SVEUČILIŠTE J.J. STROSSMAYER U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij

UGRADBENI RAČUNALNI SUSTAVI

Profesor/mentor**: prof. dr. sc. Tomislav Keser**

TEHNIČKA DOKUMENTACIJA PROJEKTA

MOBILE ROBOT (THENKICH)

Samuel Adžić, Valentin Veselčić

Akademska godina 2024/2025

Osijek, 22.09.2025.

SADRŽAJ

1. UVOD

Mobilni roboti postaju sve značajniji u industriji, sigurnosti, logistici i edukaciji. Ključna sposobnost suvremenih mobilnih platformi je autonomno kretanje uz percepciju okoline i donošenje odluka u stvarnom vremenu. U praksi se najčešće kombiniraju senzori udaljenosti, poput ultrazvučnih, i vizualni sustavi (npr. *ESP32-CAM*), uz bežičnu povezanost (*Wi-Fi* (bežična mreža) radi nadzora i interakcije s korisnikom. Dostupna su brojna otvorena (*open-source*) rješenja koja omogućuju usporedbu pristupa, arhitekture i algoritama upravljanja.

Kao reprezentativan primjer mobilne platforme s udaljenim upravljanjem i prijenosom u stvarnom vremenu, projekt Acrome prikazuje *ESP32/ESP32-CAM* robota s *web*-sučeljem za teleoperaciju i prikaz slike; dokumentiraju se korištene komponente, mrežna konfiguracija i integracija kamere i upravljanja pogonom, što je izravno relevantno za pristup povezanosti i vizualne potvrde cilja u ovom radu [1].

Za radarsku vizualizaciju udaljenosti i *web*-prikaz mjerenja, otvoreni projekt „ESP32 Radar System“ oslanja se na *ESP32*, ultrazvučni senzor i servo za skeniranje prostora te hosta *web*-sučelje na samom mikrokontroleru. Rješenje je korisno kao referenca za organizaciju mjerenja, radarsku projekciju i objavu podataka preko integriranog poslužitelja [2].

Treći relevantan primjer je autonomni robot za izbjegavanje prepreka temeljen na *ESP32* s ultrazvučnim senzorom i servo-skenerom. Projekt jasno dokumentira ulogu skeniranja pri donošenju odluka i osnovni algoritam izbjegavanja, što je korisno za usporedbu logike kretanja i rasporeda sklopovskih cjelina u jednostavnim mobilnim platformama [3].

Usporedba navedenih rješenja ukazuje na česte zajedničke elemente: ultrazvučno mjerenje s rotacijskim skeniranjem, prikaz rezultata na *web*-u te osnovne obrasce kretanja (ravno, skretanje, izbjegavanje). Istodobno, uočen je prostor za nadogradnju u pogledu potvrde cilja dodatnim senzorom (npr. *PIR* - pasivna infracrvena detekcija, za razlikovanje živog i neživog objekta) te integracije vizualne potvrde putem kamere, uz jasniju podjelu programskih funkcija (mjerenje, odlučivanje, upravljanje pogonom i komunikacija).

Specifičnost i doprinos ovoga rada jest objedinjavanje više tehnika u kompaktnoj *ESP32* platformi: diferencijalni pogon sa servomotorima *MG90S*, ultrazvučno skeniranje za prostornu percepciju, *PIR* detekcija kao potvrda toplinskog cilja, vizualna potvrda putem *ESP32-CAM* modula te *web*-poslužitelj s komunikacijom u stvarnom vremenu. Programsko rješenje strukturirano je u zasebne cjeline (senzorska obrada, odlučivanje, aktuacija, komunikacija), što

olakšava testiranje, usporedbu i buduće nadogradnje. Motivacija za izradu sustava je provjeriti koliko se jednostavne i lako dostupne komponente mogu iskoristiti za implementaciju očekivanih funkcionalnosti u autonomnim robotima — percepcija, potvrda cilja, vizualni nadzor i mrežna povezanost — te na toj osnovi izgraditi funkcionalan, reproducibilan i mjerljiv demonstrator koji spaja elektroniku, ugradbeno programiranje, obradu podataka i *web*-tehnologije.

