UNIVERSITATEA POLITEHNICA BUCUREȘTI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

PROIECT DE DIPLOMĂ

Estimarea locației unui vehicul RC bazată pe markeri și camere
(Automatizarea unui proces de localizare în spațiu folosind tehnologii actuale)

Vișan Ionuț

Coordonator științific:
ȘI. Dr. Ing. Jan-Alexandru Văduvă

BUCUREŞTI

2024

UNIVERSITY POLITEHNICA OF BUCHAREST
FACULTY OF AUTOMATIC CONTROL AND COMPUTERS
COMPUTER SCIENCE DEPARTMENT

DIPLOMA PROJECT

RC Vehicle Location Estimation based on Tags and Cameras
(Automation of a localization process in space using current technologies)

Vișan Ionuț

Thesis advisor:
ȘI. Dr. Ing. Jan-Alexandru Văduvă

BUCHAREST

2024

CUPRINS

Sinopsis 3
Abstract 4
Mulțumiri 5
1 Introducere 6
1.1 Context 6
1.2 Problema 6
1.3 Obiective 7
1.4 Structura lucrării 7
2 Motivația 8
3 Studiu de piață / Abordări existente 8
4 Soluția propusă 14
4.1 Scopul proiectului 16
5 Detalii de implementare 17
5.1 Aplicația mobilă 17
5.1.1 Tehnologiile alese 17
5.1.2 Integrarea dependențelor 18
5.1.3 Implementarea aplicației 19
5.1.4 Detalii tehnice 28
5.1.5 Perspectiva utilizatorului 28
5.1.6 Perspectiva dezvoltatorului 31
5.1.7 Funcționalitate 31
5.3 Server 34
5.3.1 Tehnologiile alese 34
5.3.2 Implementarea serverului 36
5.3.3 Perspectiva serverului 43
5.4 Ansamblu aplicație mobilă – server 45
5.5 Scheletul robotizat 47
5.5.1 Tehnologiile alese 47
5.5.2 Componente 48
5.5.3 Implementare cod ESP32 53

5.5.4 Detalii tehnice 62
5.5.5 Perspectiva dezvoltatorului 63
6 Studiu de caz / Evaluarea rezultatelor 64
6.1 Rezultate obținute 64
6.2 Criterii de performanță 68
6.3 Evaluarea rezultatelor 69
6.3.1 Identificarea corectă a markerilor 69
6.3.2 Timpi de execuție 72
6.3.3 Număr de mișcări 74
6.3.4 Precizia 75
6.3.5 Corectitudine în depanare 77
7 Utilizare 78
7.1 Utilizare actuală 78
7.2 Eficiența comparativă 78
8 îmbunătățiri 79
9 Concluzii 79
10 Bibliografie 79
11 Acces la cod 81

SINOPSIS

Automatizarea a devenit un element central în evoluția tehnologică recentă [1], transformând procese lungi și instabile, afectate de numeroase variabile externe, în operațiuni eficiente și de succes. Din succesul crescut al acestor sisteme de automatizare rezultă beneficii semnificative pentru o societate în dinamică transformare [10]. Acest interes crescând pentru procesele de automatizare subliniază importanța studiului acestora pentru cei care doresc să rămână la curent cu ultimele inovații tehnologice.

Lucrarea propune o metodă accesibilă prin care oricine poate implementa un sistem automatizat pentru efectuarea diverselor sarcini. Folosind un telefon, un sistem stabil de comunicații și un dispozitiv controlabil, utilizatorii pot comanda acest ultim dispozitiv să execute sarcinile atribuite prin intermediul telefonului. Această abordare democratizează accesul la tehnologiile de automatizare, facilitând adoptarea lor pe scară largă.

ABSTRACT

Automation has become a central element in recent technological evolution, transforming lengthy and unstable processes, affected by numerous external variables, into efficient and successful operations. The increased success of these automation systems results in significant benefits for a dynamically changing society. This growing interest in automation processes underscores the importance of studying them for those who wish to stay abreast of the latest technological innovations.

This paper proposes an accessible method by which anyone can implement an automated system for performing various tasks. Using a phone, a stable communication system, and a controllable device, users can command the latter device to perform tasks assigned via the phone. This approach democratizes access to automation technologies, facilitating their widespread adoption.

MULȚUMIRI

Efortul pe care l-am depus în timpul studiului diverselor materii din cadrul facultății a fost mereu sprijinit de colegi, laboranți, seminariști și profesori. Mereu am primit răspunsurile pe care le căutam și am fost ajutat să prosper în aprofundarea unor subiecte mai ample. Doresc să le mulțumesc acestora pentru că au reușit sa însumeze această experiență pe care am avut-o în timpul facultății.

De asmenea, un rol esențial l-a avut coordonatorul meu. Acesta a fost alături de mine în realizarea lucrării mele, mi-a oferit toate detaliile și mi-a ordonat pașii pentru a duce la bun sfârșit acest proiect. Îi sunt profund recunoscător pentru timpul oferit.

1 INTRODUCERE

1.1 Context

Procesul de automatizare este definit ca aplicarea tehnologiilor și metodologiilor pentru a opera sisteme într-o manieră eficientă, productivă și fiabilă, cu un minim de intervenție umană [10]. Automatizarea este prezentată ca esențială pentru îmbunătățirea calității, eficienței și productivității în medii structurate, având un impact semnificativ asupra economiei globale și extinzându-se în diverse domenii precum sănătatea, transport și energie [2].

În medii complexe precum Comanda și Control (C2), unde erorile umane pot avea consecințe tragice, sistemele automate inteligente sunt esențiale. Cu creșterea volumului de informații de procesat, capacitatea umană de prelucrare a informațiilor devine rapid suprasolicitată [12]. În plus, numeroasele surse de informații, ritmul și complexitatea mediului sunt de asemenea amplificate odată cu dezvoltarea tehnologiei. Mediile cu mize mari, cum ar fi C2, generează o cantitate considerabilă de stres care afectează performanța umană [2]. Performanța este, de asemenea, influențată de nivelul de oboseală resimțit de om. Toți acești factori pot contribui la variabilitatea performanței, erori umane și nerealizarea sarcinilor.

Dorim să înțelegem cum putem crea un astfel de automatism care să îndeplinească sarcini pe care i le putem atribui cu ușurință. Vom crea un proces complex de comunicație între diferite tehnologii pentru a controla un schelet robotizat fără să fie nevoie să intervenim constant pentru a duce la bun sfârșit o sarcină.

1.2 Problema

Procesele de automatizare rezolvă diverse probleme semnificative, în special în medii complexe și cu mize mari. Principalele probleme pe care le abordează includ:

Suprasolicitarea umană	Automatizarea reduce sarcina de lucru a operatorilor, preluând procesarea unui volum mare de informații, ceea ce depășește adesea capacitățile umane.
Erori umane	Sistemele automate pot diminua considerabil apariția erorilor umane, crescând astfel fiabilitatea și acuratețea execuției sarcinilor.
Stres și oboseală	Automatizarea poate reduce factorii de stres și oboseală la care sunt supuși operatorii, contribuind astfel la menținerea unei performanțe optime pe durate lungi de timp.
Consistența performanței	Sistemele automate oferă un nivel de stabilitate în execuția sarcinilor, menținând performanța constantă indiferent de condițiile externe.

1.3 Obiective

Automatizarea poate fi văzută ca o soluție potențială pentru reducerea diverselor probleme, oferind numeroase beneficii cognitive [2]. Printre acestea se numără:

Eficiență crescută	Automatizarea poate îmbunătăți semnificativ eficiența proceselor, de la fabricație până la logistica și gestionarea datelor, permițând realizarea sarcinilor mai rapid și cu mai puține erori comparativ cu metodele manuale.
Îmbunătățirea accesibilității și calității serviciilor	În sănătate, de exemplu, roboții pot asista în proceduri chirurgicale sau pot monitoriza pacienții, crescând precizia intervențiilor și accesul la tratamente de înaltă calitate.
Sustenabilitate	Automatizarea poate contribui la practici mai sustenabile în agricultură și producție, optimizând utilizarea resurselor și reducând deșeurile.
Crearea de noi oportunități	Deși există o preocupare legată de pierderea locurilor de muncă datorată automatizării, aceasta poate de asemenea crea noi joburi în domenii tehnologice și poate stimula necesitatea de recalificare profesională, contribuind la evoluția forței de muncă.

Prin urmare, automatizarea poate juca un rol crucial în ameliorarea vieții de zi cu zi și în susținerea creșterii economice pe termen lung. Realizarea unui astfel de mecanism ne poate ajuta să realizăm sarcini complexe fără a fi necesară implicarea în mod constant.

1.4 Structura lucrării

Lucrarea a fost împarțită în 10 capitole, după cum urmează:

Capitol	Descriere
1	Familiarizarea cititorului cu tema proiectului.
2	Prezentarea motivației aflate la baza proiectului.
3	Prezentarea unor comparații și elemente clare care consolidează utilitatea proiectului.
4	Detalierea soluției propuse spre implementare.
5	Explicații amănunțite despre rolul, funcționalitea și perspectiva fiecărei componenete din ansamblul ce formează proiectul.
6	Analiza asupra rezultatelor obținute.
7	Moduri de utilizare și comparație cu o tehnologie actuală.
8	Îmbunătățiri ce pot fi aduse proiectului pentru a obține performanțe mai bune.
9	Concluzii despre proiect și rezultatele obținute.
10	Articolele utilizate pentru marcarea anumitor detalii ce stau la baza proiectului.

2 MOTIVAȚIA

Motivația din spatele realizării acestui proiect este de a aborda și soluționa problemele identificate anterior, aducând totodată numeroase beneficii importante. Ne propunem să simplificăm considerabil procesul de automatizare, făcându-l mai accesibil și mai economic pentru publicul larg. Prin această inițiativă, ne dorim să democratizăm accesul la tehnologia avansată, oferind o soluție eficientă și rentabilă care să răspundă nevoilor variate ale utilizatorilor. În plus, acest avans tehnologic este conceput pentru a fi susținut de o eficiență operațională ridicată, asigurând astfel un impact pozitiv pe termen lung și o utilizare optimă a resurselor disponibile.

3 STUDIU DE PIAȚĂ / ABORDĂRI EXISTENTE

3.1 Impactul automatizării asupra industriei

Nikhil Kumar a menționat in lucrarea “Impact of the Business Process Automation on Human Resource Management” [6] că industria investește sume mari de bani în automatizarea afacerilor și că adoptarea instrumentelor tehnologice de automatizare a afacerilor influențează semnificativ operațiunile de afaceri, comportamentul resurselor umane, structura organizațională, precum și eficiența și eficacitatea acestora. Se observă un impact pozitiv al automatizării robotice asupra dezvoltării economice, unde o creștere a investițiilor în robotică cu 1% este asociată cu un câștig pe termen lung în PIB-ul pe cap de locuitor de 0,03%. Imbunătățirea randamentului este un beneficiu al automatizării și magnitudinea acestui beneficiu variază semnificativ în funcție de utilizator.

Zierahn, Gregory și Arntz au afirmat in lucrarea “Racing with or against the Machine? Evidence from Europe” [8] că producția joacă un rol vital în creșterea salariilor, ocupării forței de muncă și cererii, influențând economia sectorului în general. Datorită câștigurilor de productivitate din robotică și automatizare, nu doar la nivel de companie, ci și la nivel industrial și național, competitivitatea a crescut. Chiar și după criza financiară din industria de fabricație a SUA, producția și productivitatea au crescut constant odată cu creșterea robotizării și automatizării. Conform lui Graetz și Michaels în lucrarea “Robots at Work” [10] utilizarea roboților în industrie a rezultat într-o creștere medie a PIB-ului de 0,37% și a ratei de creștere a productivității de 0,36%, demonstrând un impact pozitiv al robotizării și automatizării asupra productivității și creșterii PIB-ului.

3.2 Tehnologii existente și eficiența lor

Într-o lume tot mai conectată, roboții și sistemele de automatizare reprezintă vârful tehnologic în domeniul confortului și productivității. Aceste tehnologii avansate promit să transforme diversele spații în medii inteligente, adaptându-se nevoilor locatarilor cu o

eficiență uimitoare. Totuși, complexitatea și costurile asociate rămân un prag semnificativ pentru mulți consumatori [2].

Roboții casnici, de la aspiratoarele autonome la sistemele avansate de asistență personală, sunt exemplul clar al inovației tehnologice. Echipați cu senzori de proximitate, camere, și algoritmi inteligenți de învățare automată, acești roboți pot naviga independent într-o casă, evitând obstacolele și optimizându-și traseele pentru indeplinirea diverselor sarcini.

Cu toate acestea, costul acestor tehnologii poate fi prohibitiv. Un sistem complet de automatizare casnică poate ajunge să coste mii de dolari, în funcție de complexitatea și numărul de dispozitive incluse. Roboții de companie, precum Aibo de la Sony, pot costa chiar și peste 2,000 de dolari.

Incorporarea unei game largi de tehnologii avansate justifică în parte aceste costuri. Utilizarea inteligenței artificiale, a procesării limbajului natural și a tehnologiilor IoT (Internet of Things) necesită dezvoltare continuă și actualizări frecvente de software, ceea ce contribuie la prețul final al produselor.

Majoritatea acestor sisteme automatizate necesită foarte multe componente speciale pentru localizare si coordonare. Sunt necesari senzori și camere cu precizie foarte bună pentru a îndruma sistemul într-un spatiu determinat. De asemena, pentru prelucrarea datelor se folosesc sisteme complicate și amănunțite, pe care majoritatea utilizatorilor nu le pot înțelege.

În concluzie, deși tehnologiile de automatizare și roboții personali sunt remarcabil de eficiente în îndeplinirea unei varietăți de funcții, natura lor complexă și costurile ridicate le fac adesea inaccesibile și dificil de utilizat pentru publicul larg. Această dicotomie între potențialul tehnologic și accesibilitatea sa pune în evidență nevoia de soluții mai simplificate și mai cost-eficiente care să aducă beneficiile automatizării la îndemâna unui număr mai mare de persoane.

3.3 Sisteme AGV cu LIDAR vs Soluția noastră

Sistemele AGV (Automated Guided Vehicles) echipate cu tehnologia LIDAR (Light Detection and Ranging) reprezintă o soluție avansată pentru automatizarea transportului într-o varietate de medii, de la fabrici și depozite, până la centre de distribuție și alte facilități industriale.

Tehnologia LIDAR

LIDAR-ul este o tehnologie care măsoară distanța până la un obiect sau suprafață folosind un fascicul laser. Un senzor LIDAR emite impulsuri de lumină laser și măsoară timpul necesar pentru ca acestea să se întoarcă după ce lovesc un obiect. Aceste date sunt utilizate pentru a genera hărți precise 3D ale mediului înconjurător.

Caracteristici Principale ale AGV-urilor cu LIDAR

Navigație și Localizare	AGV-urile cu LIDAR sunt capabile să navigheze și să se localizeze cu o mare precizie în medii complexe. Ele pot evita obstacolele detectate în timp real și pot ajusta traseul în funcție de condițiile din teren.
Flexibilitate	Spre deosebire de AGV-urile care urmează trasee fixe bazate pe magneți sau benzi pe podea, AGV-urile cu LIDAR sunt mult mai flexibile. Ele pot naviga în noi rute prin simpla actualizare a hărților lor digitale, fără a necesita modificări fizice în infrastructura locației.

Aplicații:

Industrie	AGV-urile sunt folosite pentru transportul materialelor între diferite secțiuni ale unei fabrici sau între diferite echipamente de producție.
Logistică	În depozite, aceste vehicule automate pot optimiza procesele de stocare și recuperare a mărfurilor, crescând eficiența și reducând timpul de așteptare.
Spitale	Unele spitale folosesc AGV-uri pentru a transporta medicamente, probe de laborator sau echipamente între diferite secții.

Costuri:

Investiția inițială în sistemele AGV cu LIDAR poate fi semnificativă, datorită costului tehnologiei LIDAR și necesității de integrare a sistemelor în infrastructura existentă. Costurile de operare sunt însă reduse pe termen lung, datorită eficienței și reducerii erorilor umane.

Limitări:

Costul inițial ridicat	Necesită o investiție capitală substanțială.
Dependența de medii structurate	Performanța optimă necesită un mediu bine definit și organizat.
Complexitatea integrării	Poate necesita modificări ale infrastructurii existente pentru a optimiza utilizarea AGV-urilor.

Aceste sisteme implică costuri inițiale mari și necesită o pregătire și o integrare complexă în infrastructurile existente, ceea ce poate constitui un obstacol semnificativ pentru unele organizații, persoane fără o pregătire specifică sau în spații în care dificultatea sarcinilor nu necesită implementări avansate.

Propunerea noastră vizează implementarea unei soluții care să minimizeze costurile, să facă uz de tehnologii accesibile și ușor de înțeles pentru publicul larg, asigurând în același timp o eficiență remarcabilă în executarea diverselor sarcini.

3.4 Formular de evaluare a impactului automatizării

Pentru a elabora o opinie bine fundamentată, am considerat esențial să explorăm perspectivele diverse ale unui număr variat de persoane în legătură cu subiectul studiat și impactul acestuia asupra lor. În acest scop, am conceput și distribuit un set de întrebări detaliate, prin intermediul cărora am urmărit să obținem o înțelegere mai profundă a opiniei publice. Am reușit să strângem răspunsuri de la 42 de participanți, care ne-au oferit perspective valoroase asupra temei investigate.

Participanții la acest studiu au fost selectați din rândul studenților a două facultăți importante: Facultatea de Automatică și Calculatoare și Facultatea de Medicină. Grupul de respondenți a fost alcătuit din studenți cu vârste cuprinse între 22 și 26 de ani.

1. Sunteți familiarizat cu noțiunea de sisteme automatizate? (ex. roboți pentru curățenie, sisteme de fluidiziare a traficului)

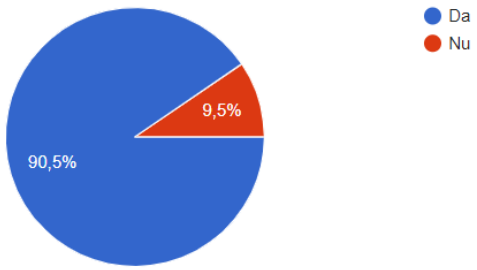

Figura 1 Raport întrebarea 1

Nu – 4 persoane

Da – 38 persoane

2. Considerați aceste sisteme automatizate utile în scopurile personale?

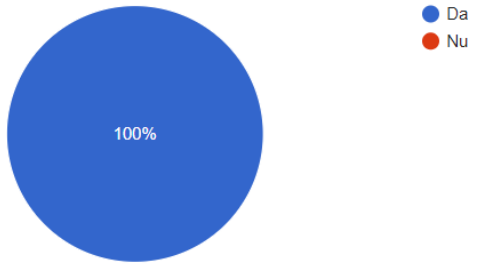

Figura 2 Raport întrebarea 2

Nu – 0 persoane

Da – 42 persoane

3. Considerați că aceste sisteme automatizate sunt prezente în diverse forme în viața dumneavoastră de zi cu zi?

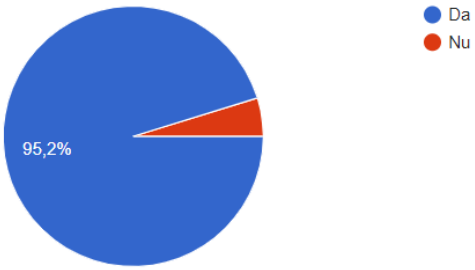

Figura 3 Raport întrebarea 3

Nu – 2 persoane

Da – 40 persoane

4. În ce masură considerați că sistemele automatizate se regăsesc în viețile noastre?

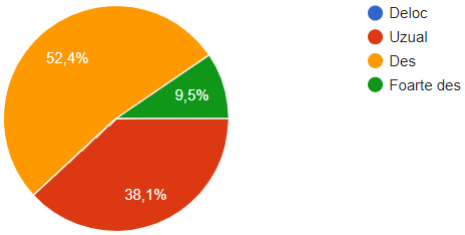

Figura 4 Raport întrebarea 4

Deloc – 0 persoane

Uzual – 16 persoane

Des – 22 persoane

Foarte des – 4 persoane

5. Considerați că utilizarea sistemelor în mai multe domenii ar putea aduce mai multe beneficii?

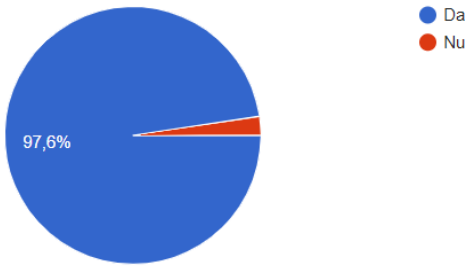

Figura 5 Raport întrebarea 5

Nu – 1 persoană

Da – 41 persoane

6. Considerați că aceste sisteme sunt scumpe?

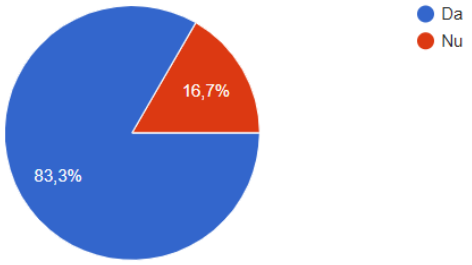

Figura 6 Raport întrebarea 6

Nu – 7 persoane

Da – 35 persoane

7. Dacă sistemele automatizate ar fi mai ieftine, le-ați folosi mai des?

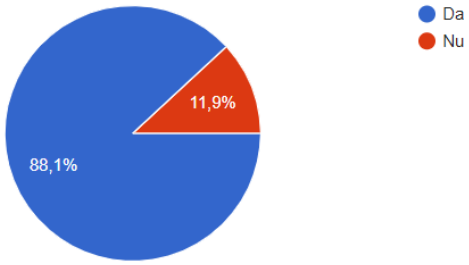

Figura 7 Raport întrebarea 7

Nu – 5 persoane

Da – 37 persoane

8. Considerați că acestea prezintă tehnologii mai avansate și sunt uneori mai greu de înțeles?

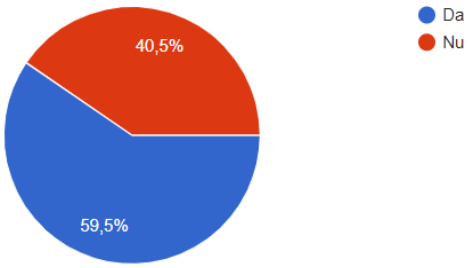

Figura 8 Raport întrebarea 8

Nu – 17 persoane

Da – 25 persoane

9. În ce măsură considerați că folosiți (în sens larg - acasă, în trafic, etc.) aceste sisteme automatizate?

