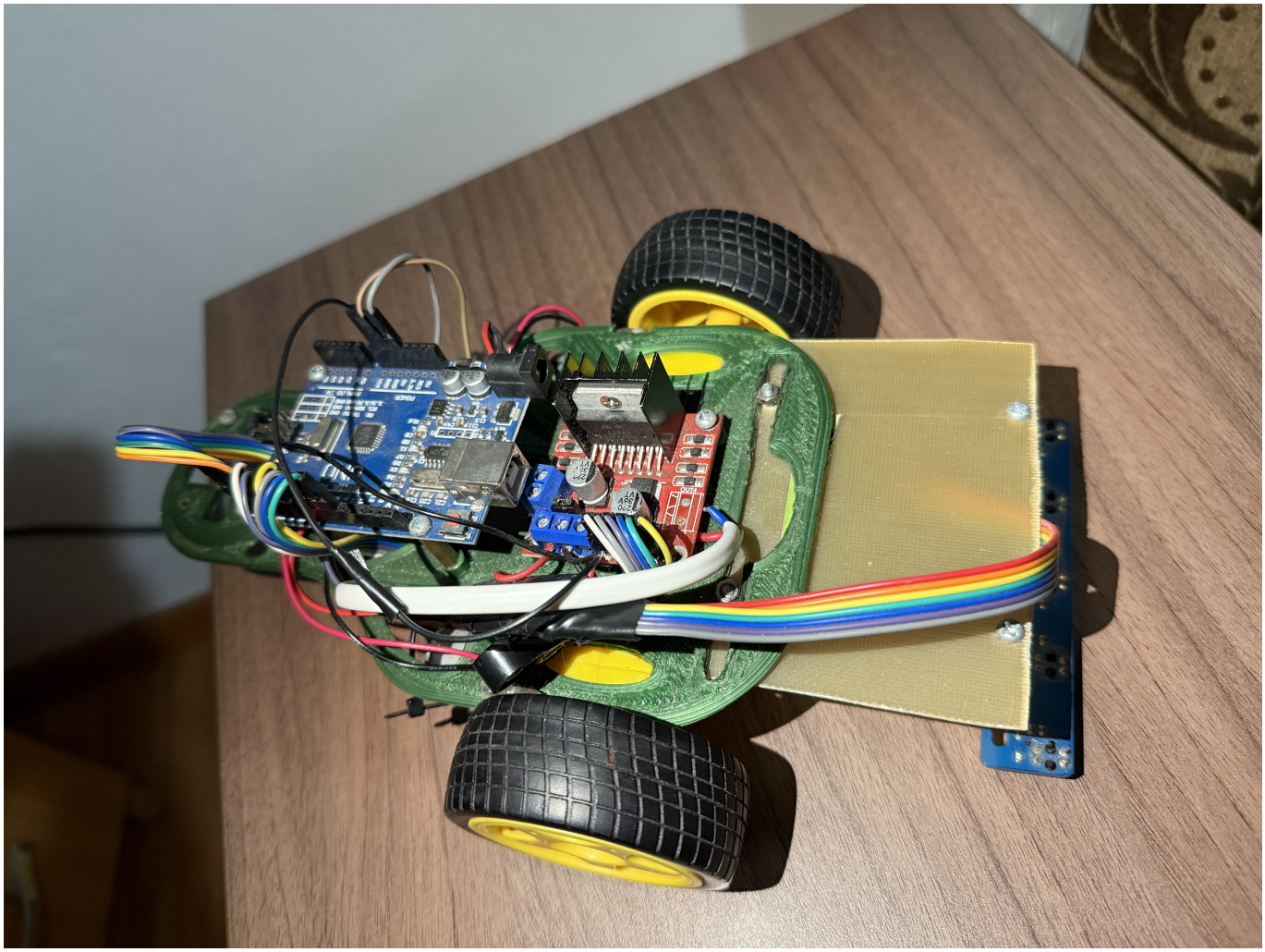
Descriere: Motoare de curent continuu pentru propulsia robotului. Interacțiune: Primesc semnale de alimentare și control de la driverul L298N prin ieșirile OUT1-OUT4, transformând semnalele electrice în mișcare mecanică. Sunt echipate cu chedere lipite pe roți pentru aderență îmbunătățită. Baterii 2x9V Descriere: Sursă de alimentare pentru întregul sistem, oferind autonomie extinsă. Interacțiune: Furnizează 12V către driverul L298N pentru alimentarea motoarelor și 5V pentru circuitele logice.

