UNIVERSITATEA POLITEHNICA BUCUREȘTI

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE

DEPARTAMENTUL AUTOMATICĂ

|  |  |
| --- | --- |
| upb | cs |

PROIECT DE DIPLOMĂ

Robot autonom de urmărire a unei persoane în timp real

versiunea 2025

Ene Matei - Emanuel

**Coordonator științific:**

Prof. dr. ing. Bogdan Dumitrescu

BUCUREŞTI

2025

**CUPRINS**

[Sinopsis 2](#_Toc201020211)

[Abstract 2](#_Toc201020212)

[Mulțumiri 3](#_Toc201020213)

[1 Introducere 4](#_Toc201020214)

[1.1 Context 4](#_Toc201020215)

[1.2 Descrierea domeniului din care face parte tema de licență 4](#_Toc201020216)

[1.3 Obiective 4](#_Toc201020217)

[1.4 Structura lucrării 4](#_Toc201020218)

[2 DESCRIEREA PROBLEMEI ABORDATE 5](#_Toc201020219)

[2.1 Formularea problemei 5](#_Toc201020220)

[2.2 Studiu asupra realizărilor similare din domeniu 5](#_Toc201020221)

[2.3 Stabilirea cerințelor funcționale și nefuncționale ale sistemului 5](#_Toc201020222)

[3 stadiul actual în domeniu și selectarea soluției tehnice 6](#_Toc201020223)

[3.1 Stadiul actual al tehnologiilor utilizate pentru dezvoltarea soluției 6](#_Toc201020224)

[3.2 Prezentarea tehnologiilor și platformelor de dezvoltare alese 6](#_Toc201020225)

[4 CONSIDERENTE LEGATE DE IMPLEMENTAREA SOLuției tehnice 12](#_Toc201020226)

[4.1 Arhitectura și implementarea simulatorului 12](#_Toc201020227)

[4.2 Arhitectura și implementarea robotului 25](#_Toc201020228)

[5 STudiu de caz. 30](#_Toc201020229)

[6 Concluzii 30](#_Toc201020230)

[7 Bibliografie 30](#_Toc201020231)

[8 Anexe 30](#_Toc201020232)

# Sinopsis

Sinopsisul proiectului are rol de introducere, conținând atât o descriere pe scurt a problemei abordate cât și o enumerare sumară a rezultatelor și a concluziilor. Se recomandă ca sinopsisul să fie redactat într-un limbaj accesibil unei persoane nefamiliarizate cu domeniul, dar în același timp destul de specific pentru a oferi rapid o vedere de ansamblu asupra proiectului prezentat.

Sinopsisul proiectului va fi redactat atât în română cât și în engleză. Ca dimensiunea recomandată aceasta secțiune va avea maxim 200 de cuvinte pentru fiecare variantă. Împreună, ambele variante se vor încadra într-o singură pagină.

# Abstract

The abstract has an introductory role and should engulf both a brief description of the issue at hand, as well as an overview of the obtained results and conclusions. The abstract should be formulated such that even somebody that is unfamiliar with the projects’ domain can grasp the objectives of the thesis while, at the same time, retaining a specificity level offering a bird’s eye view of the project.

The projects’ abstract will be elaborated in both Romanian and English. The recommended size for this section is limited to 200 words for each version. Together, both versions will fit in one page.

# Mulțumiri

(opțional) Aici puteți introduce o secțiunea specială de mulțumiri / acknowledgments.

# Introducere

## Context

O scurtă introducere a proiectului, motivație, explicație de ce este relevant domeniul proiectului.

## Descrierea domeniului din care face parte tema de licență

Care este problema pe care proiectul o va rezolva?

## Obiective

Care sunt obiectivele proiectului/soluției/abordării/ideii; Ce creșteri sau evoluții determină rezolvarea proiectului.

## Structura lucrării

Un paragraf în care fiecare dintre secțiunile următoare este prezentată în 1-2 fraze, punând accentul pe elementele cele mai semnificative din fiecare secțiune.

# DESCRIEREA PROBLEMEI ABORDATE

## Formularea problemei

## Studiu asupra realizărilor similare din domeniu

## Stabilirea cerințelor funcționale și nefuncționale ale sistemului

# stadiul actual în domeniu și selectarea soluției tehnice

## Stadiul actual al tehnologiilor utilizate pentru dezvoltarea soluției

## Prezentarea tehnologiilor și platformelor de dezvoltare alese

În cadrul acestei secțiuni se va prezenta o analiză a tehnologiilor și a platformelor folosite pentru dezvoltarea proiectului

Tehnologiile au fost selectate în scopul de a permite modularitatea, performanța în timp real, accesibilitatea și integrarea între un mediu de simulare 3D și procesarea de imagine.

Prima etapă a proiectului a constat în dezvoltarea unui simulator de detecție și urmărire a unei persoane. Acest simulator servește drept model demonstrativ pentru robotul fizic ce urmează a fi construit și facilitează testarea, reglarea algoritmilor de viziune computerizată și ajustarea strategiilor de control, cu particularitate, regulatorul PID.

1. Motorul grafic Unity (C#)

Unity este un motor de jocuri utilizat pentru a dezvolta aplicații interactive 2D și 3D. Oferă un editor vizual, suport pentru scripting în C# și un sistem puternic de fizică și randare grafică în timp real. Acest motor grafic, a fost ales ca și platformă de simulare datorită suportului solid pentru randarea 3D, accesibilității limbajului C# pentru scrierea scripturilor, simplității utilizării mediului de lucru și bineînțeles datorită suportului oferit pentru simulări fizice.

Printre avantaje se numără următoarele:

* Compatibilitate cu mai multe platforme: Unity funcționează pe toate cele 3 platforme (Windows, macOS și Linux), întrucât am testat pe 2 dintre aceste platforme.
* Sprijin larg din partea comunității și acces la numeroase resurse (ex. Unity Asset Store)
* Programarea C#: C# este un limbaj de nivel înalt, orientat pe obiecte suport adecvat atât din punct de vedere al documentației cât și al ecosistemului bine pus la punct
* Randare în timp real cu o calitate grafică ridicată

Ca și principal dezavantaj este faptul că Unity nu este conceput special pentru robotică, fiind necesar un efort mai ridicat pentru a simula dinamica reală a robotului.

1. Python 3

Python este un limbaj de programare de nivel înalt și interpretat. Este folosit pe scară largă în dezvoltare software, automatizări, data science, inteligența artificială și aplicații web. Python a fost folosit în dezvoltarea clientului pentru procesarea de imagine datorită ecosistemului excelent de învățare autoamtă și vizune computerizată. În cadrul acestei tehnologii s-au utilizat biblioteci precum Ultralytics, folosind modelul YOLO pentru detecția de persoane în timp real și OpenCV, cefacilitează decodificarea, vizualizarea și prelucrarea imaginilor într-un mod eficient.

Printre avantaje se numără:

* Simplitate și rapiditate în prototipare: Python facilitează implementarea rapidă a algoritmilor
* Biblioteci variate ce permit o dezvoltare mai rapidă
* Oferă flexibilitate în experimentarea cu modele diferite și fluxuri de prelucrare a datelor

Iar ca dezavantaje:

* Execuție mai lentă: Python are o viteză mai redusă de execuție comparativ cu limbaje compilate precum C++
* Execuția în timp real impune limitări care necesită optimizări în cazul aplicațiilor unde rata cadrelor este importantă

1. Comunicare client-server prin socket-uri TCP

A diagram of a computer program

AI-generated content may be incorrect.

Protocolul TCP (Transmission Control Protocol) este un protocol de comunicație din cadrul TCP/IP care asigură transmiterea fiabilă și ordonată a datelor între două calculatoare. El stabilește o conexiune între emițător și receptor, controlează fluxul de date, asigurând corectitudinea și consistența acestora. Un socket TCP este un punct final într-o conexiune TCP. Socket-urile TCP au fost utilizate pentru a garanta o comunicare stabilă între simulatorul Unity și modulul de procesare a imaginilor scris în Python.

Ca și avantaje:

* Nu este dependent de o platformă, fiind compatibil cu toate sistemele de operare importante
* Oferă o interfață de programare (API) simplă ce permite o implementare rapidă cu ajutorul bibliotecilor existente atât in C# cât și în Python
* Transmisie fiabilă, mesajele fiind comunicate fără pierderi

Faptul că are un timp de răspuns mai ridicat în comparație cu protocolul UDP (User Datagram Protocol), aduce un dezavantaj acestui protocol.

Putem rezuma aceste alegeri corespunzătoare primei etape în următorul tabel:

|  |  |  |  |
| --- | --- | --- | --- |
| **Tehnologie** | **Justificare alegere** | **Avantaje** | **Dezavantaje** |
| Unity (C#) | Simulare 3D, sistem de fizică și randare 3D | Randare în timp real, simulare fizică și dezvoltare cu C# | Efort ridicat în simulare robotică |
| Python 3 | Procesare de imagine și detecția de persoane | Simplitate, rapiditate și flexibilitate în dezvoltare | Execuție mai lentă, necesită optimizări în cazul aplicațiilor unde contează rata cadrelor |
| Socket TCP | Comunicare între Unity și Python | Transmisie fiabilă, independență față de platforme | Timp de răspuns mai ridicat în comparație cu protocolul UDP |

Următoarea etapă a proiectului a constat în dezvoltarea fizică a robotului în scopul de a valida simulatorul realizat.

1. Raspberry Pi 5 (4GB RAM)

A close-up of a circuit board

AI-generated content may be incorrect.