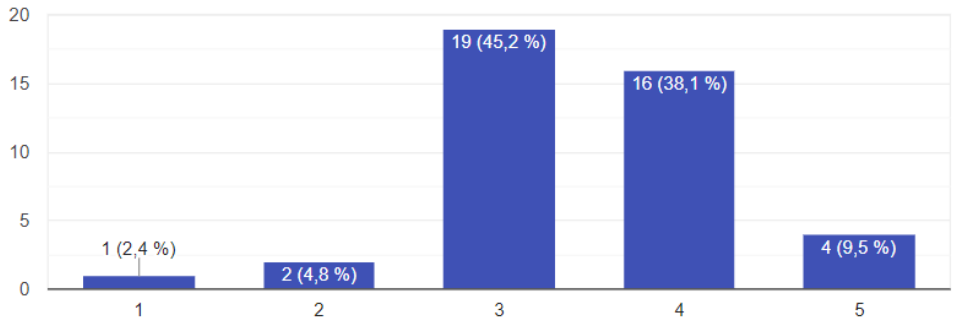

Figura 9 Raport întrebarea 9

10. Cum vă poziționați față de următoarea afirmație?

Dacă sistemele automatizate ar fi mai ușor de înțeles și ar avea prețuri mai mici aș apela mai des la utilizarea acestora în scopuri personale.

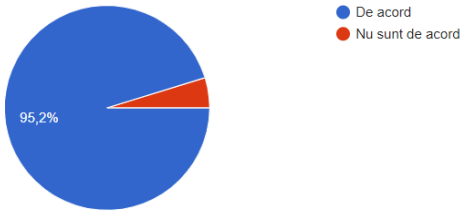

Figura 10 Raport întrebarea 10

Nu sunt de acord – 2 persoane

De acord – 40 persoane

Concluzie:

Analiza răspunsurilor colectate din cadrul studiului nostru indică o percepție majoritar pozitivă și o receptivitate înaltă față de sistemele automatizate printre studenții chestionați. Utilitatea percepută este de asemenea ridicată, cu toți participanții afirmând că găsesc tehnologiile automatizate utile pentru îmbunătățirea vieții personale.

Aceste date subliniază oportunitatea clară pentru inovare în domeniul reducerii costurilor tehnologiilor automatizate. Scăderea prețurilor ar putea deschide calea spre o adoptare și mai largă, facilitând o integrare și mai profundă a acestor sisteme în diferite aspecte ale vieții cotidiene.

4 SOLUȚIA PROPUSĂ

În urma analizelor anterioare putem deprinde că sistemele automatizate cu prețuri reduse și tehnologii mai ușor de înțeles pot aduce beneficii în multe domenii. Ne propunem să implementăm un sistem care se poate orienta singur în spațiu și poate realiza diverse sarcini în mod automat, fără intervenția constantă a utilizatorului.

Sistemul nostru utilizează smartphone-ul ca instrument principal pentru capturarea și transmiterea informațiilor, oferind o modalitate accesibilă și eficientă de comunicare. Aplicația, dezvoltată în Dart și Flutter, gestionează accesul la camera dispozitivului, capturând imagini și gestionând permisiunile necesare. Utilizatorul poate vizualiza și controla în timp real fluxul de captură și transmitere a imaginilor.

Folosim markeri ArUco plasați în mediul analizat pentru a orienta și coordona obiectele de interes. Acești markeri sunt esențiali în determinarea poziției și relațiilor spațiale dintre obiecte, fiind capturați prin camera telefonului mobil și trimiși către server pentru procesare.

Serverul, construit pe micro-framework-ul web Flask în Python, gestionează încărcarea imaginilor și detectarea markerilor ArUco folosind biblioteca OpenCV. Acesta identifică markerii, extrage ID-urile acestora și calculează distanțele relative, stocând informațiile necesare pentru acces ulterior.

Informațiile despre markerii detectați sunt disponibile prin API, permițând utilizatorilor să obțină detalii precise despre configurația spațială capturată. Serverul include funcționalități de testare a conexiunii și confirmare a funcționalității, fiind configurat să fie accesibil de pe orice adresă IP.

Pe baza datelor acumulate, un script Arduino coordonează un vehicul motorizat, folosind Wi-Fi pentru a comunica cu serverul. Scriptul procesează informațiile despre poziția markerilor și ajustează orientarea și viteza vehiculului în funcție de alinierea necesară, navigând eficient spre destinația dorită. Această integrare între aplicația mobilă, server și vehiculul motorizat demonstrează o simbioză între tehnologia software și hardware, optimizând interacțiunea cu mediul analizat.

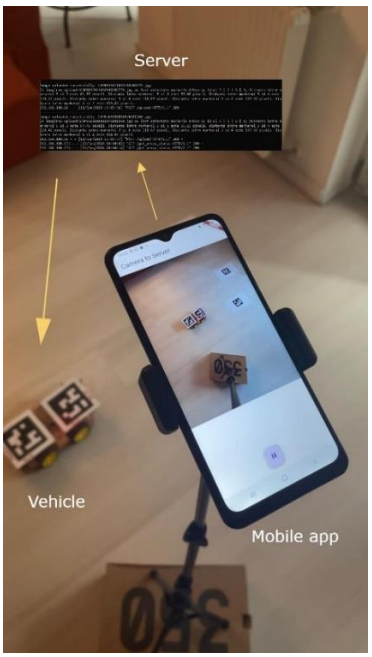

Figura 11 Logica ansamblului

4.1 Scopul Proiectului

Ansamblul oferă o serie de avantaje semnificative în raport cu tehnologiile existente pe piață, mai ales în contextul eficienței și accesibilității. În comparație cu sistemele tradiționale de automatizare și robotizare, care adesea implică costuri ridicate și complexitate tehnică, soluția noastră aduce inovație printr-o abordare simplificată și mai accesibilă.

Reducerea costurilor	Tehnologiile de automatizare tradiționale pot fi extrem de costisitoare, implicând investiții substanțiale în echipamente specifice și software avansat. În contrast, soluția noastră folosește dispozitive smartphone, care sunt deja omniprezente și accesibile pentru majoritatea populației. Aceasta reduce semnificativ bariera de cost, făcând tehnologia de automatizare disponibilă pentru o gamă mai largă de utilizatori, inclusiv pentru micii antreprenori și consumatori individuali.
Utilizare simplificată	Aplicația dezvoltată în Dart și Flutter pentru gestionarea accesului la camera dispozitivului și capturarea imaginilor este intuitivă și ușor de utilizat. Aceasta permite utilizatorilor să controleze și să monitorizeze procesul de automatizare direct de pe smartphone-ul lor, eliminând necesitatea de a învăța să folosească sisteme complexe de software care pot fi intimidante.
Flexibilitate și scalabilitate	Soluția noastră este proiectată să fie flexibilă și să se adapteze la o varietate de scenarii de utilizare. Folosirea markerilor ArUco pentru orientare și coordonare permite adaptarea rapidă la diferite medii și necesități, fără a fi nevoie de reconfigurări costisitoare ale echipamentelor hardware. De asemenea, serverul bazat pe Flask poate fi ușor scalat pentru a gestiona un volum mai mare de date sau pentru a fi integrat cu alte sisteme și tehnologii.
Accesibilitate și interoperabilitate	Prin utilizarea tehnologiei IoT și a conexiunilor de tip API, soluția noastră poate interacționa eficient cu alte dispozitive și sisteme, extinzându-și funcționalitățile și îmbunătățind experiența utilizatorilor. Acest lucru permite o integrare mai bună în ecosistemele smart home existente, precum și adaptabilitate la noi inovații tehnologice.

Prin urmare, soluția propusă nu doar că îmbunătățește accesibilitatea tehnologiilor de automatizare prin reducerea costurilor și simplificarea utilizării, dar oferă și o flexibilitate crescută, fiind astfel o opțiune viabilă și sustenabilă pentru un spectru larg de utilizatori, de la consumatori individuali la întreprinderi. Aceasta permite o adopție mai largă a automatizării în viața cotidiană.

5 DETALII DE IMPLEMENTARE

Proiectul nostru constă într-o integrare complexă de tehnologii software și hardware pentru a crea un sistem de automatizare eficient și accesibil. Pentru a înțelege acest ansamblu, trebuie să cunoaștem modul de funcționare al fiecărei componente și relațiile dintre ele.

5.1 Aplicația mobilă

5.1.1 Tehnologiile alese

Dart și Flutter, o combinație puternică introdusă de Google, au devenit o opțiune atrăgătoare pentru construirea aplicațiilor robuste și cu un aspect vizual placut.

Dart: Un limbaj pentru dezvoltare versatilă

Dart, dezvăluit de Google în 2011, este un limbaj proiectat pentru construirea de aplicații eficiente și scalabile. Inițial destinat dezvoltării web, Dart și-a extins capacitățile, făcându-l potrivit pentru o gamă largă de aplicații, de la mobile la desktop.

Caracteristici cheie:

Reîncărcare rapidă	Funcția Hot Reload a lui Dart permite dezvoltatorilor să vadă instantaneu modificările de cod în timpul procesului de dezvoltare, facilitând iterația rapidă și depanarea.
Tipizare puternică	Dart utilizează un sistem de tipizare puternic și static, îmbunătățind fiabilitatea codului și reducând riscul de erori.
Orientat pe obiecte	Dart este un limbaj orientat pe obiecte, promovând organizarea și reutilizarea codului prin utilizarea de clase și obiecte.

Flutter: Îmbunătățirea dezvoltării de aplicații Cross-Platform

Flutter, un sistem avansat de dezvoltare a interfeței de utilizator open-source, completează perfect limbajul Dart. Introdus de Google, Flutter permite crearea de aplicații compilate nativ pentru mobil, web și desktop dintr-un singur cod sursă.

Caracteristici cheie:

Un singur cod sursă	Caracteristica distinctivă a lui Flutter este capacitatea sa de a crea aplicații pentru mai multe platforme folosind un singur cod sursă, simplificând eforturile de dezvoltare și reducând provocările de întreținere.
Interfețe de utilizator	Biblioteca extinsă de widget-uri a lui Flutter îi împuternicește pe dezvoltatori să creeze interfețe de utilizator vizual remarcabile și personalizabile, asigurând o experiență plăcută a utilizatorului pe toate dispozitivele.
Dezvoltare rapidă	Cu funcții precum Reîncărcarea Rapidă, dezvoltatorii pot vedea actualizări în timp real, accelerând procesul de dezvoltare și îmbunătățind productivitatea.

5.1.2 Integrarea dependențelor

Pentru a ne asigura ca aplicația functionează corespunzător și că avem toate pachetele necesare la îndemană, trebuie să introducem toate dependențele necesare pe care le vom folosi.

Dependențele sunt pachete externe sau biblioteci de cod pe care aplicația noastră le folosește pentru a îndeplini anumite funcționalități. Fiecare dependență are un rol specific în aplicație.

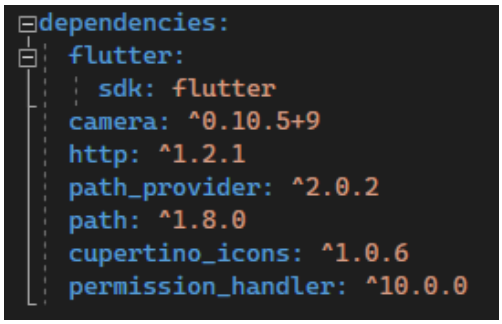

Figura 12 Dependențele aplicației mobile

camera	Acesta este un pachet care oferă funcționalități pentru accesul la camera dispozitivului. Este folosit pentru a captura imagini folosind camera dispozitivului mobil.
http	Acest pachet permite efectuarea de cereri HTTP către servere web. Este folosit pentru a trimite imaginile capturate de la camera la un server web pentru procesare sau stocare.
path_provider	Acest pachet oferă funcționalități pentru obținerea căilor către directoare specifice ale sistemului de fișiere al dispozitivului. Este util pentru a obține căile către fișierele generate sau stocate de aplicație.
path	Acest pachet furnizează un set de utilități pentru lucrul cu căile de fișiere în Dart. Este folosit împreună cu path_provider pentru manipularea căilor către fișierele din sistemul de fișiere al dispozitivului.
cupertino_icons	Acesta este un pachet care conține o colecție de pictograme de stil iOS. Este folosit pentru a adăuga pictograme personalizate în aplicație.
permission_handler	Acest pachet facilitează gestionarea permisiunilor de acces la resursele dispozitivului, cum ar fi camera, microfonul și stocarea. Este folosit pentru a solicita și gestiona permisiunile necesare pentru accesul la cameră și alte resurse ale dispozitivului.

Cu aceste dependențe configurate, putem trece la utilizarea efectivă a aplicatiei. Vom trece în detaliu prin fiecare secțiune a codului pentru a explica cum funcționează fiecare parte și cum sunt utilizate diversele pachete și clase.

5.1.3 Implementarea Aplicației

Importul Pachetelor Necesare

Pentru a facilita dezvoltarea acestei aplicații, importăm mai multe pachete esențiale care ne ajută să gestionăm diverse aspecte ale funcționalității aplicației:

dart:async	Acest pachet permite utilizarea operațiunilor asincrone, care sunt fundamentale pentru gestionarea proceselor ce nu trebuie să blocheze interfața utilizator. Acesta oferă acces la Future-uri și Stream-uri, esențiale pentru a răspunde la evenimente sau pentru a efectua operații care durează mai mult timp, cum ar fi interogări pe rețea.
dart:convert	Include utilități pentru conversia între diferite formate de date, inclusiv JSON, care este adesea folosit în comunicarea cu servere web. Aceasta facilitează manipularea datelor recepționate de la API-uri, convertindu-le în formate utilizabile în aplicație.
package:flutter/material.dart	Acest pachet reprezintă baza designului vizual în Flutter, oferind o gamă largă de widget-uri stilizate conform principiilor Material Design. Este esențial pentru construirea unei interfețe utilizator atractive și responsive.
package:camera/camera.dart	Furnizează funcționalități necesare pentru integrarea și controlul camerei dispozitivului. Permite aplicației să acceseze camera pentru a captura fotografii sau a înregistra videoclipuri, ceea ce este vital pentru aplicații care necesită interacțiune directă cu hardware-ul dispozitivului.
package:http/http.dart as http	Crucial pentru realizarea cererilor HTTP. Această importare adaugă capacitatea de a comunica cu servere externe, a trimite și primi date, facilitând astfel interacțiunile cu diverse servicii web.
package:permission_handler/permission_handler.dart	Gestionează cererile de permisiuni la nivel de sistem de operare. Este important pentru a asigura că aplicația respectă normele de confidențialitate ale utilizatorilor și are accesul necesar pentru a funcționa corect, în special în domeniile care implică accesul la componentele sensibile ale dispozitivului, cum ar fi camera și locația.

Prin utilizarea acestor pachete, aplicația noastră poate gestiona eficient comunicarea între interfața utilizator și sistemul de operare al dispozitivului, îmbunătățind experiența generală a utilizatorului și facilitând dezvoltarea de funcționalități complexe.

Variabila pentru stocarea camerelor disponibile

Pentru a gestiona camerele disponibile pe dispozitiv, aplicația noastră utilizează o variabilă globală numită cameras. Aceasta este de tipul List<CameraDescription>?, unde CameraDescription este o clasă care descrie proprietățile unei camere fizice pe dispozitiv, cum ar fi lentilele frontale sau cele aranjate pe spate. Utilizarea unei liste permite aplicației să gestioneze multiple camere, dacă sunt disponibile.

Variabila este declarată ca nullable (semnul întrebării ? la sfârșitul tipului), permițându-i să fie null în cazul în care nu sunt detectate camere disponibile sau dacă permisiunile necesare nu sunt acordate.

Funcția principală a aplicației

Aplicația noastră începe cu funcția main(), care este marcată ca asincronă (async) pentru a permite utilizarea operațiunilor asincrone în cadrul acesteia. Această funcție este esențială pentru inițializarea și lansarea aplicației.

Detalii de implementare:

Inițializarea Flutter	Primul pas este apelul WidgetsFlutterBinding.ensureInitialized(). Acest apel este crucial pentru a se asigura că toate serviciile de backend Flutter sunt inițializate înainte de a încărca orice widget sau de a rula aplicația. Aceasta este o practică standard în aplicațiile Flutter pentru a preveni erorile legate de accesul la resursele Flutter înainte de inițializarea completă a acestora.
Detectarea Camerelor	Următorul pas implică inițializarea variabilei cameras prin executarea funcției await availableCameras(). Acest apel asincron detectează și returnează o listă a camerelor disponibile pe dispozitiv. Funcția availableCameras() este parte din pachetul camera, pe care l-am importat anterior, și este esențială pentru aplicațiile care necesită acces la camera dispozitivului.
Lansarea Aplicației	Ultimul pas în funcția main() este apelul runApp(MyApp()). Acesta este punctul în care widget-ul rădăcină al aplicației, MyApp(), este construit și afișat. runApp() este funcția care încarcă și afișează interfața utilizator definită în MyApp(), începând ciclul de viață al aplicației.

Funcția main este punctul de intrare în aplicație, asigurându-se că totul este setat în mod corespunzător înainte de a permite utilizatorului să interacționeze cu aplicația. Inițializăm binding-ul widget-urilor, obținem lista camerelor disponibile și lansăm aplicația cu widget-ul MyApp ca rădăcină.

Clasa „MyApp”

Clasa MyApp este definită ca un widget StatelessWidget, ceea ce înseamnă că nu gestionează niciun fel de stare internă. Acesta este widget-ul rădăcină al aplicației noastre și este punctul de plecare pentru construcția interfeței utilizator.

Detalii de implementare:

Constructor	Clasa MyApp primește un argument opțional key, care este transmis constructorului clasei de bază StatelessWidget. Aceasta permite widget-ului să fie identificat în mod unic în cadrul arborelui de widget-uri, facilitând gestionarea stării în cazul în care ar fi necesar.
Metoda build	Aceasta este o metodă esențială în orice widget Flutter și este responsabilă pentru descrierea părții de UI pe care widget-ul o reprezintă. În cazul nostru, build returnează un obiect MaterialApp, care este un wrapper convenabil ce encapsulează mai multe funcționalități de bază ale unei aplicații, incluzând navigația, temele și așa mai departe.
Configurarea MaterialApp	1. title: Setează titlul aplicației, care este utilizat de dispozitivul pe care rulează aplicația pentru a gestiona resursele asociate. 2. theme: Definim tema aplicației folosind ThemeData, specificând primarySwatch ca fiind Colors.blue, ceea ce va influența culoarea principală a temei și diverse aspecte vizuale ale widget-urilor Material. 3. visualDensity: Ajustează densitatea vizuală a elementelor de interfață, permițând aplicației să se adapteze la diferite platforme și dimensiuni de display.
Ecranul Principal	home este setat ca CameraScreen(), care indică că prima pagină afișată când aplicația este lansată va fi CameraScreen. Acesta este un widget necesar care gestionează funcționalitățile legate de camera dispozitivului.

Prin configurarea acestor parametri, clasa MyApp setează fundamentul pentru aspectul și funcționalitatea aplicației, asigurând o structură de bază coerentă și o navigare eficientă.

Clasa „CameraScreen”

Clasa CameraScreen este un StatefulWidget în Flutter, ceea ce înseamnă că acest widget poate gestiona starea internă care se poate schimba de-a lungul timpului. Aceasta este folosită pentru a construi și menține interfața utilizatorului care va gestiona afișarea și controlul camerei pe dispozitiv.

Detalii de implementare:

Constructor	Similar cu alte widget-uri Flutter, CameraScreen acceptă un parametru key opțional care este transmis constructorului de bază StatefulWidget. Aceasta permite widget-ului să fie identificat în mod unic în cadrul arborelui de widget-uri și poate fi util în gestionarea stării sau în performanța optimizărilor.
Metoda createState	Aceasta este o caracteristică crucială a oricărui StatefulWidget. Metoda createState este responsabilă pentru crearea stării asociate widget-ului. În cazul nostru, aceasta returnează o instanță a clasei _CameraScreenState, care reprezintă locul în care logica de stare și construcția UI vor fi gestionate. Prin separarea logicii de stare de reprezentarea UI, Flutter facilitează gestionarea ciclului de viață al stării și interfeței utilizator.

CameraScreen este esențial pentru funcționalitatea de bază a aplicației care implică camera, oferind utilizatorilor posibilitatea de a interacționa cu hardware-ul camerei într-un mod eficient. Aceasta reprezintă punctul de integrare între utilizator și funcționalitățile dispozitivului de captură video sau fotografică.

Clasa „ CameraScreenState”

Clasa _CameraScreenState extinde State<CameraScreen> și este responsabilă pentru gestionarea stării interne și a comportamentului widget-ului CameraScreen.