1.1. Zadatak i struktura rada

Zadatak ovog projekta je izraditi mobilnog robota koji se autonomno kreće pomoću diferencijalnog pogona (dva servo motora), te koristi senzore i kameru za detekciju, identifikaciju i klasifikaciju objekata u okolini u interakciji s korisnikom. Sustav koristi ultrazvučni senzor, postavljen na dodatni servo motor, za pregledavanje prostora ispred robota. U trenutku kada senzor detektira objekt, aktivira se *PIR* senzor koji ispituje postoji li toplinski izvor, što može ukazivati na prisutnost živog bića. Ukoliko *PIR* senzor vrati signal da postoji izvor topline (logička jedinica), robot se zaustavlja i korisniku omogućuje da putem kamere, spojene na *ESP32-CAM* modul, snimi sliku i vizualno potvrdi o kakvom se objektu radi – živom biću ili neživom objektu. Nakon potvrde, sustav nastavlja s radom na zahtjev korisnika.

Cilj projekta je prikazati mogućnost kombiniranja osnovnih elektroničkih komponenti za ostvarenje naprednijeg ponašanja mobilnog robota uključujući izbjegavanje objekata, razlikovanje živih bića, reakciju na okolinu i uključivanje korisnika u donošenje odluke. Konačni rezultat je funkcionalan sustav koji može poslužiti kao temelj za daljnji razvoj u području autonomnih jedinica i jednostavne strojne percepcije.

Struktura dokumentacije organizirana je kroz sljedeća poglavlja:

- Poglavlje 1 – Uvod: Pregled konteksta i motivacija za projekt.
- Poglavlje 2 – Teorijska podloga i analiza problema: Opis principa rada korištenih senzora i logike sustava, prijedlog sklopovskog i programskog rješenja.
- Poglavlje 3 – Realizacija sustava: Detaljan opis korištenih komponenti, izrada električkih shema, konstrukcijskih nacrta i blok dijagrama programskog toka.
- Poglavlje 4 – Testiranje i rezultati: Metodologija testiranja, provedeni eksperimenti i analiza rezultata.
- Poglavlje 5 – Zaključak: Završni osvrt s prednostima, nedostacima i prijedlozima poboljšanja.

- Literatura: Popis svih izvora korištenih tijekom projekta.
- Prilozi i dodaci: Izvorni programski kod, tehnički podaci i dodatni materijali.

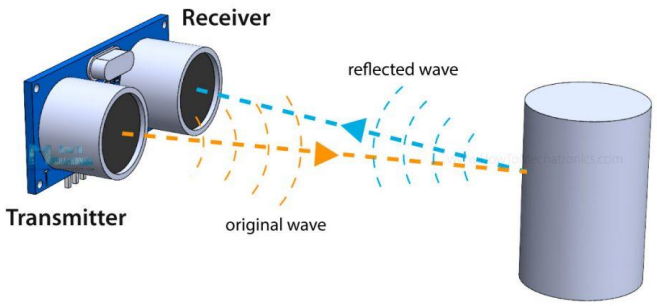
2. SUSTAV ZA DETEKCIJU I IDENTIFIKACIJU OBJEKATA POMOĆU ESP32 I SENZORA OKOLINE

U ovom poglavlju prikazuje se teorijska podloga i osnovni principi na kojima se temelji sustav. U 2.1 se daje pregled fizikalnih zakonitosti i algoritamskih načela koja omogućuju detekciju i klasifikaciju objekata. U 2.2 prikazuje se planirano sklopovsko rješenje kroz funkcionalne blokove i njihove međusobne odnose. U 2.3 se predlaže koncept programskog toka kroz grubi blok dijagram koji objašnjava osnovnu logiku rada.

2.1. Teorijski osvrt na problem i rješenje projekta

Rješenje razvijeno u sklopu ovog projekta temelji se na potrebi da se mobilni robotski sustav sposoban za samostalno kretanje ponaša inteligentno u dinamičnom okruženju. Pritom se zahtijeva da robot bude u mogućnosti detektirati objekte u svojoj okolini, razlikovati živa bića od neživih prepreka, orijentirati se u prostoru te uključiti korisnika u donošenje konačne odluke u specifičnim slučajevima. Funkcionalnost sustava oslanja se na skup međusobno povezanih principa i zakonitosti koji čine osnovu za logičko odlučivanje i upravljanje ponašanjem.