Raspberry Pi 5 este o placă de dezvoltare de înaltă performanță, dezvoltat de Raspberry Pi Foundation, fiind proiectată pentru a oferi o platformă de calcul accesbiliă, compactă și eficientă energetic, capabilă să susțină aplicații complexe, de la sisteme embedded și Internet of Things (IoT) până la stații de lucru de tip desktop cu cerințe moderate. Această placă reprezintă “creierul” robotului si include următoarele specificații cheie: procesorul Arm Cortex-A76 pe 64 biți, cu patru nuclee și frecvența de ceas de 2.4 GHz, ce produce o îmbunătățire de aproximativ 2-3 ori mai bună față de generația anterioară, Raspberry Pi 4, lucru crucial pentru sarcina computațională ridicată impusă de procesarea de imagine. Pe partea de procesare grafică, Raspberry Pi 5 integrează un GPU VideoCore VII, cu frecvența de ceas de 800 MHz, lucru ce oferă îmbunătățiri din punct de vedere al procesării grafice și al sarcinilor de procesare paralelă. Deși nu conține un accelerator de inteligența artificială dedicat (precum un TPU), GPU-ul este utilizat în fluxurile de procesare a imaginilor. Din punct de vedere al memoriei, am optat pentru varianta de 4GB, memorie suficientă pentru rularea sistemului de operare (Raspberry Pi OS), interpretorului Python și a sarcinilor complexe de procesare. De asemenea, placa oferă interfețe de conectivitate și porturi I/O, cruciali pentru dezvoltarea robotului. Mai exact, găsim:

* 40 de pini GPIO (General-Purpose Input/Output) care permit interfațarea directă cu componente electronice precum drivere de motoare, senzori și LED-uri.
* 2 porturi MIPI CSI/DSI 4-lane, compatibile cu camere și ecrane ce necesită comunicație rapidă și latență scăzută
* Wi-Fi și Bluetooth
* 2 porturi USB 3.0 și 2 porturi USB 2.0 pentru conectarea perifericelor

Ca și avantaje:

* Raport performanță-cost ridicat
* Sistem de operare complet, bazat pe Linux, ce permite flexibilitate în instalare și configurare
* Comunitate și documentație vastă
* Funcționalități avansate de I/O

Iar ca și dezavantaje:

* Comparativ cu un microcontroller, PI 5 are un consum ridicat
* Procesorul produce temperaturi ridicate atunci când este solicitat

1. A small green computer chip with a white paper strip

   AI-generated content may be incorrect.Raspberry Pi Camera Module V2

Raspberry Pi Camera Module V2 este o cameră proiectată special pentru Raspberry Pi. Aceasta este o cameră compactă și ușoară ce conține un senzor de imagine CMOS de 8 megapixeli și interacționează direct cu hardware-ul Raspberry Pi prin intermediul portului CSI (Camera Serial Interface). Spre deosebire de camerele conectate prin USB, conexiunea CSI minimizează întârzierea dintre cadre.

Principalele avantaje sunt:

* Latența este redusă și lățimea de bandă este mare, oferind conexiune directă și rapidă către procesorul plăcii Raspberry Pi.
* Biblioteca *picamera2* oferă un suport software de calitate, simplificând conectarea camerei la aplicație
* Are un format compact și greutate redusă
* Raport calitate-preț al imaginii bun

Principalul dezavantaj se remarcă în medii slab iluminate unde performanța camerei este afectată, producând imagini zgomotoase, ceea ce compromite acuratețea detecției

1. Sistemul actuator: motoare DC și driver L298N

Un motor DC este cel mai simplist tip de motor electric, format dintr-un rotor și un stator cu magneți. Viteza de rotație a motorului este direct proporțională cu sarcina aplicată, însă acest lucru este controlat eficient folosind tehnica PWM (Pulse Width Modulation).

Pinii GPIO ai unui Raspberry Pi nu pot alimenta direct un motor. Aceștia funcționează la 3.3V și pot furniza doar un curent foarte mic (de ordinul miliamperilor), în timp ce motoarele DC necesită frecvent tensiuni între 6 și 12V și pot consuma sute de miliamper sau chiar câțiva amperi. Driverul de motor L298N acționează ca un intermediar – un comutator de putere ridicată, controlat de semnale logice de joasă putere ale Raspberry Pi-ului.

Ca și principale avantaje:

* Cost redus
* Logică de control simplă
* Comunitate de suport vastă

Iar principalul dezavantaj este de dat de faptul că este sensibil la scăderile de tensiune. Acestea determină o reducere a vitezei de deplasare.

Și aceste alegeri corespunzătoare ultimei etape le putem rezuma în următorul tabel:

|  |  |  |  |
| --- | --- | --- | --- |
| **Tehnologie** | **Justificare alegere** | **Avantaje** | **Dezavantaje** |
| Raspberry Pi 5 | Oferă puterea de procesare necesară pentru viziune artificială în timp real | Raport performanță-cost ridicat, suport al comunității ridicat | Consum ridicat, temperaturi ridicate |
| Raspberry Pi Camera V2 | Interfața CSI directă oferă o latență mică și un consum redus de resurse CPU | Latență redusă, suport prin biblioteca *picamera2*, compactă | Performanța scăzută în medii slab iluminate |
| Motoare DC | Reprezintă un echilibru bun între viteză de rotație continuă, cuplu și simplitate, pentru antrenarea roților | Simplu de controlat, accesibil ca preț și disponibil pe scară largă | Necesită un driver pentru control |
| Driver L298N | Oferă o soluție eficientă și bine susținută prin documentație, ce simplifica interfațarea pinilor GPIO ai plăcii Raspberry Pi cu motoarele DC | Cost redus, control logic simplu | Generează căldură considerabilă |

# CONSIDERENTE LEGATE DE IMPLEMENTAREA SOLuției tehnice

Trecerea de la proiectarea conceptuală la implementarea unui sistem robotic funcțional reprezintă o etapă critică în orice proiect de inginerie, un punct în care teoria se confruntă cu realitatea practică. Acest capitol are rolul de a detalia acest proces, explorând deciziile pragmatice și strategiile tehnice care au stat la baza dezvoltării robotului autonom. Aici sunt analizate nu doar alegerile fundamentale, precum selecția bibliotecilor software, ci și raționamentul din spatele metodelor de integrare a componentelor software și hardware și de reglare a algoritmilor de control. Se va evidenția nu doar "ce" s-a construit, ci, mai important, "de ce" și "cum" s-a ajuns la sistemul prezentat, oferind o perspectivă transparentă asupra provocărilor și compromisurilor inerente procesului de implementare.

## Arhitectura și implementarea simulatorului

Dezvoltarea oricărui sistem robotic autonom presupune o interdependență complexă între detecție, luarea deciziilor și acționare. Înainte de implementarea fizică a sistemului, o etapă preliminară esențială constă în crearea unui mediu de simulare cu acuratețe ridicată. În cadrul acestui proiect, s-a realizat un astfel de mediu, având mai multe scopuri fundamentale. În primul rând, acesta a oferit un spațiu de testare controlat, determinist și repetabil, necesar pentru dezvoltarea iterativă și validarea algoritmilor de viziune artificială și control. În al doilea rând, a permis ajustarea precisă a parametrilor controlerului Proporțional–Integral–Derivativ (PID), un proces care, în context fizic, este adesea consumator de timp și susceptibil la variații cauzate de hardware.

Cel mai important, această abordare bazată pe simulare a servit drept strategie esențială de reducere a riscurilor. Prin izolarea logicii de control față de hardware-ul fizic, eventualele erori arhitecturale sau de logică au putut fi identificate și corectate încă de la începutul dezvoltării. Pentru a face posibil acest lucru, a fost proiectată o arhitectură de tip Client–Server, care a permis decuplarea eficientă a „creierului” robotului de „corpul” acestuia.

Serverul, implementat folosind motorul Unity, funcționează ca un mediu digital al lumii reale. Acesta se ocupă de randarea mediului 3D, simularea mișcării robotului pe baza fizicii, capturarea imaginilor printr-o cameră virtuală și executarea comenzilor de mișcare.

Clientul, o aplicație scrisă în Python, reprezintă partea de inteligență a robotului. Primește fluxul video de la serverul Unity, folosește modelul YOLO pentru detecția persoanelor, procesează imaginile, calculează erorile, pe care le trimite înapoi către server pentru a fi folosite corespunzător în algoritmul de control. Acest model urmează paradigma clasică Software-in-the-Loop (SIL), în care software-ul final de control este testat riguros într-un sistem fizic simulat, pentru a-i verifica robustețea și eficiența înainte de a fi implementat pe hardware-ul real.

Interacțiunea dintre serverul Unity și clientul Python formează un sistem de control în buclă închisă, coordonat printr-o conexiune de rețea TCP/IP. Diagrama de mai jos ilustrează componentele fundamentale și fluxul de informații care definesc această arhitectură.

A diagram of a system

AI-generated content may be incorrect.

Procesul de funcționare începe în Unity, unde serverul capturează un cadru de la camera virtuală a robotului. Această imagine este codificată și transmisă prin rețea către clientul Python. Clientul decodează imaginea, aplică logica sa de percepție avansată pentru a identifica ținta și a determina poziția acesteia în raport cu poziția dorită. Pe baza acestei analize, calculează o eroare care este transmisă înapoi către serverul Unity. Serverul interpretează eroarea, calculează ieșirile necesare pentru motoare printr-un controler PID și aplică forțele rezultate asupra componentei Rigidbody a robotului simulat, generând mișcare. Această mișcare modifică perspectiva camerei, declanșând astfel următorul ciclu al buclei de control.

1. Componentele Serverului Unity

Serverul, construit în motorul Unity, are rolul de a crea un mediu fizic interactiv. Funcționalitatea acestuia este susținută de trei scripturi principale C#, care comunică împreună: TCPServer.cs, RobotController.cs și RobotPhysicsController.cs. Aceste scripturi sunt atașate unui GameObject ce reprezintă robotul într-o scenă 3D, scenă ce conține și o țintă pe care robotul trebuie să o urmărească.

Scriptul TCPServer.cs acționează ca punct unic de acces pentru toate datele care intră și ies din mediul de simulare. Responsabilitățile sale sunt esențiale în cadrul arhitecturii client–server. La inițializare, acesta stabilește programatic un TcpListener pe o adresă IP și un port ales —127.0.0.1:2737 pentru comunicare locală. Ulterior, începe să asculte conexiunile client în mod asincron, asigurând astfel că firul principal al aplicației rămâne neblocat și receptiv.