Detalii de implementare:

Variabile de stare	1. CameraController? controller: Această variabilă gestionază instanța controllerului pentru camera dispozitivului, facilitând captura de imagini sau video. 2. Timer? timer: Utilizat pentru a executa acțiuni la intervale regulate, de exemplu, pentru a actualiza interfața utilizator sau pentru a limita timpul de captură. 3. bool isUploading = false: Un indicator de stare care marchează dacă datele sunt în proces de încărcare, util pentru gestionarea UI, cum ar fi afișarea spinner-ului de încărcare.
Ciclul de viață al widget-ului	initState: Această metodă este apelată atunci când starea widget-ului este inițial creată. Se folosește pentru a efectua inițializări, cum ar fi solicitarea permisiunilor necesare pentru camera și configurarea inițială a controllerului camerei. dispose: Metoda dispose este apelată când starea widget-ului este permanent eliminată din arborele de widget-uri. Aici, resursele sunt

	eliberate, cum ar fi controllerul camerei și timerul, pentru a preveni scurgerile de memorie.
Gestionarea permisunilor	_requestPermissions(): O metodă care gestionează solicitarea permisiunilor necesare pentru utilizarea camerei. Aceasta este crucială pentru asigurarea accesului la hardware-ul camerei în conformitate cu politicile de confidențialitate și securitate.

Prin gestionarea ciclului de viață al resurselor și starea de încărcare, _CameraScreenState asigură o experiență fluidă și responsabilă utilizatorului, permițând captura și afișarea imaginilor sau videoclipurilor în timp real, în timp ce menține performanța aplicației și respectarea normelor de securitate.

Cererea permisunilor

În clasa _CameraScreenState, gestionarea accesului la resursele dispozitivului este crucială pentru funcționalitatea corectă a aplicației. Funcția _requestPermissions() este responsabilă pentru acest aspect, asigurându-se că aplicația are permisiunile necesare pentru a opera cu camera, microfonul și stocarea.

Detalii de implementare:

Cererea de permisiuni	Lista de permisiuni include Permission.camera, Permission.microphone și Permission.storage, fiecare fiind necesar pentru diferite aspecte ale aplicației: 1. Permission.camera: Necessar pentru accesul și controlul camerei dispozitivului pentru captură foto sau video. 2. Permission.microphone: Esențial pentru înregistrarea audio, util în scenarii unde aplicația captează video cu sunet. 3. Permission.storage: Permite aplicației să salveze datele capturate, cum ar fi fotografii sau videoclipuri, pe dispozitiv.
Inițializarea camerei	După obținerea permisiunilor, funcția _initCamera() este apelată pentru a configura și pregăti camera pentru utilizare.

Prin implementarea metodei _requestPermissions(), aplicația se asigură că toate permisiunile necesare sunt acordate înainte de a accesa hardware-ul dispozitivului. Aceasta este o practică standard în dezvoltarea aplicațiilor mobile, esențială pentru protejarea confidențialității utilizatorilor și asigurarea unei funcționări fără erori a aplicației.

Inițializarea camerei

Funcția _initCamera() este esențială în configurarea și pregătirea camerei pentru utilizare în aplicație.

Detalii de implementare:

Crearea controlerului camerei	1. CameraController: Acesta este creat folosind prima cameră disponibilă din lista cameras și este setat să folosească un preset de rezoluție medie (ResolutionPreset.medium). Acest controler facilitează interacțiunea directă cu hardware-ul camerei. 2. controller!.initialize(): Această metodă asincronă inițializează camera pentru utilizare, pregătind dispozitivul pentru capturarea de imagini sau video.
Setarea modului de blitz	controller!.setFlashMode(FlashMode.off): Asigură că blitzul camerei este dezactivat. Aceasta este o configurație importantă pentru a evita utilizarea neașteptată a blitzului care poate influența calitatea capturilor în anumite condiții de lumină.
Verificarea montării widget-ului	if (!mounted) return;: Această verificare este crucială pentru a preveni actualizări ale stării dacă widget-ul nu este montat în arborele de widget-uri, prevenind astfel erori legate de starea widget-ului.
Actualizarea stării UI	setState(() {});: După inițializarea camerei, acest call pentru setState este folosit pentru a declanșa reconstruirea interfeței utilizator, astfel încât orice elemente vizuale legat de cameră să fie actualizat corespunzător.

Prin utilizarea acestei funcții, aplicația se asigură că camera este configurată corect și gata de utilizare imediat ce utilizatorul dorește să înceapă captura de imagini sau video. Această abordare ajută la gestionarea eficientă a resurselor hardware și la asigurarea unei experiențe utilizator optimizate.

Trecerea la modul de upload

Metoda _toggleUploading() din clasa _CameraScreenState este folosită pentru a comuta între stările de captură și de oprire a imaginilor din camera dispozitivului. Această metodă este esențială pentru controlul fluxului de date generat de capturile camerei.

Detalii de implementare:

Comutarea stării de încărcare	isUploading = !isUploading;: Schimbă starea variabilei isUploading. Dacă starea era false, devine true și invers. Această comutare este

	folosită pentru a controla dacă aplicația trebuie să capteze și să încarce imagini continuu sau să oprească acest proces.
Inițierea capturii de imagini	_startImageCaptureLoop(): Dacă noua stare isUploading este true, se inițiază procesul de captură continuă a imaginilor prin această metodă.
Oprirea timerului	timer?.cancel();: Dacă starea isUploading devine false, orice timer existent care gestiona captura periodică a imaginilor este oprit. Aceasta este o măsură importantă pentru a economisi resursele dispozitivului când captura continuă nu este necesară.

Această funcție este esențială pentru gestionarea eficientă a resurselor și pentru controlul fluxului de captare și încărcare a datelor în aplicație. Permite utilizatorilor să controleze când și cum dorește aplicația să capteze datele, oferind flexibilitate și eficiență în utilizarea resurselor dispozitivului.

Capturarea periodică a imaginilor

Metoda _startImageCaptureLoop() din clasa _CameraScreenState este proiectată pentru a iniția un ciclu de capturare automată a imaginilor la intervale regulate. Aceasta folosește un timer pentru a declanșa captura de imagini la fiecare n secunde.

Detalii de implementare:

Configurarea timer-ului	Timer.periodic(Duration(seconds: n), ...): Această instrucțiune creează un timer care declanșează un eveniment la fiecare n secunde. Duration(seconds: n) specifică intervalul de timp după care timerul trebuie să fie redeclanșat.
Funcția callback a timer-ului	Timer t => _takePicture(): Funcția callback este invocată de fiecare dată când timerul expiră, adică la fiecare n secunde. _takePicture() este metoda responsabilă pentru capturarea efectivă a unei imagini folosind camera dispozitivului.

Metoda _startImageCaptureLoop este crucială pentru funcționalități care necesită monitorizare sau înregistrare continuă, cum ar fi sistemele de supraveghere video. Prin automatizarea procesului de captură a imaginilor, reduce necesitatea intervenției manuale și asigură că aplicația poate colecta imagini constant, fără întârzieri.

Capturarea unei imagini

Metoda _takePicture() din clasa _CameraScreenState este folosită pentru a captura o imagine folosind camera dispozitivului. Aceasta este o parte esențială a procesului de captură și încărcare a imaginilor în aplicație.

Detalii de implementare:

Verificarea inițializării și stării de încărcare	if (!controller!.value.isInitialized || !isUploading): Această condiție asigură că metoda nu încearcă să captureze o imagine dacă camera nu este inițializată sau dacă starea isUploading este false. Acest lucru previne erori în cazul în care camera nu este pregătită sau dacă aplicația nu trebuie să capteze imagini.
Captura imaginii	final image = await controller!.takePicture(): Această linie captează efectiv o imagine folosind camera. Utilizarea await indică faptul că takePicture() este o operație asincronă care poate dura un timp, deoarece implică accesul la hardware-ul camerei.
Trimiterea Imaginii la Server	_sendImageToServer(image.path): Dacă condiția isUploading este încă true după capturarea imaginii, aceasta este trimisă la server pentru procesare și stocare ulterioară.
Gestionarea Erorilor	catch (e): Captura erorilor este esențială pentru a trata situațiile în care captura imaginii eșuează. Mesajul de eroare este afișat în consolă, ceea ce ajută la diagnosticarea problemei.

Metoda _takePicture este crucială pentru funcționalitățile aplicației care necesită interacțiune directă cu camera, oferind un control detaliat asupra procesului de captură și gestionare a datelor de intrare de la camera dispozitivului.

Trimiterea imaginilor la server

Metoda _sendImageToServer(String imagePath) din clasa _CameraScreenState este responsabilă pentru încărcarea imaginilor capturate pe un server. Aceasta este o componentă esențială pentru funcționalitățile de rețea ale aplicației, permițând partajarea sau procesarea ulterioară a datelor.

Detalii de implementare:

Comutarea stării de încărcare	isUploading = !isUploading;: Această linie schimbă starea variabilei isUploading. Dacă starea era false, devine true și invers. Această comutare este folosită pentru a controla dacă aplicația trebuie să capteze și să încarce imagini continuu sau să oprească acest proces.

Inițierea capturii de imagini	_startImageCaptureLoop(): Dacă noua stare isUploading este true, se inițiază procesul de captură continuă a imaginilor prin această metodă.
Oprirea Timerului	timer?.cancel();: Dacă starea isUploading devine false, orice timer existent care gestiona captura periodică a imaginilor este oprit. Aceasta este o măsură importantă pentru a economisi resursele dispozitivului când captura continuă nu este necesară.

Această funcție este esențială pentru gestionarea eficientă a resurselor și pentru controlul fluxului de captare și încărcare a datelor în aplicație. Permite utilizatorilor să controleze când și cum dorește aplicația să capteze datele, oferind flexibilitate și eficiență în utilizarea resurselor dispozitivului.

Construirea UI-ului

Metoda build() în clasa de stare a widget-ului CameraScreen este responsabilă pentru construirea interfeței utilizator a aplicației. Aceasta definește cum sunt aranjate și afișate elementele UI pe ecran.

Detalii de implementare:

Structura Scaffold	Oferă structura de bază pentru layout-ul ecranului, incluzând un appBar și un body.
Bara de aplicație	AppBar: Conține un titlu, în acest caz „Camera to Server”, care descrie funcționalitatea principală a aplicației.
Conținutul principal	1. Condițional controller == null || !controller!.value.isInitialized: Dacă controller-ul camerei nu este inițializat, se afișează un indicator de progres (CircularProgressIndicator), semnalând că aplicația este în proces de inițializare. 2. CameraPreview(controller!): Odată ce camera este inițializată, se afișează previzualizarea camerei, permițând utilizatorului să vadă în timp real ce capturează camera.
Butonul de acțiune	FloatingActionButton: Butonul este configurat să apeleze metoda _toggleUploading când este apăsat. Iconița butonului se schimbă din „pause” în „cloud_upload” în funcție de starea isUploading, permițând utilizatorului să controleze încărcarea imaginilor.
Locația butonului	FloatingActionButtonLocation.centerFloat: Plasează butonul plutitor în centrul inferior al ecranului, facilitând accesul pentru utilizator.

Această configurație UI este esențială pentru asigurarea unei experiențe intuitive pentru utilizatori, permițându-le să interacționeze eficient cu funcționalitățile aplicației, de la vizualizarea camerei până la controlul procesului de încărcare a imaginilor.

5.1.4 Detalii tehnice

1. Rolul timerului setat la n secunde

Captură periodică	Capturarea periodică de imagini asigură că aplicația colectează date în mod constant fără intervenție manuală.
Monitorizare continuă	Ideal pentru aplicații de monitorizare, unde este necesar să se capteze imagini la intervale regulate pentru a urmări schimbările.
Performanță și rată de date	Intervalul de n secunde asigură un echilibru între performanța aplicației și cantitatea de date trimise la server.

Putem varia captura de imagini de către camera telefonului mobil într-un interval mai scurt sau mai lung, în funcție de lipsa/necesitatea detaliilor. Un interval mai scurt ne ajută să analizăm mediul de lucru mai atent si să luăm decizii mai rapide si mai eficiente. In același timp, un interval de timp mai scurt poate necesita stocarea mai multor imagini si prelucrarea mai multor informații, ceea ce reprezintă un cost computațional mai mare.

2. Comunicarea dintre aplicația mobilă și server

Aplicația mobilă comunică cu serverul folosind cereri HTTP, care sunt gestionate în Dart prin intermediul pachetului http. Acest pachet permite aplicației să trimită cereri HTTP către server pentru a transfera date, cum ar fi imaginile capturate de cameră. În această aplicație, folosim metodele POST pentru a încărca imagini către server.

URL-ul Serverului: Aplicația trimite datele către un URL specificat, care este serverul nostru.

Trebuie sa verificăm că aplicația mobilă are acces la url-ul corect al serverului pentru a putea comunica cu acesta.

5.1.5 Perspectiva utilizatorului

Ecranul de lansare şi cerere de permisiuni

La lansarea aplicației, utilizatorul vede un ecran de încărcare cu un indicator de progres rotativ (spinner).

Aplicația solicită permisiunea de a accesa camera pentru a putea face fotografii și a înregistra videoclipuri. Utilizatorul are opțiunea de a permite sau de a refuza accesul.

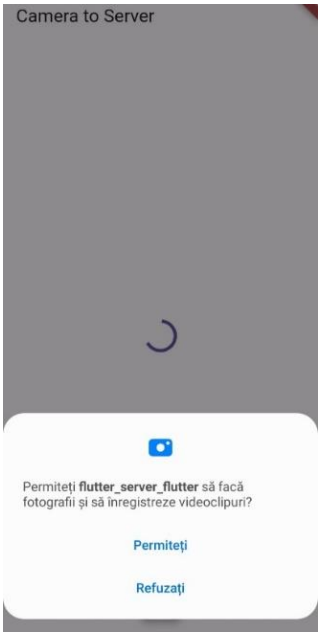

Figura 13 Permisiuni foto/video

Urmează o a doua cerere de permisiune pentru a accesa fotografiile și conținutul media de pe dispozitiv. Aceasta este necesară pentru a stoca și gestiona imaginile capturate.

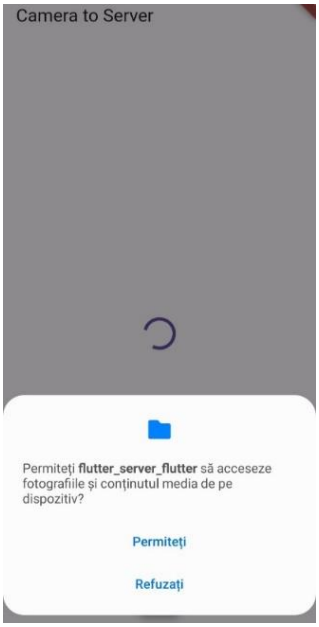

Figura 14 Permisiuni de acces

Ecranul principal cu camera activă

După ce utilizatorul acordă permisiunile necesare, ecranul principal al aplicației afișează fluxul video de la camera dispozitivului.

În partea superioară a ecranului, se află bara de titlu cu textul "Camera to Server", titlu cu scop informativ.

Butonul de control pentru incărcarea imaginilor

În partea inferioară a ecranului, se află un buton de acțiune rapidă (Floating Action Button) sub formă de pictogramă de cloud. Acest buton permite utilizatorului să înceapă sau să oprească procesul de încărcare a imaginilor pe server.

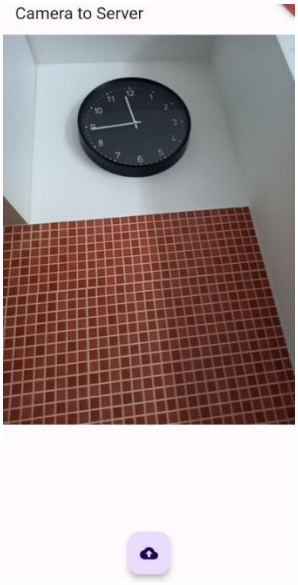

Figura 15 Preview app fără transmitere de cadre

Când utilizatorul apasă butonul, iconița acestuia se schimbă pentru a indica starea curentă: o pictogramă de pauză pentru a opri încărcarea sau o pictogramă de cloud pentru a începe încărcarea.

Indicator de progres și mesaje de stare: În timpul procesului de încărcare, aplicația poate afișa indicatori de progres și mesaje pentru a informa utilizatorul despre starea actuală a încărcărilor.

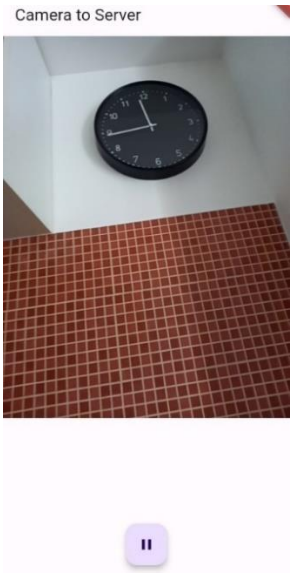

Figura 16 Preview cu transmitere de cadre

5.1.6 Perspectiva dezvoltatorului

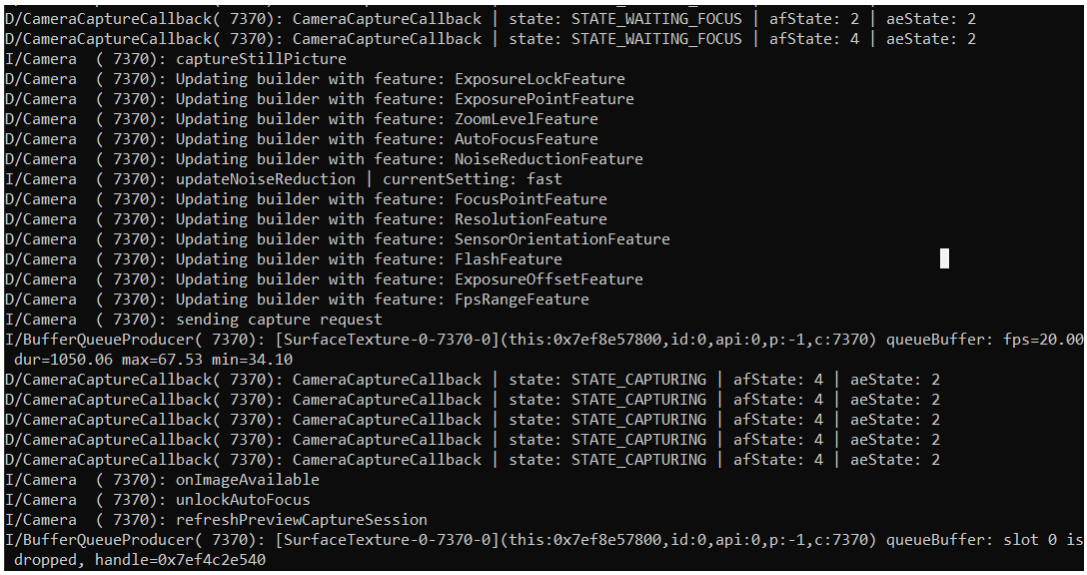

Figura 17 Log cu informații despre starea execuției aplicației

Putem observa jurnalul de debug al aplicației mobile în timpul rulării acesteia, evidențiind procesele interne și schimbările de stare ale camerei.

Monitorizarea și depanarea	Dezvoltatorul folosește aceste jurnale pentru a monitoriza comportamentul aplicației și a depana eventualele probleme care apar în timpul capturării imaginilor.
Optimizarea performanței	Analizarea jurnalelor ajută la optimizarea performanței aplicației, de exemplu, ajustarea funcționalităților camerei și asigurarea unei experiențe fluide pentru utilizator.
Verificarea funcționalităților	Confirmarea că toate funcționalitățile implementate sunt corect actualizate și utilizate în timpul capturării imaginilor.

5.1.7 Funcționalitate

Aplicația a fost dezvoltată pentru a studia un proces de automatizare, concentrându-se pe o singură funcționalitate esențială: capturarea imaginilor din mediul real și transmiterea acestora către un server. În acest scop, interfața utilizatorului este simplă și minimalistă, cu puține detalii și acțiuni, permițând o utilizare ușoară și eficientă.

Utilizatorul poate iniția capturarea periodică a imaginilor printr-o simplă apăsare a unui buton. Imaginile sunt apoi trimise automat către serverul specificat. Acest proces este optimizat pentru a asigura o transmisie stabilă și eficientă a datelor, indiferent de condițiile de rețea.

Deși aplicația are un design de bază pentru a facilita înțelegerea procesului de capturare și transmitere a imaginilor, ea oferă flexibilitatea necesară pentru a fi extinsă cu ușurință. Putem adăuga noi funcționalități și acțiuni pentru a îmbunătăți și diversifica utilizarea aplicației.

5.2 Markeri ArUco

5.2.1 Aspect si dicționare

Un marker ArUco este un marker sintetic pătrat, compus dintr-o margine neagră lată și o matrice binară interioară care determină identificatorul său (id). Marginea neagră facilitează detectarea rapidă în imagine, iar codificarea binară permite identificarea și aplicarea tehnicilor de detectare și corecție a erorilor. Dimensiunea markerului determină dimensiunea matricei interne. De exemplu, un marker de dimensiune 4x4 este compus din 16 biți.