PC

Descriere: Computer utilizat pentru debugging și monitorizare. Interacțiune: Comunică cu Arduino prin conexiune UART (serial) pentru a primi date de diagnostic și a trimite comenzi de test.

Șasiu Imprimat 3D

Descriere: Structură fizică a robotului realizată prin imprimare 3D. Interacțiune: Integrează toate modulele electronic într-un design compact și robust, permițând poziționarea optimă a senzorilor IR față de suprafață. Sistemul funcționează în buclă închisă: senzorii IR detectează poziția liniei, Arduino procesează aceste date și calculează corecțiile necesare, apoi trimite comenzi către driverul motor pentru a ajusta direcția și viteza robotului, menținându-l astfel pe traseul dorit.



Design Hardware

Listă de Componente

Arduino UNO

- **Descriere**: O placă microcontroler bazată pe ATmega328P.
- **Pini**: UNUSED, IOREF, Reset, 3.3V, 5V, GND, Vin, A0, A1, A2, A3, A4, A5, SCL, SDA, AREF, D13, D12, D11, D10, D9, D8, D7, D6, D5, D4, D3, D2, D1, D0.

Driver de Motor DC L298N

- **Descriere**: Un driver de motor cu punte H dublă care permite controlul a două motoare DC.
- **Pini**: OUT1, OUT2, 12V, GND, 5V, OUT3, OUT4, 5V-ENA-JMP-I, 5V-ENA-JMP-O, +5V-J1, +5V-J2, ENA, IN1, IN2, IN3, IN4, ENB.

Motor DC (x2)

- **Descriere**: Motoare DC standard utilizate pentru propulsia robotului.
- **Pini**: pin 1, pin 2.

Matrice IR cu 5 Canale

- **Descriere**: O matrice de senzori infraroșu utilizată pentru detectarea liniilor de pe sol.
- **Pini**: OUT5, OUT4, OUT3, OUT2, OUT1, 5V, GND.

Baterie de 9V (x2)

- **Descriere**: Sursă de alimentare pentru circuit.
- **Pini**: -, +.

Mufă Universală Tip Baril (tată)

- **Descriere**: Conector pentru sursa de alimentare.
- **Pini**: Power Out, V+, V-.

Detalii de Cablare

Arduino UNO

- **5V**: Conectat la 5V al Matricei IR cu 5 Canale.
- **GND**: Conectat la GND al Matricei IR cu 5 Canale.
- **D2**: Conectat la OUT1 al Matricei IR cu 5 Canale.
- **D3**: Conectat la OUT2 al Matricei IR cu 5 Canale.
- **D4**: Conectat la OUT3 al Matricei IR cu 5 Canale.
- **D7**: Conectat la OUT4 al Matricei IR cu 5 Canale.
- **D12**: Conectat la OUT5 al Matricei IR cu 5 Canale.
- **D5**: Conectat la ENB al Driverului de Motor DC L298N.
- **D6**: Conectat la ENA al Driverului de Motor DC L298N.
- **D8**: Conectat la IN4 al Driverului de Motor DC L298N.
- **D9**: Conectat la IN3 al Driverului de Motor DC L298N.
- **D10**: Conectat la IN2 al Driverului de Motor DC L298N.
- **D11**: Conectat la IN1 al Driverului de Motor DC L298N.

Driver de Motor DC L298N

- **GND**: Conectat la V- al Mufei Universale Tip Baril (tată) și - al ambelor Baterii de 9V.
- **5V**: Conectat la V+ al Mufei Universale Tip Baril (tată).
- **12V**: Conectat la + al ambelor Baterii de 9V.
- **OUT1**: Conectat la pinul 2 al primului Motor DC.
- **OUT2**: Conectat la pinul 1 al primului Motor DC.
- **OUT3**: Conectat la pinul 2 al celui de-al doilea Motor DC.
- **OUT4**: Conectat la pinul 1 al celui de-al doilea Motor DC.

