

**PORTOFOLIU DE PRACTICĂ**

**MOBILITATE CIPRU**

**10.05.2026 – 23.05.2026**

**BRÂNZĂ DENIS**

**GONDOȘ CRISTIAN**

**NISTOR LUCA**

# Cuprins

PROIECT: Control al Luminozității unui LED prin Senzor Analogic..... 3

PROIECT: Sistem de Alertă și Monitorizare a Stării unui Senzor Digital..... 7

PROIECT: Interfață de Monitorizare și Conversie de Tensiune pentru Joystick ..... 10

PROIECT : Alarmă de distanță cu LED-uri si Buzzer ..... 13

PROIECT : Motor Pas-cu-Pas Unipolar cu Driver ..... 19

PROIECT: Motor Pas cu Pas ..... 25

PROIECT: Conveyor Belt ..... 29

PROIECT: Sistem Conveior cu Motor Pas-cu-Pas, Tastatură I2C și EEPROM |  
Stepper Belt Conveyor with I2C Keypad & EEPROM..... 33

PROIECT FINAL: Conveyor Belt with Ejection Platform ..... 49

# PROIECT: Control al Luminozității unui LED prin Senzor Analogic

## Scopul Proiectului

Proiectul își propune să ajusteze dinamice luminozitatea unui LED folosind un senzor analogic conectat la placă. Valoarea citită de la senzor este convertită într-un semnal de modulare în lățime a impulsurilor (**PWM**) pentru a modifica intensitatea luminoasă a LED-ului.

## Funcționare și Logică de Control

### 1. Citirea Semnalului Analogic:

- Placa citește valoarea furnizată de senzor prin pinul analogic A0.
- Convertorul analog-digital din Arduino returnează o valoare între **0 și 1023** (rezoluție de 10 biți).

### 2. Conversia Tensiunii / Scalarea:

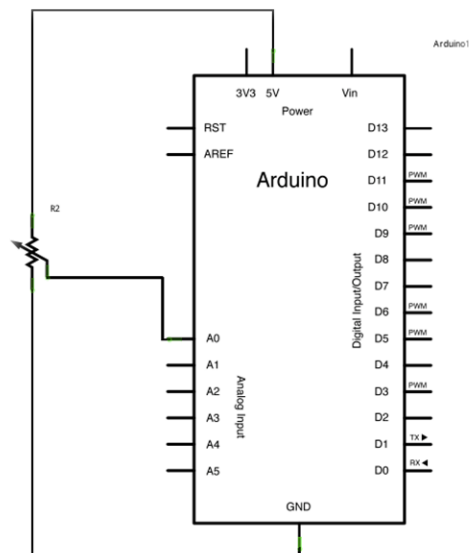
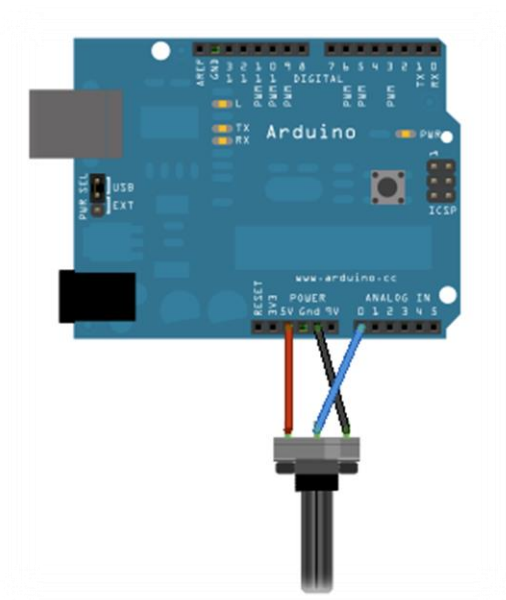
- Semnalul de ieșire pentru LED (pinul digital 9) folosește tehnica PWM, care acceptă valori între **0 și 255** (rezoluție de 8 biți).
- Codul încearcă să adapteze intervalul de citire (0 ... 1023) la intervalul de ieșire (0 ... 255).

### 3. Monitorizarea Datelor:

- Valorile brute furnizate de senzor și valorile calculate pentru luminozitate sunt trimise către monitorul serie al calculatorului la o viteză de comunicare de 9600 baud.

## Conexiuni Hardware (Alocare Pini)

- **Senzor Analogic (Potențiometrul / Senzor de lumină):** Conectat la pinul analogic **A0**
- **LED:** Conectat la pinul digital **9** (pin compatibil cu semnal de modulare în lățime a impulsurilor)
- **Consolă serie:** Transmisie de date prin portul USB la viteza de 9600 baud



## 💡 Aplicații Practice

- **Variatoare de lumină (Dimmer):** Reglarea manuală a intensității lămpilor prin rotirea unui buton.
- **Sisteme automate de iluminat:** Ajustarea automată a luminii dintr-o încăpăre în funcție de nivelul de lumină ambientală măsurat de o fotorezistență.

### COD:

```
int ledPin = 9;           // the PWM pin the LED is attached to

int brightness = 0;      // how bright the LED is

// the setup routine runs once when you press reset:
void setup() {
    pinMode(ledPin, OUTPUT);

    Serial.begin(9600);
}

// the loop routine runs over and over again forever:
void loop() {
```

```
int sensorValue = analogRead(A0);  
// set the brightness of pin 9:  
brightness = sensorValue * int(255 / 1023);  
Serial.println(sensorValue);  
Serial.println(brightness);  
analogWrite(ledPin, brightness);  
}
```



# PROIECT: Sistem de Alertă și Monitorizare a Stării unui Senzor Digital

## Scopul Proiectului

Proiectul își propune să monitorizeze continuu un senzor digital, să aprindă un LED de avertizare atunci când senzorul detectează o acțiune (cum ar fi prezența sau mișcarea) și să afișeze mesaje de stare pe monitorul serie al calculatorului doar în momentele în care starea senzorului se schimbă.

## Funcționare și Logică de Control

### 1. Citirea și Interpretarea Senzorului:

- Placa citește starea pinului digital 5 (thingPin).
- Când senzorul trimite un semnal de nivel înalt (HIGH), înseamnă că a fost detectată o acțiune sau o mișcare.

### 2. Avertizarea Luminoasă și Semnalizarea pe Portul Serie:

- **La detectarea acțiunii (HIGH):**
  - LED-ul conectat la pinul 9 se aprinde.
  - Dacă sistemul se afla anterior în stare de repaus (state == LOW), se trimite mesajul "Motion detected" către calculator și starea se actualizează la HIGH.
- **La încetarea acțiunii (LOW):**
  - LED-ul se stinge.
  - Dacă sistemul înregistra anterior mișcare (state == HIGH), se trimite mesajul "The action/motion has stopped" și starea revine la LOW.

### 3. Prevenirea Raportării Repetitive:

- Prin utilizarea variabilei state, codul implementează o logică de schimbare de stare (edge detection simplificat). Mesajele text nu sunt redactate continuu pe ecran la fiecare citire, ci **doar o singură dată**, la tranziția dintre activ și inactiv.
- Pauza de 500 ms (delay(500)) stabilizează citirile și previne alternanțele foarte rapide cauzate de zgomotul electric al senzorului.

## Conexiuni Hardware (Alocare Pini)

- **Senzor Digital (PIR / Tilt / Buton):** Conectat la pinul digital 5

- **LED (Indicator Stare):** Conectat la pinul digital 9
- **Consolă Serie:** Transmisie de date prin USB la viteză de 9600 baud

### 💡 Aplicații Practice

- **Sisteme simple de alarmă / securitate:** Aprinderea unei lumini și trimiterea unei alerte la detectarea intrușilor într-o încăpăre.
- **Monitorizarea ușilor/ferestrelor:** Detectarea deschiderii sau închiderii unei căi de acces folosind senzori magnetici sau de contact.
- **Sisteme de economisire a energiei:** Activarea iluminatului automat atunci când o persoană intră într-o zonă monitorizată.

#### CODE:

```
int ledPin = 9;
int thingPin = 5;
int state = LOW;
int val = 0;

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(thingPin, INPUT);
  Serial.begin(9600);
}

void loop() {
  val = digitalRead(thingPin);

  if (val == HIGH) {
    digitalWrite(ledPin, HIGH);
    delay(500);

    if (state == LOW) {
      Serial.println("Motion detected");
      state = HIGH;
    }
  }
  else {
    digitalWrite(ledPin, LOW);
```

```
delay(500);

if (state == HIGH) {
  Serial.println("The action/motion has stopped"); [cite: 49]
  state = LOW; [cite: 49]
}
}
}
```

# PROIECT: Interfață de Monitorizare și Conversie de Tensiune pentru Joystick

## Scopul Proiectului

Proiectul își propune să citească poziția pe două axe ( $X$  și  $Y$ ) și starea butonului integrat ale unui modul joystick. Sistemul convertește valorile brute preluate de la senzor în tensiune electrică ( $V$ ) și le afișează periodic pe monitorul serie al calculatorului.

