Rubik Cube Solver

Introducere

Proiectul constă într-un robot capabil să recunoască culorile fețelor unui Cub Rubik și să le reasambleze în ordinea corectă. Acesta funcționează ca o punte între lumea digitală și cea fizică, primind instrucțiuni și transformându-le în mișcări reale pentru a rezolva cubul.

Scopul principal este realizarea unui sistem interactiv care să poată demonstra vizual etapele de rezolvare ale cubului. Nu este doar o mașinărie care face totul singură, ci un dispozitiv care permite utilizatorului să urmărească și să controleze întreg procesul, pas cu pas.

Totul a plecat de la dorința de a vedea cum concepte precum inteligența artificială pot fi aplicate într-un obiect tangibil. Am pornit de la ideea unui robot care să poată comunica cu noi: el ne spune ce vede pe cub, iar noi îi putem spune ce mișcări să facă, evitând astfel o funcționare rigidă și complet automată.

Pentru mine, utilitatea este practică: învăț să stăpânesc interacțiunea dintre senzori și partea mecanică într-un mod fluid. Pentru ceilalți, robotul are un rol strict didactic. Este mult mai simplu să înveți să rezolvi cubul urmărind o demonstrație fizică, executată în timp real, decât încercând să descifrezi scheme abstracte dintr-o carte sau de pe internet.

Descriere generală

PC (Interfața de control): Reprezintă nivelul superior de decizie. Acesta rulează algoritmii de rezolvare a cubului, calculează soluția optimă și transmite instrucțiunile către microcontroler prin protocolul UART.

ATmega328P (Microcontrolerul): Este unitatea centrală de procesare. Acesta acționează ca un interpret între software și hardware: „traduce” comenzile primite prin interfața serială în semnale de comandă pentru motoare și gestionează interogarea senzorilor.

Senzori C (Senzorii de culoare): Șase unități de citire care identifică culorile fețelor cubului. Aceștia convertesc lumina reflectată în date digitale ce sunt transmise către microcontroler prin bus-ul de date I2C.

Modulul I2C (Multiplexor): Deoarece senzorii partajează aceeași adresă sau sunt numeroși, acest modul permite gestionarea lor eficientă folosind doar doi pini de date (SDA/SCL), optimizând astfel fluxul de informații și cablajul.

CNC Shield: Plăci de extensie ce facilitează distribuția semnalelor către drivere. Acestea organizează

pinii de control, permițând microcontrolerului să gestioneze simultan cele 6 motoare necesare proiectului.

Drivere: Module de putere care primesc semnalele logice de joasă tensiune de la microcontroler și livrează curentul necesar motoarelor. Acestea interpretează impulsurile primite pentru a executa pașii fizici.

Motoare Stepper: Actuatori de precizie care rotesc fețele cubului. Controlul lor este realizat prin semnale de tip Step/Dir, generate software prin rutine de întreruperi (ISR) pentru a asigura un unghi de rotație exact (90 de grade sau 180 de grade).

LCD: Afișează feedback vizual și starea sistemului în timp real.

Sursa de tensiune & Distribuitorul de putere: Asigură energia necesară întregului ansamblu. Distribuitorul separă ramura de forță (pentru motoare) de cea de control, pentru a evita interferențele electrice.

Regulator de tensiune: Coboară și stabilizează tensiunea pentru componentele logice sensibile (microcontroler și senzori), protejându-le împotriva fluctuațiilor.

Cooler: Sistem de răcire activ care previne degradarea termică a driverelor și a regulatorului în timpul funcționării prelungite.