Motoare DC

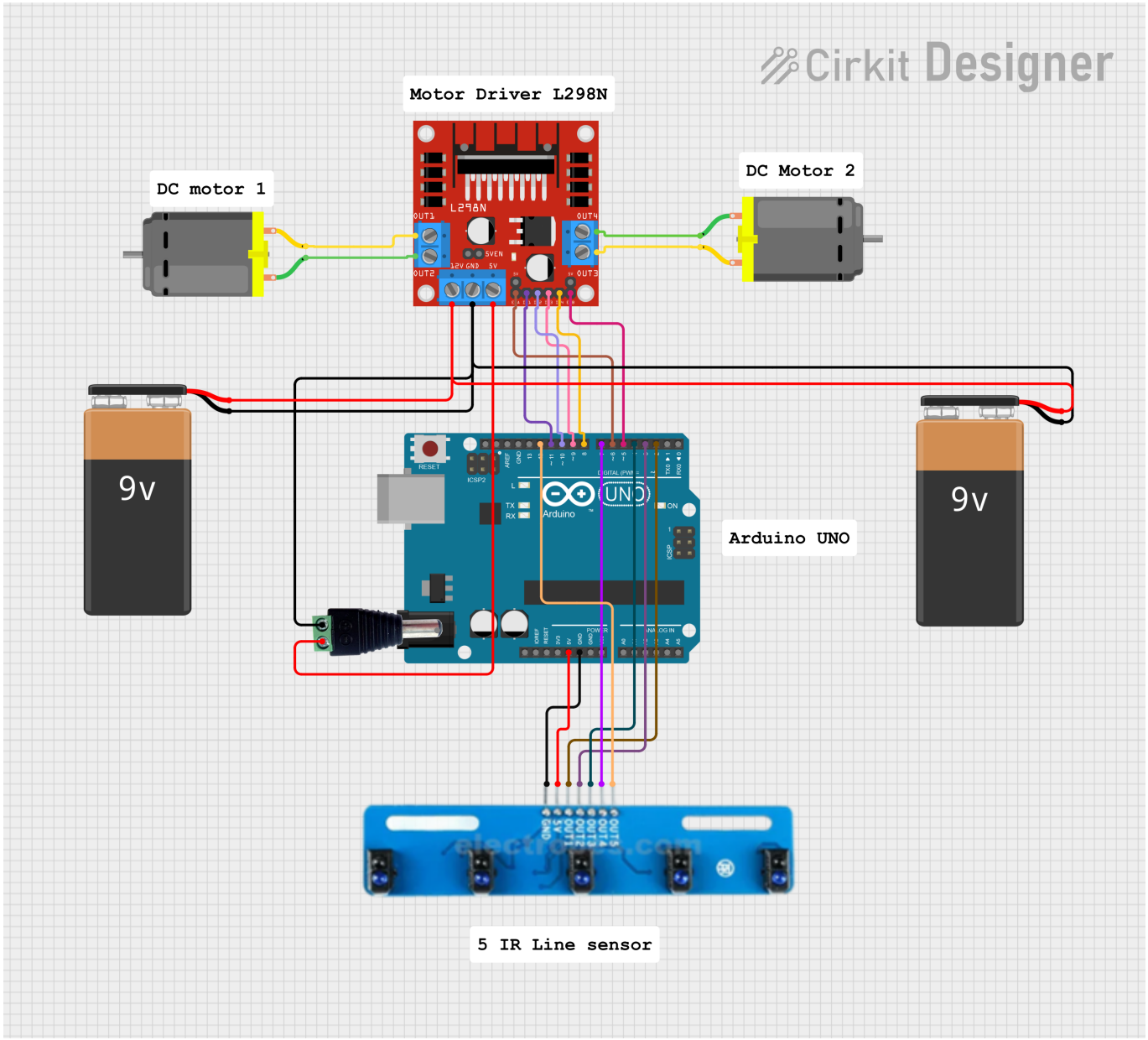
- **Primul Motor DC**:
  - pin 1: Conectat la OUT2 al Driverului de Motor DC L298N.
  - pin 2: Conectat la OUT1 al Driverului de Motor DC L298N.
- **Al Doilea Motor DC**:
  - pin 1: Conectat la OUT4 al Driverului de Motor DC L298N.
  - pin 2: Conectat la OUT3 al Driverului de Motor DC L298N.