## Funcționare și Logică de Control

### 1. Masurarea și Conversia Axei $X$ și $Y$ :

- Cele două axe ale joystick-ului sunt potențiometre interne conectate la pinii analogici A0 și A1.
- Convertorul analog-digital din Arduino citește semnalul sub formă de valoare brută întregă între 0 și 1023.
- Codul aplică o formulă de conversie pentru a afla tensiunea reală:

$$\text{Tensiune} = \text{ValoareBrută} \times \left( \frac{5.0}{1023.0} \right)$$

- În poziția centrală (repaus), tensiunea citită pe ambele axe este de aproximativ 2.5 V. Prin deplasarea manetei, valoarea variază între 0.0 V și 5.0 V.

### 2. Citirea Butonului Integrat:

- Apăsarea pe manetă acționează un buton conectat la pinul digital 3.
- Se folosește rezistorul intern de tragere în sus (pull-up): linia `digitalWrite(buttonPin, HIGH);` menține pinul în stare HIGH când butonul este liber.
- La apăsare, butonul leagă pinul la masă (GND), schimbând starea în LOW (afisat ca "pressed").

### 3. Monitorizarea și Temporizarea:

- Datele (tensiunea pe  $X$  cu 4 zecimale, tensiunea pe  $Y$  și starea butonului) sunt transmise prin portul serie la fiecare secundă (`delay(1000)`).

## Conexiuni Hardware (Alocare Pini)

- **Axa X (VRx):** Pinul analogic A0
- **Axa Y (VRy):** Pinul analogic A1
- **Buton (SW):** Pinul digital 3 (cu rezistor intern de pull-up activat)
- **Consolă Serie:** Transmisie de date la o viteză de 9600 baud

## Aplicații Practice

- **Telecomenzi și Controller-e de Jocuri:** Controlul mișcării unui personaj sau cursor pe un ecran.
- **Control Roboti și Gimbal-uri:** Orientarea brațelor robotice, a platformelor pan-tilt sau a vehiculelor de tip rover.

- **Navigare în Meniuri:** Selectarea opțiunilor pe un ecran grafic sau LCD (sus/jos/stânga/dreapta + apăsare pentru Enter).

**CODE:**

```
int xPin = A0;           // X-axis of joystick connected to analog pin A0
int yPin = A1;           // Y-axis of joystick connected to analog pin A1
int buttonPin = 3;       // Joystick button connected to digital pin 3

float xValue = 0.0;      // Current voltage on X-axis
float yValue = 0.0;      // Current voltage on Y-axis
int buttonState = HIGH;  // Current button state (HIGH = not pressed)

void setup() {
    // Run once after power-up or reset
    pinMode(xPin, INPUT);      // X-axis is input
    pinMode(yPin, INPUT);      // Y-axis is input
    pinMode(buttonPin, INPUT);  // Button is input
    digitalWrite(buttonPin, HIGH); // Enable internal pull-up resistor for button
    Serial.begin(9600);        // Start serial communication at 9600 bps
}

void loop() {
    // Read joystick positions
    xValue = analogRead(xPin) * (5.0 / 1023.0);
    yValue = analogRead(yPin) * (5.0 / 1023.0);

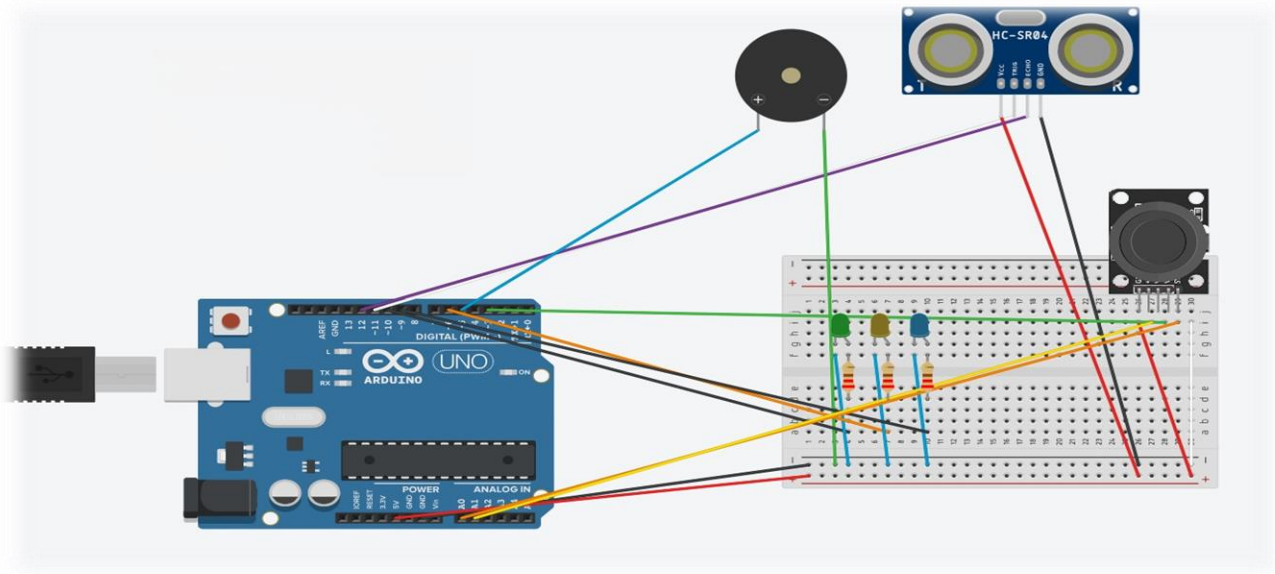
    // Read button state
    buttonState = digitalRead(buttonPin);

    // Output results to serial monitor
    Serial.print("X-axis: ");
    Serial.print(xValue, 4);
    Serial.print(" V, \tY-axis: ");
    Serial.print(yValue, 4);
    Serial.print(" V, \tButton: ");

    if (buttonState == HIGH) {
        Serial.println("not pressed");
    }
}
```

```
    } else {  
        Serial.println("pressed");  
    }  
  
    delay(1000); // Wait 1 second before next reading  
}
```

# PROIECT : Alarmă de distanță cu LED-uri si Buzzer



## Obiectivul proiectului / Project Objective

**RO:** Crearea unui sistem de alarma de distanță similar cu un senzor de parcare auto. Senzorul ultrasonic HC-SR04 măsoară distanța pana la un obiect, iar 3 LED-uri (verde, albastru, roșu) se aprind progresiv pe masura ce obiectul se apropie. Buzzerul emite semnale din ce în ce mai puternice. Joystick-ul controlează volumul maxim al buzzerului, iar butonul sau comuta modul silențios.

**EN:** Creating a distance alarm system similar to a car parking sensor. The HC-SR04 ultrasonic sensor measures the distance to an object, and 3 LEDs (green, blue, red) light up progressively as the object gets closer. The buzzer beeps louder as the object approaches. The joystick controls the maximum buzzer volume, and its button toggles mute mode.

## Lista de piese necesare / Required Components

- 1 x Placa Arduino Uno R3 / Arduino Uno R3 Board
- 1 x Senzor ultrasonic HC-SR04 / HC-SR04 Ultrasonic Sensor
- 1 x Buzzer pasiv / Passive Buzzer
- 1 x Modul Joystick (cu buton) / Joystick Module (with button)
- 1 x LED Verde / Green LED

- **1 x** LED Albastru / Blue LED
- **1 x** LED Roșu / Red LED
- **3 x** Rezistente 220 Ohmi (pentru LED-uri) / 220 Ohm Resistors (for LEDs)
- **1 x** Breadboard și fire de conexiune (jumper wires) / Breadboard and jumper wires
- **1 x** Cablu USB pentru Arduino / USB cable for Arduino

---

## Schema de conectare / Wiring / Connection Diagram

---

### Senzor HC-SR04 / HC-SR04 Sensor:

- VCC -> 5V pe Arduino / 5V on Arduino
- TRIG -> Pin digital 11 / Digital Pin 11
- ECHO -> Pin digital 12 / Digital Pin 12
- GND -> GND pe Arduino / GND on Arduino

### LED-uri (cu rezistentă 220 Ohm în serie) / LEDs (with 220 Ohm resistor in series):

- Catodul (piciorusul scurt) -> GND prin rezistentă 220 Ohmi / Cathode (short leg) -> GND via 220 Ohm resistor
- LED Verde: Anod -> Pin digital 10 / Green LED: Anode -> Digital Pin 10
- LED Albastru: Anod -> Pin digital 6 / Blue LED: Anode -> Digital Pin 6
- LED Rosu: Anod -> Pin digital 9 / Red LED: Anode -> Digital Pin 9

### Buzzer pasiv / Passive Buzzer:

- Pinul + (pozitiv) -> Pin digital 5 (PWM) pe Arduino / + pin -> Digital Pin 5 (PWM) on Arduino
- Pinul - (negativ) -> GND pe Arduino / - pin -> GND on Arduino

### Modul Joystick / Joystick Module:

- VCC -> 5V pe Arduino / 5V on Arduino
- GND -> GND pe Arduino / GND on Arduino
- VRx -> Pin analogic A0 (controlează volumul / controls volume)
- VRy -> Pin analogic A1
- SW (buton) -> Pin digital 3 (cu INPUT\_PULLUP) / SW (button) -> Digital Pin 3 (with INPUT\_PULLUP)