După stabilirea conexiunii cu un client, scriptul pornește două corutine paralele. Prima, SendImagesContinuously, rulează la un interval fix (de exemplu, 10 cadre pe secundă). La fiecare iterație, aceasta capturează ieșirea camerei virtuale a robotului dintr-un RenderTexture, o convertește într-un obiect Texture2D standard și apoi o codifică sub forma unui șir de octeți JPEG comprimat. Pentru a asigura o transmisie fiabilă, trimite mai întâi un întreg pe 4 octeți ce reprezintă dimensiunea datelor imaginii, urmat de datele imaginii propriu-zise. Acest protocol simplu, dar robust, permite clientului Python să știe exact câți octeți trebuie să primească pentru fiecare cadru.

A doua corutină, ReceiveAndProcessData, este activă în mod constant pentru a asculta eventualele date primite de la client. Aceasta folosește același protocol bazat pe dimensiune prefixată pentru a citi șirul de comandă de control. Un aspect esențial al designului său este gestionarea execuției multithread. Deoarece API-ul Unity nu este thread-safe și poate fi accesat doar din firul principal (main thread), comenzile primite prin rețea nu pot manipula direct obiecte din scenă. Pentru a rezolva această limitare, TCPServer.cs utilizează o coadă (\_mainThreadQueue). Atunci când o comandă este recepționată, acțiunea de procesare a acesteia este pusă în coadă. Metoda Update(), care rulează pe firul principal, extrage și execută aceste acțiuni într-un mod sigur și sincronizat. Acest design asigură stabilitatea aplicației și previne cazurile limită, formând o conexiune robustă între operațiile de rețea și logica motorului de joc.

Scriptul RobotController.cs funcționează ca nivel de control logic de înalt nivel al robotului simulat. Acesta primește șirul de comandă simbolică de la TCPServer și îl traduce în comenzi numerice concrete pentru controlerul fizic de nivel jos. Principala sa responsabilitate este interpretarea valorilor de eroare parțial analizate și aplicarea unui algoritm de control Proporțional–Integral–Derivativ (PID) pentru a calcula răspunsul corespunzător.

Comanda primită reprezintă eroarea dintre poziția persoanei țintă detectate și poziția dorită a persoanei în imagine (in pixeli). În cazul ideal, persoana țintă este centrată orizontal și vertical în limitele impuse. Inevitabil însă, persoana se va mișca, creând o discrepanță între starea curentă și starea dorită. Astfel, un controlerul în buclă închisă, măsoară continuu această eroare și calculează o comandă de corecție pentru a anula eroarea.

Schema buclei de reglare se poate vedea în figura de mai jos:

A diagram of a computer

AI-generated content may be incorrect.

Am folosit două controlere PID independente:

1. Controlerul responsabil de rotație. Acesta corectează eroarea pe axa orientală.
2. Controlerul responsabil de translație. Acesta corectează eroarea pe axa verticală.

Controlerul PID calculează o comandă de ieșire (output) pe baza a trei termeni, fiecare contribuind într-un mod unic la corectarea erorii, eroare calculată ca fiind diferența dintre poziția dorită și poziția actuală măsurată.

Formula de bază a unui controler PID este:

|  |  |  |
| --- | --- | --- |
|  |  | () |

Unde:

* e(t) reprezintă eroarea la momentul t
* reprezintă câștigul proporțional
* reprezintă câștigul integral
* reprezintă câștigul derivativ

Componenta proporțională este cea mai simplă și intuitivă. Ea generează o comandă de corecție care este direct proporțională cu mărimea erorii curente. Spre exemplu, dacă persoana este mult la stânga poziției dorite, eroarea este mare. Deci termenul P va genera o comandă de rotație puternică spre stânga. În sens opus, dacă persoana este puțin deplasată la stânga față de poziția dorită, sau este chiar în poziția dorită, eroarea este mică sau chiar nulă. Astfel termenul P va genera o comandă de rotație mai lentă la stânga, sau nu generează nicio comandă, în cazul în care eroarea este 0.

Avantajul principal al controlului proporțional este răspunsul rapid și imediat la orice eroare. Totuși, el are o limitare fundamentală: aproape întotdeauna lasă în urmă o eroare staționară. Robotul s-ar putea apropia de poziția dorită, dar se va opri într-o poziție unde forța de corecție generată de mica eroare rămasă nu mai este suficientă pentru a învinge inerția sau fricțiunea sistemului. Practic, nu ajunge niciodată exact la țintă, oscilând în mod continuu.

Componenta integrativă se ocupă exact de limitarea termenului P. Ea acumulează erorile din trecut. Avantajul principal este eliminarea erorii staționare, asigurând o precizie ridicată.

Componenta derivativă acționează ca o frână inteligentă. Ea nu se uită la eroarea curentă, ci la rata de schimbare a erorii. Practic, încearcă să prezică unde se va afla eroarea în viitorul apropiat. Spre exemplu, dacă robotul se deplasează prea rapid spre țintă, eroarea scade foarte repede. Termenul D detectează această scădere și generează o ieșire opusă mișcării (o frână), pentru a încetini apropierea și a „amortiza” astfel ajungerea la țintă.

Avantajul principal este reducerea semnificativă a depășirii (overshoot) și a oscilațiilor, stabilizând sistemul. Dezavantajul este că este foarte sensibil la zgomot (noise) în semnalul de intrare.

În alegerea parametrilor , și , am recurs la un proces de tuning iterativ cu scopul de a găsi un echilibru între un răspuns rapid și stabilitate, fără oscilații sau depășiri mari.

Pentru acest proces, am avut în vedere următoarele considerente din următorul tabel:

|  |  |  |  |
| --- | --- | --- | --- |
| **Parametru** | **Efectul creșterii valorii** | **Avantaje** | **Dezavantaje / Riscuri** |
|  | * Mărește viteza de răspuns * Reduce eroarea staționară (dar nu o elimină) * Crește overshoot-ul * Poate duce la oscilații și instabilitate | Reacție rapidă la erori. | Risc mare de instabilitate și overshoot. |
|  | * Elimină complet eroarea staționară * Crește semnificativ overshoot-ul * Poate degrada stabilitatea sistemului * Încetinește ușor răspunsul inițial | Precizie foarte mare pe termen lung. | Cauzează overshoot și poate destabiliza sistemul dacă este prea mare. |
|  | * Reduce overshoot-ul și oscilațiile * Crește stabilitatea sistemului * Permite folosirea unor valori mai mari pentru și * Poate încetini răspunsul la schimbări bruște | Sistem stabil, lin și fără overshoot. | Sensibilitate la zgomot. |

Am repetat acest proces pentru ambele controlere (rotație și translație), deoarece dinamica sistemului este diferită pe fiecare axă.

Scriptul RobotPhysicsController.cs reprezintă cel mai de jos nivel din ierarhia de control a simulării. Cum eroarea provenită este reprezentată de distanța în pixeli, scopul său este acela de a transforma comenzile numerice abstracte de viteză și direcție în mișcare fizică, utilizând motorul fizic integrat al Unity. Pentru a asigura calcule fizice deterministe și stabile, toată logica din acest script este executată în metoda FixedUpdate(), care este apelată la un interval de timp fix, sincronizat cu ciclul de actualizare al motorului fizic.

La pornire, scriptul obține o referință către componenta Rigidbody a robotului, care reprezintă elementul central al tuturor manipulărilor bazate pe fizică în Unity. Logica principală este împărțită în două funcții. Funcția ApplyMotorPhysics preia comanda de viteză (speedCommand) de la RobotController și simulează comportamentul unui motor electric. Aceasta normalizează comanda și calculează o viteză liniară țintă, dar în loc să aplice această viteză instantaneu, utilizează Mathf.Lerp pentru a interpola lin între viteza curentă și cea dorită. Astfel, sunt simulate efectele inerției și accelerației, rezultând o mișcare mult mai realistă și fluidă decât în cazul unei schimbări bruște de viteză.

În mod similar, funcția ApplySteeringPhysics procesează comanda de rotație (steeringCommand) pentru a calcula un unghi de direcție țintă. Și în acest caz, se folosește Mathf.Lerp pentru a realiza o tranziție lină de la unghiul de direcție curent al robotului către cel dorit, simulând astfel constrângerile fizice ale unui mecanism de direcție.

În final, funcția MoveRobot aplică valorile calculate asupra componentei Rigidbody. Mai întâi, calculează o rotație pe baza valorii currentSteeringAngle și o aplică folosind \_rBody.MoveRotation(). Apoi, determină vectorul de deplasare înainte în funcție de noua orientare a robotului și îl aplică utilizând \_rBody.MovePosition(). Utilizarea acestor metode este esențială, deoarece sunt concepute să funcționeze corect cu motorul fizic al Unity, gestionând corespunzător coliziunile și oferind un comportament mai stabil și previzibil decât manipularea directă a obiectului.

1. Componentele Clientului Python

Clientul Python funcționează ca bază computațional al sistemului autonom, acolo unde datele vizuale brute sunt transformate într-o estimare a erorii de poziționare față de țintă. Acesta rulează într-un mediu Python, utilizând biblioteci specializate în viziune computerizată și comunicații în rețea.

Funcționalitatea clientului se bazează pe un set de biblioteci open-source standard. Biblioteca socket oferă instrumentele de bază necesare pentru stabilirea și gestionarea conexiunii de rețea TCP cu serverul Unity. Elementul central al logicii de percepție este OpenCV (cv2), o bibliotecă extinsă pentru viziune computerizată, utilizată aici pentru decodificarea fluxului JPEG primit, conversia spațiului de culoare (din BGR în HSV pentru un filtru de culoare mai robust) și desenarea de vizualizări pe cadrele procesate. Deoarece OpenCV reprezintă imaginile sub formă de matrici NumPy, biblioteca NumPy devine o componentă esențială pentru toate manipulările numerice și la nivel de pixeli.

Capabilitatea de detecție a obiectelor este asigurată de biblioteca ultralytics, care oferă o interfață de nivel înalt și eficientă pentru rularea modelului de detecție YOLO (You Only Look Once).

În final, biblioteca struct este utilizată pentru codificarea și decodificarea datelor binare, în special pentru gestionarea prefixelor de tip întreg pe 4 octeți care indică dimensiunea mesajelor, asigurând respectarea strictă a protocolului de comunicare stabilit.