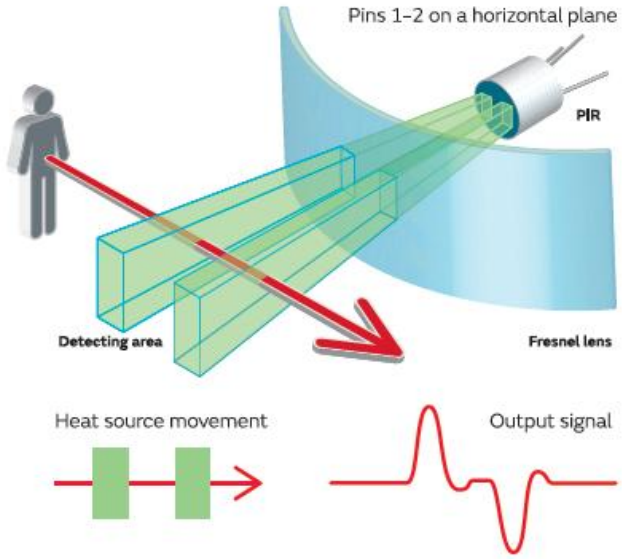
Jedan od temeljnih principa korišten u sustavu jest određivanje udaljenosti do objekta pomoću zvučnog vala. Riječ je o reflektivnoj metodi u kojoj se emitira kratki ultrazvučni impuls, a zatim mjeri vrijeme potrebno da se taj impuls vrati reflektiran od objekta. Na osnovu poznate brzine širenja zvuka u zraku moguće je izračunati udaljenost do objekta. Ovaj mehanizam omogućuje robotu da „vidi“ prepreke ispred sebe, odnosno da uoči njihovu pojavu unutar definiranog radijusa detekcije.



Slika 2.1. *Princip rada ultrazvučnog senzora.* [4]

Nakon što je objekt detektiran na određenoj udaljenosti, sustav koristi metodu infracrvene detekcije kako bi provjerio sadrži li taj objekt vlastito toplinsko zračenje. Taj postupak se temelji na pasivnoj registraciji infracrvenih valova koji se emitiraju iz toplih tijela, kao što su ljudi ili

životinje. Živa bića emitiraju toplinsku energiju u spektru koji je moguće registrirati, što omogućuje donošenje osnovne klasifikacijske odluke: je li objekt potencijalno živi organizam ili neživa prepreka.



Slika 2.2. *Princip rada PIR senzora*. [5]

Dodatna vrijednost sustava očituje se u njegovoj mogućnosti prostorne orijentacije. Za to se koristi rotirajući mehanizam koji omogućuje senzorsko skeniranje okoline pod različitim kutovima. Na temelju promjene detekcije u ovisnosti o kutu rotacije senzora, sustav može zaključiti u kojem se smjeru objekt nalazi te odgovarajuće reagirati – zakrenuti udesno, ulijevo ili nastaviti ravno, ovisno o orijentaciji objekta u odnosu na trenutni položaj robota.

Sva ova senzorska saznanja objedinjena su kroz algoritamsku logiku koja je razvijena u obliku jednostavnog stanja sustava. U svakom trenutku robot se nalazi u određenom stanju – mirovanja, kretanja naprijed, zaokreta ili zaustavljanja zbog detekcije cilja. Ovisno o ulaznim podacima iz senzora i vremenskim uvjetima, sustav prelazi iz jednog stanja u drugo, čime se osigurava odziv u stvarnom vremenu. Ovakva arhitektura ponašanja temelji se na uzročno-posljedičnim vezama – detekcija objekta izaziva reakciju sustava koja je definirana logikom algoritma.