---

## Codul proiectului / Project Code

---

```
// Distance Alarm - Arduino Uno R3

// HC-SR04 ultrasonic sensor measures the distance to an object.
```

```

// 3 LEDs light up like a bar graph - the closer you get, the more LEDs turn on.

// Buzzer beeps louder as you get closer, like a parking sensor.

// Joystick X controls the maximum volume. Joystick button toggles mute.


int trigPin    = 11;

int echoPin    = 12;

int joyXPin    = A0;

int joyYPin    = A1;

int joyBtnPin  = 3;

int redPin     = 9;

int greenPin   = 10;

int bluePin    = 6;

int buzzerPin  = 5;


bool muted      = false;

bool btnWasDown = false;


void setup() {

    pinMode(trigPin,    OUTPUT);

    pinMode(echoPin,    INPUT);

    pinMode(joyBtnPin,  INPUT_PULLUP);

    pinMode(redPin,     OUTPUT);

    pinMode(greenPin,   OUTPUT);

    pinMode(bluePin,    OUTPUT);

    pinMode(buzzerPin,  OUTPUT);

    Serial.begin(9600);

}

void loop() {

```

```

int xValue = analogRead(joyXPin);

bool btnDown = (digitalRead(joyBtnPin) == LOW);

int cm      = readDistanceCm();


// Joystick button: toggle mute on each new press

if (btnDown && !btnWasDown) {

    muted = !muted;

}

btnWasDown = btnDown;


// Joystick X -> volume (0..255). Right = louder, left = quieter.

int volume = map(xValue, 0, 1023, 0, 255);


// LEDs as a distance bar graph: closer = more LEDs lit

// Green:  something detected (within 30 cm)

// + Blue:  getting closer (within 20 cm)

// + Red:   very close (within 10 cm)

digitalWrite(greenPin, (cm > 0 && cm <= 30) ? HIGH : LOW);

digitalWrite(bluePin,  (cm > 0 && cm <= 20) ? HIGH : LOW);

digitalWrite(redPin,   (cm > 0 && cm <= 10) ? HIGH : LOW);


// Buzzer: silent if muted or no object in range.

// Otherwise louder as the object gets closer,

// capped at the joystick volume.

int buzz = 0;

if (!muted && cm > 0 && cm <= 30) {

    buzz = map(cm, 30, 5, 0, volume);

    if (buzz < 0)        buzz = 0;

```

```

    if (buzz > volume) buzz = volume;

}

analogWrite(buzzerPin, buzz);

Serial.print("cm=");    Serial.print(cm);

Serial.print(" vol=");   Serial.print(volume);

Serial.print(" buzz=");  Serial.print(buzz);

Serial.print(" muted="); Serial.println(muted ? "yes" : "no");


delay(50);

}

int readDistanceCm() {

    digitalWrite(trigPin, LOW);

    delayMicroseconds(2);

    digitalWrite(trigPin, HIGH);

    delayMicroseconds(10);

    digitalWrite(trigPin, LOW);

    long duration = pulseIn(echoPin, HIGH);

    return duration / 58;

}

```

---

## Exerciții și Provocări / Exercises & Challenges

---

- **Ajustarea pragurilor / Adjust thresholds:** Modifică valorile 30, 20, 10 (cm) din cod pentru a schimba sensibilitatea barului de LED-uri. / Change the values 30, 20, 10 (cm) in the code to adjust the LED bar graph sensitivity.
- **Control volum / Volume control:** Misca joystick-ul pe axa X și observa cum se schimbă volumul buzzerului în Serial Monitor. / Move the joystick on the X axis and observe how the buzzer volume changes in Serial Monitor.
- **Modul silențios / Mute mode:** Apăsă butonul joystick-ului pentru a dezactiva/activa buzzerul. Ce se întâmplă cu LED-urile? / Press the joystick button to toggle the buzzer. What happens to the LEDs?

- **Adaugă un display / Add a display:** Poți conecta un LCD 16x2 pentru a afișa distanța în cm în timp real? / Can you connect a 16x2 LCD to display the distance in cm in real time?
- **Monitorizare / Monitor:** Deschide Serial Monitor (Ctrl+Shift+M) și observa valorile cm, vol, buzz și muted pe măsura ce apropii mâna de senzor. / Open Serial Monitor (Ctrl+Shift+M) and watch the cm, vol, buzz, and muted values as you move your hand toward the sensor.

---

**!!! Sfat / Tip:** Verifica de două ori orientarea senzorului HC-SR04 înainte să alimentezi placa! Pinii TRIG și ECHO trebuie conectate corect (11 și 12). Buzzerul este pasiv și folosește PWM (analogWrite) – nu va funcționa cu tone(). Asigura-te că este pe pinul 5 (PWM). / Double-check the HC-SR04 orientation before powering the board! TRIG and ECHO must be on pins 11 and 12. The buzzer is passive and uses PWM (analogWrite) – it will NOT work with tone(). Make sure it is on pin 5 (PWM).

# PROIECT : Motor Pas-cu-Pas Unipolar cu Driver

## Obiectivul proiectului / Project objective

**RO:** Controlul unui motor pas-cu-pas unipolar (28BYJ-48) printr-un driver dedicat (ULN2003APG), conectat la o placă Arduino Uno R3 și alimentat de o baterie externă de 5V. Motorul va efectua rotații complete și parțiale, în ambele direcții, cu viteză reglabilă. **EN:** *Control a unipolar stepper motor (28BYJ-48) through a dedicated driver (ULN2003APG), connected to an Arduino Uno R3 board and powered by an external 5V battery. The motor will perform full and partial rotations in both directions, with adjustable speed.*

## Lista de piese necesare / Required components

- 1 x Placă Arduino Uno R3 / *Arduino Uno R3 Board*
- 1 x Motor pas-cu-pas unipolar 28BYJ-48 / *28BYJ-48 Unipolar Stepper Motor*
- 1 x Modul driver ULN2003APG (cu placă de test) / *ULN2003APG Driver Module (with breakout board)*
- 1 x Baterie externă 5V (power bank sau sursă de 5V/1A) / *External 5V battery (power bank or 5V/1A power supply)*
- 1 x Breadboard și fire de conexiune (jumper wires) / *Breadboard and jumper wires*
- 1 x Cablu USB pentru Arduino / *USB cable for Arduino*

## Schema de conectare (Fritzing) / Wiring diagram (Fritzing)

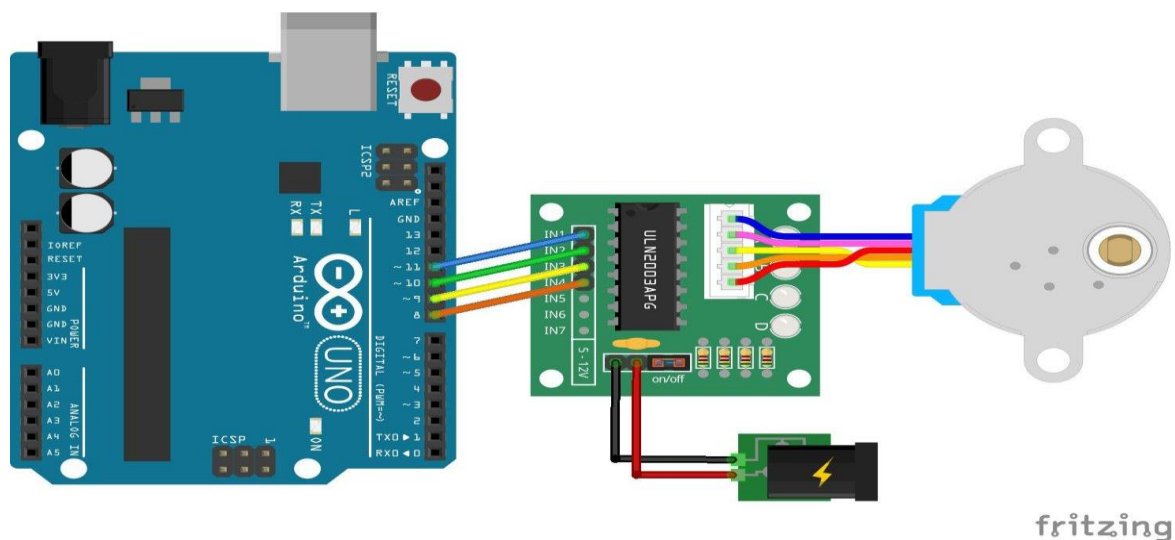


Fig. 1 – Schema de conectare motor pas-cu-pas cu driver ULN2003 și Arduino Uno / Stepper motor wiring diagram with ULN2003 driver and Arduino Uno

## Schema de conectare detaliată / Detailed wiring / connection diagram

### Motor 28BYJ-48 → Driver ULN2003 / Motor Pins → Driver

| Pin Motor / Motor Pin                                    | Pin Driver / Driver Pin                      |
|----------------------------------------------------------|----------------------------------------------|
| Conector 5 pini<br>(albastru/roz/galben/portocaliu/roșu) | Conector motor de pe modul ULN2003           |
| Fir roșu (VCC +5V)                                       | VCC pe modulul driver / VCC on driver module |