Scriptul main\_yolov11n.py acționează ca punct principal de intrare și coordonator al aplicației de pe partea client. Rolul său principal este de a gestiona comunicația prin rețea și de a coordona fluxul de date către și dinspre modulul de detecție.

La execuție, scriptul încearcă mai întâi să stabilească o conexiune TCP prin socket cu serverul Unity, folosind adresa și portul specificate. Odată stabilită conexiunea, este instanțiată clasa *ColorFollowerSmooth*, realizând astfel o separare clară între logica de gestionare a rețelei și algoritmii complecși de procesare a imaginilor. Această structură modular îmbunătățește mentenanța și lizibilitatea codului.

După ce datele imaginii au fost complet încărcate în memorie, acestea sunt transmise direct metodei processImage a obiectului follower. După ce modulul de percepție returnează un șir formatat reprezentând eroarea transmisa regulatorului PID, scriptul principal îl codifică în octeți, împachetează dimensiunea acestuia într-un întreg pe 4 octeți și trimite atât dimensiunea, cât și datele comenzii înapoi către serverul Unity.

Algoritmul de detecție și urmărire, implementat în clasa ColorFollowerSmooth, nu a fost construit dintr-o singură etapă, ci a evoluat treptat, prin îmbunătățiri succesive. Am urmat o abordare iterativă, în care fiecare componentă a fost testată și verificată înainte de a trece la următorul nivel de complexitate. Rezultatul final este o soluție hibridă echilibrată. Dezvoltarea a început cu partea de bază — detecția — și a fost extinsă treptat cu logică de control mai avansată și capacitatea de a diferenția ținta dorită.

Faza inițială a dezvoltării a fost dedicată stabilirii și validării capacității de detecție de bază a sistemului. Obiectivul principal a fost confirmarea fezabilității utilizării unui model de învățare profundă pentru detectarea în timp real a unei persoane.

În acest scop, am optat pentru un model din familia YOLO (You Only Look Once), mai exact varianta yolo11n.pt. Alegerea nu a fost întâmplătoare, ci dictată de cerințele impuse de procesare în timp real pe o platformă hardware cu resurse limitate, Raspberry Pi.

Modelul YOLOv11, dezvoltat de Ultralytics, reprezintă un salt semnificativ în evoluția detecției de obiecte în timp real. Spre deosebire de versiunile anterioare, care aduceau îmbunătățiri incrementale, YOLOv11 introduce o arhitectură cu două căi paralele, concepută special pentru a echilibra nevoia de viteză cu cea de acuratețe. Astfel, informația vizuală este procesată simultan printr-un model mare, axat pe precizie, și un model mic, optimizat pentru eficiență.

Componenta deosebită a modelului mare este integrarea unui model lingvistic-vizual (VLM), care îi oferă o înțelegere contextuală avansată, asemănătoare celei umane. Acest lucru îi permite să identifice obiecte chiar și în situații vizuale complexe. Legătura dintre cele două căi este realizată printr-un proces complex de transfer generativ de cunoștințe, prin care modelul mare — „profesorul” — transferă cunoștințele sale semantice către modelul mic — „elevul”. În urma acestui proces, modelul mic devine capabil să funcționeze eficient pe dispozitive cu resurse limitate, fără a compromite semnificativ acuratețea sau capacitatea de generalizare.

A graph with colored lines

AI-generated content may be incorrect.Această abordare hibridă se traduce în performanțe superioare: YOLOv11 depășește versiunile anterioare printr-o acuratețe mai mare (mAP), folosind în același timp mai puțini parametri și o putere de calcul redusă (FLOPs). Astfel, utilizarea modelului YOLOv11 pe platforma Raspberry Pi este ideală, oferind un echilibru excelent între latență scăzută — necesară pentru o urmărire în timp real — și o inteligență contextuală avansată, esențială pentru detecții precise în medii dinamice. Astfel, performanța este maximizată în limitele resurselor hardware disponibile.

[https://docs.ultralytics.com/models/yolo11/#supported-tasks-and-modes]

Indiferent de versiune, arhitectura oricărui model YOLO modern este structurată în trei componente principale, care lucrează în serie pentru a transforma o imagine de intrare într-o listă de obiecte detectate.

1. Backbone-ul joacă rolul „ochilor” rețelei — este responsabil pentru procesarea imaginii de intrare și extragerea de caracteristici vizuale la diferite niveluri de abstractizare.
2. Neck-ul funcționează asemenea unei „măduve spinării” — preia hărțile de caracteristici generate de Backbone și le combină într-un mod inteligent, generând reprezentări bogate care facilitează detecția obiectelor de dimensiuni variate.
3. Head-ul reprezintă „creierul decizional” al arhitecturii — utilizează informațiile agregate de Neck pentru a produce predicțiile finale: coordonatele căsuțelor de delimitare (bounding boxes), scorurile de încredere și probabilitățile de clasificare.

A computer screen shot of a computer program

AI-generated content may be incorrect.

[https://medium.com/@nikhil-rao-20/yolov11-explained-next-level-object-detection-with-enhanced-speed-and-accuracy-2dbe2d376f71]

În această etapă preliminară, logica algoritmului a fost intenționat menținută minimalistă: primea un cadru video, folosea modelul YOLO pentru a detecta toate instanțele clasei „person” și evidenția vizual prima persoană detectată, prin desenarea unei căsuțe de delimitare (bounding box). Acest pas fundamental a demonstrat cu succes că modelul ales putea fi rulat eficient și era capabil să identifice persoane în cadrul fluxului video simulat. Deși a confirmat capacitatea sistemului de a „vedea”, a scos totodată la iveală și limitările sale: incapacitatea de a face diferența între mai multe persoane și lipsa totală a unei logici de control. În acel moment, sistemul se limita la a observa pasiv mediul, fără să intervină sau să influențeze ceea ce vedea.

Plecând de la această etapa de detecție deja validată, a doua fază a dezvoltării a introdus principiile controlului în buclă închisă. Scopul s-a mutat de la simpla detecție la urmărirea activă, prin cuantificarea poziției țintei și generarea unui semnal de eroare corectiv. Acest lucru a fost realizat prin definirea unei poziții dorite, în câmpul vizual al camerei și compararea continuă a acesteia cu poziția actuală a țintei. Logica de control a fost împărțită pentru a gestiona cele două axe principale de mișcare ale robotului. Pentru controlul rotației, a fost definită o zonă țintă în centrul orizontal al imaginii. Algoritmul calcula apoi centrul orizontal al căsuței de delimitare a persoanei detectate și determina eroarea ca fiind deplasarea față de poziția centrală dorită. Această valoare de eroare este folosită ulterior direct pentru a genera comenzile de direcție ale robotului, ghidându-l să vireze la stânga sau la dreapta pentru a menține ținta centrată.

Pentru controlul translației — adică al distanței robotului față de țintă — a fost necesară o abordare mai nuanțată. În loc să se bazeze pe poziția verticală absolută a căsuței de delimitare, care poate varia semnificativ, s-a optat pentru o metrică mai robustă: dimensiunea aparentă a țintei. Pe măsură ce robotul se apropie sau se îndepărtează de o persoană, dimensiunea căsuței de delimitare din imagine se modifică proporțional.

Prin urmare, obiectivul controlului a fost menținerea unei dimensiuni verticale constante a țintei. Pentru a standardiza această măsurătoare, coordonata superioară a căsuței (y1), care corespunde vârfului capului persoanei, a fost utilizată ca indicator stabil al distanței. A fost definită o zonă verticală ideală, iar eroarea a fost calculată ca diferența dintre coordonata y1 față și această zonă. Semnalul de eroare rezultat controlează ulterior mișcarea de înaintare și retragere a robotului, determinându-l să păstreze o distanță constantă față de țintă. Integrarea cu succes a acestor două mecanisme de control a transformat sistemul dintr-un simplu detector pasiv într-un robot capabil să urmărească o țintă izolată.

Totuși, această evoluție a scos în evidență o problemă esențială rămasă nerezolvată: ambiguitatea țintei. Într-un scenariu cu mai multe persoane, algoritmul nu dispunea de niciun mecanism prin care să distingă persoana urmărită, ceea ce ducea la un comportament haotic în cazul în care apar mai multe persoane în scenă, sistemul comutând aleatoriu între persoane.

Pentru a depăși această provocare, a fost introdusă în a treia etapă de dezvoltare, o componentă esențială: diferențierea țintei. Soluția a constat în completarea detecției generale de tip „person” oferită de YOLO cu o trăsătură identificabilă specifică — în acest caz, culoarea tricoului. Această extindere a transformat sistemul dintr-un simplu urmăritor de persoane într-un urmăritor dedicat unei ținte anume. S-a stabilit o convenție conform căreia persoana desemnată drept țintă trebuie să poarte o culoare verde vizibilă.

Algoritmul de percepție a fost reconstruit fundamental pentru a implementa această strategie hibridă. În loc să acționeze asupra primei detecții, algoritmul evaluează acum fiecare persoană detectată ca potențială candidată. Pentru fiecare căsuță de delimitare, este extrasă regiunea de interes (ROI), care este apoi convertită în spațiul de culoare HSV (Hue, Saturation, Value).

Spațiul HSV poate fi vizualizat ca un cilindru, unde fiecare componentă are o semnificație distinctă și intuitivă:

* Hue (H - Nuanța): Aceasta este componenta cea mai importantă pentru noi. Ea reprezintă culoarea pură și este măsurată ca un unghi pe un cerc de culori (de la 0 la 360 de grade). Roșul se află la 0 grade, verdele la 120, albastrul la 240, etc. Avantajul crucial este că nuanța unei culori (de exemplu, "verde") rămâne relativ constantă, indiferent dacă obiectul este puternic iluminat sau se află în umbră. În implementarea OpenCV, pentru a încăpea într-un singur byte (8 biți), acest interval este mapat la 0-179.
* Saturation (S - Saturația): Această componentă descrie intensitatea sau puritatea culorii. O saturație mare (valoare apropiată de 255) reprezintă o culoare vie, bogată. O saturație mică (valoare apropiată de 0) reprezintă o culoare ternă, spălăcită, care tinde spre o nuanță de gri. Această componentă ne permite să distingem între un verde vibrant și un gri-verzui.
* Value (V - Valoarea/Strălucirea): Această componentă reprezintă luminozitatea sau întunecimea culorii. O valoare mare (apropiată de 255) indică o culoare foarte luminoasă, care tinde spre alb. O valoare mică (apropiată de 0) indică o culoare foarte întunecată, care tinde spre negru.