Hardware Design

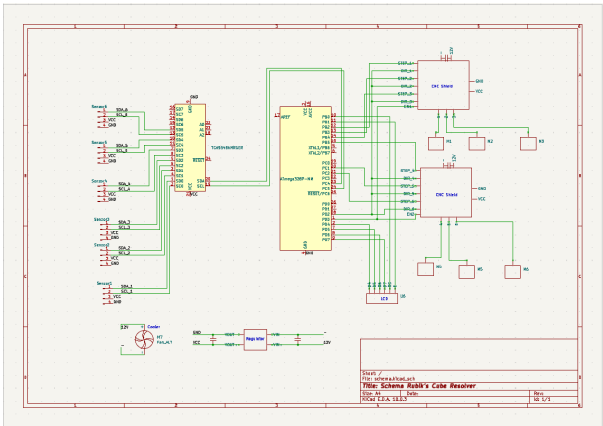
Listă de componente

Componentă	Cantitate	Categorie	Rol / Detalii tehnice
Motor Pas cu Pas Nema 17	6 bucăți	Mișcare / Actuatoare	Asigură acționarea mecanică directă a celor 6 fețe ale cubului.
Driver Motor A4988	6 bucăți	Mișcare / Actuatoare	Controlul precis al curentului și pașilor pentru fiecare motor Nema 17.
Shield CNC v3	2 bucăți	Mișcare / Actuatoare	Plăci de extensie pentru interfațarea și montarea rapidă a driverelor.
Senzor de Culoare TCS34725	6 bucăți	Senzori & Optică	Achiziția de date cromatice (RGB) de pe fiecare fațetă a cubului.
Multiplexor I2C (TCA9548A)	1 bucată	Senzori & Optică	Permite citirea simultană a celor 6 senzori prin izolarea aceleiași adrese I2C.
Placă Microcontroler ATmega328p	1 bucată	Control / Logică	Unitatea centrală de procesare (ex: Arduino Uno) care rulează algoritmul.
Placă distribuție putere (9 cai)	1 bucată	Alimentare / Protecție	Sistem de distribuție 12V echipat cu siguranțe fuzibile pentru protecție la scurtcircuit.
Modul Buck Converter	1 bucată	Alimentare / Protecție	Coborâtor de tensiune reglabil (1.25V - 37V) pentru stabilizarea liniei de logică (5V).

Sursă de alimentare	1 bucată	Alimentare / Protecție	Sursa principală de energie, asigură tensiunea necesară motoarelor (max. 30V).
Condensatori Electrolitici	4 bucăți	Alimentare / Protecție	Filtrarea tensiunii și absorbția vârfurilor de curent pentru protecția driverelor.
Ventilator / Cooler (12V)	1 bucată	Alimentare / Protecție	Răcirea activă a driverelor A4988 pentru prevenirea protecției termice.
Breadboard	1 bucată	Prototipare	Placă de test pentru realizarea rapidă a conexiunilor electrice fără lipire.
Cub Rubik	1 bucată	Altele	Obiectul fizic central supus procesului de scanare și rezolvare mecanică.

Analiza Circuitului Electronic și Logica Conexiunilor

Implementarea fizică a sistemului presupune interconectarea modulelor electrice. Configurația aleasă asigură imunitate la perturbații electromagnetice. De asemenea, circuitul garantează o rată optimă de transfer pe magistrale. Conexiunile au fost structurate pe blocuri funcționale clare. Fiecare nod de legătură a fost analizat critic din punct de vedere al semnalului.



Discuția 1: Cum am rezolvat conflictul adreselor la senzori
Când am început proiectul, am observat o mare problemă. Toți cei 6 senzori de culoare au din fabrică exact aceeași adresă hardware. Din acest motiv, nu am putut să îi leg în paralel direct la microcontroler. Ca să rezolv asta, am introdus în schemă multiplexorul TCA9548A. Acesta funcționează ca un comutator digital pornit din cod. Acum, eu selectez pe rând câte un canal și citesc fețele cubului una câte una, fără erori.

Discuția 2: Separarea liniilor de Logică și de Forță
Am observat rapid că motoarele pas cu pas consumă foarte mult curent. În plus, ele produc un zgomot electric masiv în circuit. Acest zgomot îmi reseta sau îmi bloca microcontrolerul în timpul testelor. Ca să scap de problemă, am separat complet traseul de alimentare al motoarelor de cel al logicei. Am alimentat motoarele direct din sursa mare. Componentelor sensibile le-am dat 5V curați dintr-un Buck Converter. Am unit doar firele de masă (GND) ca referință.

Discuția 3: Protecția driverelor cu condensatori
La porniri și opriri brute, am văzut că motoarele generează vârfuri inverse de tensiune. Aceste spike-uri pot prăji instantaneu tranzistorii scumpi din drivere. Pentru a-mi proteja piesele, am montat condensatori electrolitici în paralel pe alimentarea fiecărui driver. Eu îi văd ca pe niște baterii tampon de foarte mare viteză. Ei absorb imediat acele vârfuri periculoase și îmi mențin tensiunea stabilă pe

linie.

Discuția 4: Transmiterea comenzilor de mișcare (STEP / DIR)
Am realizat din start că microcontrolerul meu nu are destulă putere ca să învârtă direct un motor Nema 17. Din acest motiv, am ales să controlez totul indirect prin driverele A4988. Eu trimit de pe pini doar două semnale logice slabe. Cu pinul **DIR** aleg în ce parte se învârte motorul (stânga sau dreapta). Prin pinul **STEP** trimit impulsuri rapide, iar frecvența lor îmi dictează viteza de rotație.