### Driver ULN2003 → Arduino Uno / Driver → Arduino

| Pin Driver / Driver Pin | Pin Arduino / Arduino Pin       |
|-------------------------|---------------------------------|
| IN1                     | Pin digital 8                   |
| IN2                     | Pin digital 9                   |
| IN3                     | Pin digital 10                  |
| IN4                     | Pin digital 11                  |
| GND                     | GND pe Arduino / GND on Arduino |

### Alimentare / Power Supply

| Sursă / Source             | Destinație / Destination               |
|----------------------------|----------------------------------------|
| Baterie 5V – fir roșu (+)  | VCC pe modulul driver ULN2003          |
| Baterie 5V – fir negru (–) | GND pe modulul driver ULN2003          |
| GND baterie (comun)        | GND Arduino (masă comună obligatorie!) |

**⚠️ Atenție: GND-ul bateriei și GND-ul Arduino TREBUIE conectate împreună (masă comună)! Fără aceasta, motorul nu va funcționa corect.**

 **Warning: Battery GND and Arduino GND MUST be connected together (common ground)!**  
*Without this, the motor will not function correctly.*

## Codul proiectului / Project code

---

```
//Include the Arduino Stepper Library
#include <Stepper.h>

// Define Constants

// Number of steps per internal motor revolution
const float STEPS_PER_REV = 32;

// Amount of Gear Reduction
const float GEAR_RED = 64;

// Number of steps per geared output rotation
const float STEPS_PER_OUT_REV = STEPS_PER_REV * GEAR_RED;

// Define Variables

// Number of Steps Required
int StepsRequired;

// Create Instance of Stepper Class
// Specify Pins used for motor coils
// The pins used are 8,9,10,11
```

```
// Connected to ULN2003 Motor Driver In1, In2, In3, In4

// Pins entered in sequence 1-3-2-4 for proper step sequencing

Stepper steppermotor(STEPS_PER_REV, 8, 10, 9, 11);

void setup()
{
  // Nothing (Stepper Library sets pins as outputs)
}

void loop()
{
  // Slow - 4-step CW sequence to observe lights on driver board
  steppermotor.setSpeed(1);
  StepsRequired = 4;
  steppermotor.step(StepsRequired);
  delay(2000);

  // Rotate CW 1/2 turn slowly
  StepsRequired = STEPS_PER_OUT_REV / 2;
  steppermotor.setSpeed(100);
  steppermotor.step(StepsRequired);
  delay(1000);

  // Rotate CCW 1/2 turn quickly
  StepsRequired = - STEPS_PER_OUT_REV / 2;
```

```
steppermotor.setSpeed(700);  
  
steppermotor.step(StepsRequired);  
  
delay(2000);  
  
}
```

## Exerciții și provocări / Exercises & challenges

---

### 1. Schimbarea vitezei / Change the Speed

**RO:** Modifică valoarea din `myStepper.setSpeed(10)` la 5, 10 sau 15 RPM și observă diferența. Ce se întâmplă dacă depășești 15 RPM? **EN:** *Change the value in `myStepper.setSpeed(10)` to 5, 10, or 15 RPM and observe the difference. What happens if you exceed 15 RPM?*

### 2. Rotire parțială / Partial Rotation

**RO:** Trimite comanda 'H' în Serial Monitor pentru a roti motorul cu exact o jumătate de rotație. Încearcă să implementezi o comandă pentru sfert de rotație. **EN:** *Send the 'H' command in Serial Monitor to rotate the motor exactly half a turn. Try to implement a command for a quarter rotation.*

### 3. Rotire continuă / Continuous Rotation

**RO:** Modifică codul astfel încât motorul să se rotească continuu în ambele direcții (du-te-vino), cu o pauză de 1 secundă între direcții. **EN:** *Modify the code so the motor rotates continuously back and forth, with a 1-second pause between directions.*

### 4. Număr de pași / Step Count

**RO:** Calculează câți pași sunt necesari pentru a roti arborele motorului cu exact 90°. Verifică: 2048 pași = 360°, deci 90° = ? **EN:** *Calculate how many steps are needed to rotate the motor shaft exactly 90°. Hint: 2048 steps = 360°, so 90° = ?*

### 5. Monitorizare Serial / Serial Monitor

**RO:** Deschide Serial Monitor (Ctrl+Shift+M), setează viteza la 9600 baud și trimite comenzile F, B, H. Observă mesajele afișate. **EN:** *Open Serial Monitor (Ctrl+Shift+M), set baud rate to 9600 and send F, B, H commands. Observe the printed messages.*

### Sfaturi importante / Important tips

---

**!!! Sfat / Tip:** Motorul 28BYJ-48 necesită ordinea specială a pinilor IN1, IN3, IN2, IN4 în biblioteca Stepper. Dacă ordinea este greșită, motorul va vibra fără să se rotească. / The 28BYJ-48 motor

requires the special pin order IN1, IN3, IN2, IN4 in the Stepper library. If the order is wrong, the motor will vibrate without rotating.

**!!! Sfat / Tip:** Nu alimenta motorul direct de la pinii de 5V ai Arduino-ului! Folosește întotdeauna bateria externă. Motorul poate consuma până la 240mA, ceea ce poate deteriora placa Arduino. / Never power the motor directly from the Arduino's 5V pin! Always use the external battery. The motor can draw up to 240mA, which may damage the Arduino board.

**!!! Sfat / Tip:** Asigură-te că masa comună (GND) este conectată între Arduino și modulul driver ULN2003. Fără aceasta, semnalele de control nu vor funcționa corect. / Make sure the common ground (GND) is connected between the Arduino and the ULN2003 driver module. Without this, control signals will not work correctly.

# PROIECT: Motor Pas cu Pas

## Scopul Proiectului

Proiectul își propune să exemplifice modul de acționare, poziționare și control al vitezei de rotație pentru un motor pas cu pas (modelul 28BYJ-48) conectat la un driver cu tranzistoare (ULN2003). Prin intermediul codului, sunt exemplificate atât mișcările fine (la nivel de pași individuali), cât și rotațiile rapide în ambele sensuri (orar și antiorar).

## Funcționare și Logică de Control

### 1. Calculul Raportului de Transmisie:

- Motorul are intern 32 de pași per rotație, dar folosește un reductor mecanic cu un raport de 1:64.
- Codul calculează numărul total de pași necesari pentru o rotație completă la axul de ieșire:

$$\text{Pași per Rotație Completă} = 32 \times 64 = 2048 \text{ pași}$$

### 2. Secvența de Mișcare (Ciclu Repetitiv):

- Pasul 1 (Observare vizuală):** Motorul efectuează doar 4 pași la o viteză foarte mică (1 RPM). Această etapă permite vizualizarea aprinderii succesive a LED-urilor de pe placa driverului ULN2003. Urmează o pauză de 2 secunde.
- Pasul 2 (Rotație în sens orar):** Motorul execută jumătate de rotație completă (1024 pași) în sensul acelor de ceasornic, la o viteză moderată (100 RPM). Urmează o pauză de 1 secundă.
- Pasul 3 (Rotație în sens antiorar):** Prin transmiterea unei valori negative (−1024 pași), motorul execută jumătate de rotație în sens contrar acelor de ceasornic, la o viteză mai ridicată (700 RPM). Urmează o pauză de 2 secunde înainte de reluarea ciclului.

### 3. Secvențierea Pinilor:

- Biblioteca nativă necesită declararea pinilor în ordinea 8, 10, 9, 11 (respectiv conexiunea la intrările IN1, IN3, IN2, IN4 ale driverului) pentru a asigura alternanța corectă a polilor electromagnetici.

## Conexiuni Hardware (Alocare Pini)

- Intrări Driver Motor (ULN2003):**
  - IN1 conectat la pinul digital 8

- IN2 conectat la pinul digital **9**
- IN3 conectat la pinul digital **10**
- IN4 conectat la pinul digital **11**
- **Alimentare Driver:** Sursă externă dedicată de 5V (pentru a proteja placa Arduino de consumul direct al motorului).

### 💡 Aplicații Practice

- **Teste de laborator și calibrare:** Verificarea funcționării corecte a motoarelor pas cu pas și a plăcilor de comandă.
- **Sisteme de poziționare simplă:** Deschideri/închideri de clapete, fante de aerisire sau bariere de acces.
- **Mecanisme de ceasornicărie sau jaluzele automatizate:** Proiecte ce necesită unghiuri precise de rotație la viteze reduse.