A diagram of a color chart

AI-generated content may be incorrect.

O abordare clasică, bazată pe filtrarea în spațiul de culoare standard RGB (Red, Green, Blue), s-ar fi dovedit ineficientă. Motivul constă în modul în care modelul RGB reprezintă culoarea: fiecare pixel este definit de o combinație a intensității celor trei canale de lumină primare. În acest model, informația despre culoare (nuanța) și informația despre luminozitate (strălucire) sunt strâns interconectate. O cămașă verde în lumină puternică și aceeași cămașă verde în umbră ar avea valori RGB diferite, făcând extrem de dificilă definirea unui singur set de praguri care să funcționeze în ambele scenarii. Această sensibilitate la condițiile de iluminare face ca modelul RGB să fie nepotrivit pentru aplicații robuste de viziune computerizată.

Astfel, spațiul HSV de culoare este deosebit de avantajos pentru analiza culorii, deoarece componenta Hue (nuanța) rămâne în mare parte constantă chiar și în condiții variabile de iluminare.

[https://medium.com/@venkatpendela96/rgb-vs-hsv-52e55e504120]

În acest spațiu de culoare, a fost generată o mască pentru a izola toți pixelii care se încadrează într-un interval predefinit pentru culoarea verde. Pentru fiecare candidat, s-a calculat o metrică cantitativă — raportul de pixeli verzi— ce reprezintă densitatea pixelilor verzi din interiorul regiunii de interes (ROI). Astfel, logica de selecție a algoritmului a fost transformată: sistemul identifică acum candidatul cu cea mai mare densitate de pixeli verzi. Pentru a preveni detecțiile eronate, a fost aplicat un prag minim, asigurând că doar un candidat cu o cantitate semnificativă din culoarea țintă este considerat valid.

Această combinație între detecția pe bază de formă, oferită de rețeaua neuronală, și analiza pe bază de trăsături — în acest caz, culoarea — a conferit robotului abilitatea esențială de a identifica într-un mod aproximativ unic (exceptând, desigur, cazul în care apar diferite persoane cu un raport de culoare verde mai ridicat) și de a se fixa pe ținta sa, chiar și într-un mediu aglomerat.

Ultima îmbunătățire, care a abordat problema blocaje vizuale temporare sau a eșecurilor de detecție, a fost implementarea unui mecanism de netezire temporală. Prin reținerea ultimei poziții valide a țintei (self.prev\_bbox), algoritmul a dobândit o formă de memorie pe termen scurt, care i-a permis să își mențină fixarea și să compenseze întreruperile scurte ale contactului vizual cu ținta dorită, realizând astfel comportamentul de urmărire robust și fluid dorit pentru sistemul final.

TO DO: O analiza a performantei

Arhitectura de simulare client-server utilizată în acest proiect s-a dovedit a fi un element esențial în procesul de dezvoltare al robotului autonom. Prin separarea clară între detcție, control și simularea fizică, aceasta a permis un proces de dezvoltare rapid, iterativ și cu un grad redus de risc. Datorită acestei separări, componentele sistemului au putut fi testate și îmbunătățite independent — spre exemplu, diferite modele de detecție a obiectelor sau metode de filtrare a culorii au putut fi evaluate direct în clientul Python, fără a necesita intervenții asupra codului serverului Unity.

Simulatorul a reprezentat un mediu de testare digital esențial. În acest cadru controlat au fost calibrați parametrii de bază ai controlului PID și a fost verificată eficiența algoritmului de urmărire hibrid (YOLO combinat cu filtrarea pe bază de culoare). Activitatea desfășurată în cadrul simulatorului a oferit o înțelegere valoroasă asupra comportamentului dinamic al sistemului și a dificultăților generate de latența comunicației.

Prin urmare, dezvoltarea acestui simulator a redus considerabil riscurile tranziției către implementarea fizică. Logica de control, implementată în Python, a fost concepută pentru a fi portabilă, iar rezultatele obținute în simulare au oferit încredere că aceasta va funcționa eficient și pe platforma hardware reală. Etapa de simulare a constituit, astfel, o fundație solidă și testată pentru realizarea prototipului fizic final.

## Arhitectura și implementarea robotului

După validarea cu succes a algoritmilor de control și a logicii sistemului în cadrul simulatorului, proiectul a avansat către faza de implementare fizică. Această etapă esențială presupune transpunerea conceptelor abstracte și a modulelor software dezvoltate în simulare într-un sistem mecatronic funcțional. Arhitectura robotului fizic reflectă structura logică a simulatorului, menținând aceleași principii fundamentale: detecție, procesare și acționare. Diferența majoră constă în înlocuirea componentelor virtuale și a comunicației în rețea cu hardware real și interfețe electrice directe.

Întregul sistem este coordonat de un calculator compact, Raspberry Pi 5, care îndeplinește rolul de unitate centrală de procesare. Acest mini-computer puternic rulează „creierul” robotului — aceeași aplicație Python dezvoltată și îmbunătățită în etapa de simulare. Ansamblul hardware include un modul de cameră Raspberry Pi Camera Module V2 pentru percepție și un driver de motoare L298N care controlează două motoare DC pentru acționare.

Diagrama de mai jos oferă o imagine de ansamblu asupra componentelor hardware și software implicate, precum și asupra relațiilor de comunicare directă dintre acestea.

A diagram of a computer

AI-generated content may be incorrect.

Iar mai jos se află schema circuitului.

A circuit board with wires connected to it

AI-generated content may be incorrect.

Bucla de control din sistemul fizic este implementată într-un mod direct și integrat. Camera Pi captează imaginea și o transmite prin interfața CSI cu lățime mare de bandă către Raspberry Pi. Pe această placă rulează un singur script Python unificat, care prelucrează imaginea utilizând logica validată anterior — detecția de obiecte prin YOLO și filtrarea cromatică. Ulterior, același script calculează acțiunea corectivă necesară folosind un algoritm PID și generează semnale electrice (PWM și digitale) prin intermediul pinilor GPIO. Aceste semnale sunt transmise către driverul L298N, care le amplifică pentru a comanda motoarele DC, rezultând în mișcarea robotului. Schimbarea perspectivei camerei, determinată de deplasarea robotului, închide astfel bucla de control.

Funcția de percepție vizuală a robotului este realizată de modulul Raspberry Pi Camera V2, echivalentul fizic al camerei virtuale și al RenderTexture-ului din simularea Unity. Alegerea acestei camere a fost motivată de integrarea nativă cu ecosistemul Raspberry Pi. În locul unei conexiuni USB convenționale, camera utilizează interfața dedicată CSI (Camera Serial Interface), care oferă o cale de transfer rapidă și directă către procesor. Această conexiune reduce semnificativ latența la captarea imaginilor, spre deosebire de protocoalele USB, mai lente și mai solicitante pentru CPU. În plus, prin utilizarea CSI, o mare parte din sarcina de transfer este preluată la nivel hardware, permițând procesorului să aloce mai multe resurse pentru rularea algoritmilor de viziune computerizată.

Din punct de vedere software, controlul camerei este realizat prin intermediul bibliotecii Picamera2, direct din scriptul principal Python. Această bibliotecă modernă oferă o interfață eficientă și flexibilă pentru setarea parametrilor camerei, cum ar fi rezoluția și numărul de cadre pe secundă, precum și pentru captarea cadrelor sub formă de array-uri NumPy. Utilizarea acestui format nativ permite o integrare directă și rapidă cu OpenCV, eliminând copierea intermediară a datelor și asigurând un flux de procesare performant — de la captură la analiză.

Raspberry Pi 5, împreună cu platforma software asociată, joacă rolul de „creier” al robotului. Acest modul reprezintă echivalentul fizic al clientului Python și al unor componente din serverul Unity, reunite acum într-o singură unitate de procesare compactă și integrată. Datorită procesorului quad-core Arm Cortex-A76, Raspberry Pi 5 oferă puterea de calcul necesară pentru a susține întreaga buclă de control — de la captarea imaginilor, la rularea modelului YOLO și calculul comenzilor — în timp real.

Mediul software este construit pe baza sistemului de operare oficial Raspberry Pi OS, o distribuție Linux stabilă, derivată din Debian. Aplicația principală constă într-un singur script Python, care orchestrează comportamentul complet al robotului. Acest script reutilizează aceleași biblioteci esențiale din faza de simulare — OpenCV, NumPy și Ultralytics YOLO — pentru realizarea sarcinilor de detecție. Logica implementată în clasa color\_follower\_smooth a fost portată direct, demonstrând valoarea abordării de tip „simulare mai întâi”. Diferența majoră constă în mecanismele de intrare și ieșire: în loc să primească imagini prin rețea, scriptul utilizează acum biblioteca picamera2 pentru a le capta direct de la dispozitivul fizic. Iar în loc să trimită comenzi prin socket TCP, scriptul controlează componentele hardware reale printr-o bibliotecă dedicată accesării pinilor GPIO.

Modulul de acționare are rolul de a transforma deciziile digitale luate de unitatea de procesare în mișcare fizică. Acesta reprezintă echivalentul real al componentelor Rigidbody și motorului fizic din simularea Unity și este alcătuit din două motoare DC și un driver L298N. Robotul utilizează o configurație clasică de viraj și deplasare, în care un motor DC controlează viteza de deplasare și acționează asupra roților cu unui sistem bazat pe roți dințate, cel de-al doilea motor DC controlează rotația robotului, acționând într-un mod similar cu primul motor. cele două seturi de roți sunt controlate independent.