Câteva exemple de markeri ArUco:

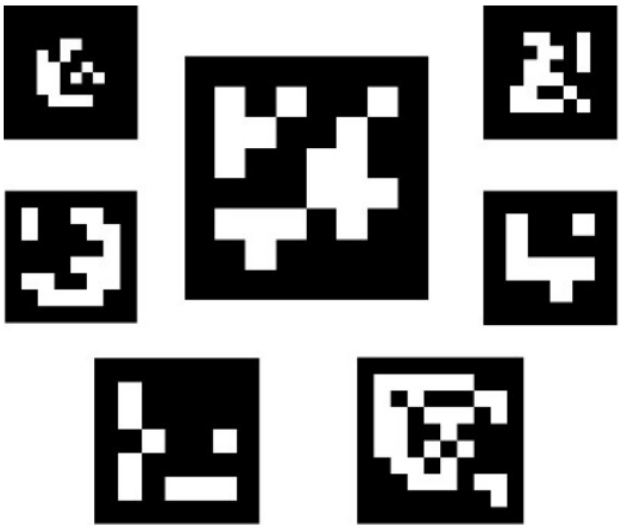

Figura 18 Markeri ArUco

Trebuie menționat că un marker poate fi găsit rotit în mediul înconjurător; cu toate acestea, procesul de detectare trebuie să fie capabil să determine rotația sa originală, astfel încât fiecare colț să fie identificat în mod neechivoc. Acest lucru se face și pe baza codificării binare.

Un dicționar de markeri este setul de markeri care sunt considerați într-o aplicație specifică. Este pur și simplu lista codificărilor binare ale fiecărui marker.

Proprietățile principale ale unui dicționar sunt dimensiunea dicționarului și dimensiunea markerului.

Dimensiunea dicționarului este numărul de markeri care compun dicționarul.

Dimensiunea markerului este dimensiunea acelor markeri (numărul de biți/module).

Modulul ArUco include câteva dicționare predefinite care acoperă o gamă de dimensiuni diferite de dicționare și markeri.

S-ar putea crede că id-ul markerului este numărul obținut prin convertirea codificării binare într-un număr în baza zecimală. Totuși, acest lucru nu este posibil deoarece, pentru dimensiuni mari ale markerilor, numărul de biți este prea mare și gestionarea unor astfel de

numere uriașe nu este practică. În schimb, un id de marker este pur și simplu indexul markerului în cadrul dicționarului căruia îi aparține. De exemplu, primii 5 markeri dintr-un dicționar au id-urile: 0, 1, 2, 3 și 4.

5.2.2 Eficiența

Am ales să folosim acești markeri în proiectul nostru deoarece sunt extrem de utili pentru:

Detecție rapidă și precisă	Markerii ArUco permit detectarea rapidă și precisă a obiectelor în imagini. Algoritmii de detectare sunt bine optimizați și pot identifica markerii chiar și în condiții de iluminare variabilă sau zgomot vizual. Aceasta este esențială pentru controlul scheletului robotizat, ne asigură ca acesta se orientează corect in ciuda oricărui impediment.
Robustețe	Markerii ArUco sunt proiectați să fie detectați fiabil în diverse condiții de mediu. Marginea neagră lată facilitează detecția, iar codificarea binară permite aplicarea tehnicilor de corecție a erorilor, asigurând o identificare corectă chiar și în prezența distorsiunilor sau a variațiilor de iluminare.
Implementare și integrare ușoară	Utilizarea markerilor ArUco este susținută de biblioteci bine documentate și ușor de utilizat, cum ar fi OpenCV. Aceasta reduce timpul și efortul necesar pentru implementarea și integrarea funcționalității de detecție a markerilor în proiectul nostru.
Costuri reduse	Markerii ArUco sunt o soluție cost-eficientă pentru detecția și localizarea obiectelor. Nu necesită hardware specializat, fiind suficientă utilizarea unei camere standard și a unui software pentru procesarea imaginii. Aceasta face ca tehnologia să fie accesibilă și scalabilă.

5.2.3 Detalii tehnice

<u>Dicționarul ales</u>

În cadrul sistemului nostru, folosim un dicționar predefinit din biblioteca ArUco inclusă în OpenCV. Dicționarul specific utilizat este DICT_6X6_250. Acest dicționar are următoarele caracteristici:

Dimensiunea markerului	Fiecare marker are o matrice binară de 6x6, adică 36 de biți.
Dimensiunea dicționarului	Dicționarul conține 250 de markeri diferiți.

Dimensiunea de 6x6 este suficient de mare pentru a include tehnici de corecție a erorilor, reducând șansele de identificare greșită a markerilor, mai ales în condiții de zgomot sau iluminare variabilă.

Vedere în plan

Markerii vor fi distribuiți atât în plan, cât și pe scheletul robotizat. În acest mod ne asigurăm că următoarele sarcini sunt realizate corect:

- orientarea direcției
- calculul distanței de la schelet (față/spate) la diferiți markeri din plan
- obținerea diverselor dependențe intre schelet si mediu

5.3 Server

5.3.1 Tehnologiile alese

Flask

Flask este un micro-framework web pentru Python, care oferă o structură ușoară și flexibilă pentru dezvoltarea aplicațiilor web. Este util din mai multe motive:

Simplitate și flexibilitate	Flask oferă doar elementele de bază necesare pentru a dezvolta o aplicație web, permițând dezvoltatorilor să adauge extensii și biblioteci după nevoie. Aceasta îl face foarte flexibil și ușor de utilizat pentru proiecte mici și mijlocii.
Dificultate de invățare redusă	Flask are o documentație excelentă și o comunitate activă, ceea ce îl face ușor de învățat și utilizat, chiar și pentru începători. API-ul său simplu și intuitiv facilitează dezvoltarea rapidă.
Extensibilitate	Chiar dacă Flask vine cu funcționalități de bază, există numeroase extensii disponibile (cum ar fi Flask-SQLAlchemy pentru baze de date, Flask-RESTful pentru API-uri REST, etc.) care pot fi integrate ușor pentru a adăuga funcționalități avansate.
Performanță	Fiind un micro-framework, Flask nu adaugă prea multe overhead-uri, ceea ce duce la performanțe mai bune în comparație cu framework-urile mai grele.
Libertatea de arhitectură	Flask nu impune o anumită arhitectură, permițând dezvoltatorilor să își organizeze codul așa cum doresc. Aceasta este ideal pentru proiectele unde se dorește control total asupra structurii aplicației.

Python

Python este un limbaj de programare interpretat, dinamic și de nivel înalt, cunoscut pentru sintaxa sa clară și concisă. Este util pentru dezvoltarea web datorită următoarelor motive:

Citibilitate și simplitate	Sintaxa Python este simplă și clară, ceea ce facilitează scrierea și înțelegerea codului. Acest lucru reduce timpul necesar pentru dezvoltare și mentenanță.

Comunitate	Python are o comunitate mare și activă, oferind suport și resurse abundente (tutoriale, biblioteci, extensii). Flask este doar un exemplu de bibliotecă puternică și bine susținută în ecosistemul Python.
Versatilitate	Python este un limbaj versatil, utilizat nu doar în dezvoltarea web, ci și în domenii precum știința datelor, automatizarea, scripting și dezvoltarea de aplicații desktop. Aceasta permite dezvoltatorilor să utilizeze același limbaj pentru diferite părți ale unui proiect.
Integrări și biblioteci	Python beneficiază de un număr mare de biblioteci și pachete care pot fi ușor integrate în aplicații web (de exemplu, Pandas pentru manipularea datelor, NumPy pentru calcule numerice, OpenCV pentru procesarea imaginilor).
Productivitate ridicată	Datorită simplității și a ecosistemului bogat, dezvoltarea cu Python tinde să fie rapidă și eficientă. Acest lucru este deosebit de important în mediile de startup-uri sau în proiectele care necesită prototipare rapidă.

Avantaje ale combinării Flask cu Python

Dezvoltare Rapidă	Combinând simplitatea Flask cu sintaxa clară a Python, putem construi și lansa aplicația rapid. Aceasta este esențială în mediile de afaceri dinamice unde timpul de lansare este crucial.
Flexibilitate și extensibilitate	Flask permite adăugarea de extensii după nevoie, iar Python oferă o multitudine de biblioteci care pot extinde funcționalitatea aplicației.
Dificultate de învățare redusă	Atât Flask, cât și Python sunt ușor de învățat, ceea ce face această combinație ideală pentru dezvoltatorii noi sau pentru echipele care doresc să adopte un nou framework rapid.
Comunitate puternică	Ambele tehnologii au comunități mari și active, oferind suport și resurse pentru dezvoltatori. Acest lucru facilitează rezolvarea problemelor și implementarea de soluții eficiente.
Performanță și Scalabilitate	Deși Flask este un micro-framework, permite crearea de aplicații scalabile prin intermediul extensiilor și a integrării cu alte servicii și tehnologii. Python, fiind eficient și versatil, completează această capacitate.

În concluzie, utilizarea Flask cu Python oferă un mediu de dezvoltare web puternic, flexibil și eficient, potrivit pentru o gamă largă de proiecte, de la prototipuri rapide până la aplicații web complexe.

5.3.2 Implementarea serverului

Această implementare a serverului Flask permite primirea și procesarea imaginilor trimise de o aplicație mobilă, detectarea markerilor ArUco în imagini și returnarea informațiilor relevante către client. Structura modulară a codului asigură o gestionare eficientă a încărcării fișierelor, procesării imaginilor și comunicării rezultatelor, facilitând integrarea cu alte componente ale proiectului, cum ar fi microcontrollerele pentru controlul dispozitivelor.

Vom analiza pe rând componentele și funcționalitățile principale ale acestui cod.

Importarea bibliotecilor și configurarea inițială

Scriptul Python folosit în proiect utilizează mai multe biblioteci puternice pentru a gestiona funcționalități diverse, de la procesarea imaginilor la gestionarea aplicațiilor web. Iată detaliile importurilor și rolul fiecărei biblioteci:

Flask	Utilizată pentru a crea și gestiona aplicația web. Flask este un micro-framework pentru web care permite dezvoltarea rapidă a aplicațiilor web, gestionând rute și cereri HTTP.
Werkzeug	Folosit pentru manipularea numelor de fișiere în siguranță. secure_filename este o funcție din Werkzeug care se asigură că numele de fișiere sunt sigure pentru stocare pe server, prevenind atacurile de tip path traversal.
NumPy	Bibliotecă esențială pentru calcul numeric. NumPy este adesea utilizată în procesarea datelor și a imaginilor pentru manipularea eficientă a matricilor și a array-urilor mari.
OS	Folosit pentru manipularea căilor fișierelor și directoarelor. Aceasta permite scriptului să interacționeze cu sistemul de fișiere al sistemului de operare, facilitând operațiuni cum ar fi schimbarea directorului de lucru sau listarea conținutului unui director.
Aruco	Importă funcționalitatea specifică pentru detectarea markerilor ArUco din OpenCV.
OpenCV	Utilizată pentru procesarea avansată a imaginilor și detectarea markerilor ArUco. OpenCV (Open Source Computer Vision Library) este o bibliotecă vastă care permite implementarea diferitelor tehnici de vizualizare computerizată.

Această configurație inițială a scriptului este esențială pentru a asigura că toate componentele necesare sunt corect configurate și pregătite pentru a efectua sarcinile de procesare a imaginilor și gestionare a cererilor web.

Biblioteca OpenCV

Folosim biblioteca OpenCV în mod special pentru a identifica markerii ArUco din imagini și pentru a afla distanțele dintre aceștia.

Identificarea markerilor ArUco în imagini se face prin următorii pași principali:

Conversia imaginii la grayscale	Markerii ArUco sunt alb-negru, iar conversia imaginii color la grayscale simplifică procesul de detecție.
Detecția contururilor	Algoritmul caută contururi pătrate în imagine, deoarece markerii ArUco au formă pătrată.
Validarea markerilor	După identificarea contururilor pătrate, se verifică dacă interiorul acestora respectă modelul binar specific unui marker ArUco. Fiecare marker ArUco are un cod unic binar care este citit și validat împotriva unui dicționar de markeri cunoscuți.
Identificarea și decodarea markerilor	După validare, codul binar din interiorul conturului pătrat este decodificat pentru a determina ID-ul unic al markerului.
Calcularea colțurilor markerilor	Se identifică coordonatele colțurilor fiecărui marker detectat, care sunt utilizate pentru diverse aplicații ulterioare (de exemplu, estimarea poziției).

Calcularea distanței între markerii ArUco într-o imagine se face prin următorii pași principali:

Detectarea markerilor	Se detectează markerii și colțurile acestora în imagine.
Calcularea centrelor markerilor	Se calculează centrul fiecărui marker folosind coordonatele colțurilor.
Calcularea distanței Euclidiene între centrele markerilor	Se calculează distanța Euclidiană între centrele markerilor folosind formula distanței Euclidiene în 2D.

Se calculează distanța Euclidiană între centrele markerilor folosind formula distanței Euclidiene în 2D.

Formula distanței Euclidiene:

Distanța Euclidiană între două puncte (x1, y1) și (x2, y2) în planul 2D este dată de formula:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Figura 19 Formula distanței Euclidiene

Inițializarea Aplicației Flask

În această secțiune a scriptului, se configurează aplicația Flask pentru a gestiona încărcarea și stocarea fișierelor pe server. Configurarea implică stabilirea unui folder pentru încărcările de fișiere și definirea tipurilor de fișiere permise.

Detalii de implementare:

Configurarea folderului de încărcări	UPLOAD_FOLDER: Variabila este setată la 'uploads', ceea ce indică directorul relativ unde serverul va stoca fișierele încărcate de utilizatori.
Setarea extensiilor de fișiere permise	ALLOWED_EXTENSIONS: Aceasta este o setare de securitate importantă care limitează tipurile de fișiere ce pot fi încărcate la server, acceptând doar imagini de tip PNG, JPG, JPEG și GIF. Aceasta previne încărcarea de fișiere potențial periculoase.
Configurarea aplicației Flask	app.config['UPLOAD_FOLDER']: Această linie atribuie folderul de încărcări configurat anterior la setările aplicației Flask, asigurându-se că, calea este accesibilă global prin aplicație.
Crearea directorului de încărcări	os.makedirs(UPLOAD_FOLDER, exist_ok=True): Această funcție asigură crearea folderului 'uploads' dacă acesta nu există deja, prevenind erori la încercarea de a salva fișierele încărcate.

Această configurație inițială este esențială pentru a asigura că aplicația Flask poate gestiona în mod eficient și sigur încărcările de fișiere, oferind baza necesară pentru funcționalități ulterioare de manipulare și procesare a fișierelor.

Dicționar pentru mesaje ArUco

Scriptul include inițializarea unui dicționar denumit aruco_messages destinat să stocheze mesaje legate de markerii ArUco detectați. Acest dicționar este utilizat pentru a comunica rezultatele detectării markerilor către client sau alte părți ale aplicației.

Funcția de detectare a markerilor ArUco

Funcția detect_aruco_markers() este concepută pentru a analiza o imagine și a identifica prezența markerilor ArUco, o tehnică specifică în domeniul viziunii computerizate pentru recunoașterea unor tipuri specifice de coduri vizuale.

Detalii de implementare:

Încărcarea și pregătirea imaginii	Imaginea este încărcată folosind cv2.imread(image_path). Se configurează dicționarul ArUco și parametrii detectorului, esențiali pentru identificarea markerilor în imagine.
Detectarea markerilor	cv2.aruco.detectMarkers(): Această funcție identifică markerii în imagine, returnând colțurile, ID-urile markerilor detectați și candidații respinși.
Calculul distanțelor între markerii detectați	Dacă sunt detectați mai mulți markeri, funcția calculează distanța euclidiană între fiecare pereche de markeri și stochează aceste distanțe într-un array.
Stocarea și afișarea rezultatelor	Mesajele cu rezultatele detectării sunt stocate în dicționarul aruco_messages și sunt, de asemenea, afișate în consolă.

Această funcție utilizează biblioteca OpenCV pentru detectarea și procesarea markerilor ArUco pentru viziune computerizată, oferind un instrument puternic pentru identificarea și măsurarea relațiilor spațiale în imagini.

Funcţia pentru verificarea extensiilor permise

Funcția allowed_file() este utilizată pentru a verifica dacă fișierele încărcate de utilizatori au extensii permise de aplicație, asigurând astfel că sunt procesate doar fișierele de tip imagine într-un mod sigur.

Detalii de implementare:

Verificarea prezenței punctului în numele fișierului	return '.' in filename: Aceasta verifică dacă numele fișierului conține un punct, care este adesea folosit pentru a separa numele de fișier de extensia acestuia. Fără un punct, se presupune că fișierul nu are o extensie specificată.
Extragerea și verificarea extensiei	filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS: Funcția rsplit este folosită pentru a împărți numele fișierului într-o listă, limitând separarea la ultima apariție a punctului pentru a obține direct extensia. Extensia este apoi convertită la litere mici cu lower() și verificată dacă se află în lista ALLOWED_EXTENSIONS definită anterior.

Această funcție este apelată în cadrul unei rute Flask care gestionează încărcarea fișierelor, pentru a determina dacă fișierul primit trebuie procesat sau respins, bazat pe tipul său.

Definirea rutelor Flask

1. Definirea rutei de bază

Funcția index() definită sub decoratorul @app.route('/') este configurată ca ruta de bază a aplicației Flask. Aceasta returnează un mesaj simplu pentru a indica faptul că serverul rulează și este accesibil.

Detalii de implementare:

Decoratorul @app.route('/')	Acesta indică faptul că funcția index() va răspunde la cererile HTTP GET trimise către rădăcina URL-ului aplicației (de exemplu, http://localhost:5000/). Decoratorul @app.route este utilizat în Flask pentru a asocia adresele URL cu funcții specifice în cod.
Funcția index()	Este o funcție simplă care nu acceptă parametri și returnează un string. Mesajul „Hello, World! The Flask server is running.” este trimis ca răspuns la orice cerere către rădăcina site-ului, servind ca o confirmare vizibilă că serverul funcționează corect.

Această rută este utilizată în dezvoltarea aplicației web pentru a verifica rapid dacă serverul este în funcțiune și configurat corect. Este, de asemenea, un punct de plecare înainte de adăugarea unor rute mai complexe și funcționalități în aplicație.

2. Ruta de testare a conexiunii

Funcția test_connection() definită în cadrul aplicației Flask servește ca un punct de verificare pentru testarea conectivității între aplicația client și server. Aceasta este o metodă HTTP GET care răspunde cu un mesaj de succes atunci când este accesată.

Detalii de implementare:

Configurarea rutei	@app.route('/test', methods=['GET']): Decoratorul specifică că ruta /test acceptă cereri de tip GET. Aceasta asigură că ruta poate fi accesată simplu de către aplicații client fără a necesita transmiterea de date complexe.
Procesarea cererii	La accesarea rutei, funcția afișează un mesaj în consola serverului ("Received a test connection request.") pentru a confirma că cererea a fost recepționată.
Răspunsul JSON	jsonify({'message': 'Success! Flutter app connected to Flask server.'}): Funcția returnează un răspuns formatat JSON, care este tipul de răspuns preferat pentru aplicațiile mobile și web moderne, facilitând parsarea datelor pe partea client.

Codul de stare HTTP	Răspunsul include, de asemenea, un cod de stare HTTP 200, care indică faptul că cererea a fost procesată cu succes.

Această rută este folosită în fazele inițiale de dezvoltare pentru a verifica configurația rețelei și ulterior pentru diagnosticarea problemelor de conectivitate. Aplicația mobilă Flutter poate apela această rută pentru a confirma că poate comunica eficient cu serverul Flask.

3. Ruta pentru obținerea statusului markerilor ArUco

Funcția get_aruco_status() este definită pentru a returna informații despre markerii ArUco detectați printr-o interfață API. Aceasta permite clientilor să interogheze serverul despre ultimele detalii detectate legate de markerii ArUco.

Detalii de implementare:

Verificarea existentei informațiilor	if aruco_messages['last_message']: Această condiție verifică dacă există informații recente stocate în dicționarul aruco_messages sub cheia last_message. Dacă există informații, acestea sunt returnate clientului.
Returnarea informațiilor	jsonify({'distances': aruco_messages['last_message']}): Dacă ultimul mesaj există, informațiile sunt împachetate într-un obiect JSON care include detaliile despre distanțele între markerii detectați, facilitând interpretarea acestora de către client.
Răspuns în cazul lipsei informațiilor	jsonify({'message': 'No recent information about markers.'}): În cazul în care nu există informații recente, se returnează un mesaj clarificator, indicând lipsa actualizărilor. Codul de stare HTTP 404 sugerează că nu există resurse de returnat la momentul interogării.

Această rută este utilă deoarece sunt necesare actualizări periodice despre starea detectării markerilor ArUco pentru a procesa sau afișa informațiile în mod corespunzător.

4. Ruta pentru încărcarea fișierelor

Funcția upload_file() în cadrul aplicației Flask gestionează încărcarea fișierelor de imagini de către utilizatori. Este configurată pentru a accepta doar metode POST, adecvată pentru transferul de date mai mari cum ar fi fișierele de imagini.

Detalii de implementare:

Verificarea selectării fișierului	Se verifică dacă utilizatorul a selectat un fișier; în caz contrar, se returnează o eroare.

Salvarea și procesarea fișierului	Dacă fișierul există și are o extensie permisă, acesta este salvat în directorul specificat. Apoi, se inițiază detectarea markerilor ArUco în imaginea încărcată.
Răspunsurile JSON	Funcția returnează răspunsuri JSON care informează clientul despre rezultatul operațiunii de încărcare, fie că a fost cu succes, fie că tipul de fișier nu este suportat.

Funcția returnează răspunsuri JSON care informează clientul despre rezultatul operațiunii de încărcare, fie că a fost cu succes, fie că tipul de fișier nu este suportat.

5. Ruta pentru servirea fișierelor încărcate

Funcția serve_file() este definită pentru a facilita accesul la fișierele încărcate pe server. Ruta acceptă o cerere GET pentru a recupera un fișier specific pe baza numelui său.

Detalii de Implementare:

Parametrul din URL	<filename>: Acesta este un parametru variabil în URL-ul rutei. Flask capturează orice este specificat în acest segment al URL-ului și îl folosește ca argument pentru funcția serve_file().
Funcția send_from_directory()	Aceasta este o funcție helper oferită de Flask pentru a servi fișiere dintr-un director specificat. Aici, se utilizează pentru a trimite fișierul solicitat din directorul de încărcare configurat (UPLOAD_FOLDER).