### COD:

```
//Include the Arduino Stepper Library

#include <Stepper.h>


// Define Constants


// Number of steps per internal motor revolution

const float STEPS_PER_REV = 32;


// Amount of Gear Reduction

const float GEAR_RED = 64;


// Number of steps per geared output rotation

const float STEPS_PER_OUT_REV = STEPS_PER_REV * GEAR_RED;


// Define Variables
```

```

// Number of Steps Required

int StepsRequired;

// Create Instance of Stepper Class

// Specify Pins used for motor coils

// The pins used are 8,9,10,11

// Connected to ULN2003 Motor Driver In1, In2, In3, In4

// Pins entered in sequence 1-3-2-4 for proper step sequencing


Stepper steppermotor(STEPS_PER_REV, 8, 10, 9, 11);


void setup()

{

// Nothing (Stepper Library sets pins as outputs)

}


void loop()

{

// Slow - 4-step CW sequence to observe lights on driver board

steppermotor.setSpeed(1);

StepsRequired = 4;

steppermotor.step(StepsRequired);

delay(2000);


// Rotate CW 1/2 turn slowly

```

```
StepsRequired = STEPS_PER_OUT_REV / 2;

steppermotor.setSpeed(100);

steppermotor.step(StepsRequired);

delay(1000);


// Rotate CCW 1/2 turn quickly

StepsRequired = - STEPS_PER_OUT_REV / 2;

steppermotor.setSpeed(700);

steppermotor.step(StepsRequired);

delay(2000);

}
```

# PROIECT: Conveyor Belt

## Scopul Proiectului

Proiectul își propune să realizeze o mișcare liniară automatizată de precizie (pe un ghidaj sau curea transportoare). Sistemul deplasează un ansamblu mobil în pași ficși de câte 100 mm (10 cm), inversează automat direcția când atinge o limită de 200 mm (20 cm) și monitorizează continuu parcursul prin consola serie.

## Funcționare și Logică de Control

### 1. Calculul de Calibrare Liniară:

- Motorul (model 28BYJ-48) are 2048 pași per rotație completă ( $32 \times 64$ ).
- O rotație completă a motorului determină o translație liniară de 40 mm (4 cm).
- Din acest raport rezultă factorul de calibrare de 51.2 pași/mm ( $\frac{2048}{40} = 51.2$ ).

### 2. Ciclul Automat de Deplasare:

- La fiecare iterare din buclă, sistemul își propune să înainteze cu o distanță fixă (`distanceToCover = 100`).
- Calculează pașii necesari:

$$\text{Pași} = 100 \text{ mm} \times 51.2 \text{ pași/mm} = 5120 \text{ pași}$$

- Inversarea de sens și limitarea cursei:**
  - Sistemul adună distanța în `totalDistance`.
  - Când distanța acumulată atinge sau depășește 200 mm, valoarea `stepsPerMlMtr` devine negativă ( $-51.2$ ), iar contorul se resetează.
  - Acest lucru face ca motorul să execute următorul pas în sens invers (întoarcere la origini).

### 3. Monitorizarea pe Portul Serie:

- La fiecare pas, trimite către calculator distanța totală parcursă și numărul de pași pe care îi execută în acel moment.

## Conexiuni Hardware (Alocare Pini)

- Driver Motor (ULN2003):** Conectat la pinii digitali **8, 10, 9, 11** (secvență specifică pentru comanda bobinelor)

- **Tensiune de Alimentare Motor:** Alimentat la o tensiune optimizată de 8.5V (conform comentariilor din cod, asigură un cuplu mai bun față de alimentarea standard de 5V)

### Aplicații Practice

- **Testarea de rezistență / ciclare (Benchmarking):** Verificarea uzurii mecanice a unei benzi sau a unui ghidaj prin mișcări repetate înainte-înapoi.
- **Mese mobile de scanare / printare:** Mișcarea capului de imprimare sau a unui senzor pe un traseu dus-întors bine definit.
- **Mecanisme de dozare sau împingere automată:** Deplasarea periodică a unui sertar sau piston pe o distanță fixă.

**COD:**

```
#include <Stepper.h>

//the motor has 32 internal steps

const int INTERNAL_STEPS_PER_REVOL=32;

//the motor also has internal gears to reduce the speed and increa ase
the torque

const int GEAR_REDUCTION=64;

//total steps per revolution

const int STEPS_PER_REVOLUTION=INTERNAL_STEPS_PER_REVOL*GEAR_REDUCTION;


int stepsRequired;

float stepsPerMlmtr=51.2;

float distanceToCover, distanceToSteps;

int totalDistance=0;


Stepper theMotor(INTERNAL_STEPS_PER_REVOL,8,10,9,11);


void setup() {

    // put your setup code here, to run once:
```

```

    Serial.begin(9600);
}

//one full revolution is equal to 4cm

//the motor is operating at 8.5 volts

//at this voltage is working much better


void loop() {

    // put your main code here, to run repeatedly:

    theMotor.setSpeed(1000);

    //lets have the motor do one full revolution so we can measure the
    distance it covers

    //in one full revolution

    /* stepsRequired=STEPS_PER_REVOLUTION;

    theMotor.step(stepsRequired);

    delay(2000);*/

    //after measurements one full revolution is equal to 4cm

    //now we can calculate the mm per step.

    //if we have 40mm per 2048 steps and we want to calculate

    //how many steps to cover 1mm. and the answer is 51.2 steps per
    millimeter

    distanceToCover=100; //lets move 50mm

    stepsRequired=distanceToCover * stepsPerMlmtr;

    totalDistance=totalDistance+distanceToCover;

    if (totalDistance>=200){

        totalDistance=0;

        stepsPerMlmtr=stepsPerMlmtr*-1;

    }
}

```

```
Serial.print("Total Distance:");  
Serial.println(totalDistance);  
Serial.print("Steps Required: ");  
Serial.println(stepsRequired);  
theMotor.step(stepsRequired);  
delay(10);  
}
```

# PROIECT: Sistem Conveior cu Motor Pas-cu-Pas, Tastatură I2C și EEPROM | Stepper Belt Conveyor with I2C Keypad & EEPROM

## Obiectivul proiectului / *Project objective*

**RO:** Controlul unui motor pas-cu-pas unipolar (28BYJ-48) printr-un driver dedicat (ULN2003APG), conectat la un Arduino Uno R3. Motorul antrenează o curea transportoare de 800 mm, pe care este prinsă cu șuruburi o mică platformă dreptunghiulară. Sistemul deplasează platforma pe o distanță programabilă, menținând poziția în EEPROM. Tastatura I2C (PCF8574) permite pornirea/oprirea, schimbarea direcției, setarea distanței țintă și recalibrarea poziției. Un buzzer semnalizează evenimentele importante. **EN:** *Control a unipolar stepper motor (28BYJ-48) through a dedicated driver (ULN2003APG), connected to an Arduino Uno R3. The motor drives an 800 mm conveyor belt, onto which a small rectangular platform is screwed. The system moves the platform a programmable distance, storing the position in EEPROM. An I2C keypad (PCF8574) allows start/stop, direction change, target distance setting, and position recalibration. A buzzer signals key events.*

## Lista de piese necesare / *Required components*

- **1 x** Placă Arduino Uno R3 / *Arduino Uno R3 Board*
- **1 x** Motor pas-cu-pas unipolar 28BYJ-48, montat mecanic pe cadrul conveiorului / *28BYJ-48 Unipolar Stepper Motor, mechanically mounted on the conveyor frame*
- **1 x** Curea transportoare (lungime 800 mm) antrenată de motor prin angrenaj / *Conveyor belt (800 mm length) driven by the motor through a gear*
- **1 x** Platformă dreptunghiulară mică, prinsă cu șuruburi de curea – transportată de-a lungul conveiorului / *Small rectangular platform, screwed onto the belt – carried along the conveyor*
- **1 x** Modul driver ULN2003APG (cu placă de test) / *ULN2003APG Driver Module (with breakout board)*
- **1 x** Tastatură matricială 4x4 cu modul I2C PCF8574 (adresă 0x20) / *4x4 Matrix Keypad with PCF8574 I2C module (address 0x20)*
- **1 x** Buzzer pasiv (5V, pin digital 4) / *Passive buzzer (5V, digital pin 4)*
- **1 x** Baterie externă 5V (power bank sau sursă 5V/1A) / *External 5V battery (power bank or 5V/1A supply)*
- **1 x** Breadboard și fire de conexiune (jumper wires) / *Breadboard and jumper wires*
- **1 x** Cablu USB pentru Arduino / *USB cable for Arduino*
- Biblioteci Arduino: EEPROM.h (built-in), Wire.h (built-in), I2CKeyPad.h (Rob Tillaart) / *Arduino Libraries: EEPROM.h (built-in), Wire.h (built-in), I2CKeyPad.h (by Rob Tillaart)*

Schema de conectare detaliată / Detailed wiring / connection diagram

Motor 28BYJ-48 → Driver ULN2003 / Motor Pins → Driver

| Pin Motor / Motor Pin                                    | Pin Driver / Driver Pin                      |
|----------------------------------------------------------|----------------------------------------------|
| Conector 5 pini<br>(albastru/roz/galben/portocaliu/roșu) | Conector motor pe modulul ULN2003            |
| Fir roșu (VCC +5V)                                       | VCC pe modulul driver / VCC on driver module |

Driver ULN2003 → Arduino Uno / Driver → Arduino

| Pin Driver / Driver Pin | Pin Arduino / Arduino Pin       |
|-------------------------|---------------------------------|
| IN1                     | Pin digital 8                   |
| IN2                     | Pin digital 9                   |
| IN3                     | Pin digital 10                  |
| IN4                     | Pin digital 11                  |
| GND                     | GND pe Arduino / GND on Arduino |

Tastatură I2C PCF8574 → Arduino Uno / I2C Keypad PCF8574 → Arduino Uno

| Pin Tastatură / Keypad Pin | Pin Arduino / Arduino Pin |
|----------------------------|---------------------------|
| VCC                        | 5V                        |
| GND                        | GND                       |
| SDA                        | Pin A4 (SDA)              |
| SCL                        | Pin A5 (SCL)              |

Buzzer → Arduino Uno / Buzzer → Arduino Uno

| Pin Buzzer / Buzzer Pin | Pin Arduino / Arduino Pin |
|-------------------------|---------------------------|
| (+) Semnal              | Pin digital 4             |
| (-) GND                 | GND                       |

Alimentare / Power Supply

| Sursă / Source             | Destinație / Destination               |
|----------------------------|----------------------------------------|
| Baterie 5V – fir roșu (+)  | VCC pe modulul driver ULN2003          |
| Baterie 5V – fir negru (-) | GND pe modulul driver ULN2003          |
| GND baterie (comun)        | GND Arduino (masă comună obligatorie!) |

 **Atenție:** GND-ul bateriei și GND-ul Arduino TREBUIE conectate împreună (masă comună)! Fără aceasta, motorul nu va funcționa corect.  **Warning:** Battery GND and Arduino GND MUST be connected together (common ground)! Without this, the motor will not function correctly.