Pinii GPIO ai plăcii Raspberry Pi funcționează la o tensiune joasă (3.3V) și pot furniza doar un curent foarte mic, ceea ce îi face inadecvați pentru alimentarea directă a unui motor. Driverul de motoare L298N joacă rolul esențial de intermediar de putere. Acesta este un circuit integrat de tip H-bridge dublu, care funcționează ca un set de comutatoare electronice controlate. El primește semnalele logice de joasă putere de la pinii GPIO ai plăcii și le folosește pentru a comuta o sursă de alimentare de tensiune mai mare (o baterie de 9V) către motoare.

Mecanismul de control are două componente principale. Viteza este reglată prin PWM (Pulse Width Modulation). Scriptul Python generează un semnal PWM pe pinii de „Enable” ai driverului (ENA și ENB). Prin varierea factorului de umplere al acestui semnal — procentul de timp în care semnalul este pe nivel logic HIGH — se controlează tensiunea medie aplicată motorului și, echivalent, viteza de rotație. Această combinație de control PWM și semnale digitale permite controlul complet și independent al celor două motoare.

Deci, un ciclu complet al buclei de control se desfășoară după următoarea succesiune de pași. Scriptul principal în Python inițiază ciclul solicitând bibliotecii picamera2 captarea unui cadru video. Acest cadru este livrat sub forma unui array NumPy direct către funcția de detecție. Modelul YOLO rulează procesul de detecție pe CPU-ul plăcii Raspberry Pi, identificând toate persoanele din imagine. Lista de detecții este apoi filtrată cu ajutorul algoritmului de verificare a culorii pentru a izola ținta specifică — persoana care poartă verde — iar mecanismul robust prev\_bbox asigură continuitatea urmăririi. Pe baza poziției țintei finale, controller-ele PID (implementate tot în cadrul aceluiași script) calculează comenzile corective necesare pentru viteză și direcție. Aceste comenzi abstracte sunt apoi traduse în semnale concrete de control hardware. O funcție de control al motoarelor calculează ciclurile de lucru PWM și stările digitale de direcție pentru fiecare motor. În final, biblioteca RPi.GPIO setează corespunzător pinii GPIO ai plăcii, iar driverul L298N interpretează aceste semnale, redirecționând curentul din baterie către motoarele DC. Robotul se mișcă, modificând astfel instantaneu câmpul vizual al camerei, iar ciclul se reia cu următorul cadru.

În concluzie, arhitectura fizică a robotului reflectă o implementare reușită și eficientă a conceptelor dezvoltate în mediul de simulare. Alegerea plăcii Raspberry Pi 5 s-a dovedit corectă, oferind puterea de procesare necesară pentru sarcinile de viziune computerizată în timp real. Combinația dintre camera CSI cu latență redusă și controlul direct și rapid prin interfața GPIO a permis realizarea unei bucle de control strânse și eficiente. Această tranziție de succes, de la un model digital la un prototip fizic funcțional, validează filosofia de proiectare adoptată și evidențiază un sistem mecatronic bine conceput și integrat.

# STudiu de caz

După analiza detaliata a fundamentelor teoretice și a detaliilor de implementare tehnică din capitolele anterioare, acest capitol este dedicat validării practice și demonstrării concrete a funcționalității sistemului dezvoltat. Acesta mută focusul de la arhitectura internă a codului la comportamentul observabil al aplicației, adoptând în mod explicit perspectiva utilizatorului final. Scopul acestui studiu de caz de a ilustra, într-un mod clar și secvențial, experiența interacțiunii cu robotul autonom și cu simulatorul său. Prin parcurgerea unui scenariu de utilizare tipic, de la inițializarea sistemului până la executarea sarcinii de urmărire într-un mediu dinamic, voi oferi o dovadă tangibilă a modului în care conceptele teoretice și soluțiile tehnice se materializează într-un produs final funcțional.

## Prezentarea simulatorului

Din perspectiva utilizatorului, pornirea simulării implică lansarea secvențială a celor două aplicații cheie: serverul Unity și clientul Python.

Primul pas constă în executarea aplicației server, compilată din proiectul Unity. La lansare, utilizatorul este întâmpinat de fereastra principală a simulatorului, așa cum este ilustrat în figura următoare.

A screenshot of a video game

AI-generated content may be incorrect.

Scena 3D prezentată este un mediu controlat, proiectat special pentru a testa și valida capabilitățile robotului.

Utilizatorul poate identifica imediat componentele esențiale:

* Robotul simulat**:** Reprezentat de un cub roșu, acesta este agentul autonom pe care îl vom controla, asupra căruia este atașată o cameră virtuală.
* Ținta desemnată**:** Un personaj umanoid, distinct prin culoarea verde a vestimentației, este plasat în proximitatea robotului. Acesta reprezintă ținta pe care robotul este programat să o urmărească.
* Elemente de distragere**:** În fundal, se poate observa un grup de alte personaje umanoide, îmbrăcate în culori neutre (roșu). Prezența acestora este deliberată are scopul de a demonstra, în etapele ulterioare, capacitatea algoritmului de a distinge corect între ținta desemnată și alte persoane din mediul înconjurător.

În acest stadiu inițial, robotul este staționar, iar în fundal, componenta server TCP/IP așteaptă în mod pasiv o conexiune de la clientul Python pentru a iniția bucla de control.

De asemenea, se poate observa capacitatea simulatorului de a expune parametrii cheie direct în interfața grafică a editorului. Această abordare permite o reglare (tuning) a comportamentului robotului în timp real, fără a necesita modificarea codului sursă și recompilarea aplicației. Utilizatorul poate ajusta valorile direct din ferestrele "Inspector" pentru scripturile atașate robotului, așa cum se poate observa în figurile de mai jos.

A screenshot of a computer

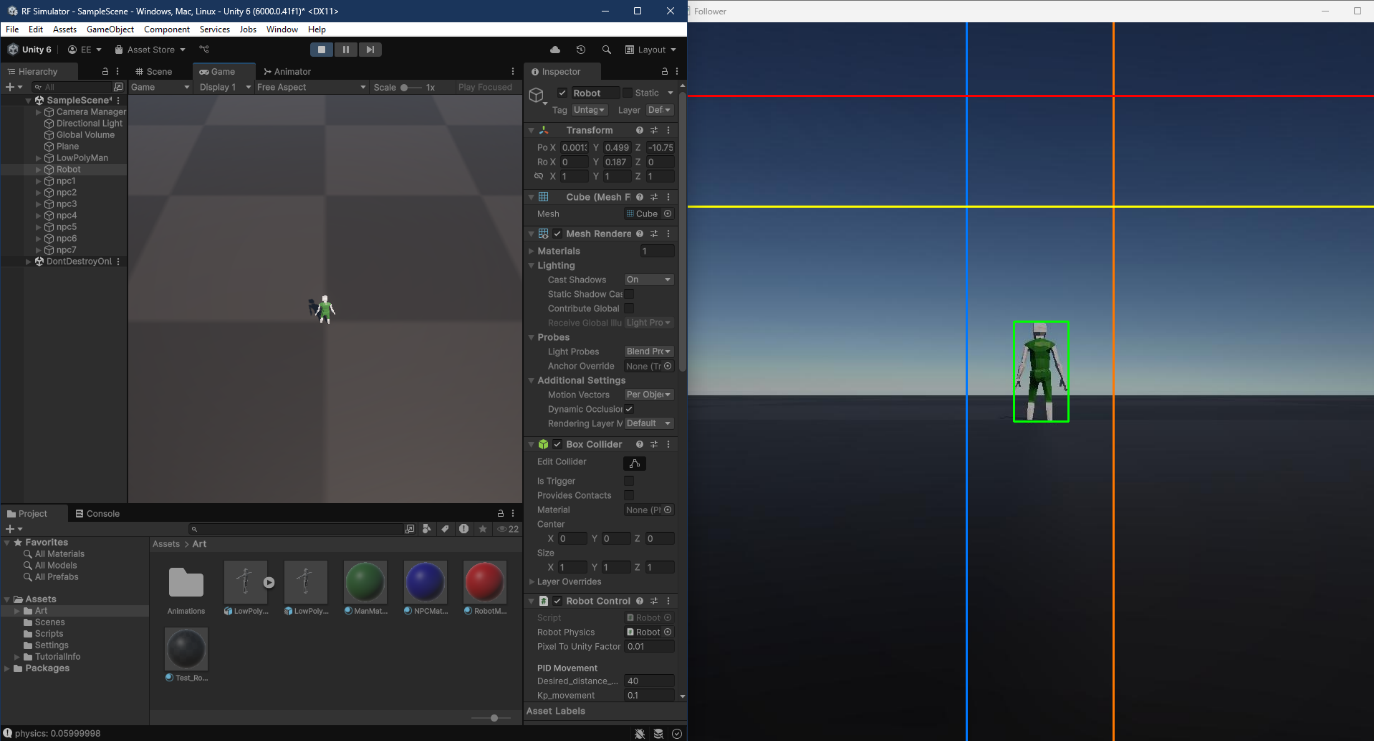
AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Odată ce clientul Python stabilește conexiunea cu serverul Unity, bucla de control devine activă, iar procesul de urmărire autonomă începe instantaneu.

În figura de mai jos observăm starea sistemului din primul cadru.



Fereastra „Follower" afișează perspectiva camerei robotului și rezultatul procesării în timp real. Această fereastră ne arată că ținta (personajul în verde) a fost corect identificată, însă bounding box-ul verde depășește considerabil zona țintă. Mai exact, partea superioară a acestuia se află mult sub linia galbenă superioară, indicând o eroare pozitivă semnificativă pe axa de translație. Din punct de vedere fizic, acest lucru înseamnă că robotul se află prea departe de țintă.

Această eroare este imediat procesată de controlerul PID din scriptul RobotController, care calculează o comandă de mișcare pozitivă (deplasare înainte) pentru a micșora distanța față de țintă. În același timp, se calculează și eroarea de rotație pentru a menține ținta centrată pe orizontală.