Discuția 5: De ce am folosit Shield-urile CNC
Când am numărat pinii de pe placa mea Arduino Uno, am văzut că nu am suficienți pini fizici pentru 6 drivere. Soluția mea a fost să folosesc două plăci de extensie Shield CNC v3. Aceste plăci se montează direct peste microcontroler. Ele îmi multiplică și îmi rutează curat toate legăturile. Așa am scăpat de fire lungi și am redus riscul de interferențe.

Discuția 6: Cum am rezolvat supraîncălzirea (Managementul termic)
În timpul funcționării, am observat că driverele A4988 se încingeau extrem de puternic. Dacă depășeau o anumită temperatură, ele intrau singure în protecție și îmi opreau robotul în mijlocul mișcării. Ca să rezolv asta, am integrat în schemă un ventilator de 12V. L-am pus să bată direct pe radiatoarele lor. Fluxul continuu de aer le ține reci și sistemul meu merge stabil ore în șir.

Discuția 7: Organizarea firelor cu distribuitorul de putere
Aveam foarte multe componente care cereau alimentare la 12V și nu doream să fac un nod aglomerat de fire. Am decis să folosesc o placă de distribuție a puterii cu 9 căi. Cu ajutorul ei, am împărțit curentul de la sursa principală în mod curat și organizat. Un mare avantaj pe care l-am observat sunt siguranțele fuzibile incluse. Ele îmi protejează piesele și izolează circuitul la orice scurtcircuit accidental.

Discuția 8: Economisirea pinilor prin legarea EN și DIR
Pentru că resursele de pini erau la limită, a trebuit să fiu inventiv. Am observat că pot grupa pinii de control de la drivere în paralel pe aceleași linii logice. Mai exact, am unit pinii de activare (**EN**) și direcție (**DIR**) în noduri comune. Deoarece motoarele mele merg în grupuri logice impuse de algoritm, această conexiune mi-a redus la jumătate pinii folosiți. Astfel, am eliberat pini vitali pentru senzorii de culoare.

Discuția 9: Cum am reglat curentul din potențiometre (VREF)
La primele teste, unele motoare se încingeau prea tare, iar altele pierdeau pași și stricau rezolvarea cubului. Am înțeles că trebuie să calibrez curentul trimis spre bobine. Am luat un multimetru și am început să măsor tensiunea VREF direct pe micile șuruburi (potențiometre) de pe drivere. Învârtind fin de ele, am limitat electronic curentul maxim. Acum am obținut cuplul perfect de rotație, iar motoarele rămân reci.

Software Design

Mediu de Dezvoltare și Configurare Platformă

Proiectul este dezvoltat pentru microcontrolerul **ATmega328P**, utilizând mediul de dezvoltare **PlatformIO** (ecosistemul atmelavr). Configurarea sistemului este definită prin următorii parametri:

* **Platformă:** atmelavr * **Placă de dezvoltare:** ATmega328P (utilizează protocolul xplainedmini ca interfață de upload prin avrdude) * **Framework:** arduino (folosit pentru structura nativă de compilare a fișierelor C) * **Frecvență procesor (F_CPU):** 16000000UL (16 MHz) * **Viteză Monitor Serial (UART):** 9600 bps * **Librării 3rd-party:** chris–a/Keypad

---

Algoritm și Structuri Core

Aplicația implementează controlul automatizat al unui robot pentru rezolvarea Cubului Rubik, dotat cu 6 motoare pas cu pas (câte unul dedicat fiecărei fețe) și 6 senzori de culoare independenți poziționați deasupra brațelor mecanice.

Maparea Mișcărilor (Singmaster Notation) Sistemul asociază caracterele primite prin UART cu funcțiile mecanice din firmware conform tabelului de execuție:

Notare Mutare	Funcție Firmware Asociată	Tip Acțiune Mecanică	Pași Executați
“U” / “U'” / “U2”	mutari.up_ceas, mutari.up_anticeas, mutari.up_dublu	Fața Superioară	50 (90°) / 50 (-90°) / 100 (180°)
“D” / “D'” / “D2”	mutari.down_ceas, mutari.down_anticeas, mutari.down_dublu	Fața Inferioară	50 (90°) / 50 (-90°) / 100 (180°)
“L” / “L'” / “L2”	mutari.left_ceas, mutari.left_anticeas, mutari.left_dublu	Fața Stângă	50 (90°) / 50 (-90°) / 100 (180°)
“R” / “R'” / “R2”	mutari.right_ceas, mutari.right_anticeas, mutari.right_dublu	Fața Dreaptă	50 (90°) / 50 (-90°) / 100 (180°)
“F” / “F'” / “F2”	mutari.front_ceas, mutari.front_anticeas, mutari.front_dublu	Fața Frontală	50 (90°) / 50 (-90°) / 100 (180°)
“B” / “B'” / “B2”	mutari.back_ceas, mutari.back_anticeas, mutari.back_dublu	Fața Posterioară	50 (90°) / 50 (-90°) / 100 (180°)