Comenzile tastaturii / Keypad commands

| Tastă / Key     | Funcție (RO)                              | Function (EN)                        |
|-----------------|-------------------------------------------|--------------------------------------|
| A               | Schimbă direcția de mișcare               | Toggle movement direction            |
| B               | Pornește / Oprește motorul                | Start / Stop the motor               |
| C               | Introduce distanța Țintă (mm)             | Set target distance (mm)             |
| D               | Setează poziția curentă (mm)              | Set current position (mm)            |
| 0-9             | Introduce cifre pentru distanță / poziție | Enter digits for distance / position |
| C (modul intr.) | Confirmă valoarea introdusă               | Confirm entered value                |

| Tastă / Key     | Funcție (RO)                    | Function (EN)                 |
|-----------------|---------------------------------|-------------------------------|
| * (modul intr.) | Șterge ultima cifră (backspace) | Delete last digit (backspace) |
| # (modul intr.) | Anulează introducerea datelor   | Cancel data entry             |

### Codul proiectului / *Project code*

**RO:** Codul complet al proiectului, include toate funcțiile: beep(), saveState(), opreste(), faUnPas(), proceseazaTasta(), setup() și loop(). **EN:** *Complete project code, including all functions: beep(), saveState(), opreste(), faUnPas(), proceseazaTasta(), setup() and loop().*

```
#include <EEPROM.h>

#include <Wire.h>

#include <I2CKeyPad.h>

const int INTERNAL_STEPS_PER_REVOL = 32;
const int GEAR_REDUCTION = 64;

const float STEPS_PER_MM = 51.2;
const int BELT_LENGTH = 800;
const int STEP_SIZE = 50;

const int BUZZER_PIN = 4;

const int motorPins[4] = {8, 9, 10, 11};

const int stepSequence[8][4] = {
    {1, 0, 0, 0},
    {1, 1, 0, 0},
    {0, 1, 0, 0},
```

```
{0, 1, 1, 0},
{0, 0, 1, 0},
{0, 0, 1, 1},
{0, 0, 0, 1},
{1, 0, 0, 1}
};

int currentStep = 0;
long stepsRamasi = 0;
int directiePas = 1;

float stepsPerMlmtr = STEPS_PER_MM;
int pozitie = 0;
bool running = false;

long distantaTarget = 0;
long distantaParcursa = 0;

bool setandDistanta = false;
bool setandPozitie = false;
String inputDistanta = "";

const int ADDR_POZITIE = 0;
const int ADDR_DIRECTIE = 2;

const int STEP_DELAY_US = 1000;
unsigned long lastStepTime = 0;

const uint8_t KEYPAD_ADDRESS = 0x20;
```

```

I2CKeyPad keypad(KEYPAD_ADDRESS);

char keys[] = "123A456B789C*0#D";
char lastKey = 0;

void beep(int ori, int durata) {
    for (int i = 0; i < ori; i++) {
        tone(BUZZER_PIN, 400);
        delay(durata);
        noTone(BUZZER_PIN);
        delay(durata);
    }
}

void saveState() {
    EEPROM.put(ADDR_POZITIE, pozitie);
    int directie = (stepsPerMlmtr > 0) ? 1 : -1;
    EEPROM.put(ADDR_DIRECTIE, directie);
}

void opreste() {
    running = false;
    stepsRamasi = 0;
    beep(3, 300);
    Serial.print("Distanța parcursă: ");
    Serial.print(distanțaParcursă);
    Serial.println(" mm. Oprit.");
}

```

```

void faUnPas() {
    if (stepsRamasi == 0) return;

    unsigned long now = micros();
    if (now - lastStepTime < STEP_DELAY_US) return;
    lastStepTime = now;

    currentStep = (currentStep + directiePas + 8) % 8;
    for (int i = 0; i < 4; i++) {
        digitalWrite(motorPins[i], stepSequence[currentStep][i]);
    }

    stepsRamasi--;

    if (stepsRamasi == 0) {
        int nextPozitie = pozitie + (stepsPerMlmtr > 0 ? STEP_SIZE : -
STEP_SIZE);
        distantaParcursa += STEP_SIZE;

        bool laCapat = false;

        if (nextPozitie >= BELT_LENGTH) {
            nextPozitie = BELT_LENGTH;
            stepsPerMlmtr = -STEPS_PER_MM;
            laCapat = true;
        } else if (nextPozitie < 0) {
            nextPozitie = 0;
            stepsPerMlmtr = STEPS_PER_MM;
            laCapat = true;
        }
    }
}

```

```

}

pozitie = nextPozitie;

saveState();

if (laCapat) {
    beep(5, 80);
}

Serial.print("Pozitie: ");
Serial.print(pozitie);
Serial.print(" mm | Directie: ");
Serial.print(stepsPerMlmtr > 0 ? "inainte" : "inapoi");
Serial.print(" | Parcurs: ");
Serial.print(distantaParcursa);
Serial.print("/");
Serial.print(distantaTarget);
Serial.println(" mm");

if (distantaTarget > 0 && distantaParcursa >= distantaTarget) {
    opreste();
    return;
}

if (running) {
    stepsRamasi = (long) (STEP_SIZE * STEPS_PER_MM);
    directiePas = (stepsPerMlmtr > 0) ? 1 : -1;
}
}

```

```

}

void proceseazaTasta(char key) {
    if (setandDistanta) {
        if (key >= '0' && key <= '9') {
            inputDistanta += key;

            Serial.print(setandPozitie ? "Pozitie introdusa: " : "Distanta
introdusa: ");
            Serial.println(inputDistanta);
        }
        else if (key == 'C') {
            long valoare = inputDistanta.toInt();
            if (setandPozitie) {
                if (valoare >= 0 && valoare <= BELT_LENGTH) {
                    pozitie = (int)valoare;
                    saveState();
                    Serial.print("Pozitie setata la: ");
                    Serial.print(pozitie);
                    Serial.println(" mm");
                } else {
                    Serial.print("Pozitie invalida! Trebuie intre 0 si ");
                    Serial.println(BELT_LENGTH);
                }
            }
            setandPozitie = false;
        } else {
            if (valoare > 0) {
                distantaTarget = valoare;
                distantaParcursa = 0;
                Serial.print("Distanta target setata: ");
            }
        }
    }
}

```

```

        Serial.print(distantaTarget);

        Serial.println(" mm");

    } else {

        Serial.println("Distanta invalida!");

    }

}

inputDistanta = "";

setandDistanta = false;

}

else if (key == '*') {

    if (inputDistanta.length() > 0) {

        inputDistanta.remove(inputDistanta.length() - 1);

        Serial.print(setandPozitie ? "Pozitie introdusa: " : "Distanta
introdusa: ");

        Serial.println(inputDistanta);

    }

}

else if (key == '#') {

    inputDistanta = "";

    setandDistanta = false;

    setandPozitie = false;

    Serial.println("Anulat.");

}

return;

}

if (key == 'A') {

    stepsPerMlmtr = -stepsPerMlmtr;

    directiePas = (stepsPerMlmtr > 0) ? 1 : -1;