Această rută este utilă în aplicații web deoarece ne oferă acces la fișierele încărcate. Permite accesul direct și eficient fără a necesita logica suplimentară pentru gestionarea și verificarea accesului la fișiere.

Pornirea serverului Flask

Pentru a pune în funcțiune serverul Flask, folosim o configurație specifică în cadrul scriptului Python care permite serverului să fie accesibil de pe orice host, facilitând astfel dezvoltarea și testarea aplicației din orice locație sau accesul de pe diferite dispozitive din rețea.

Detalii de implementare:

Modul de depanare	debug=True: Acest parametru activează modul de depanare în Flask, care oferă informații detaliate despre erori și reîncarcă automat serverul la modificările în cod, ceea ce este extrem de util în dezvoltarea aplicației.

Accesibilitatea serverului	host='0.0.0.0': Setarea aceasta face serverul accesibil pe toate adresele IP ale mașinii gazdă, permițând oricărui dispozitiv din aceeași rețea să acceseze aplicația.
Portul	port=5000: Specifică portul pe care serverul Flask va accepta cererile. 5000 este portul implicit pentru aplicațiile Flask, dar poate fi modificat în funcție de necesități sau restricții de rețea.

Această configurație este ideală pentru fazele de dezvoltare și testare, oferind flexibilitate și facilitând eficiența.

5.3.3 Perspectiva serverului

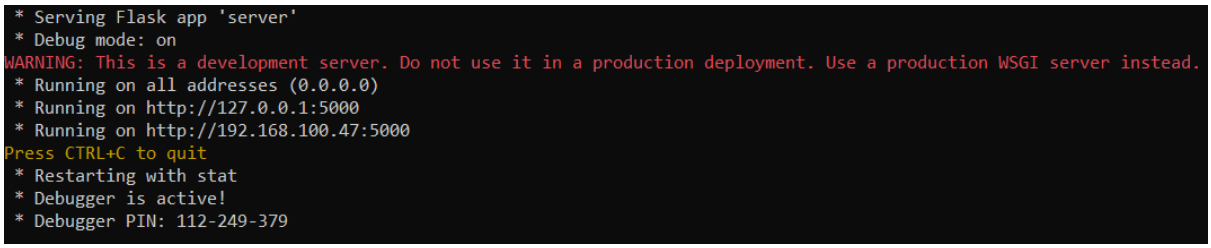

Figura 20 Log inițial al serverului

Acestea sunt informațiile de bază pe care le putem observa după pornirea serverului. Serverul Flask este accesibil pe adresa locală (127.0.0.1:5000) și pe adresa IP a rețelei (192.168.100.47:5000). Adresa 0.0.0.0 indică faptul că serverul acceptă conexiuni pe toate interfețele de rețea.

Pentru a realiza conexiunea de la aplicația mobilă la server trebuie să ne asigurăm ca ambele comunică pe adresa IP (192.168.100.47:5000). Dacă acest aspect este realizat corect, aplicația mobilă este pornită si butonul de control pentru încărcarea imaginilor este comutat pentru a permite încarcarea cadrelor din fluxul live, putem observa următoarele informații in log-ul serverului:

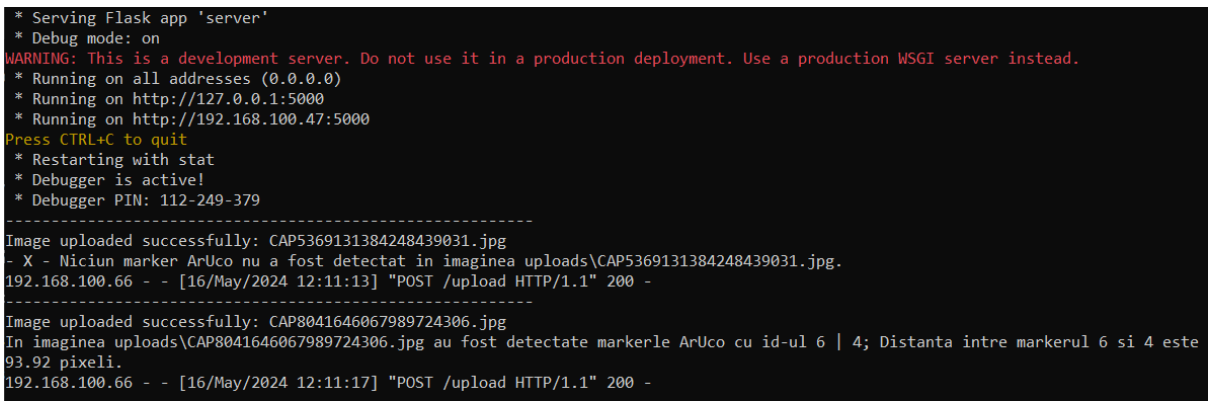

Figura 21 Log al serverului cu informații despre markeri

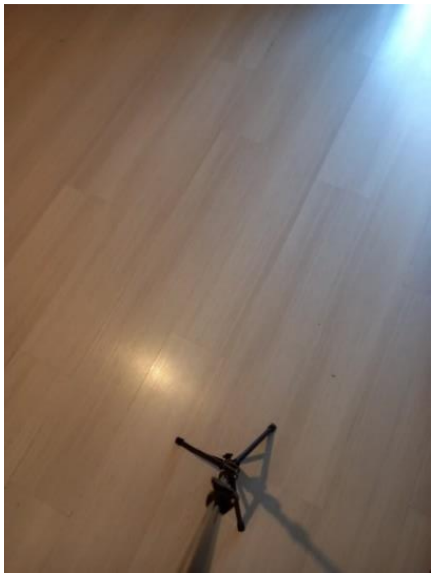

Figura 22 Perspectiva 1 a camerei asupra mediului

Figura 23 Perspectiva 2 a camerei asupra mediului

Cu ajutorul celor două imagini și a informațiilor din log aflăm că serverul încarcă cu succes imaginile primite cu ajutorul camerei telefonului mobil, prelucrează datele din acestea și întoarce cu succes informațiile de care avem nevoie.

Vom explica în următoarea secțiune, în detaliu, rețeaua pe care o realizează aplicația mobilă cu serverul.

5.4 Ansamblu aplicație mobilă – server

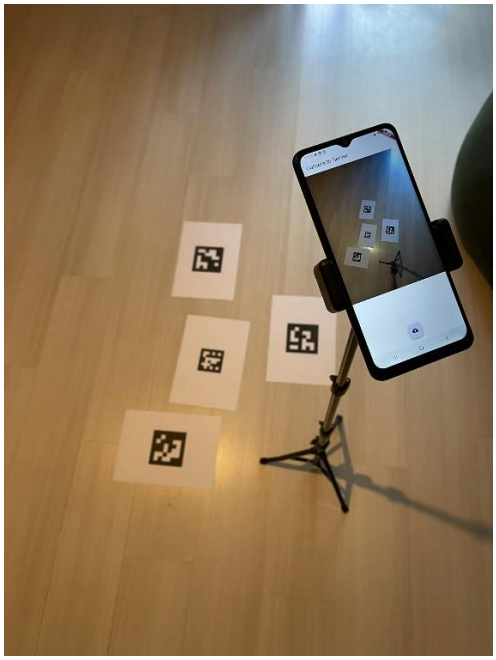

Figura 24 Perspectiva detaliată a camerei asupra mediului

Conform explicațiilor din secțiunile precedente, aplicația mobilă permite accesarea camerei și trimiterea cadrelor către server la un interval de n secunde.

În poza de mai sus, putem observa cum folosim aplicația pentru a procesa un mediu real, cu un caz simplu, în care vrem sa detectăm prezența a 4 markeri și distanțele dintre aceștia.

O dată realizată conexiunea dintre aplicația mobilă și server, cel din urmă preia cadrele și le încarcă în fișierul de upload. Fișier creat în același director cu serverul nostru.

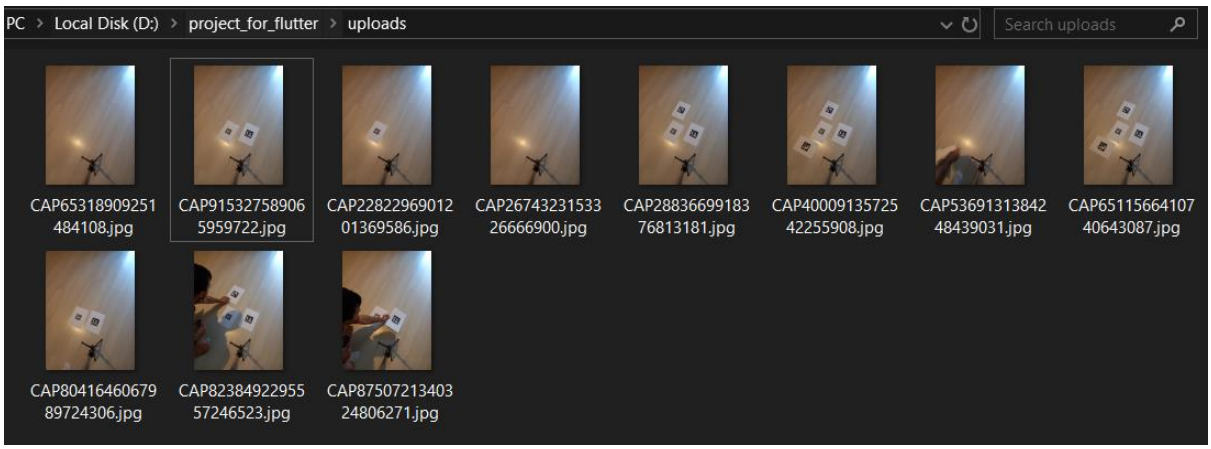

Figura 25 Stocarea cadrelor de către server în fișier

Cadrele sunt preluate constant până aplicația este oprită.

Imaginile sunt procesate corespunzător, conform explicațiilor din secțiunea precedentă, iar datele obținute din aceste informații sunt întoarse către utilizator prin log-ul serverului.

Cazuri pe care serverul le poate întâmpina

1. În imaginea încărcată nu există markeri ArUco.

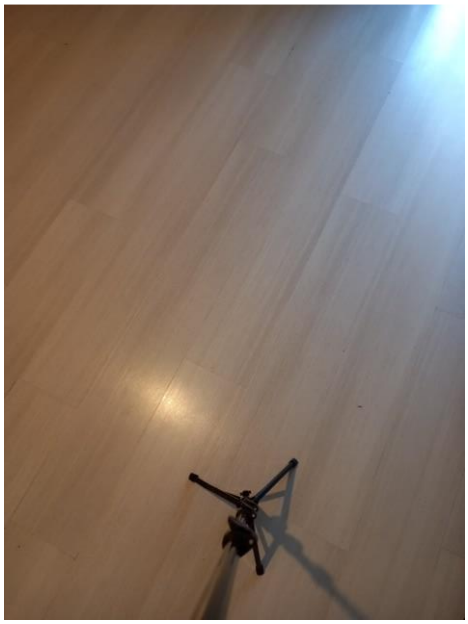

Figura 26 Perspectiva cazului 1 a camerei

Mesajul indicat:

```
Image uploaded successfully: CAP653189092514841O8.jpg
- X - Niciun marker ArUco nu a fost detectat in imaginea uploads\CAP653189092514841O8.jpg.
192.168.100.66 - - [16/May/2024 12:06:17] "POST /upload HTTP/1.1" 200 -
```

Figura 27 Log al serverului cu informații pentru cazul 1

2. În imaginea încărcată a fost detectat un singur marker ArUco.

Figura 28 Perspectiva cazului 2 a camerei

Mesajul indicat:

```
Image uploaded successfully: CAP2282296901201369586.jpg
- V - In imaginea uploads\CAP2282296901201369586.jpg a fost detectat un marker ArUco cu id-ul 4.
192.168.100.66 - - [16/May/2024 12:06:24] "POST /upload HTTP/1.1" 200 -
```

Figura 29 Log al serverului cu informații pentru cazul 2

Putem observa că serverul citește corespunzător markerul și întoarce id-ul acestuia.

3. În imaginea încărcată au fost detectați mai mulți markeri ArUco.

Figura 39 Perspectiva cazului 3 a camerei

Mesajul indicat:

```
Image uploaded successfully: CAP6511566410740643087.jpg
In imaginea uploads\CAP6511566410740643087.jpg au fost detectate markerle ArUco cu id-ul 3 | 6 | 5 | 4; Distanta intre markerul 3 si
 6 este 207.93 pixeli. Distanta intre markerul 3 si 5 este 233.60 pixeli. Distanta intre markerul 3 si 4 este 117.79 pixeli. Distant
a intre markerul 6 si 5 este 145.18 pixeli. Distanta intre markerul 6 si 4 este 111.82 pixeli. Distanta intre markerul 5 si 4 este 1
22.13 pixeli.
192.168.100.66 - - [16/May/2024 12:06:44] "POST /upload HTTP/1.1" 200 -
```

Figura 31 Log al serverului cu informații pentru cazul 3

Putem observa că serverul citește corespunzător markerii și întoarce id-ul fiecăruia și distanțele dintre aceștia.

Informațiile pe care le obținem (id-urile și distanțele) ne vor ajuta sa direcționăm scheletul robotizat în mediul nostru.

5.5 Scheletul robotizat

5.5.1 Tehnologiile alese

Arduino

Arduino este o platformă open-source folosită pentru construirea de proiecte electronice. Ea constă dintr-un microcontroler fizic programabil (adesea sub forma unei plăci de dezvoltare) și un mediu de dezvoltare integrat (IDE) care rulează pe calculatorul propriu și este folosit pentru a scrie și încărca cod pe placa Arduino. Prin intermediul său, atât amatorii cât și profesioniștii pot explora, experimenta și crea soluții tehnologice adaptate nevoilor contemporane.

De ce este util Arduino?

Accesibilitate și Simplitate	Interfața și limbajul de programare (bazat pe C/C++) sunt concepute pentru a fi accesibile atât începătorilor, cât și celor cu experiență. Plăcile Arduino sunt relativ ieftine, făcând tehnologia accesibilă pentru o gamă largă de utilizatori.
Flexibilitate și versatilitate	Există o varietate de plăci Arduino, fiecare cu specificații diferite pentru diverse aplicații, de la simpla automatizare a unor procese până la proiecte complexe de robotică. Arduino poate fi conectat la o gamă largă de module și senzori, permițând crearea unor proiecte diverse și inovative.
Resurse abundente și comunitate activă	Există o cantitate vastă de resurse online, inclusiv ghiduri, tutoriale video și cărți care pot ajuta utilizatorii să învețe și să dezvolte proiecte. O comunitate globală activă oferă suport, împărtășind proiecte, soluții și idei pe forumuri și rețele sociale.
Aplicații educative	Arduino este folosit pe scară largă în educație pentru a preda concepte de programare, electronică și inginerie. Este o unealtă excelentă pentru învățare practică. Permite elevilor și studenților să creeze proiecte de la zero, încurajând inovarea și gândirea critică.
Prototipare rapidă	Datorită simplității și flexibilității sale, Arduino este ideal pentru prototiparea rapidă a ideilor și testarea conceptelor înainte de a trece la dezvoltarea produselor finale.
Integrare cu alte tehnologii	Arduino poate fi integrat în proiecte IoT pentru a colecta și transmite date, a controla dispozitive de la distanță și a automatiza procese. Arduino poate comunica cu diverse aplicații software, oferind posibilitatea de a crea soluții integrate hardware-software.

5.5.2 Componente

Pentru a aduce la viață scheletul robotizat, este esențial să alegem și să utilizăm o gamă variată de componente electronice și mecanice. Fiecare componentă joacă un rol crucial în funcționarea și performanța generală a robotului, contribuind la realizarea mișcărilor precise, la interacțiunea cu serverul și la gestionarea datelor.

Șasiu

Figura 32 Componentele șasiului

2x Plăci acrilic	Acestea reprezintă baza robotului, pe care vor fi montate toate celelalte componente.
4x Cauciucuri	Acestea permit deplasarea robotului pe diferite suprafețe.
4x Motoare 3-6V cu reductor	Utilizate pentru a pune în mișcare roțile robotului. Motoarele sunt echipate cu reductoare pentru a oferi un cuplu mai mare și un control mai bun al vitezei.
1x Suport baterii 4AA	Acesta, impreuna cu 4 baterii AA vor alimenta cele 4 motoare, permițând executarea corectă a diverselor mișcări ale roților.
Componente de conectivitate și fixare	Necesare pentru a fixa motoarele pe plăcile acrilice, pentru a suprapune plăcile și pentru a conecta componentele.

Punte h dubla L298N V1

Figura 33 Driver

Driverul L298 este o punte H dublă de mare curent gândită să accepte niveluri logice TTL standard și să acționeze sarcini inductive precum relee, electromagneți, motoare DC și motoare pas cu pas. Este compatibilă cu Arduino și alte placi de dezvoltare. Microcontrolerul poate comunica prin pinii dedicați al acestui modul prin I/O si prin PWM, permițând controlul motorului în ceea ce priveste direcția și viteza. Dispune de pini Enable pentru a activa sau dezactiva terminalul corespondent.

Acesta realizează conexiunea dintre sursa de alimentare, motoare și esp32. Primește comenzile de la creier (esp32) și le transmite motoarelor pentru îndeplinrea diverselor sarcini definite.

ESP32

Figura 34 ESP32

ESP32 este un microcontroler puternic și versatil dezvoltat de Espressif Systems. Este o soluție completă de sistem pe cip (SoC) care integrează un microprocesor dual-core sau single-core, memorie RAM, stocare flash și diverse periferice.

Conectivitatea Wi-Fi este una dintre cele mai importante caracteristici ale ESP32, permițându-i să se conecteze la rețele wireless și să comunice cu alte dispozitive prin intermediul internetului. Această capacitate este esențială pentru multe aplicații IoT (Internet of Things), unde ESP32 poate trimite și primi date de la senzori, poate controla dispozitive la distanță și poate interacționa cu servicii cloud.

Aceasta a fost caracteristica principală pentru care am decis sa folosim un esp32 in cadrul proiectului nostru, să poata comunica prin Wi-Fi cu serverul informațiile procesate prin cadrul aplicației mobile.

Alimentare ESP32

Figura 35 Baterie externă

Folosim o baterie externă pentru alimentarea Esp32. Alimentarea dintr-o sursă separată față de motoare ne asigură că nu vom avea probleme în circuit.

Ansamblu

Informațiile primite de la server sunt procesate in cadrul ESP32 Acesta trimite catre driver comenzile obținute pe baza informatiilor procesate. Pe baza comenzilor, driverul trimite către motoare semnale pentru punerea in funcțiune a acestora. Prin aceste semnale, scheletul robotizat reușește sa pună în mișcare roțile și să se deplaseze în spațiu.

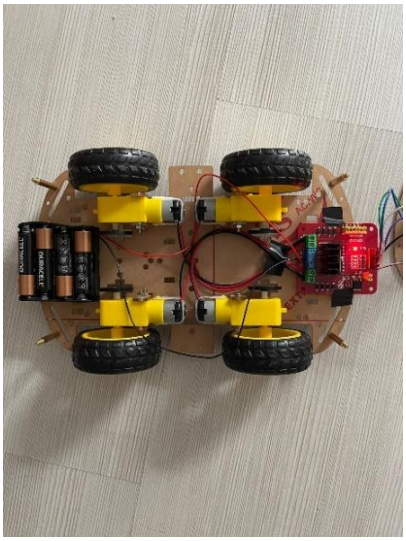

Figura 36 Secțiunea 1 a ansamblului

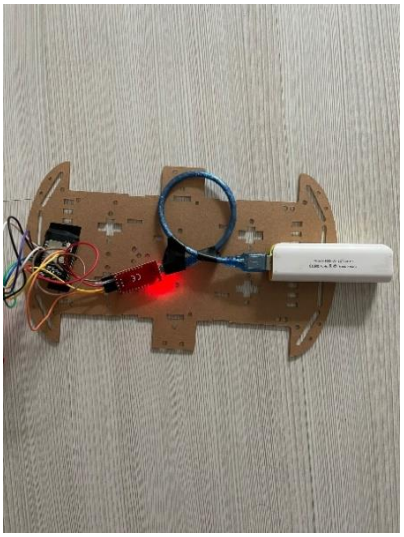

Figura 37 Secţiunea 2 a ansamblului

Figura 38 Ansamblul format din suprapunerea secţiunilor 1 şi 2

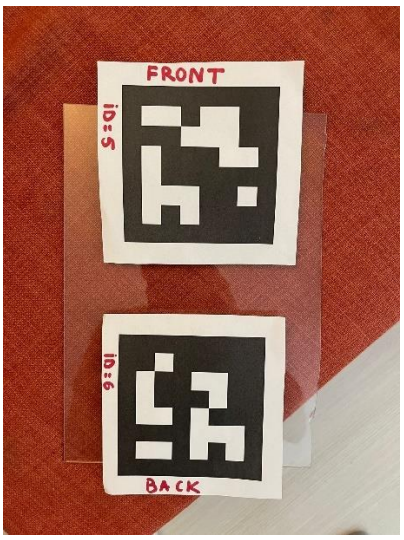

Figura 39 Secţiunea 3 cu markerii folosiţi pentru orientare

Figura 40 Ansamblul format din suprapunerea celor 3 secţiuni

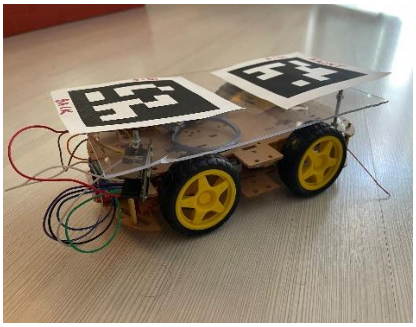

Figura 41 Perspectiva laterală a scheletului robotizat

Rolul secțiunii 3 cu cei doi markeri este foarte importantă, aceasta ne ajută să definim care este fața și care este spatele ansamblului robotizat. Folosim un marker cu id-ul 5 pentru fața mașinii și un marker cu id-ul 6 pentru spatele mașinii. Cu ajutorul acestora putem orienta în mod corect ansamblul pentru a se deplasa în spațiu și pentru a ajunge la destinția noastră. Vom vedea în secțiunea următoare cum ne ajută acestea în cadrul calculelor necesare orientării.