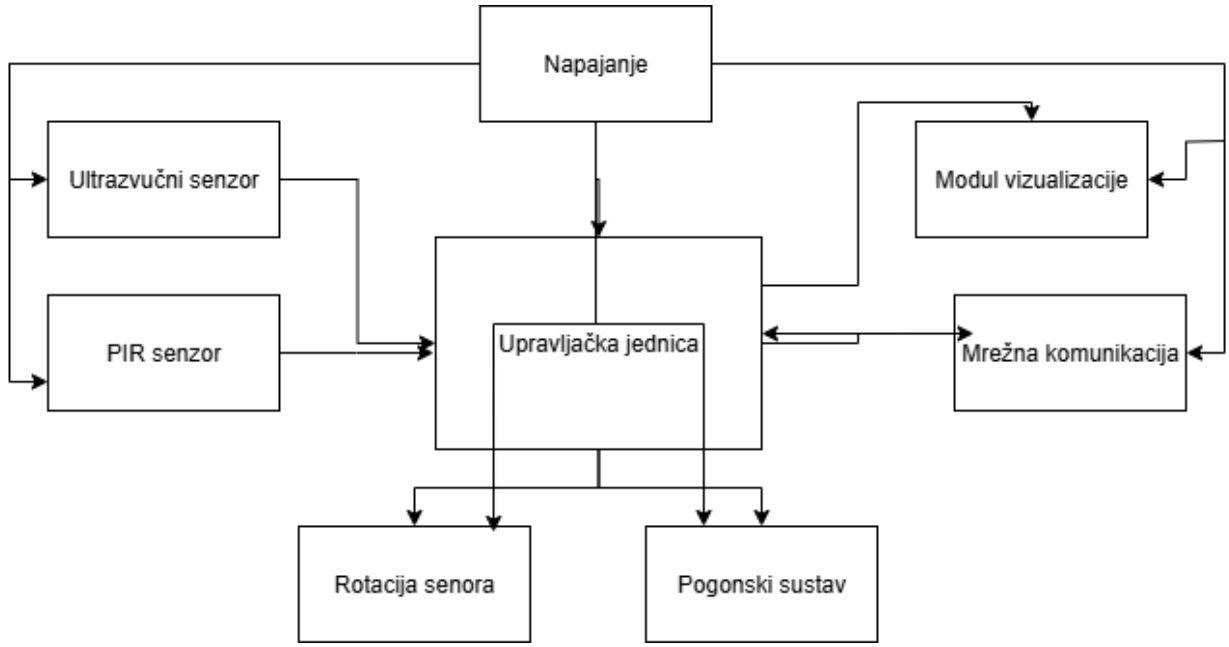
Kao dodatak sustavu, uvedena je i mogućnost vizualne potvrde cilja. U trenutku kada sustav zaključi da je ispred njega živi objekt, prelazi u pasivno stanje u kojem se korisniku omogućuje pristup vizualnoj informaciji o objektu. Na taj način, krajnji korisnik ima mogućnost vizualno procijeniti o čemu se radi i donijeti odluku o daljnjem kretanju robota. Takva kombinacija

automatiziranog odlučivanja i korisničke interakcije osigurava veću sigurnost i pouzdanost sustava u stvarnim uvjetima rada.

Sve opisane metode i zakonitosti objedinjene su u funkcionalno rješenje koje omogućuje robusno i kontekstualno ponašanje mobilnog sustava u nepoznatom i promjenjivom okruženju. Projekt je time ostvario osnovnu razinu inteligentnog odlučivanja koristeći jednostavne senzorske principe i jasno definiranu algoritamsku logiku upravljanja.

2.2. Prijedlog sklopovskog rješenja

U ovom potpoglavlju prikazuje se planirano sklopovsko rješenje mobilnog robota kroz funkcionalne blokove i njihove međusobne odnose. Na slici 2.3 prikazan je funkcionalni blok dijagram koji obuhvaća upravljačku jedinicu, senzore, aktuatorske podsustave, modul vizualizacije, mrežnu komunikaciju i napajanje.