```

```
    saveState();

    beep(2, 100);

    Serial.println(stepsPerMlmtr > 0 ? "Directie: inainte" : "Directie:
inapoi");
}

else if (key == 'B') {
    running = !running;
    beep(1, 750);
    Serial.println(running ? "Pornit" : "Oprit");
    if (running && stepsRamasi == 0) {
        if (distantaParcursa >= distantaTarget) {
            distantaParcursa = 0;
        }
        stepsRamasi = (long)(STEP_SIZE * STEPS_PER_MM);
        directiePas = (stepsPerMlmtr > 0) ? 1 : -1;
    }
    if (!running) {
        stepsRamasi = 0;
    }
}

else if (key == 'C') {
    setandDistanta = true;
    setandPozitie = false;
    inputDistanta = "";

    Serial.println("Introdu distanta (mm) si apasa C pentru confirmare,
# pentru anulare:");
}

else if (key == 'D') {
    setandDistanta = true;
    setandPozitie = true;
```

```

        inputDistanta = "";

        Serial.println("Introdu pozitia curenta (mm) si apasa C pentru
confirmare, # pentru anulare:");

    }
}

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

    pinMode(BUZZER_PIN, OUTPUT);
    digitalWrite(BUZZER_PIN, LOW);

    for (int i = 0; i < 4; i++) {
        pinMode(motorPins[i], OUTPUT);
        digitalWrite(motorPins[i], LOW);
    }

    if (!keypad.begin()) {
        Serial.println("Keypad negasit! Verifica conexiunile.");
        while (1);
    }

    int savedPozitie;
    int savedDirectie;

    EEPROM.get(ADDR_POZITIE, savedPozitie);
    EEPROM.get(ADDR_DIRECTIE, savedDirectie);

    bool pozitieValida = (savedPozitie >= 0 && savedPozitie <=
BELT_LENGTH);

```

```

bool directieValida = (savedDirectie == 1 || savedDirectie == -1);

if (pozitieValida && directieValida) {
    pozitie = savedPozitie;
    stepsPerMlmtr = STEPS_PER_MM * savedDirectie;
    Serial.print("Pozitie restaurata: ");
    Serial.print(pozitie);
    Serial.println(" mm");
    Serial.print("Directie: ");
    Serial.println(savedDirectie == 1 ? "inainte" : "inapoi");
} else {
    pozitie = 0;
    stepsPerMlmtr = STEPS_PER_MM;
    Serial.println("Prima pornire, start de la 0.");
    saveState();
}

Serial.println("A=directie, B=stop/start, C=seteaza distanta,
D=seteaza pozitie");
}

void loop() {
    char key = 0;

    if (keypad.isPressed()) {
        uint8_t idx = keypad.getKey();
        if (idx != I2C_KEYPAD_NOKEY) {
            key = keys[idx];
        }
    }
}

```

```

}

if (key != 0 && lastKey == 0) {

    Serial.print("Tasta apasata: ");

    Serial.println(key);

    proceseazaTasta(key);

}

lastKey = key;

faUnPas();

}

```

## Exerciții și provocări / Exercises & challenges

### 1. Calculul pașilor pe mm / 1. Steps per mm calculation

**RO:** Motorul 28BYJ-48 are 32 de pași interni pe rotație și un raport de reducere 64:1. Verifică:  $32 \times 64 = 2048$  pași/rotație. În modul half-step (8 pași/ciclu) și cu diametrul de angrenaj dat,  $STEPS\_PER\_MM = 51.2$ . Calculează diametrul rodanței folosite. **EN:** The 28BYJ-48 has 32 internal steps per revolution and a 64:1 gear ratio. Verify:  $32 \times 64 = 2048$  steps/revolution. In half-step mode (8 steps/cycle) with the given gear diameter,  $STEPS\_PER\_MM = 51.2$ . Calculate the pitch diameter of the gear used.

### 2. Schimbarea lungimii curelei / 2. Change belt length

**RO:** Modifică constanta  $BELT\_LENGTH$  de la 800 la 500 mm și  $STEP\_SIZE$  de la 50 la 25 mm. Ce se întâmplă cu frecvența bip-urilor la capăt de cursă? Ajustează logica de inversare a direcției. **EN:** Change the  $BELT\_LENGTH$  constant from 800 to 500 mm and  $STEP\_SIZE$  from 50 to 25 mm. What happens to the end-of-travel beep frequency? Adjust the direction-reversal logic accordingly.

### 3. Comandă Serial / 3. Serial control

**RO:** Extinde codul astfel încât comenzile A, B, C, D să poată fi trimise și prin Serial Monitor (9600 baud). La primirea comenzii 'S' prin serial, oprește imediat motorul și afișează poziția în mm. **EN:** Extend the code so that A, B, C, D commands can also be sent via Serial Monitor (9600 baud). Upon receiving the 'S' command via serial, immediately stop the motor and print the current position in mm.

#### 4. Viteză variabilă / 4. Variable speed

**RO:** STEP\_DELAY\_US controlează viteza motorului (implicit 1000 µs între paşi). Adaugă tastele 1, 2, 3 de pe tastatură pentru a selecta viteze de 500 µs, 1000 µs şi 2000 µs. Observă efectele asupra torsiunii. **EN:** STEP\_DELAY\_US controls motor speed (default 1000 µs between steps). Add keypad keys 1, 2, 3 to select speeds of 500 µs, 1000 µs, and 2000 µs. Observe the effects on torque.

#### 5. EEPROM şi restaurarea stării / 5. EEPROM & state restoration

**RO:** Desconectează Arduino-ul după ce motorul a ajuns la poziţia 400 mm. Reconectează-l şi verifică prin Serial Monitor că poziţia şi direcţia sunt restaurate corect din EEPROM. Adaugă un mesaj suplimentar de confirmare la pornire. **EN:** Disconnect the Arduino after the motor reaches position 400 mm. Reconnect it and verify through Serial Monitor that position and direction are correctly restored from EEPROM. Add an extra confirmation message on startup.

#### 6. Monitorizare Serial / 6. Serial Monitor

**RO:** Deschide Serial Monitor (Ctrl+Shift+M), setează viteza la 9600 baud. Apasă B pentru pornire şi observă mesajele de poziţie afisate la fiecare 50 mm. Interpretează mesajul "Distanţa parcursă: X mm. Oprit." **EN:** Open Serial Monitor (Ctrl+Shift+M), set baud rate to 9600. Press B to start and observe position messages printed every 50 mm. Interpret the message "Distanţa parcursă: X mm. Oprit."

#### Sfaturi importante / Important tips

**!!! Sfat / Tip:** Motorul 28BYJ-48 foloseşte modul half-step cu 8 paşi pe ciclu (secvenţa stepSequence[8][4]). Dacă ordinea pinilor este greşită, motorul va vibra fără să se rotească. / The 28BYJ-48 uses half-step mode with 8 steps per cycle (stepSequence[8][4] array). If the pin order is wrong, the motor will vibrate without rotating.

**!!! Sfat / Tip:** Nu alimenta motorul direct de la pinii de 5V ai Arduino-ului! Foloseşte întotdeauna bateria externă. Motorul poate consuma până la 240 mA, ceea ce poate deteriora placa Arduino. / Never power the motor directly from the Arduino 5V pin! Always use the external battery. The motor can draw up to 240 mA, which may damage the Arduino board.

**!!! Sfat / Tip:** Biblioteca I2CKeyPad necesită instalare manuală din Library Manager (autor: Rob Tillaart). Fără aceasta, codul nu se va compila. / The I2CKeyPad library must be installed manually from Library Manager (author: Rob Tillaart). Without it, the code will not compile.

**!!! Sfat / Tip:** Valoarea poziţiei şi direcţia sunt salvate în EEPROM după fiecare pas de 50 mm. La repornire, sistemul reia automat de la ultima poziţie cunoscută. / Position and direction are saved to EEPROM after each 50 mm step. On restart, the system automatically resumes from the last known position.

**!!! Sfat / Tip:** Asigură-te că masa comună (GND) este conectată între Arduino, modulul driver ULN2003 şi bateria externă. Fără aceasta, semnalele de control nu vor funcţiona corect. / Make sure the common ground (GND) is connected between the Arduino, the ULN2003 driver module, and the external battery. Without this, control signals will not work correctly.



# PROIECT FINAL: Conveyor Belt with Ejection Platform

**Project Objective:** Design and build an automated conveyor belt system controlled by an Arduino. The belt moves a platform across a 1,650 mm track using a stepper motor. When the platform reaches the end of the belt, a servo motor physically ejects an object. The system features a 4x4 keypad for user commands, a 16x2 LCD for live status feedback, a buzzer for audio alerts, and EEPROM memory to save the current position and direction across power cycles.