5.5.3 Implementare cod ESP32

Importarea bibliotecilor și includerea definițiilor necesare

Codul pentru ESP32 utilizează mai multe biblioteci esențiale pentru conectivitatea Wi-Fi, procesarea datelor și comunicarea prin HTTP.

WiFi.h	Utilizată pentru a gestiona conectivitatea Wi-Fi a dispozitivului ESP32. Această bibliotecă permite conectarea la rețele Wi-Fi, gestionarea conexiunilor și transmiterea datelor prin rețea.

Arduino.h	Biblioteca de bază pentru programele Arduino, inclusiv pentru ESP32. Aceasta include funcții esențiale pentru manipularea pinilor, temporizatorilor și alte funcții de bază ale hardware-ului.
HTTPClient.h	Folosit pentru a efectua cereri HTTP de la ESP32 către server sau alte dispozitive în rețea. Este crucială pentru proiectele IoT care necesită comunicare între dispozitiv și un backend sau servicii cloud.
ArduinoJson.h	Utilizată pentru manipularea datelor JSON, care sunt adesea folosite în comunicarea dintre dispozitivele IoT și server. Permite serializarea și deserializarea datelor JSON, facilitând astfel schimbul de date structurate.
Variabila FLT_MAX	#define FLT_MAX 3.4028235E+38 este o definiție preprocesor care stabilește valoarea maximă pentru tipul de date float în cadrul programului. Aceasta asigură o constantă de referință pentru verificarea sau compararea valorilor de tip float.

Această configurare inițială în documentație asigură că toate bibliotecile și definițiile necesare sunt corect importate și configurate pentru funcționalitatea ESP32.

Definirea informațiilor de conectare și a pinilor

Necesare pentru conectarea la rețeaua Wi-Fi prin care se realizează comunicarea cu serverul și configurarea pinilor, necesari în controlul motoarelor. Aceasta este esențială pentru stabilirea comunicației și interacțiunea cu hardware-ul.

Detalii de implementare:

Conectare Wi-Fi	const char* ssid = "xxxxx"; - SSID-ul rețelei Wi-Fi, folosit pentru a identifica și conecta ESP32 la rețeaua specifică. const char* password = "xxxxx"; - Parola pentru autentificarea în rețeaua Wi-Fi menționată. const char* serverUrl = "http://192.168.100.47:5000/get_aruco_status"; - URL-ul serverului la care ESP32 va face cereri HTTP pentru a obține statusul markerilor ArUco, indicând interacțiunea cu serverul nostru.
Configurarea pinilor	const int motor1Pin1 = 16; și const int motor1Pin2 = 14; - Pinii pentru primul motor, controlând direcția de rotație. const int enable1Pin = 4; - Pinul de activare pentru driverul motorului 1, folosit pentru a controla puterea sau pentru a opri motorul. const int motor2Pin1 = 15; și const int motor2Pin2 = 13; - Pinii pentru al doilea motor, similar cu primul motor. const int enable2Pin = 12; - Similar cu enable1Pin, pentru al doilea motor.

Variabile pentru control și feedback	int speed = 255; - Viteza maximă pentru motoare, folosită pentru setarea PWM-ului (modulare a lățimii pulsului). float d_front_back_car = 0; - variabilă pentru distanța față-spate a mașinii, utilizată pentru calculul poziționării mașinii. float distance_from_min_to_back_car = 0; - variabilă pentru distanța de la spatele mașinii la cel mai apropiat marker, utilizată pentru orientare. float distance = 0; și float error = 0; - Variabile pentru distanță și eroare, utilizate pentru orientarea corespunzătoare a mașinii.

Această secțiune din documentație asigură că toate informațiile esențiale și configurațiile sunt stabilite corect pentru dezvoltarea funcțională și integrarea componentelor software și hardware în proiectul cu ESP32.

Funcția setup() pentru configurarea inițială

Funcția setup() este esențială pentru inițializarea corectă a dispozitivului, pregătind toate componentele hardware pentru funcționare.

Detalii de implementare:

Comunicație Serială	Serial.begin(115200); - Inițializează comunicația serială la 115200 baud, esențială pentru monitorizarea comportamentului dispozitivului.
Configurarea Pinilor	Pinii motorului (motor1Pin1, motor1Pin2, motor2Pin1, motor2Pin2) și pinii de activare (enable1Pin, enable2Pin) sunt configurați ca OUTPUT.
Inițializarea Pinilor	Toți pinii sunt setați în starea LOW pentru a asigura că motoarele rămân oprite la începutul programului, prevenind mișcările neintenționate.

Aceste setări garantează că sistemul este configurat corespunzător pentru a începe execuția programului cu toate componentele în stare inițială controlată.

Verificarea conexiunii la Wi-Fi

Codul gestionează conectarea la rețeaua Wi-Fi și oferă feedback vizual despre starea conexiunii prin intermediul portului serial.

Detalii de implementare:

Inițierea Conexiunii	WiFi.begin(ssid, password); - Porneste conexiunea la rețeaua Wi-Fi folosind SSID-ul și parola specificate.
Monitorizarea Progresului	Un ciclu while verifică periodic (la fiecare 500 ms, delay(500);) starea conexiunii și afișează un punct pe serial pentru fiecare iterație, semnalând progresul.
Confirmarea Conexiunii	La conectarea cu succes, se afișează "WiFi connected" și "IP address: ", urmat de afișarea adresei IP locale cu WiFi.localIP();.

Această secțiune este vitală pentru asigurarea unei conexiuni reușite la Wi-Fi și pentru comunicarea eficientă cu serverul, fiind esențială pentru aplicații ce necesită interacțiunea cu rețelele. Implementarea oferă un mecanism clar și eficient de raportare și depanare a stării conexiunii.

Execuția cererilor HTTP în funcția loop()

Această secțiune a codului facilitează efectuarea cererilor HTTP periodice, esențiale pentru menținerea unei interacțiuni constante cu serverul cât timp dispozitivul este conectat la Wi-Fi.

Detalii de implementare:

Verificarea Conexiunii Wi-Fi	Se verifică dacă dispozitivul este conectat la Wi-Fi înainte de a începe trimiterea cererilor.
Configurarea Clientului HTTP	HTTPClient http; - Creează un obiect de tip HTTPClient pentru gestionarea cererilor. http.begin(serverUrl); - Inițiază o conexiune HTTP către URL-ul specificat.
Trimiterea și Gestionarea Cererilor	Se trimite o cerere de tip GET la server și se stochează codul de răspuns HTTP primit, evaluând succesul conexiunii și disponibilitatea serverului.

Această implementare demonstrează eficiența ESP32 în aplicații care necesită conectivitate și comunicare de date continue, asigurând că dispozitivul poate reacționa și interacționa cu datele de pe server în timp real. Astfel, codul permite o integrare robustă în sisteme care depind de actualizări frecvente de la server pentru funcționarea optimă.

Procesarea răspunsului HTTP și extracția datelor

Vom trata răspunsul primit la o cerere HTTP pentru a utiliza informațiile. Codul extrage și procesează datele JSON din răspuns, afișându-le și utilizându-le pentru calcule ulterioare.

Detalii de implentare:

Verificăm dacă cererea HTTP a fost succesful (cod pozitiv).

String response = http.getString(); - Extrage corpul răspunsului HTTP ca un șir de caractere.

Afișează răspunsul în consola serială pentru depanare.

DynamicJsonDocument doc(1024); - Creează un document JSON dinamic pentru a stoca și parsa datele JSON.

deserializeJson(doc, response); - Parsează șirul JSON în documentul creat.

JsonArray distances = doc["distances"]; - Extrage un array JSON numit "distances" din document.

Prin extracția și utilizarea datelor din răspunsurile JSON, dispozitivul poate efectua decizii bazate pe date în timp real. În acest mod putem analiza orice schimbare apare în cadrul fluxului de funționare și interveni rapid pentru depanare.

Procesarea și extragerea distanței de la spatele mașinii la fața mașinii

Este necesar să extragem și să convertim distanțele dintre markerii ArUco specifici, folosind datele preluate din răspunsul HTTP anterior procesat. Dorim să aflăm distanța de la fața mașinii la spatele mașinii, distanță ce ne va ajuta ulterior să calibrăm direcția mașinii.

Detalii de implentare:

Ciclul for parcurge un array distances care conține informații despre distanțe sub formă de string-uri.

Condițiile if verifică dacă string-ul curent conține referințe la markerii atribuiți pentru fața și spatele mașinii, respectiv pentru spatele și fața mașinii (este important sa verificăm corelația dintre markeri ca fiind din ambele orientări), asigurând că datele procesate sunt relevante pentru markerii specifici.

Este important să verificăm diverse combinații (text) dintre markeri deoarece ordinea în care ne sunt trimise markerele de la server nu este mereu la fel. Acest lucru se poate schimba din cauza distanței relative față de punctul de referință (locul în care este amplasată camera).

indexOf și substring sunt folosite pentru a localiza și extrage segmentul numeric al distanței dintre markeri.

toFloat convertește string-ul numeric extras într-o valoare float, care este apoi atribuită unei variabile destinată stocării distanței (d_front_back_car).

Prin utilizarea acestei secvențe, putem extrage informații despre distanța pe care dorim sa o aflăm din mesajele transmise de la server.

Găsirea distanțelor dintre fața mașinii și restul markerilor

Dorim să aflăm restul markerilor și distanțele acestora în raport cu fața mașinii.

Detalii de implementare:

Folosim un ciclu if pentru a verifica prezența anumitor șiruri de caractere care indică relațiile dintre markerul atribuit feței mașinii și alții și exclude cazurile care includ explicit atribuirea de la spatele la fața mașinii.

Utilizăm funcțiile indexOf și substring pentru a localiza și extrage segmentul numeric al distanței dintr-un șir mai mare.

Conversia acestui segment de șir într-o valoare numerică de tip float se realizează folosind metoda toFloat, care permite ulterior utilizarea acestei valori în calcule.

Această secțiune ne asigură că mașina găsește toți markerii din jur. Precum calculul distanței de la fața mașinii la spatele mașinii, este necesar sa verificăm diversele combinații dintre mesaje.

Identificarea markerului cel mai apropiat

Deoarece taskul mașinii este să ajungă la cea mai apropiată țintă, este necesar să identificăm cel mai apropiat marker dintre cei căutați anterior.

Detalii de implementare:

Codul folosește două blocuri if (mesajele de la server pot fi diferite în get-uri diferite) pentru a verifica prezența anumitor markeri specificați în string-uri de distanțe.

În interiorul fiecărui bloc if, codul compară distanța curentă cu cea mai mică distanță găsită anterior (minDistance).

Dacă o distanță mai mică este găsită, aceasta este actualizată ca fiind noua distanță minimă, și se extrage numele markerului asociat cu acea distanță.

Extracția numelui markerului implică identificarea pozițiilor de început și de sfârșit în string-ul de distanțe și folosirea metodei substring pentru a captura numele markerului.

Explorăm markerii din jur pentru a-l găsi pe cel mai apropiat de fața mașinii. Acest detaliu ne asigură că ajungem în mod corect la destinația stabilită. Vom denumi markerul cel mai apropiat de mașină ca fiind minMarker.

Identificarea distanței de la cel mai apropiat marker la spatele mașinii

Vom folosi această disanță ulterior pentru orientarea corectă a mașinii.

Detalii de implementare:

Ciclul for iterează prin fiecare element al array-ului distances, care conține distanțe sub formă de string-uri.

Condițiile if verifică prezența unei perechi specifice de markeri în fiecare string de distanțe, inclusiv combinații ale minMarker cu markerul reprezentativ pentru spatele mașinii.

Funcțiile indexOf și substring sunt utilizate pentru a localiza și extrage segmentul numeric al distanței din string.

toFloat() convertește string-ul extras într-o valoare de tip float, care este apoi atribuită unei variabile destinată stocării distanței.

Pentru a realiza calculul orientării cu precizie, este necesar să avem in vedere toate detaliile legate de mașina noastră și sistemul de referință (minMarker).

Calculul erorii pentru verificarea orientării vehiculului

Vom calcula eroarea de orientare a vehiculului în raport cu un marker ArUco. Aceasta este folosită pentru a determina dacă vehiculul este aliniat corect față de marker, o componentă esențială pentru sistemul nostru de navigație.

Detalii de implementare:

Variabila error calculează diferența dintre suma distanței minime până la markerul cel mai apropiat și distanța dintre partea frontală și cea din spate a mașinii (d_front_back_car), minus distanța de la markerul cel mai apropiat până la partea din spate a mașinii (distance_from_min_to_back_car).

Eroarea calculată este esențială pentru ajustări în controlul direcției sau pentru feedback în sistemele automate.

Acest calcul permite evaluarea alinierii mașinii față de markerul cel mai apropiat, oferind o metrică a poziționării relative a mașinii față de marker. Astfel, error indică dacă mașina este orientată corespunzător pentru a se deplasa direct către markerul targetat, ajutând la corectarea traseului dacă este necesar.

Condiția de verificare a orientării corecte a mașinii

Vom determina dacă mașina este suficient de bine aliniată cu markerul cel mai apropiat. Dacă error este într-un interval, mașina este considerată corect orientată față de marker, ceea ce este esențial pentru navigația precisă.

Detalii de implementare:

Eroarea este calculată ca diferența dintre distanța teoretică ideală și distanța reală măsurată între mașină și marker.

Codul evaluează dacă această eroare se încadrează între 0 și 4 pixeli.

Condiția if verifică dacă eroarea este în limita specificată, caz în care se consideră că mașina este aliniată corect.

Acest mecanism de verificare asigură că mașina rămâne pe traiectoria optimă, reducând riscul de deviere datorată alinierii incorecte.

Verificarea distanței până la destinație

Această verificare este destinată să observe dacă distanța dintre mașină și markerul cel mai apropiat este mai mare decât un prag specific, în acest caz, 75 de pixeli. Această verificare este importantă pentru a asigura că mașina nu este prea aproape de marker.

Detalii de implementare:

Condiția if (minDistance > 75) evaluează dacă distanța calculată până la markerul cel mai apropiat depășește 75 de pixeli.

Dacă distanța este mai mare de 75 de pixeli, mașina este considerată a fi într-o poziție sigură pentru a continua operațiunile fără riscul de coliziune sau alte probleme legate de proximitatea excesivă.

Această verificare este crucială pentru a ne asigura că mașina noastră nu produce o coliziune cu destinația noastră.

Mișcarea înainte a mașinii

Dacă condițiile anterioare (orientarea în funcție de eroare și distanța până la destinație) nu sunt atinse, mașina este poziționată corect față de marker și este la o distanță mai mare față de limita impusă. În aceste condiții ne putem deplasa înainte până ajungem la destinație.

Detalii de implementare:

Codul utilizează funcția digitalWrite pentru a seta starea pinilor fiecărui motor, asigurând direcția corectă de rotație.

Pentru motorul 1, pinul 1 este setat pe LOW și pinul 2 pe HIGH.

Similar, pentru motorul 2, pinul 1 este setat pe LOW și pinul 2 pe HIGH.

Funcția analogWrite este folosită pentru a controla viteza motoarelor prin pinii enable1Pin și enable2Pin, unde viteza este specificată de variabila speed.

O pauză scurtă de n milisecunde este introdusă cu delay(n) pentru a permite motorului să execute comanda de mișcare.

După ce mașina se deplasează o perioadă scurtă înainte dorim să oprim motoarele pentru a face noi calcule:

Codul folosește funcția digitalWrite pentru a seta pinii motorului 1 și motorului 2 pe LOW, ceea ce oprește alimentarea la motoare.

Funcția analogWrite este utilizată pentru a seta pinii de enable ai ambelor motoare pe 0, ceea ce reduce tensiunea aplicată la motoare la zero, asigurând că acestea sunt complet oprite.

O întârziere de n milisecunde este adăugată prin delay(n) pentru a permite sistemului o pauză suficientă între oprirea și eventuala repornire sau pentru a procesa alte comenzi.

Această secțiune a codului asigură controlul precis al mișcării înainte a mașinii, utilizând funcții de setare a direcției și vitezei motoarelor, urmat de o oprire controlată pentru recalcularea poziției. Aceste mecanisme sunt vitale pentru menținerea unui traseu corect și pentru adaptarea continuă la condițiile de navigație.

Oprirea motoarelor la destinație

După scurta deplasare înainte, verificăm iar condiția pentru distanța față de destinație. Dacă aceasta este îndeplinită, putem opri mașina de tot.

Detalii de implementare:

Rezultatul începe cu afișarea unui mesaj în consola serială: "Mașina a ajuns la destinație.", indicând utilizatorului sau sistemului de monitorizare că vehiculul a finalizat cursa planificată.

Următoarele linii de cod (digitalWrite și analogWrite) setează pinii motorului pe LOW și reduc tensiunea la zero (prin setarea pinilor enable pe 0). Acest lucru oprește motoarele efectiv, prevenind orice mișcare ulterioară.

Codul include un delay de n milisecunde, oferind un timp de pauză suficient pentru a asigura că toate procesele se opresc complet înainte de a continua cu orice alte acțiuni automate sau manuale.

Prin asigurarea opririi complete la destinație, sistemul asigură că mașina și-a îndeplinit sarcina de a ajunge la destinație.

Calibrarea mașinii

În situația în care mașina nu se află în poziție dreaptă față de markerul considerat destinație, este necesar să mișcăm mașina până condiția de aliniere (verificarea erorii) este respectată.

Detalii de implementare:

Codul configurează motorul 1 pentru a se mișca într-o direcție (pin2 HIGH, pin1 LOW) și motorul 2 în direcția opusă (pin2 LOW, pin1 HIGH) pentru a roti vehiculul pe loc.

Viteza fiecărui motor este controlată prin analogWrite la pinii enable, cu o valoare specificată de variabila speed.

Această rotație este menținută pentru n milisecunde, după care ambele motoare sunt oprite pentru n milisecunde pentru a evalua noua orientare a vehiculului.

Cu ajutorul acestui pas, putem roti mașina într-o anumită direcție și verifica dacă condiția de aliniere este respectată. Acest detaliu ne asigură că mașina ajunge în mod corect la destinație.

5.5.4 Detalii tehnice

În programul Arduino, întârzierea de n secunde setată la sfârșitul funcției loop() servește la controlul frecvenței cu care se execută bucla, influențând în special frecvența cererilor HTTP GET către server. Acest interval de timp ajută la prevenirea suprasolicitării rețelei și serverului, gestionând mai eficient utilizarea datelor și evitând congestia în rețea. De asemenea, contribuie la o mai bună gestionare a energiei, fiind important în cazul dispozitivelor alimentate de baterii.

Utilizarea distanței în pixeli este benefică pentru simplificarea complexității calculelor, deoarece elimină necesitatea conversiilor complexe în unități fizice. Acest lucru permite o procesare mai rapidă și mai eficientă a datelor, fiind ideal în contextul sistemului nostru unde timpul de răspuns este crucial.

Delay-urile din codul Arduino controlează temporizarea acțiunilor motorului pentru a asigura o navigație precisă.

5.5.5 Perspectiva dezvoltatorului

În secțiunea aceasta vom exemplifica situațiile de comunicație dintre server și ESP32. Vom lucra cu ESP32 conectat prin configurator la laptop pentru a vedea informațiile din transmisia serializată.

Situații în care se poate găsi ESP32 după ce acesta este pornit, cu codul încarcat:

1. Acesta se conectează cu succes la rețeaua wi-fi dar serverul nu este pornit. Cererile get lansate către server vor întoarce un mesaj „Error on HTTP request" în acest caz.

```
10:45:30.711 -> mode:DIO, clock div:1
10:45:30.711 -> load:0x3fff0030,len:1344
10:45:30.711 -> load:0x40078000,len:13964
10:45:30.711 -> load:0x40080400,len:3600
10:45:30.711 -> entry 0x400805f0
10:45:30.990 -> E (215) psram: PSRAM ID read error: 0xffffffff
10:45:31.597 -> ......WiFi connected
10:45:34.114 -> IP address:
10:45:34.114 -> 192.168.100.133
10:45:39.131 -> Error on HTTP request
10:45:46.103 -> Error on HTTP request
```

Figura 42 Informaţii obţinute din serializare pentru cazul 1

2. Acesta se conectează cu succes la rețeaua wi-fi și la server dar aplicația mobilă nu a fost pornită și nu avem informații după care sa ne orientăm.

```
10:56:37.149 -> mode:DIO, clock div:1
10:56:37.149 -> load:0x3fff0030,len:1344
10:56:37.149 -> load:0x40078000,len:13964
10:56:37.149 -> load:0x40080400,len:3600
10:56:37.149 -> entry 0x400805f0
10:56:37.427 -> E (215) psram: PSRAM ID read error: 0xffffffff
10:56:38.080 -> .....WiFi connected
10:56:40.078 -> IP address:
10:56:40.078 -> 192.168.100.133
10:56:40.685 -> {
10:56:40.685 ->   "message": "Nu exista informatii recente despre markere."
10:56:40.685 -> }
10:56:40.685 ->
10:56:40.685 -> Distanţa dintre fata masinii şi spatele masinii este: 0.00
10:56:40.685 -> Cel mai apropiat marker de masina este , aflat la distanta: ovf
10:56:40.685 -> Distanţa de la markerul  la spatele masinii este: 0.00
10:56:40.685 -> Verificam daca masina este orientata corect catre markerul
10:56:40.685 -> Masina nu este in linie dreapta catre markerul
10:56:40.685 -> Incepem calibrarea directiei masinii catre markerul
```

Figura 43 Informaţii obţinute din serializare pentru cazul 2

3. Acesta se conectează cu succes la wi-fi și la server iar aplicația mobilă este inclusă în proces. Informațiile sunt procesate corespunzător și ajung corect la ESP32. Pe baza acestora vom urma acțiunile definite în cod.