Slika 2.3. *Funkcionalni blok dijagram planiranog sklopovskog rješenja.*

Središnji dio sustava čini mikrokontrolerska upravljačka jedinica koja ima zadatak obraditi ulazne podatke, upravljati aktuatorima, održavati komunikaciju i kontrolirati logičke prijelaze u ponašanju sustava. Podaci ulaze u sustav kroz dva senzorska podsustava: ultrazvučni senzor za detekciju udaljenosti i položaja prepreka te *PIR* senzor za potvrdu toplinske prisutnosti. Njihovi izlazi prenose se prema upravljačkoj jedinici, gdje se analiziraju i povezuju s algoritmom odlučivanja. Rotacijski mehanizam, povezan s upravljačkom jedinicom, omogućuje prostorno skeniranje senzora i pruža informacije o distribuciji objekata u horizontalnoj ravnini. Pogonski

sustav, također upravljan iz središnje jedinice, omogućuje kretanje robota prema naprijed, natrag te izvođenje zaokreta. Kada senzorski ulazi potvrde postojanje toplinskog izvora, upravljačka jedinica aktivira modul vizualizacije koji snima sliku objekta i prenosi je korisniku. Komunikacija se ostvaruje putem mrežnog sučelja koje omogućuje dvosmjernu razmjenu podataka – korisnik prima informacije o udaljenosti, statusu toplinske detekcije i vizualne potvrde, a može također inicirati naredbe za pokretanje ili zaustavljanje rada sustava. Napajanje je prikazano kao zajednički blok koji osigurava energiju svim funkcionalnim jedinicama. Veze sa slike 2.3 simbolično označavaju ovisnost podsustava o izvoru napajanja, bez ulaska u detalje distribucije naponskih razina.

Predloženo rješenje time osigurava modularnost, fleksibilnost i jasno razdvajanje funkcionalnih uloga, što omogućuje lakšu nadogradnju i prilagodbu sustava u kasnijim fazama razvoja.

2.3. Prijedlog programskog rješenja

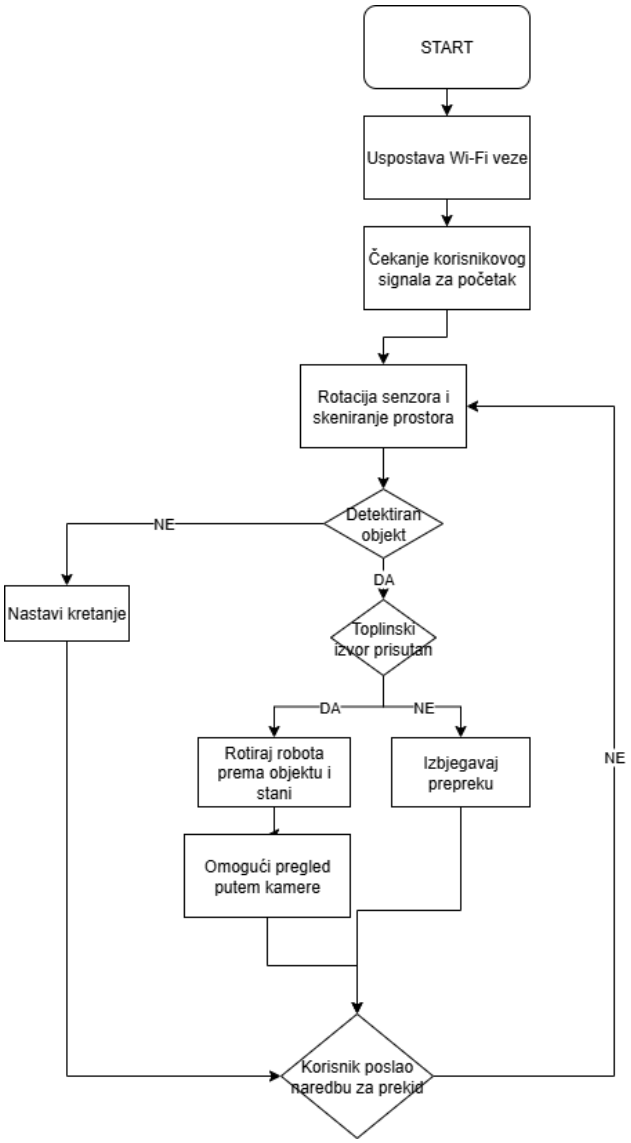
Programsko rješenje zamišljeno je kao niz uzročno-posljedičnih logičkih koraka koji se neprekidno izvršavaju dok je sustav aktivan. Glavni algoritam se temelji na beskonačnoj petlji koja upravlja ponašanjem robota na temelju podataka dobivenih iz senzora i korisničkih naredbi. Svaki senzorski ulaz ima jasno definiranu ulogu i neposredno utječe na stanje sustava i naredne korake izvršavanja.