Protocolul de Comunicație UART Pentru viteză și eficiență mecanică, firmware-ul procesează direct două coduri de acțiune primitive primite de la PC:

1. **Codul '1' (Scanare Cub):** Pornește secvența de citire a configurării. Pentru fiecare din cele 6 fețe, senzorul citește rândul expus (cele 3 pătrățele cele mai apropiate), rotește fața la 90° și repetă algoritmul de 4 ori. La final, fața este readusă în poziția inițială. Firmware-ul returnează prin UART direct o matrice curată de text (6×9) conținând codurile de culoare stabilite.
2. **Codul '2' (Listă Mișcări):** Oprește bucla principală și așteaptă un șir lung de mutări (maxim 20 de mișcări, sub 60 de caractere), acceptând lipirea textului direct în UART. Spațiul (” ”) este tratat ca separator, iar caracterul Enter (\n sau \r) marchează finalul șirului. Mișcările sunt executate secvențial cu o pauză de stabilizare fizică între ele.

Surse și Funcții Implementate

Codul aplicației este modularizat în mai multe fișiere pentru a separa perifericele hardware de logica

de business și a permite un debugging facil:

- **config.h:** Centralizează maparea tuturor pinilor de direcție (DIR_M1 … DIR_M6) și pași (STEP_M1 … STEP_M6), controlul pinului de activare generală (EN_PIN pe PB0), adresa I2C hardware și frecvența F_CPU.
- **uart.h / uart.c:** Gestionează interfața serială hardware asincronă a microcontrolerului. Include funcțiile uart_init() pentru configurarea la 9600 bps (8N1), uart_receive() pentru recepție blocantă, uart_write_str() pentru transmitere text și uart_read_line() pentru bufferizarea șirului de mișcări până la Enter.
- **i2c.h / i2c.c:** Implementează protocolul de magistrală I2C Master pe pinii dedicați ai MCU-ului (PC4 → SDA, PC5 → SCL). Rulează funcțiile primitive de control la frecvența standard de 100kHz: i2c_start(), i2c_stop(), i2c_write(), i2c_read_ack() și i2c_read_nack().
- **senzor_culoare.h / senzor_culoare.c:** Gestionează inițializarea (tcs34725_init_sensor()) și citirea registrelor brute de date (Clear, Red, Green, Blue) de la senzorii optici punctuali TCS34725 prin intermediul funcției citeste_culoare_senzor().
- **Algoritmul de conversie:** Funcția determina_cod_culoare() implementează o logică bazată pe rapoarte cromatice (R/C, G/C, B/C). Deoarece senzorii sunt fixați pe brațe și nu pot fi centrați perfect pe fiecare pătrățel, acest algoritm elimină erorile cauzate de variațiile de distanță sau lumină ambientală. Funcția izolează și returnează doar codurile celor 6 culori țintă: 0 → Galben, 1 → Alb, 2 → Verde, 3 → Albastru, 4 → Roșu, 5 → Portocaliu.
- **motoare.h / motoare.c:** Configurează pinii I/O ca ieșiri și asigură liniștea totală în repaus prin trecerea EN_PIN în starea HIGH (motoare_init_pini()). Funcția step_motor() activează exclusiv motorul selectat, setează direcția și generează trenul de impulsuri cu un delay stabil de 800µs între pași. Funcția executa_lista_miscari() parsează lexical șirul primit prin UART folosind strtok(), rulând secvențial mutările cu un delay de 400ms între ele.
- **main.c:** Reprezintă punctul central de intrare al aplicației. Conține rutina scanare_completa_cub() pentru interogarea senzorilor și trimiterea matricei, precum și bucla infinită (while(1)) care citește asincron octetul de comandă de la PC și direcționează execuția către Sarcina 1 (Scanare) sau Sarcina 2 (Listă mișcări).
- **Codul sursă al proiectului (GitHub):** Repository Rubik-Cube-Solver

Rezultate Obţinute

Care au fost rezultatele obţinute în urma realizării proiectului vostru.