Dacă conexiunea dintre ESP32 si server s-a realizat cu succes, cererile get se realizeaza corect la un interval de n secunde pentru a updata informația.

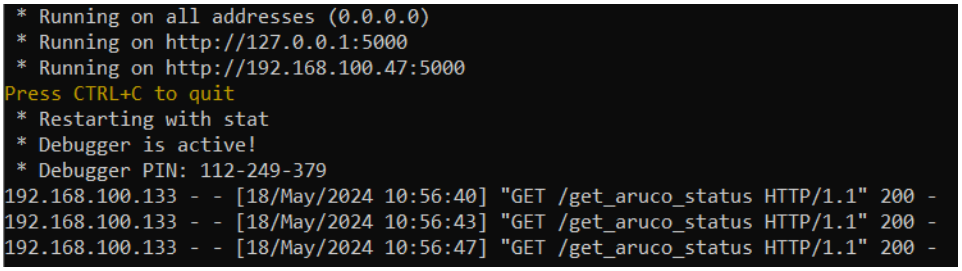

Figura 44 Informații obținute din logul serverului despre cererile get

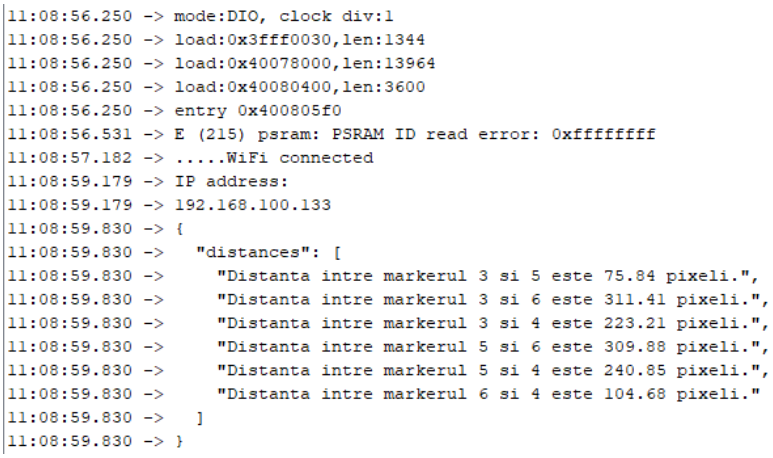

Figura 45 Informații obținute din serializare pentru cazul 3

Figura 46 Informații obținute din logul serverului despre markerii din imagine

Serverul procesează în mod corect informațiile de la aplicația mobilă și le trimite cu ajutorul get-urilor către ESP32.

6 STUDIU DE CAZ / EVALUAREA REZULTATELOR

6.1 Rezultate obținute

În urma implementării pașilor anteriori, ansamblul poate realiza următoarea sarcină: poate controla scheletul robotizat pentru a se alinia și ulterior deplasa înainte spre cel mai apropiat marker ArUco, dintre cei prezenți în plan.

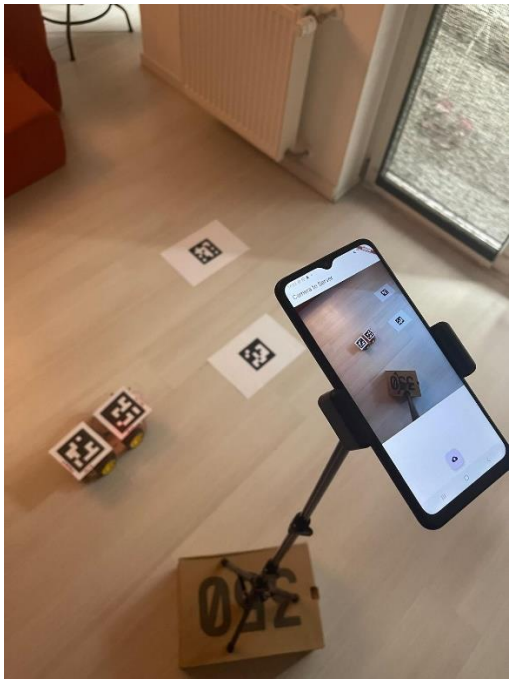

Figura 47 Perspectiva de ansamblu

Figura 48 Perspectiva orientării

După cum se poate observa în figura 47 și figura 48 scheletul robotizat se află orientat invers către cei doi markeri (id 3 și id 4). Aplicația mobilă este pornită și putem observa un preview al camerei dar butonul pentru încărcarea cadrelor nu a fost activat.

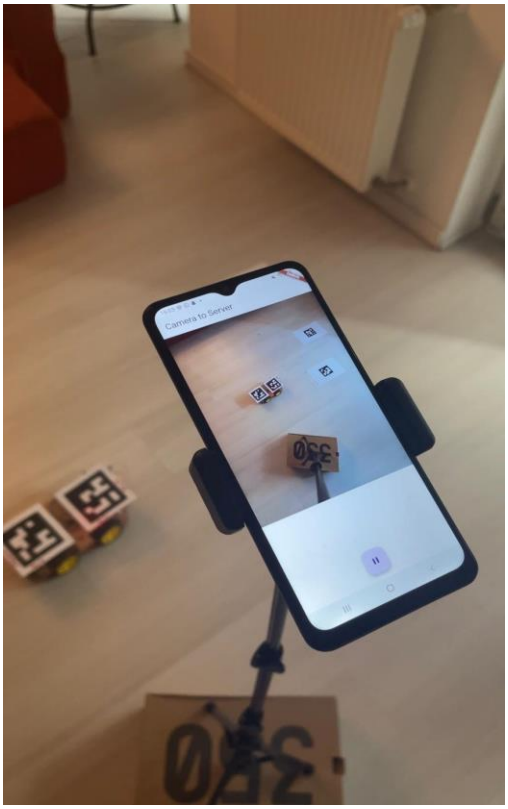

Figura 49 Perspectiva de ansamblu cu activarea trimiterii de cadre

Am acționat manual și am schimbat starea pentru trimiterea cadrelor către server pentru a incepe analiza informațiilor din mediu și pentru a trimite sarcinile către schelet.

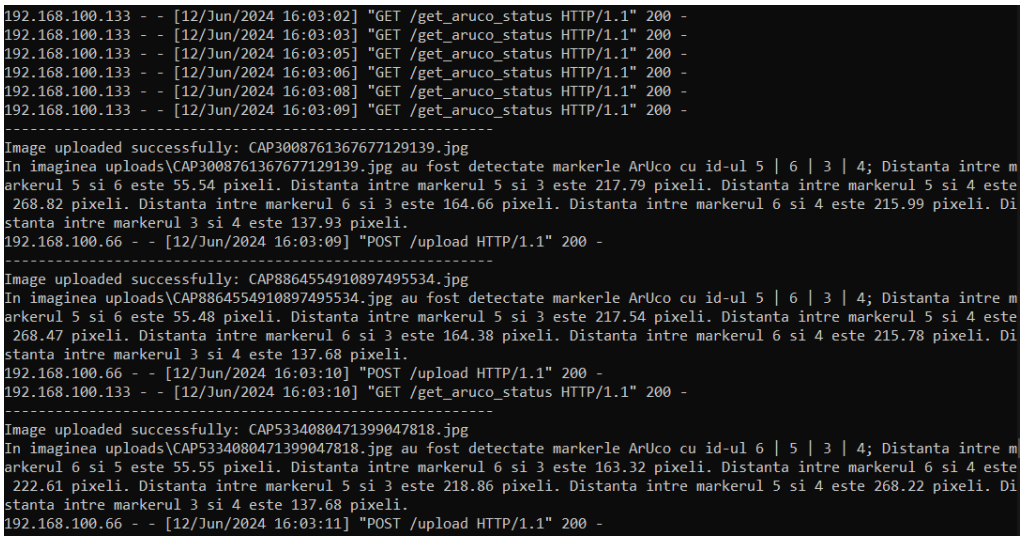

Figura 50 Log al serverului înainte și după pornirea trimiterii de cadre

Inițial, scheletul este conectat la server și putem vedea că acesta trimite cereri de get în mod repetat și se realizează corect. Aceste cereri de get nu obțin informații până nu este pusă in schemă și aplicația mobilă.

După ce comutăm starea butonului pentru trimiterea de cadre la un interval stabilit de timp putem observa informațiile importante din planul nostru:

1. Distanța între markerul 5, asociat feței mașinii, și markerul 6, asociat spatelui mașinii, este egală cu 55.48 pixeli.

2. Distanțele de la markerul 5, asociat feței mașinii, la ceilalți markeri din plan sunt: 217.79 pixeli până la markerul 3 și 268.82 pixeli până la markerul 4. După cum am explicat în secțiunea cu Implementarea cod ESP32, vom extrage markerul 3 ca fiind cel mai apropiat de mașină (având distanța cea mai mică).

3. Ne interesează să aflăm și distanța de la markerul 6, asociat spatelui mașinii, până la markerul cel mai apropiat, respectiv 3 în cazul acesta pentru a putea face calculele de calibrare a direcției. Disanța aceasta este 164.66 pixeli.

Folosind aceste informații dorim sa ghidăm mașina spre markerul cel mai apropiat dar observăm că aceasta nu se află orientată corect către acesta.

Eroarea pentru considerarea direcției corecte trebuie sa fie în intervalul [0, 4].

Eroarea = (distanța de la fața mașinii până la cel mai apropiat marker + distanța între fața mașinii și spatele mașinii) – distanța de la spatele mașinii până la cel mai apropiat marker

Eroarea = (217.79 + 55.48) – 164.66 = 108.61 pixeli

Rolul acestor calcule sunt explicate în secțiunea cu Implementarea codului ESP32.

Figura 51 Reorientare 1

Figura 52 Reorientare 2

Figura 53 Reorientare 3

Putem observa că mașina execută periodic rotiri spre dreapta și verifică dacă aceasta a fost orientată corect. În ultima imagine mașina este orientată corect către marker și începe deplasarea înainte până se împlinește condiția de oprire (< 75 pixeli).

Figura 54 Deplasarea spre destinție

Mașina a ajuns la destinație - a ajuns la o distanță mai mică de 75 pixeli.

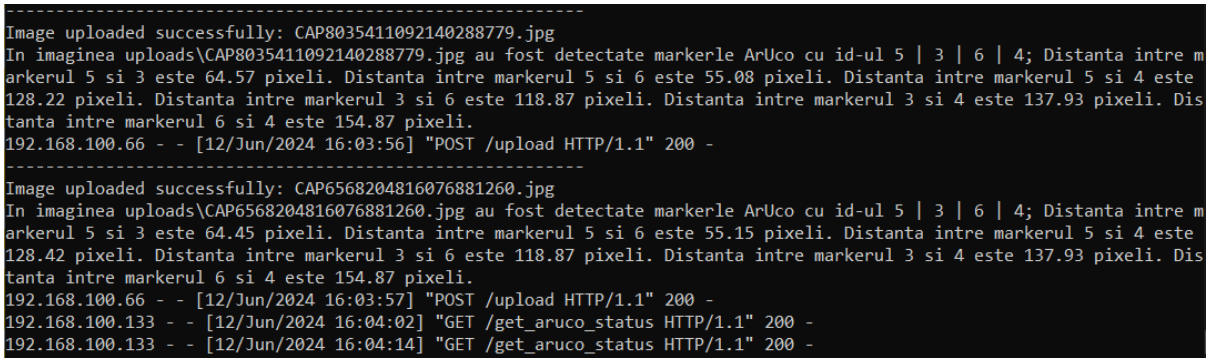

Figura 55 Log cu informații după ce mașina a ajuns la destinație

După cum putem observa distanța de la markerul 5, asociat feței mașinii, la markerul 3, asociat celui mai apropiat marker, este 64.57 pixeli în punctul final, în care se împlinește condiția discutată anterior.

În acest punct mașina v-a staționa deoarece a ajuns la destinație.

6.2 Criterii de performanță

Este foarte important de menționat că performanța ansamblului ține de mai multe criterii care pot fi schimbate în funcție de context.

Trei criterii esențiale sunt:

1. Intervalul de timp pentru încărcarea cadrelor.

2. Intervalul de timp între cererile de get făcute de ESP32.

3. Perspectiva camerei

Dacă dorim să studiem un mediu mai complex, ce necesită multă atenție și calcule mai multe din pas în pas – vom seta primele două criterii cât mai mici pentru a trimite cadre cât mai multe și a face cereri pentru informații pe acestea cât mai des. Putem analiza fiecare mică mișcare a mașinii sau orice variabilă apărută care necesită un răspuns.

Dacă dorim să studiem un mediu mai puțin complex, în care trebuie să se facă puține schimbări – vom seta primele două criterii mai mari pentu a trimite cadre cât mai puține și a face cereri pentru informații pe acestea cât mai rar. Mediul necesită mișcări puține si nu prezintă multe schimbări care necesită un răspuns.

Perspectiva camerei ne indică cadrul pe care il putem studia:

Dacă telefonul mobil se poate mișca, mediul poate fi analizat continuu fără necesitatea unui cadru mare.

Dacă telefonul mobil trebuie să stea pe loc, capacitatea analizei detaliilor si a numărului de detalii din mediu depinde de unghiul și poziția camerei.

În cazul nostru, mediul nu prezintă schimbări și nici nu necesită foarte multe mișcări amănunțite. Am decis ca intervalul între cadre să fie de o secundă și intervalul între cererile de get pentru informații să fie de o secundă. Am așezat telefonul pe un trepied pentru a avea un unghi bun dar nu foarte mare. Aceste criterii au fost alese pentru a avea un raport bun între viteza de execuție, consumul de memorie și capacitatea de a putea verifica eventualele erori în același timp cu rularea programelor.

Ansamblul a fost implementat pentru a funcționa în mod corect și pentru sarcini care necesită o analiză mai amănunțită și mai complexă.

Cu ajutorul acestor criterii, am putut analiza în mod corect un mediu real și am putut realiza sarcini corect.

6.3 Evaluarea rezultatelor

6.3.1 Identificarea corectă a markerilor

După cum se poate observa din log-uri, execuția programelor și validarea unor teste separate folosind un număr mai mare de markeri în plan (2, 3, 4, 5), putem deprinde că aplicația găsește mereu corect:

1. Distanțele folosite pentru orientarea scheletului.

2. Markerul aflat la cea mai mică distanță de fața scheletului.

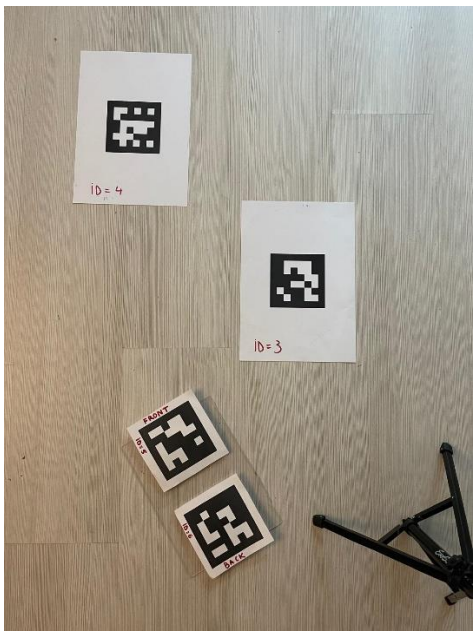

Figura 56 Caz 1 - minim dintre 2 markeri

```
{
  "distances": [
    "Distanta intre markerul 6 si 5 este 71.04 pixeli.",
    "Distanta intre markerul 6 si 3 este 183.93 pixeli.",
    "Distanta intre markerul 6 si 4 este 236.53 pixeli.",
    "Distanta intre markerul 5 si 3 este 140.82 pixeli.",
    "Distanta intre markerul 5 si 4 este 168.90 pixeli.",
    "Distanta intre markerul 3 si 4 este 114.61 pixeli."
  ]
}
```


```
Distanța dintre fata masinii și spatele masinii este: 71.04
Cel mai apropiat marker de masina este 3, aflat la distanta: 140.82
Distanța de la markerul 3 la spatele masinii este: 183.93
Verificam daca masina este orientata corect catre markerul 3
Masina nu este in linie dreapta catre markerul 3
Incepem calibrarea directiei masinii catre markerul 3
```

Figura 57 Informații pentru cazul 1

Din 2 markeri, cel mai apropiat este cel cu id-ul 3, aflat la distanța 140.82 pixeli.

Figura 58 Caz 2 - minim dintre 3 markeri


```
{
  "distances": [
    "Distanta intre markerul 6 si 5 este 71.21 pixeli.",
    "Distanta intre markerul 6 si 15 este 206.69 pixeli.",
    "Distanta intre markerul 6 si 3 este 326.15 pixeli.",
    "Distanta intre markerul 6 si 4 este 266.60 pixeli.",
    "Distanta intre markerul 5 si 15 este 158.40 pixeli.",
    "Distanta intre markerul 5 si 3 este 278.54 pixeli.",
    "Distanta intre markerul 5 si 4 este 198.50 pixeli.",
    "Distanta intre markerul 15 si 3 este 120.61 pixeli.",
    "Distanta intre markerul 15 si 4 este 117.70 pixeli.",
    "Distanta intre markerul 3 si 4 este 157.34 pixeli."
  ]
}
```


```
Distanța dintre fata masinii și spatele masinii este: 71.21
Cel mai apropiat marker de masina este 15, aflat la distanta: 158.40
Distanța de la markerul 15 la spatele masinii este: 206.69
Verificam daca masina este orientata corect catre markerul 15
Masina nu este in linie dreapta catre markerul 15
Incepem calibrarea directiei masinii catre markerul 15
```

Figura 59 Informații pentru cazul 2

Din 3 markeri, cel mai apropiat este cel cu id-ul 15, aflat la distanța 158.40 pixeli.

Figura 60 Caz 3 - minim dintre 4 markeri


```
{
  "distances": [
    "Distanta intre markerul 6 si 5 este 71.48 pixeli.",
    "Distanta intre markerul 6 si 42 este 197.46 pixeli.",
    "Distanta intre markerul 6 si 3 este 326.52 pixeli.",
    "Distanta intre markerul 6 si 4 este 263.23 pixeli.",
    "Distanta intre markerul 6 si 15 este 352.85 pixeli.",
    "Distanta intre markerul 5 si 42 este 146.76 pixeli.",
    "Distanta intre markerul 5 si 3 este 278.62 pixeli.",
    "Distanta intre markerul 5 si 4 este 195.06 pixeli.",
    "Distanta intre markerul 5 si 15 este 291.34 pixeli.",
    "Distanta intre markerul 42 si 3 este 131.89 pixeli.",
    "Distanta intre markerul 42 si 4 este 108.98 pixeli.",
    "Distanta intre markerul 42 si 15 este 161.31 pixeli.",
    "Distanta intre markerul 3 si 4 este 155.83 pixeli.",
    "Distanta intre markerul 3 si 15 este 95.55 pixeli.",
    "Distanta intre markerul 4 si 15 este 111.70 pixeli."
  ]
}

Distanța dintre fata masinii și spatele masinii este: 71.48
Cel mai apropiat marker de masina este 42, aflat la distanta: 146.76
Distanța de la markerul 42 la spatele masinii este: 197.46
Verificam daca masina este orientata corect catre markerul 42
Masina nu este in linie dreapta catre markerul 42
Incepem calibrarea directiei masinii catre markerul 42
```


Figura 61 Informații pentru cazul 3

Din 4 markeri, cel mai apropiat este cel cu id-ul 42, aflat la distanța 146.76 pixeli.

Figura 62 Caz 4 - minim dintre 5 markeri


```
{
  "distances": [
    "Distanta intre markerul 6 si 5 este 74.58 pixeli.",
    "Distanta intre markerul 6 si 4 este 249.21 pixeli.",
    "Distanta intre markerul 6 si 3 este 355.00 pixeli.",
    "Distanta intre markerul 6 si 60 este 311.05 pixeli.",
    "Distanta intre markerul 6 si 15 este 386.14 pixeli.",
    "Distanta intre markerul 6 si 42 este 395.58 pixeli.",
    "Distanta intre markerul 5 si 4 este 181.74 pixeli.",
    "Distanta intre markerul 5 si 3 este 295.54 pixeli.",
    "Distanta intre markerul 5 si 60 este 236.95 pixeli.",
    "Distanta intre markerul 5 si 15 este 316.80 pixeli.",
    "Distanta intre markerul 5 si 42 este 321.82 pixeli.",
    "Distanta intre markerul 4 si 3 este 122.80 pixeli.",
    "Distanta intre markerul 4 si 60 este 97.76 pixeli.",
    "Distanta intre markerul 4 si 15 este 137.29 pixeli.",
    "Distanta intre markerul 4 si 42 este 164.28 pixeli.",
    "Distanta intre markerul 3 si 60 este 164.84 pixeli.",
    "Distanta intre markerul 3 si 15 este 94.55 pixeli.",
    "Distanta intre markerul 3 si 42 este 171.39 pixeli.",
    "Distanta intre markerul 60 si 15 este 112.18 pixeli.",
    "Distanta intre markerul 60 si 42 este 85.40 pixeli.",
    "Distanta intre markerul 15 si 42 este 81.07 pixeli."
  ]
}

Distanța dintre fata masinii și spatele masinii este: 236.95
Cel mai apropiat marker de masina este 4, aflat la distanta: 181.74
Distanța de la markerul 4 la spatele masinii este: 97.76
Verificam daca masina este orientata corect catre markerul 4
Masina nu este in linie dreapta catre markerul 4
Incepem calibrarea directiei masinii catre markerul 4
```


Figura 63 Informații pentru cazul 4

Din 5 markeri, cel mai apropiat este cel cu id-ul 4, aflat la distanța 181.74 pixeli.