Required Parts List: 1 x **Arduino Uno** (or compatible ATmega328P board)

1 x **28BYJ-48 Stepper Motor** with ULN2003 driver module

1 x **SG90 / MG90S Servo Motor** (for the ejection arm)

1 x **16x2 LCD Display** (standard 4-bit parallel interface)

1 x **4x4 Matrix Keypad** (I2C via PCF8574, address 0x20)

1 x **Active Buzzer**

1 x **Breadboard** and jumper wires

1 x **9V power supply** (USB or external)

- Wiring Diagram: **LCD (LiquidCrystal, 4-bit mode):**
- RS → Pin 6
- E → Pin 7
- D4 → Pin 12 | D5 → Pin 13 | D6 → Pin 3 | D7 → Pin 5
- VSS, RW, K → GND | VDD, A → 5V (use 220 Ω resistor on A for backlight)
- Stepper Motor (via ULN2003): IN1 → Pin 8 | IN2 → Pin 9 | IN3 → Pin 10 | IN4 → Pin 11
- Motor VCC → 5V | GND → GND
- Servo Motor: Signal → Pin A0 | VCC → 5V | GND → GND
- I2C Keypad (PCF8574):
- SDA → A4 | SCL → A5 | VCC → 5V | GND → GND
- Confirm I2C address is 0x20 (all address jumpers open)
- Buzzer: Positive → Pin 4 | Negative → GND

## Project Code:

```
#include <EEPROM.h>
#include <Wire.h>
#include <I2CKeyPad.h>
#include <LiquidCrystal.h>
#include <Servo.h>
```

```

// ===== LCD =====
LiquidCrystal lcd(6, 7, 12, 13, 3, 5);
// ===== SERVO =====
Servo ejectServo;
const int SERVO_PIN = A0;
const int SERVO_IDLE = 0;
const int SERVO_THROW = 90;
// ===== MOTOR =====
const float STEPS_PER_MM = 51.2;
const int BELT_LENGTH = 1650;
const int STEP_SIZE = 50;
const int BUZZER_PIN = 4;
const int motorPins[4] = {8, 9, 10, 11};
const int stepSequence[8][4] = {
{1,0,0,0},{1,1,0,0},{0,1,0,0},{0,1,1,0},
{0,0,1,0},{0,0,1,1},{0,0,0,1},{1,0,0,1}
};
int currentStep = 0;
long stepsRamasi = 0;
int directiePas = 1;
float stepsPerMlmtr = STEPS_PER_MM;
int pozitie = 0;
bool running = false;
long distantaTarget = 0;
long distantaParcursa = 0;
bool setandDistanta = false;
bool setandPozitie = false;
String inputDistanta = "";
const int ADDR_POZITIE = 0;
const int ADDR_DIRECTIE = 2;
const int STEP_DELAY_US = 1000;
unsigned long lastStepTime = 0;
const uint8_t KEYPAD_ADDRESS = 0x20;
I2CKeyPad keypad(KEYPAD_ADDRESS);
char keys[] = "123A456B789C*0#D";
char lastKey = 0;
unsigned long lastLCDUpdate = 0;
// --- LCD update helpers ---

```

```

void updateLCD(String line1, String line2) {
  lcd.clear();
  lcd.setCursor(0, 0); lcd.print(line1);
  lcd.setCursor(0, 1); lcd.print(line2);
}

void afisareStatus() {
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("P:"); lcd.print(pozitie); lcd.print("mm ");
  lcd.print(running ? "RUN" : "STOP");
  lcd.setCursor(0, 1);
  lcd.print(stepsPerMlmtr > 0 ? "DIR:>" : "DIR:<");
  lcd.print(" T:"); lcd.print(distantaTarget - distantaParcursa);
}

void beep(int times, int duration) {
  for (int i = 0; i < times; i++) {
    tone(BUZZER_PIN, 400); delay(duration);
    noTone(BUZZER_PIN); delay(duration);
  }
}

void saveState() {
  EEPROM.put(ADDR_POZITIE, pozitie);
  int dir = (stepsPerMlmtr > 0) ? 1 : -1;
  EEPROM.put(ADDR_DIRECTIE, dir);
}

void aruncaObiect() { // Eject object via servo
  bool wasRunning = running;
  running = false;
  for (int i = 0; i < 4; i++) digitalWrite(motorPins[i], LOW);
  updateLCD("OBJECT ARRIVED", "EJECTING...");
  beep(1, 200);
  delay(1000);
  ejectServo.write(SERVO_THROW);
  delay(600);
  ejectServo.write(SERVO_IDLE);
  delay(500);
}

```



Cofinanțat de  
Uniunea Europeană



## Start în Robotică cu Erasmus+ 2025-1-RO01-KA122-VET-000340573

```
beep(2, 80);
updateLCD("EJECTION", "COMPLETE");

delay(700);
if (wasRunning) {
    running = true;
    if (stepsRamasi == 0)
        stepsRamasi = (long) (STEP_SIZE * STEPS_PER_MM);
    directiePas = (stepsPerMlmtr > 0) ? 1 : -1;
}
}

void opreste() { // Stop conveyor
    running = false; stepsRamasi = 0;
    beep(3, 300);
    updateLCD("CONVEYOR", "STOPPED");
}

void faUnPas() { // Execute one motor step
    if (stepsRamasi == 0) return;
    unsigned long now = micros();
    if (now - lastStepTime < STEP_DELAY_US) return;
    lastStepTime = now;
    currentStep = (currentStep + directiePas + 8) % 8;
    for (int i = 0; i < 4; i++)
        digitalWrite(motorPins[i], stepSequence[currentStep][i]);
    stepsRamasi--;
    if (stepsRamasi == 0) {
        int next = pozitie + (stepsPerMlmtr > 0 ? STEP_SIZE : -STEP_SIZE);
        distantaParcursa += STEP_SIZE;
        bool atEnd = false;
        if (next >= BELT_LENGTH) {
            next = BELT_LENGTH;
            aruncaObiect();
            stepsPerMlmtr = -STEPS_PER_MM;
            atEnd = true;
        }
    }
}
```



Cofinanțat de  
Uniunea Europeană



**Start în Robotică cu Erasmus+**  
**2025-1-RO01-KA122-VET-000340573**

```

} else if (next < 0) {
next = 0;
stepsPerMlmtr = STEPS_PER_MM;
atEnd = true;
}
pozitie = next;
saveState();
if (atEnd) beep(5, 80);
if (distanțaTarget > 0 && distanțaParcursa >= distanțaTarget) {
opreste(); return;
}
if (running) {
stepsRamasi = (long)(STEP_SIZE * STEPS_PER_MM);
directiePas = (stepsPerMlmtr > 0) ? 1 : -1;
}
}
}

void proceseazaTasta(char key) { // Handle keypad input
if (key == '*') { updateLCD("MANUAL EJECT","ACTIVATED");
aruncaObiect(); return; }
if (setandDistanța) {
if (key >= '0' && key <= '9') { inputDistanța += key; /* update LCD */
}
else if (key == 'C') {
long val = inputDistanța.toInt();
if (setandPozitie) {
if (val >= 0 && val <= BELT_LENGTH) { pozitie = (int)val; saveState();
}
setandPozitie = false;
} else {
if (val > 0) { distanțaTarget = val; distanțaParcursa = 0; }
}
inputDistanța = ""; setandDistanța = false; delay(1200);
}
}

```



Cofinanțat de  
Uniunea Europeană



## Start în Robotică cu Erasmus+ 2025-1-RO01-KA122-VET-000340573

```

else if (key == '#') { inputDistanța = ""; setandDistanța = false;
setandPozitie = false; delay(1000); }

return;
}

if (key == 'A') { // Reverse direction
stepsPerMlmtr = -stepsPerMlmtr;
directiePas = (stepsPerMlmtr > 0) ? 1 : -1;
saveState(); beep(2, 100);
} else if (key == 'B') { // Start / Stop toggle
running = !running; beep(1, 750);
if (running && stepsRamasi == 0) {
if (distanțaParcursa >= distanțaTarget) distanțaParcursa = 0;
stepsRamasi = (long) (STEP_SIZE * STEPS_PER_MM);
directiePas = (stepsPerMlmtr > 0) ? 1 : -1;
}
if (!running) stepsRamasi = 0;
delay(1000);
} else if (key == 'C') { // Set target distance
setandDistanța = true; setandPozitie = false; inputDistanța = "";
updateLCD("SET DISTANCE", "Enter mm");
} else if (key == 'D') { // Set current position
setandDistanța = true; setandPozitie = true; inputDistanța = "";
updateLCD("SET POSITION", "Enter mm");
}
}

void setup() {
Serial.begin(9600); Wire.begin();
lcd.begin(16, 2);
updateLCD("CONVEYOR", "BOOTING...");
pinMode(BUZZER_PIN, OUTPUT); digitalWrite(BUZZER_PIN, LOW);
for (int i = 0; i < 4; i++) { pinMode(motorPins[i], OUTPUT);
digitalWrite(motorPins[i], LOW); }
ejectServo.attach(SERVO_PIN); ejectServo.write(SERVO_IDLE);
if (!keypad.begin()) { lcd.clear(); lcd.print("KEYPAD ERROR");
while(1); }

```



Cofinanțat de  
Uniunea Europeană



## Start în Robotică cu Erasmus+ 2025-1-RO01-KA122-VET-000340573

```
int savedPos, savedDir;

EEPROM.get(ADDR_POZITIE, savedPos); EEPROM.get(ADDR_DIRECTIE,
savedDir);

if (savedPos >= 0 && savedPos <= BELT_LENGTH && (savedDir == 1 ||
savedDir == -1)) {
pozitie = savedPos; stepsPerMlmtr = STEPS_PER_MM * savedDir;
} else { pozitie = 0; stepsPerMlmtr = STEPS_PER_MM; saveState(); }
updateLCD("SYSTEM READY", "READY"); delay(1500);
}

void loop() {
char key = 0;
if (keypad.isPressed()) {
uint8_t idx = keypad.getKey();
if (idx != I2C_KEYPAD_NOKEY) key = keys[idx];
}
if (key != 0 && lastKey == 0) proceseazaTasta(key);
lastKey = key;
faUnPas();
if (millis() - lastLCDUpdate > 400 && !setandDistanta) {
afisareStatus(); lastLCDUpdate = millis();
}
}
```