Matrice IR cu 5 Canale

- **5V**: Conectat la 5V al Arduino UNO.
- **GND**: Conectat la GND al Arduino UNO.
- **OUT1**: Conectat la D2 al Arduino UNO.
- **OUT2**: Conectat la D3 al Arduino UNO.
- **OUT3**: Conectat la D4 al Arduino UNO.
- **OUT4**: Conectat la D7 al Arduino UNO.
- **OUT5**: Conectat la D12 al Arduino UNO.

Baterii de 9V

- **-**: Conectat la V- al Mufei Universale Tip Baril (tată) și GND al Driverului de Motor DC L298N.
- **+**: Conectat la 12V al Driverului de Motor DC L298N.



Software Design

1. Motivarea alegerii bibliotecilor folosite

Pentru acest proiect am optat pentru o abordare minimalistă, folosind doar **Arduino.h** ca bibliotecă principală. Această alegere a fost motivată de:

- **Simplicitate și performanță**: Evitarea dependențelor externe reduce overhead-ul și îmbunătățește timpul de răspuns al sistemului
- **Control direct asupra hardware-ului**: Accesul direct la funcțiile Arduino oferă control precis asupra pin-ilor digitali și PWM
- **Stabilitate**: Cod mai puțin predispus la erori cauzate de conflicte între biblioteci
- **Optimizare pentru microcontrolere**: Codul este optimizat pentru resursele limitate ale Arduino-ului

2. Elementul de noutate al proiectului

Principalul element inovator al proiectului constă în **implementarea unui filtru low-pass pe eroarea PID**:

```
error = error * 0.4 + input * 0.6;
```

Această abordare oferă:

- **Reducerea zgomotului**: Filtrarea fluctuațiilor bruște ale senzorilor
- **Stabilitate îmbunătățită**: Mișcări mai line ale robotului
- **Adaptabilitate**: Posibilitatea de fine-tuning prin modificarea coeficienților (0.4/0.6)

Un alt aspect inovator este **sistemul de detectare a cazurilor speciale** pentru diferite configurații ale senzorilor, cu tratare separată pentru intersecții și pierderea liniei.

3. Justificarea utilizării funcționalităților din laborator

Funcționalitățile implementate sunt direct derivate din conceptele de laborator:

Control PID

- **Proporțional (KP=17.00)**: Corecție bazată pe eroarea curentă
- **Derivativ (KD=600.00)**: Anticiparea schimbărilor pentru stabilitate
- **Integral (KI=0.00)**: Dezactivat pentru evitarea oscilațiilor în acest caz specific

Controlul motoarelor

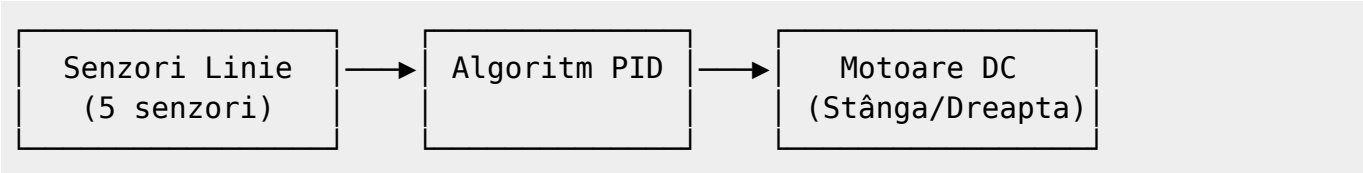
- **PWM pentru viteza**: Utilizarea analogWrite pentru control precis
- **Control direcțional**: Utilizarea pin-ilor digitali pentru forward/backward
- **Maparea valorilor**: Conversie din procentaj (-100 la 100) în valori PWM (0-255)