6.3.2 Timpi de execuție

Raportat la unghiul si poziția camerei, am analizat timpul necesar mașinii de a ajunge la destinație pentru două cazuri:

1. Mașina este orientată invers față de destinație.

Distanța de la fața mașinii la cel mai apropiat marker	Timpul în care mașina ajunge din punctul inițial la destinație
120 pixeli	16 secunde
170 pixeli	23 secunde
220 pixeli	30 secunde
300 pixeli	40 secunde
350 pixeli	47 secunde

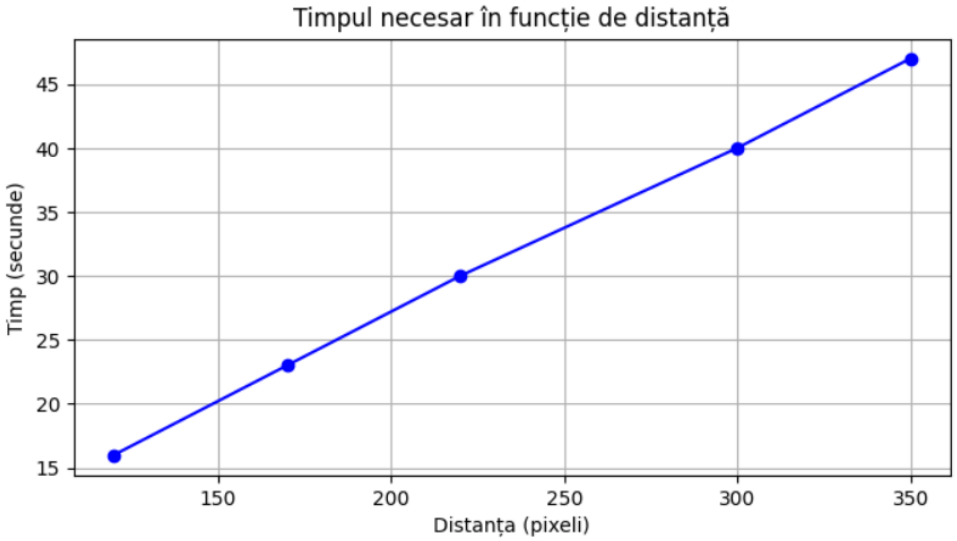

Figura 64 Grafic 1 - Timpul necesar în funcție de distanță

2. Mașina este orientată corect față de destinație.

Distanța de la fața mașinii la cel mai apropiat marker	Timpul în care mașina ajunge din punctul inițial la destinație
120 pixeli	6 secunde
170 pixeli	9 secunde
220 pixeli	12 secunde
300 pixeli	16 secunde
350 pixeli	19 secunde

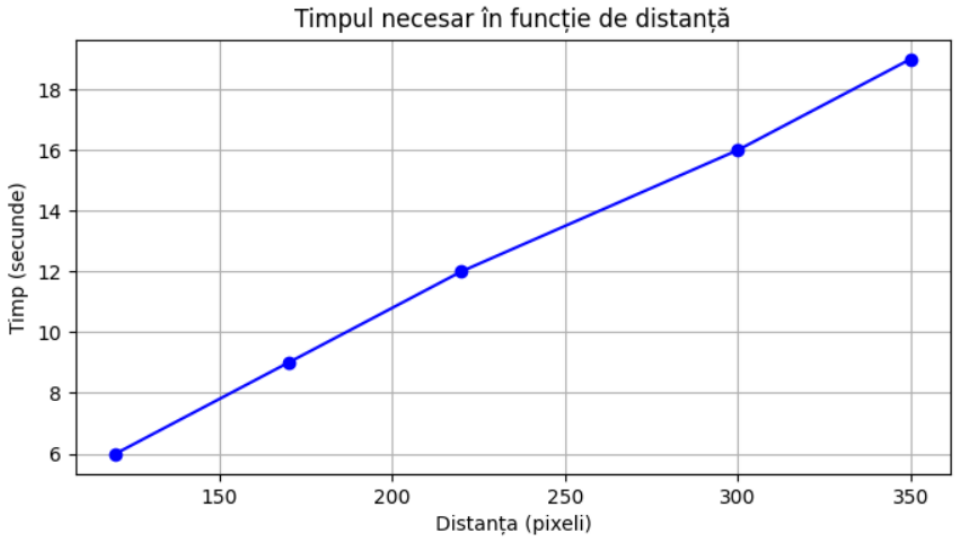

Figura 65 Grafic 2 - Timpul necesar în funcție de distanță

Putem deprinde din rezultate că mașina are nevoie de aproximativ 2.5-3 secunde pentru a parcurge o distanță de 50 pixeli în linie dreaptă și are nevoie de aproximativ 15 secunde în plus pentru a se orienta corect față de destinație dacă a fost poziționat cu spatele la aceasta.

6.3.3 Număr de mișcări

Raportat la unghiul si poziția camerei, am analizat numărul de mișcări necesar mașinii de a ajunge la destinație pentru două cazuri:

1. Mașina este orientată invers față de destinație.

Distanța de la fața mașinii la cel mai apropiat marker	Numărul de mișcări necesare pentru a ajunge la destinație
120 pixeli	16
170 pixeli	17
220 pixeli	19
300 pixeli	21
350 pixeli	22

2. Mașina este orientată corect față de destinație.

Distanța de la fața mașinii la cel mai apropiat marker	Numărul de mișcări necesare pentru a ajunge la destinație
120 pixeli	2
170 pixeli	3
220 pixeli	4
300 pixeli	6
350 pixeli	7

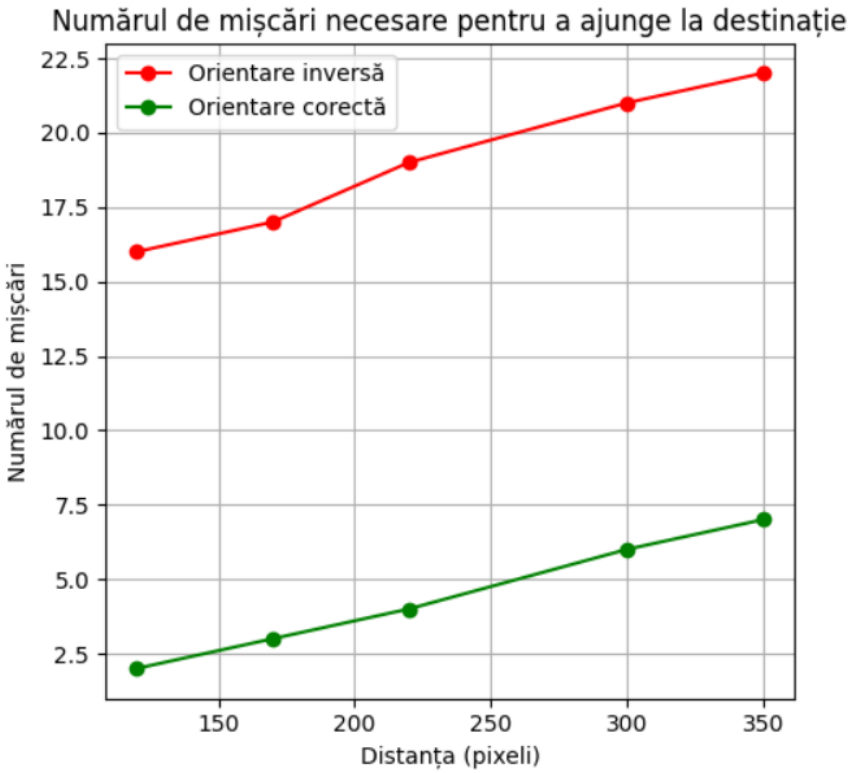

Figura 66 Grafic - Numărul de mișcări necesare pentru a ajunge la destinație

Putem deprinde din rezultate că mașina poate parcurge o distanță de aproximativ 50 de pixeli cu o mișcare înainte și are nevoie de aproximativ 13-14 mișcări pentru a se orienta corect față de destinație dacă a fost plasată invers (în linie dreaptă dar cu spatele mai aproape de destinație).

6.3.4 Precizia

După cum am putut observa din execuția programelor și din log-uri, distanțele între markeri sunt calculate în pixeli. Acest lucru se datorează bibliotecii care face aceste prelucrări pe imagini. Acuratețea mișcării este avantajată de folosirea distanței în pixeli deoarece prezintă un mod mai rapid și mai corect de prelucrare a deciziilor în timp real.

De asemenea putem face o conversie foarte ușor pentru a afla distanța în centimetri/metri pentru calcule ce necesită acest lucru. Ne sunt necesare doar distanța de la cameră la obiectul de referință, unghiul folosit și distanța în pixeli.

Dorim să examinăm cât de precisă este detecția mișcării folosind două cereri la un interval de o secundă în care mașina execută o mișcare foarte mică.

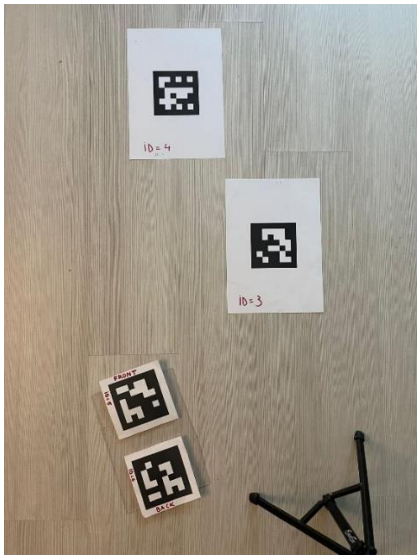

Figura 67 Caz 1 pentru studiul preciziei


```
{
  "distances": [
    "Distanta intre markerul 6 si 5 este 69.40 pixeli.",
    "Distanta intre markerul 6 si 3 este 211.24 pixeli.",
    "Distanta intre markerul 6 si 4 este 286.58 pixeli.",
    "Distanta intre markerul 5 si 3 este 163.58 pixeli.",
    "Distanta intre markerul 5 si 4 este 221.30 pixeli.",
    "Distanta intre markerul 3 si 4 este 120.39 pixeli."
  ]
}
```


```
Distanța dintre fata masinii și spatele masinii este: 69.40
Cel mai apropiat marker de masina este 3, aflat la distanta: 163.58
Distanța de la markerul 3 la spatele masinii este: 211.24
Verificam daca masina este orientata corect catre markerul 3
Masina nu este in linie dreapta catre markerul 3
Incepem calibrarea directiei masinii catre markerul 3
```

Figura 68 Informații pentru cazul 1

Din informațiile inițiale, aflăm că fața mașinii se află la distanța de 163.58 pixeli de cel mai apropiat marker.

Figura 69 Caz 2 pentru studiul preciziei

```
{
  "distances": [
    "Distanta intre markerul 6 si 5 este 62.37 pixeli.",
    "Distanta intre markerul 6 si 3 este 172.87 pixeli.",
    "Distanta intre markerul 6 si 4 este 246.79 pixeli.",
    "Distanta intre markerul 5 si 3 este 162.03 pixeli.",
    "Distanta intre markerul 5 si 4 este 205.62 pixeli.",
    "Distanta intre markerul 3 si 4 este 120.39 pixeli."
  ]
}

Distanța dintre fata masinii și spatele masinii este: 62.37
Cel mai apropiat marker de masina este 3, aflat la distanta: 162.03
Distanța de la markerul 3 la spatele masinii este: 172.87
Verificam daca masina este orientata corect catre markerul 3
Masina nu este in linie dreapta catre markerul 3
Incepem calibrarea directiei masinii catre markerul 3
```


Figura 70 Informații pentru cazul 2

La următorul get, realizat după o secundă, mașina s-a deplasat din poziția inițială și putem observa că aceste detalii au fost sesizate și modificate.

Fața mașinii se află la distanța de 162.03 pixeli de cel mai apropiat marker. Orientarea acesteia s-a modificat și în același timp și relațiile față de punctele de referință din jur.

Aceste rezultate ne indică capacitatea ansamblului de a răspunde cu o precizie ridicată la variabilele din jur și la situații ce necesită reacție rapidă.

6.3.5 Corectitudine în depanare

Un detaliu foarte important pentru corectitudinea implementării este reprezentat de capacitatea de a putea răspunde rapid la diversele probleme sau sarcini ce se pot modifica în timp real.

După cum au fost explicate anterior, toate cele 3 componente de bază (aplicație mobilă, server, schelet robotizat) sunt însoțite de interfețe de comunicare (log) cu mesaje amănunțite și explicite pentru a putea urma un fir logic pe parcursul rulării acestora. Aceste mesaje conțin toate detaliile necesare verificării oricărui amănunt pentru a putea fi rezolvat orice aspect.

De asemenea, în fișierul pentru uploads sunt stocate toate cadrele pe care aplicația mobilă le capturează. Fiecare imagine primește o denumire diferită care este utilizată ca referință ulterior in log.

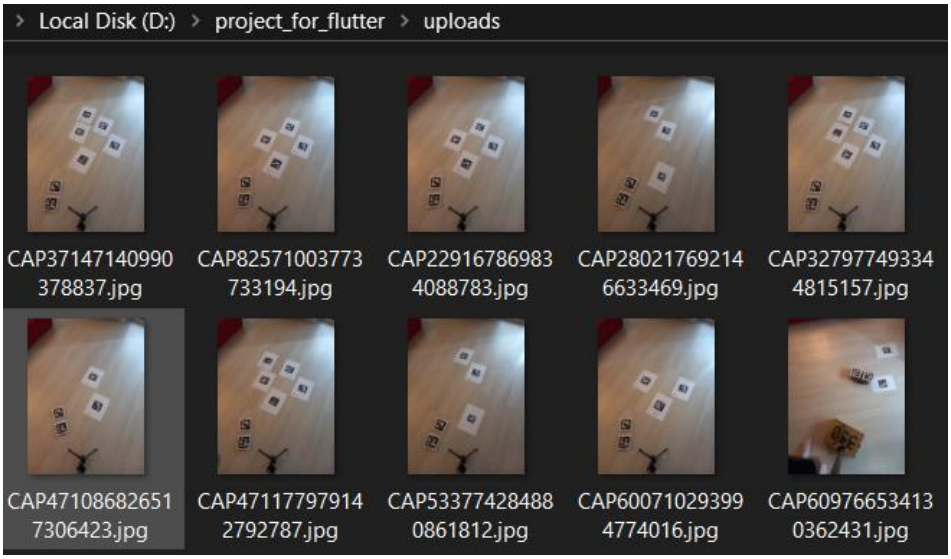

Figura 72 Imaginile încărcate în fișierul de upload

Aceste detalii amănunțite ne asigură că putem urma cursul de execuție al ansamblului pentru a înțelege orice aspect și pentru a putea interveni rapid dacă este necesar.

7 UTILIZARE

7.1 Utilizare actuală

Ansamblul realizat poate fi programat pentru a îndeplini diverse sarcini folosind structura prezentată. Cu ajutorul implementării actuale acesta poate fi utilizat cu ușurință și eficiență pentru a se orienta într-un spațiu determinat.

Dacă folosim un telefon mobil cu o cameră bună, poziționat la un unghi mai mare scheletul se poate deplasa pe o suprafață plană pentru a atinge diverse puncte considerate destinații. Avem acces la 250 de markeri ArUco, suficienți pentru a marca traseul pe care trebuie sa îl urmeze.

Scheletul poate fi folosit pentru: transport de materiale într-un depozit, transport de probe biologice într-un laborator, asistent personal la cumpărături, asistent personal în casă, transportul diverselor sisteme în agricultură, unde condițiile de lucru (căldură excesivă, deshidratare) sunt dificile pentru oameni dar inofensive pentru scheletul robotizat, etc.

7.2 Eficiența comparativă

Inițial, am comparat structura pe care doream sa o implementăm cu tehnologiile AGV cu sistemul LIDAR. Principalele îmbunătățiri pe care doream să le aducem au fost: reducerea costurilor, simplificarea procesului, a tehnologiilor folosite și obținerea unui sistem cu eficiență ridicată pentru sarcini mai puțin complexe decât cele rezolvate de LIDAR.

Consider că structura prezentată aduce îmbunătățirile menționate anterior. Costul pentru producția sistemului actual nu depășește 250$, tehnologiile folosite sunt mai apropiate de înțelesul utilizatorilor, calculele necesare nu prezintă un grad ridicat de complexitate și aceasta poate fi utilizată eficient pentru multe sarcini.

8 ÎMBUNĂTĂȚIRI

Îmbunătățirea algoritmilor de navigație	O potențială îmbunătățire a proiectului ar fi optimizarea algoritmilor de navigație pentru a crește eficiența și precizia deplasării robotului. Aceasta poate include implementarea unor algoritmi avansați de machine learning pentru recunoașterea și evitarea obstacolelor în timp real.
Optimizarea consumul energetic	Un alt aspect important de îmbunătățit este consumul energetic al sistemului. Implementarea unor algoritmi de gestionare a energiei și utilizarea unor componente hardware mai eficiente din punct de vedere energetic poate prelungi durata de funcționare a robotului.
Îmbunătățirea interfeței utilizator	Interfața utilizator poate fi optimizată pentru a fi mai intuitivă și ușor de utilizat. Aceasta include simplificarea procesului de configurare și operare a robotului, precum și adăugarea unor funcționalități suplimentare care să ofere feedback în timp real utilizatorului.
Extinderea funcționalităților	Dezvoltarea unor noi funcționalități poate extinde domeniul de aplicabilitate al robotului. De exemplu, adăugarea de module pentru manipularea obiectelor sau integrarea cu alte sisteme automatizate poate crește versatilitatea și utilitatea robotului în diverse scenarii.

9 CONCLUZII

Proiectul a demonstrat cu succes eficiența utilizării markerilor ArUco și a camerelor pentru localizarea precisă a vehiculelor RC. Implementarea unui sistem automatizat accesibil, bazat pe tehnologii moderne, a evidențiat potențialul democratizării automatizării. Utilizarea unui smartphone și a unor tehnologii open-source a permis crearea unei soluții cost-eficiente și ușor de utilizat, accesibilă unui public larg. Integrarea componentelor hardware și software a fost realizată cu succes, asigurând o funcționare armonioasă și eficientă a sistemului. Acest proiect deschide calea pentru dezvoltări ulterioare și extinderea aplicabilității sistemelor de automatizare în diverse domenii.

10 BIBLIOGRAFIE

Referințele au fost sortate alfabetic după denumirea lucrării.

1	Sonali Jadhav, Rupali Gawande, „A Study of Impact of Automation on Industry and Employees", 2020. [Online]. Available: https://aissmschmct.in/wp-content/uploads/2020/07/Study-of-Impact-of-Automation-on-Industry-Employees-sonali-jadhav.pdf
2	Executive Office of the President of the United States, „Artificial Intelligence, Automation, and the Economy", 2016 [Online], Available: https://obamawhitehouse.archives.gov/blog/2016/12/20/artificial-intelligence-automation-and-economy
3	Linda Shapiro, George Stockman, „Computer Vision", 2000. [Online]. Available: https://nana.lecturer.pens.ac.id/index_files/referensi/computer_vision/Computer%20Vision.pdf
4	Rafael C. Gonzalez, Richard E. Woods, „Digital Image Processing (4th edition)", 2008. [Online]. Available: https://dl.icdst.org/pdfs/files4/01c56e081202b62bd7d3b4f8545775fb.pdf
5	Nikhil Kumar, „Impact of the Business Process Automation on Human Resource Management", 2014. [Online]. Available: https://www.worldwidejournals.com/paripex/recent_issues_pdf/2014/March/March_2014_1395995589_2927c_33.pdf
6	S. K. Roy, „Localization Algorithms and Strategies for Real-Time Systems", 2017
7	Sebastian Thrun, Wolfram Burgard, Dieter Fox, „Probabilistic Robotics: The Intelligence for Robots Moving in Uncertain Environments", 2005. [Online]. Available: https://docs.ufpr.br/~danielsantos/ProbabilisticRobotics.pdf
8	Terry Gregory, Anna Salomons, Ulrich Zierahn, „Racing With or Against the Machine? Evidence on the Role of Trade in Europe", 2022. [Online]. Available: https://academic.oup.com/jeea/article/20/2/869/6382134
9	Ruben Barrera-Michel, „Robotics and automation [The Way Ahead]", 2023. [Online]. Available: https://ieeexplore.ieee.org/document/10120613
10	Georg Graetz, Guy Michaels, „Robots at Work", 2015. [Online]. Available: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2589780
11	Richard Breton, Éloi Bossé, „The Cognitive Costs and Benefits of Automation", 2003. [Online]. Available: https://www.researchgate.net/publication/235171183_The_Cognitive_Costs_and_Benefits_of_Automation
12	Radhamadhavan Madhavan, Ludovic Righetti, William D. Smart, „The Impact of Robotics and Automation on Working Conditions and Employment [Ethical, Legal, and Societal Issues]", 2018. [Online]. Available: https://www.researchgate.net/publication/325770874_The_Impact_of_Robotics_and_Automation_on_Working_Conditions_and_Employment_Ethical_Legal_and_Societal_Issues
13	K. Goldberg, „What is Automation?", 2011. [Online]. Availbale: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6104197

11 ACCES LA COD

Codul utilizat în realizarea proiectului poate fi găsit aici:

https://github.com/visan-ionut/Complex_RC_Vehicle_Estimation