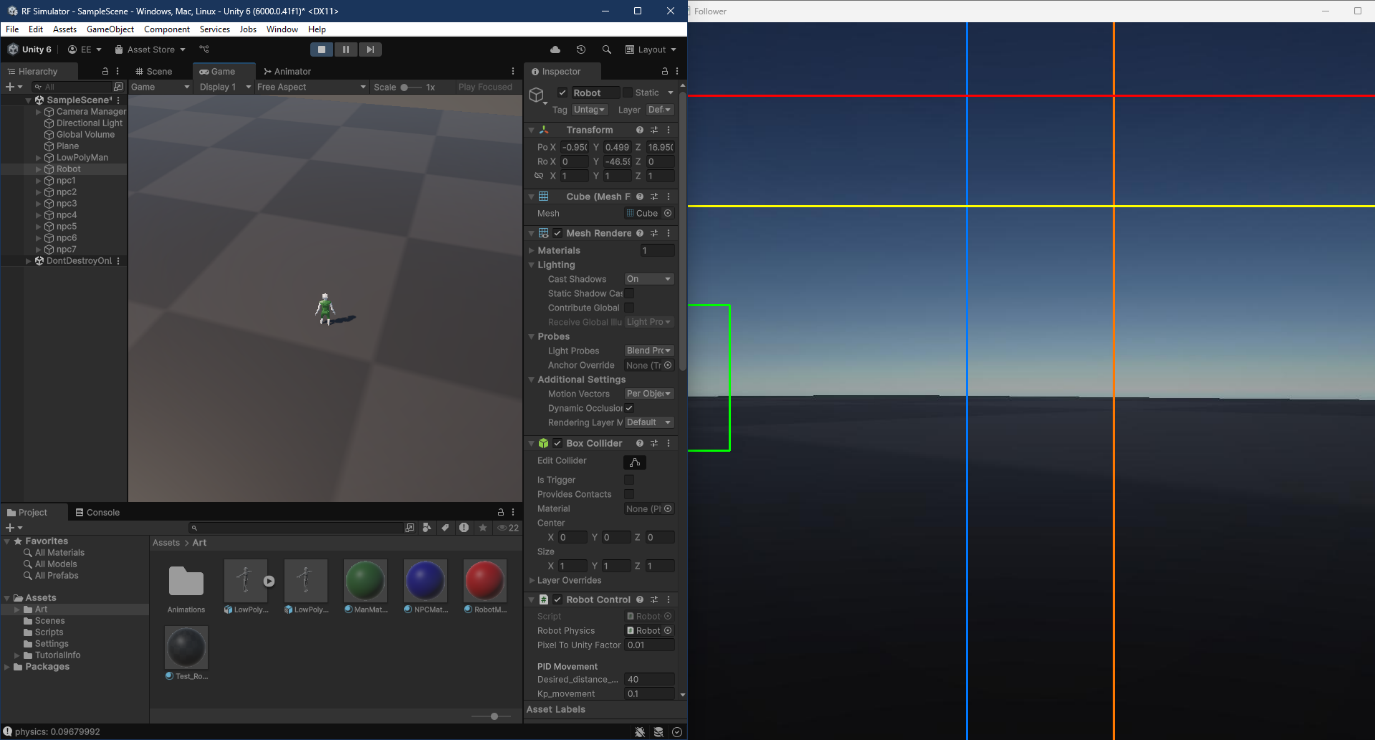
Rezultatul acestei acțiuni corective este ilustrat în figura de mai jos.

A screenshot of a computer

AI-generated content may be incorrect.

Putem observa în scena Unity (stânga) că robotul s-a deplasat fizic înainte. În consecință, în fereastra "Follower" (dreapta), bounding box-ul care încadrează ținta s-a mărit, ajungând la dimensiunea dorită și încadrându-se perfect în zona de siguranță delimitată de liniile de ghidaj orizontale și verticale. În acest moment, eroarea calculată de sistem este aproape de zero, iar controlerul PID încetează să mai trimită comenzi de mișcare semnificative. Sistemul a atins o stare de echilibru stabil, îndeplinindu-și cu succes obiectivul de a se poziționa la distanța și unghiul corecte față de țintă.

Următoarea figură demonstrează un scenariu în care, personajul țintă a fost deplasat intenționat rapid spre stânga, ieșind complet din câmpul vizual al camerei robotului.

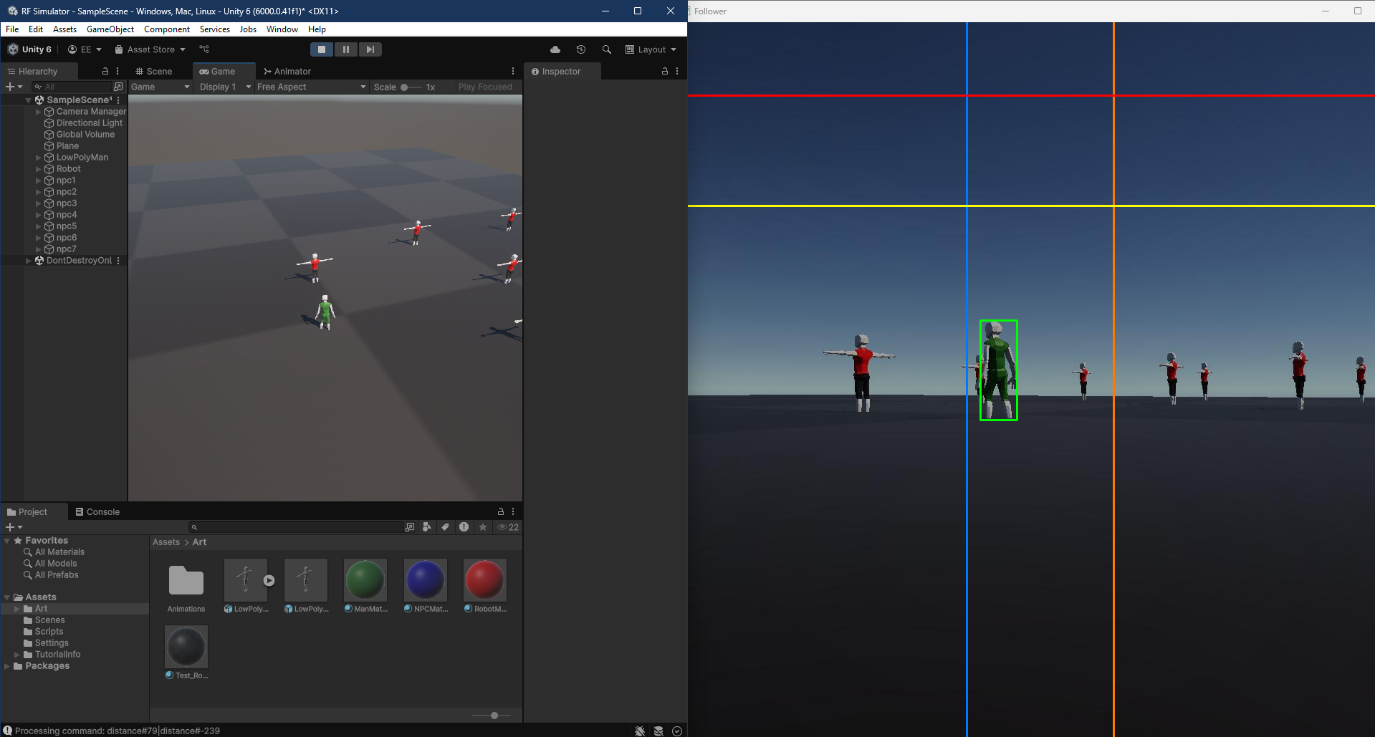


Aici intervine mecanismul de robustețe implementat în clasa ColorFollowerSmooth. Logica specifică faptul că, în absența unei noi detecții valide (o persoană care îndeplinește criteriul de culoare), algoritmul nu abandonează sarcina, ci recurge la ultima poziție validă cunoscută a țintei, stocată în variabila self.prev\_bbox.

Observând fereastra "Follower" din figura, putem vedea exact acest comportament. Deși personajul nu mai este prezent în cadru, chenarul verde, care reprezintă ținta din perspectiva "creierului", rămâne "înghețat" la ultima sa poziție cunoscută, la marginea stângă a ecranului. Pentru controlerul PID, această situație este clară: există o eroare orizontală masivă între poziția memorată a țintei și centrul cadrului. Prin urmare, robotul nu se oprește. El continuă să execute o comandă de virare puternică spre stânga, într-o mișcare anticipativă de a "căuta" ținta în direcția în care a fost văzută ultima oară. Acest comportament conferă sistemului o formă de "memorie pe termen scurt". Este o strategie esențială care permite robotului să re-achiziționeze ținta după o ocluzie scurtă (de exemplu, dacă persoana trece pe după un stâlp) sau după o mișcare bruscă. Această demonstrație evidențiază trecerea de la un sistem pur reactiv, care răspunde doar la ce vede în prezent, la un sistem mai inteligent și proactiv, care utilizează informații din trecut pentru a anticipa și a corecta stări de eroare complexe.

Scenariul final al studiului de caz este conceput pentru a evalua o ultimă funcționalitate a sistemului: capacitatea de a menține urmărirea țintei desemnate într-un mediu vizual aglomerat, populat de alte persoane. Succesul în acest test este dovada că arhitectura hibridă, care combină detecția generală de obiecte cu filtrarea specifică de atribute, este eficientă.

Pentru a simula această condiție, ținta (personajul verde) a fost plasată intenționat în proximitatea unui grup de personaje non-țintă, care au rol de "distractori" vizuali, așa cum se poate observa în scena de ansamblu din figura de mai jos.



Rezultatul, vizibil în fereastra "Follower" (dreapta), este o demonstrație a succesului arhitecturii implementate. Deși modelul YOLO, în prima sa etapă de procesare, detectează prezența tuturor persoanelor din cadru, al doilea strat de verificare – filtrul de culoare HSV – intră în acțiune ca un mecanism de distingere esențial.

Algoritmul parcurge lista tuturor detecțiilor de "persoană" și evaluează fiecare candidat în parte, calculând procentul de pixeli verzi din bounding box-ul corespunzător. După cum se poate observa, doar ținta desemnată trece cu succes de acest filtru, fiind singura încadrată de un chenar verde și, prin urmare, singura luată în considerare de logica de control PID. Personajele non-țintă, deși percepute de rețeaua neuronală ca fiind "persoane", sunt efectiv ignorate de sistemul de urmărire, deoarece nu îndeplinesc criteriul de culoare.