Senzoristica digitală

- **Array de 5 senzori**: Configurație standard pentru detectarea precisă a liniei
- **Logică inversată**: `!digitalRead()` pentru adaptarea la tipul de senzor utilizat

4. Scheletul proiectului și interacțiunea dintre funcționalități

Arhitectura sistemului:



Fluxul de execuție:

1. **Inițializare (setup())**:
   - Configurarea pin-ilor pentru motoare și senzori
   - Setarea orientării motoarelor
   - Inițializarea comunicației seriale

1. **Bucla principală (loop())**:
   - Citirea valorilor de la senzori
   - Calculul erorii de poziție
   - Aplicarea algoritmului PID
   - Ajustarea vitezei motoarelor

Validarea funcționalităților:

- **Testarea senzorilor**: Funcția `SerialLineAnalyzer()` permite monitorizarea în timp real
- **Calibrarea PID**: Constante ajustabile pentru fine-tuning
- **Debugging**: Output serial pentru verificarea valorilor

5. Mediul de dezvoltare utilizat

Pentru acest proiect am utilizat **CLion IDE** în combinație cu **PlatformIO**, o alegere motivată de următoarele avantaje:

CLion IDE:

- **IntelliSense avansat**: Autocompletare inteligentă și detecția erorilor în timp real
- **Debugging integrat**: Posibilitatea de debug pas cu pas pentru identificarea problemelor
- **Refactoring tools**: Restructurarea facilă a codului pentru optimizare
- **Git integration**: Control versiuni integrat pentru managementul modificărilor

PlatformIO:

- **Managementul dependențelor**: Gestionarea automată a bibliotecilor și framework-urilor
- **Suport multi-platformă**: Compatibilitate cu diverse tipuri de microcontrolere
- **Build system optimizat**: Compilare rapidă și eficientă
- **Library manager**: Acces facil la biblioteca vastă de componente Arduino

Configurarea proiectului:

```
; platformio.ini
[env:uno]
platform = atmelavr
board = uno
framework = arduino
monitor_speed = 9600
```

Această combinație oferă:

- **Productivitate crescută**: Mediu profesional de dezvoltare
- **Debugging eficient**: Identificarea rapidă a problemelor
- **Managementul dependencies**: Evitarea conflictelor de versiuni
- **Scalabilitate**: Posibilitatea de extindere pentru proiecte mai complexe

6. Calibrarea elementelor de senzoristica

Strategia de calibrare implementată:

1. **Maparea pozițiilor**: Fiecare combinație de senzori este mapată la o valoare specifică:
   - Centru (00100): valoare 0
   - Stânga extremă (10000): valoare -6
   - Dreapta extremă (00001): valoare +6

1. **Modificatorul nearLineModifier (0.50)**:
   - Reduce sensibilitatea pentru pozițiile apropiate de centru
   - Evită corecțiile bruște pentru deviații mici

1. **Tratarea cazurilor speciale**:
   - **Intersecții** (11111, 01110): Păstrarea ultimei valori valide
   - **Curbe strânse** (11100, 00111): Valori extreme pentru rotire rapidă

Procesul de calibrare:

```
// Testare individuală a senzorilor
lineSensorModule.SerialLineAnalyzer("every sensor");

// Monitorizare completă
lineSensorModule.SerialLineAnalyzer("all");
```

7. Optimizări implementate

7.1 Filtrul Low-Pass pe eroare

- **Unde**: În funcția `PID()`
- **De ce**: Reducerea zgomotului de la senzori
- **Cum**: `error = error * 0.4 + input * 0.6;`

7.2 Limitarea integratorului

- **Unde**: `errorInt = constrain(error + errorInt, -20, 20);`
- **De ce**: Prevenirea wind-up-ului integratorului
- **Cum**: Limitarea valorii cumulate între -20 și 20