Concluzii

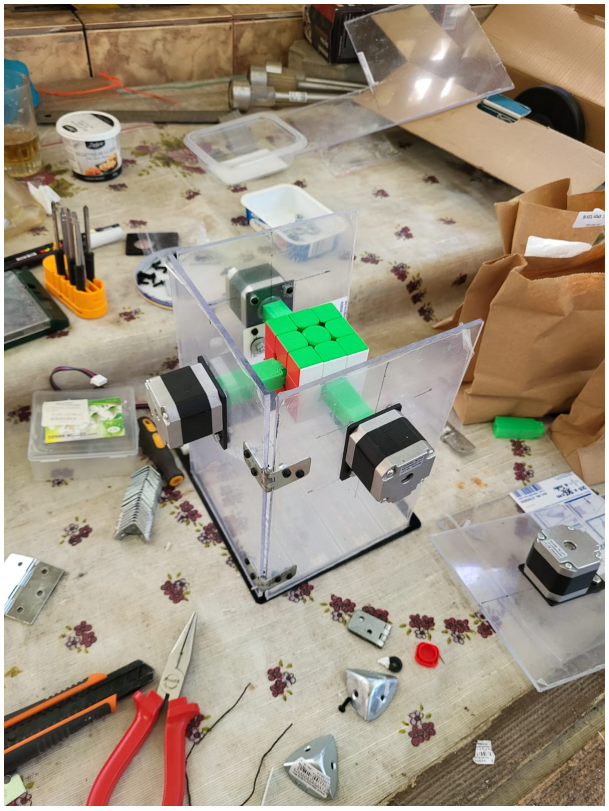
Download

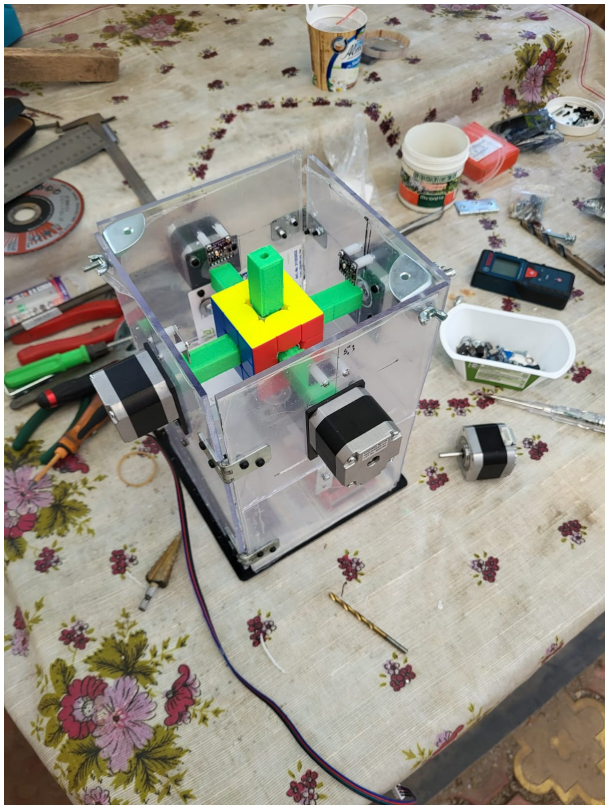
O arhivă (sau mai multe dacă este cazul) cu fişierele obţinute în urma realizării proiectului: surse, scheme, etc. Un fişier README, un ChangeLog, un script de compilare şi copiere automată pe uC

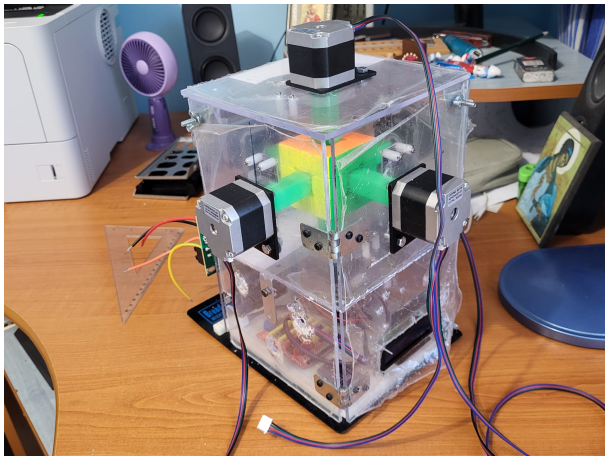
crează întotdeauna o impresie bună .

Fişierele se încarcă pe wiki folosind facilitatea **Add Images or other files**. Namespace-ul în care se încarcă fişierele este de tipul **:pm:prj20??:c?** sau **:pm:prj20??:c?:nume_student** (dacă este cazul). **Exemplu:** Dumitru Alin, 331CC → **:pm:prj2009:cc:dumitru_alin**.

Jurnal







Bibliografie/Resurse

Listă cu documente, datasheet-uri, resurse Internet folosite, eventual grupate pe **Resurse Software** şi **Resurse Hardware**.

Export to PDF