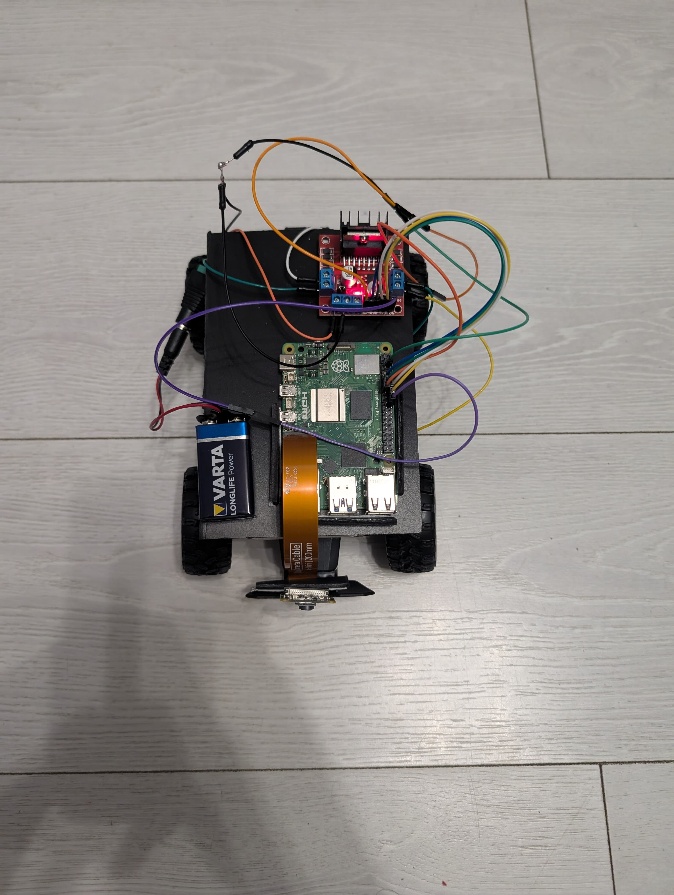
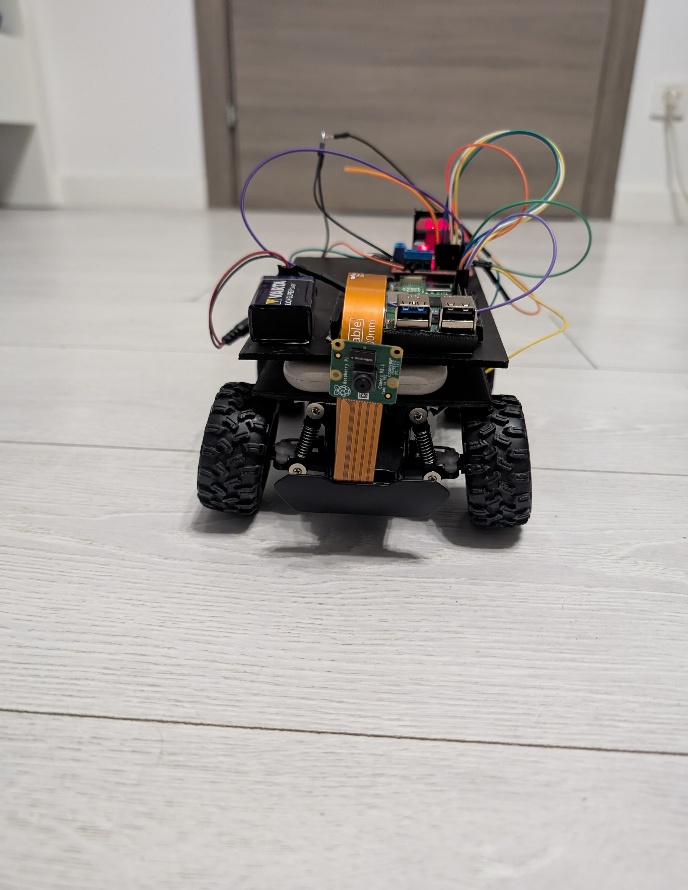
Acest test confirmă în mod concludent că sistemul nu este un simplu urmăritor generic de oameni, ci un agent autonom capabil de o identificare specifică.

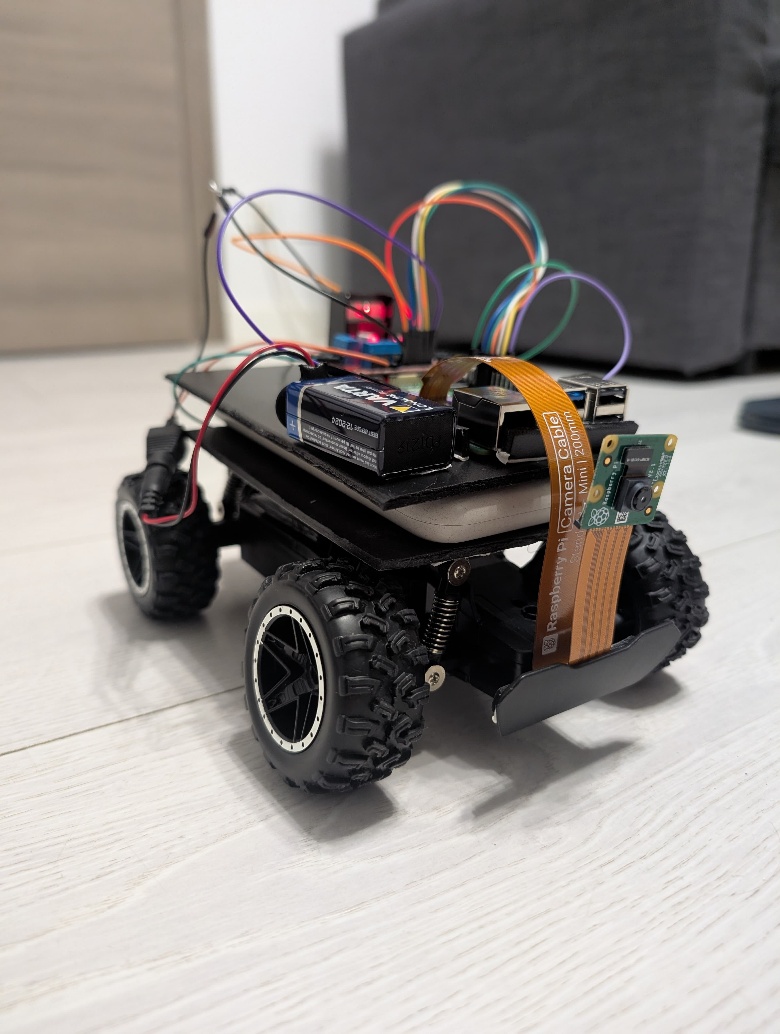
Așadar putem afirma ca în acest simulator s-a reușit îndeplinirea cerinței fundamentale a proiectului: urmărirea fiabilă a unui individ unic, chiar și în diferite situații critice.

## Prezentarea robotului

După etapele de proiectare teoretică și validare în mediul simulat, s-a trecut la construcția prototipului fizic.

Imaginile următoare oferă o perspectivă detaliată asupra integrării componentelor cheie pe șasiul robotului.





Analizând prototipul, putem identifica modul în care fiecare componentă hardware își ocupă rolul definit în arhitectura sistemului:

* Unitatea Centrală de Procesare: În centrul platformei superioare, vizibilă cel mai clar în vederea de sus, se află placa de dezvoltare Raspberry Pi 5. Aceasta este inima computațională a robotului, responsabilă pentru rularea sistemului de operare, a scriptului Python și a modelului de detecție YOLOv11. Poziționarea sa centrală facilitează accesul la pinii GPIO și la porturile de conectivitate.
* Modulul de Percepție: Montată în partea frontală a șasiului, Camera Raspberry Pi V2 este orientată pentru a avea un câmp vizual neobstrucționat. Conexiunea sa la placa Raspberry Pi se realizează prin cablul panglică portocaliu, conectat la portul CSI. Această alegere de design, așa cum s-a discutat în capitolele anterioare, este esențială pentru a asigura o transmisie a datelor video cu latență redusă, o condiție critică pentru reactivitatea în timp real a robotului.
* Modulul de Actuație: Deasupra plăcii Raspberry Pi, se află driverul de motoare L298N, placa de culoare roșie. Se poate observa rețeaua de fire de tip "jumper" (multicolore) care reprezintă calea fizică a semnalelor de control (PWM pentru viteză și semnale digitale pentru direcție) de la Pi la driver.
* Sistemul de Alimentare: Un aspect critic al implementării fizice, vizibil în special în vederea laterală, este sistemul de alimentare dual. Placa Raspberry Pi 5, fiind un computer sensibil la fluctuațiile de tensiune, este alimentată de un acumulator extern de tip power bank, care îi asigură o sursă stabilă de 5V. Separat de acesta, driverul de motoare L298N este alimentat de o baterie de 9V. Această decuplare energetică este o decizie de inginerie deliberată pentru a preveni zgomotul electric generat de motoare să interfereze cu funcționarea plăcii Raspberry Pi, asigurând astfel stabilitatea întregului sistem.

Mai mult decât o simplă asamblare de componente, prototipul fizic prezentat este pe deplin funcțional și îndeplinește cu succes obiectivul fundamental al acestui proiect. Testele practice efectuate într-un mediu real au demonstrat un comportament care reflectă fidel și validează rezultatele obținute anterior în mediul simulat, confirmând succesul tranziției de la digital la fizic.

Astfel, implementarea fizică nu servește doar ca o demonstrație tehnică a componentelor asamblate, ci și ca o validare finală a întregului proces de proiectare. De la arhitectura software și algoritmii de viziune computerizată, până la logica de control și selecția hardware, prototipul funcțional confirmă că soluția propusă este atât teoretic solidă, cât și practic viabilă.

# Concluzii

# Bibliografie

# Anexe