7.3 Optimizarea structurilor

- **Unde**: Structurile `motor` și `lineSensor`
- **De ce**: Organizarea logică a codului și reutilizarea
- **Cum**: Encapsularea funcționalităților în metode specifice

7.4 Maparea eficientă a puterii

- **Unde**: Metoda `setPower()` din structura motor

- **De ce**: Conversie directă din procente în valori PWM
- **Cum**: Utilizarea funcției `map()` și `constrain()`

7.5 Detecția inteligentă a lipsei liniei

- **Unde**: Metoda `noDetection()`
- **De ce**: Păstrarea comportamentului predictibil când linia se pierde
- **Cum**: Returnarea ultimei valori valide în loc de 0

Concluzii

Proiectul implementează un sistem robust de urmărire a liniei cu:

- **Control PID optimizat** cu filtru low-pass
- **Senzoristica redundantă** cu 5 senzori
- **Gestionarea cazurilor speciale** pentru intersecții și curbe
- **Arhitectură modulară** pentru ușurința în dezvoltare și debug

Rezultatul este un robot stabil și predictibil, capabil să urmărească trasee complexe cu precizie ridicată.

```
#include <Arduino.h>
double error;
double lastError;
double errorDiff;
double errorInt;

#define KP 17.00
#define KD 600.00
#define KI 0.00
#define baseSpeed 40.00
#define nearLineModifier 0.50

struct motor {
    int motorControllerForwardPIN{};
    int motorControllerBackwardPIN{};
    int motorControllerSpeedPIN{};
    int internalPower{};
    int mappedPower{};
    bool internalOrientation = false;

    void setOrientation(const bool orientation) {
        internalOrientation = orientation;
    }

    void setPins(const int forwardPIN, const int backwardPIN, const int
speedPWMPIN) {
```

```
        motorControllerForwardPIN = forwardPIN;
        motorControllerBackwardPIN = backwardPIN;
        motorControllerSpeedPIN = speedPWMPIN;
        pinMode(motorControllerForwardPIN, OUTPUT);
        pinMode(motorControllerBackwardPIN, OUTPUT);
        pinMode(motorControllerSpeedPIN, OUTPUT);
    }

    void setPower(int power) {
        power = constrain(power, -100, 100);
        mappedPower = power;
        if (internalOrientation)
            power *= -1;
        internalPower = static_cast<int>(map(abs(power), 0, 100, 0, 255));
        if (power < 0) {
            digitalWrite(motorControllerForwardPIN, LOW);
            digitalWrite(motorControllerBackwardPIN, HIGH);
        } else {
            digitalWrite(motorControllerBackwardPIN, LOW);
            digitalWrite(motorControllerForwardPIN, HIGH);
        }
        analogWrite(motorControllerSpeedPIN, internalPower);
    }
};

motor Right;
motor Left;

struct lineSensor {
    bool leftValue = false;
    bool leftCenterValue = false;
    bool centerValue = false;
    bool rightCenterValue = false;
    bool rightValue = false;

    double globalLineValue = 0;
    String lineOutput;
    int lastGlobalLineValue = 0;
    int leftSensorPIN = 0;
    int leftCenterSensorPIN = 0;
    int centerSensorPIN = 0;
    int rightCenterSensorPIN = 0;
    int rightSensorPIN = 0;

    void setPins(const int left, const int leftCenter, const int center,
const int rightCenter, const int right) {
        leftSensorPIN = left;
        leftCenterSensorPIN = leftCenter;
        centerSensorPIN = center;
        rightCenterSensorPIN = rightCenter;
        rightSensorPIN = right;
```

```
        pinMode(leftSensorPIN, INPUT);
        pinMode(leftCenterSensorPIN, INPUT);
        pinMode(centerSensorPIN, INPUT);
        pinMode(rightCenterSensorPIN, INPUT);
        pinMode(rightSensorPIN, INPUT);
    }

    void LineValueAnalyzer() {
        if (!noDetection())
            lastGlobalLineValue = static_cast<int>(globalLineValue);
        leftValue = !digitalRead(leftSensorPIN);
        leftCenterValue = !digitalRead(leftCenterSensorPIN);
        centerValue = !digitalRead(centerSensorPIN);
        rightCenterValue = !digitalRead(rightCenterSensorPIN);
        rightValue = !digitalRead(rightSensorPIN);
        lineOutput = leftValue + static_cast<String>(leftCenterValue) +
static_cast<String>(centerValue) + static_cast<
                        String>(rightCenterValue) + static_cast<String>(
rightValue);

        if (lineOutput == "10000") {
            globalLineValue = -6; // -5
        } else if (lineOutput == "11000") {
            globalLineValue = -5; // -3
        } else if (lineOutput == "01000") {
            globalLineValue = -2 * nearLineModifier;
        } else if (lineOutput == "01100") {
            globalLineValue = -1 * nearLineModifier;
        } else if (lineOutput == "00100") {
            globalLineValue = 0;
        } else if (lineOutput == "00110") {
            globalLineValue = 1 * nearLineModifier;
        } else if (lineOutput == "00010") {
            globalLineValue = 2 * nearLineModifier;
        } else if (lineOutput == "00011") {
            globalLineValue = 5; // 3
        } else if (lineOutput == "00001") {
            globalLineValue = 6; // 5
        } else if (lineOutput == "11100" || lineOutput == "11110" ||
lineOutput == "10100" || lineOutput == "10010") {
            globalLineValue = -6;
        } else if (lineOutput == "00111" || lineOutput == "01111" ||
lineOutput == "00101" || lineOutput == "01001") {
            globalLineValue = 6;
        } else if (lineOutput == "11111" || lineOutput == "01110" ||
lineOutput == "01010") {
            globalLineValue = lastGlobalLineValue;
        }
    }
```

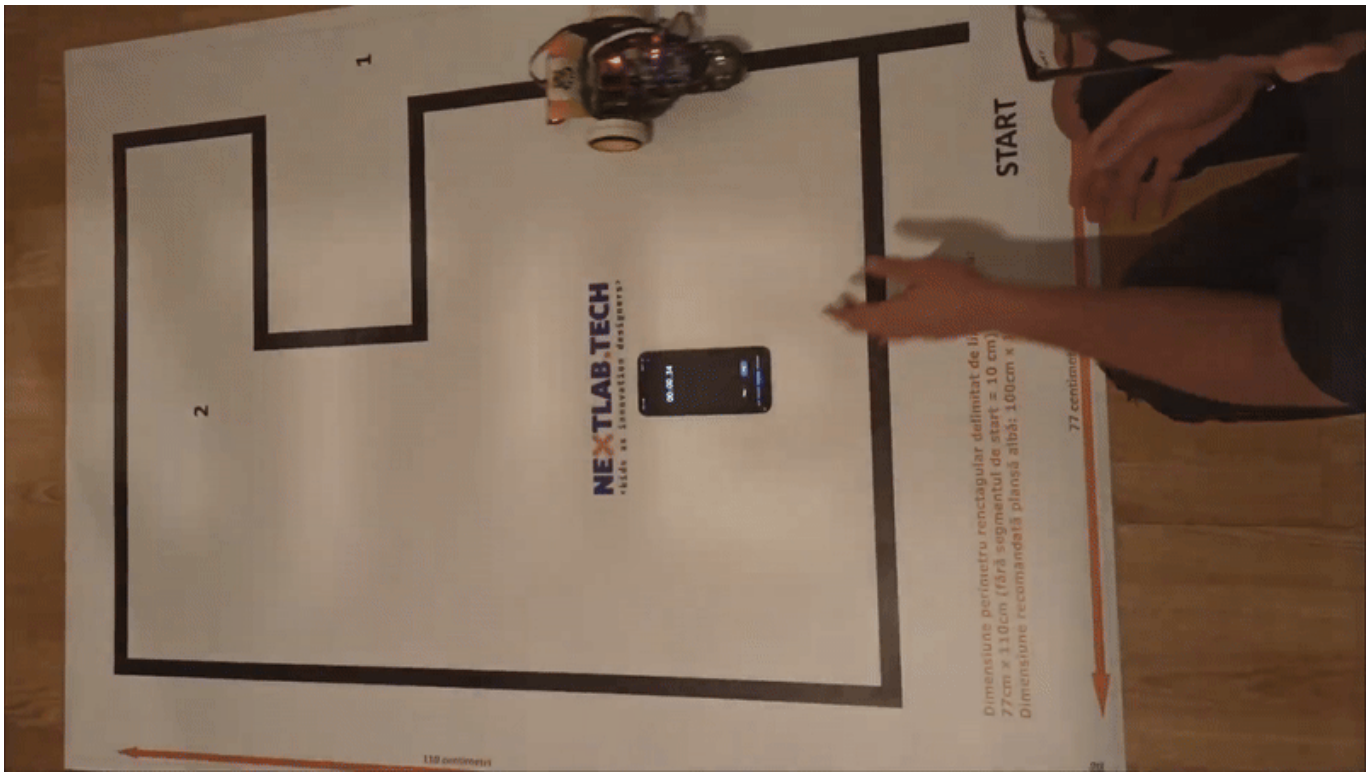
```
    bool noDetection() {
        leftValue = digitalRead(leftSensorPIN);
        leftCenterValue = digitalRead(leftCenterSensorPIN);
        centerValue = digitalRead(centerSensorPIN);
        rightCenterValue = digitalRead(rightCenterSensorPIN);
        rightValue = digitalRead(rightSensorPIN);

        if (leftValue == 1 && leftCenterValue == 1 && centerValue == 1 &&
rightCenterValue == 1 && rightValue == 1)
            return true;

        return false;
    }

    int returnLineValue() {
        if (noDetection()) {
            Serial.println(lastGlobalLineValue);
            return lastGlobalLineValue;
        }

        Serial.println(globalLineValue);
        return static_cast<int>(globalLineValue);
    }

    void SerialLineAnalyzer(const String &command) {
        LineValueAnalyzer();

        if (command == "every sensor") {
            Serial.print(leftValue);
            Serial.print(" ");
            Serial.print(leftCenterValue);
            Serial.print(" ");
            Serial.print(centerValue);
            Serial.print(" ");
            Serial.print(rightCenterValue);
            Serial.print(" ");
            Serial.print(rightValue);
            Serial.print("\n");
        }

        if (command == "all") {
            Serial.print(leftValue);
            Serial.print(" ");
            Serial.print(leftCenterValue);
            Serial.print(" ");
            Serial.print(centerValue);
            Serial.print(" ");
            Serial.print(rightCenterValue);
            Serial.print(" ");
            Serial.print(rightValue);
            Serial.print("       ");
```

```
            Serial.print(globalLineValue);
            Serial.print(" ");
        }
        if (command == "value") {
            Serial.print(globalLineValue);
            Serial.print("\n");
        }
    }
};

lineSensor lineSensorModule;

void setup() {
    Serial.begin(9600);
    Right.setOrientation(false);
    Left.setOrientation(true);
    Right.setPins(4, 5, 3);
    Left.setPins(7, 8, 6);
    lineSensorModule.setPins(9, 10, 11, 12, 13);
}

double output;

double PID(int input) {
    error = error * 0.4 + input * 0.6;
    errorDiff = error - lastError;
    errorInt = constrain(error + errorInt, -20, 20);
    output = KP * error + KD * errorDiff + KI * errorInt;
    lastError = error;

    return output;
}

void loop() {
    lineSensorModule.LineValueAnalyzer();

    Left.setPower(baseSpeed - PID(lineSensorModule.returnLineValue()));
    Right.setPower(baseSpeed + PID(lineSensorModule.returnLineValue()));
}
```

Download

linefollower.zip

Rezultate Obţinute



- Robotul urmărește linia neagră pe fundal alb
- La intersecții, utilizează ultima valoare validă pentru a continua
- Viteza de bază este 40% din capacitatea maximă pentru stabilitate
- Corecțiile PID sunt vizibile prin mișcările de ajustare ale robotului