Nakon inicijalizacije sustava i uspostave *Wi-Fi* veze, čeka se korisnički signal za početak rada. Po aktivaciji, sustav ulazi u glavni ciklus. U tom ciklusu neprekidno se mjeri udaljenost do objekata pomoću senzora, te se istovremeno obavlja rotacija senzora (radara) kako bi se pokrilo šire područje ispred robota.

Ukoliko je detektiran objekt na određenoj udaljenosti, aktivira se provjera prisutnosti toplinskog zračenja putem *PIR* senzora. Ako se potvrdi toplinski izvor, robot se zaustavlja, a korisniku se omogućuje vizualni pregled objekta putem kamere.

Ako toplinski izvor nije prisutan, algoritam zaključuje da se radi o običnoj fizičkoj prepreci. U tom slučaju, aktivira se procedura izbjegavanja prepreke – promjena smjera kretanja i pokušaj zaobilaženja objekta. Ako objekt nije detektiran ili je sigurno udaljen, robot nastavlja s kretanjem.

Nakon svakog ciklusa sustav se vraća na početak petlje i proces se ponavlja, sve dok korisnik ne pošalje naredbu za prekid ili dok se ne završi definirana misija.



Slika 2.4. *Blok dijagram programskog toka.*

3. REALIZACIJA MOBILNOG ROBOTSKOG SUSTAVA ZA DETEKCIJU OBJEKATA I VIZUALNU POTVRDU CILJA

U ovom poglavlju prikazana je realizacija mobilnog robotskog sustava za detekciju objekata i vizualnu potvrdu cilja. Najprije su u poglavlju 3.1 opisane korištene komponente, programski alati i razvojna okruženja, uz kratka pojašnjenja njihove funkcije u sustavu. Poglavlje 3.2 donosi detaljnu električnu shemu sustava te konstrukcijska rješenja s pripadajućim tehničkim nacrtima i fotografijama gotovog robota. U poglavlju 3.3 prikazana je realizacija programskog rješenja, kroz detaljan blok dijagram toka i opis logike izvođenja algoritma. Time se u potpunosti dokumentira prijelaz iz idejne faze (poglavlje 2) u konkretnu implementaciju funkcionalnog prototipa.

3.1. Korištene komponente, alati i programska okruženja

Komponente sustava:

- Mikrokontrolerska upravljačka jedinica – *ESP32 CP2102*

Glavni kontrolni sklop s ugrađenom podrškom za bežičnu komunikaciju i višezadaćno izvođenje. Koristi se za upravljanje senzorima, aktuatorima i obradu logike ponašanja robota u stvarnom vremenu. Ujedno omogućuje povezivanje s korisnikom putem *Wi-Fi* mreže [3].

Slika 3.1. *Prikaz ESP32 CP2102 upravljačke jedinice* [6]

- Kamera modul – *ESP32-CAM* s *OV2640* lećom

Korišten za snimanje slike objekta kada je detektirana prisutnost toplinskog zračenja. Modul ima integriranu kameru s poboljšanom lećom za bolju kvalitetu slike i povezan je direktno na mrežno sučelje za vizualni prikaz korisniku [7].

Slika 3.2. *Prikaz ESP32-Cam modula* [8]

- Ultrazvučni senzor – *HC-SR04*

Ovaj senzor koristi refleksiju zvučnog impulsa za mjerenje udaljenosti do objekata. Precizan je u kratkim dometima i koristi se za inicijalnu detekciju prepreka ispred robota. Montiran je na rotirajući mehanizam kako bi se omogućilo skeniranje prostora pod različitim kutovima [9].

Slika 3.3. *Prikaz ultrazvučnog senzora HC-SR04* [10]

- Infracrveni *PIR* senzor – *HC-SR501*

Omogućuje detekciju infracrvenog (toplinskog) zračenja koje emitiraju ljudi i životinje. Aktivira se tek nakon što je ultrazvučni senzor detektirao objekt, čime se sprječava nepotrebna aktivacija senzora i povećava efikasnost sustava [11].

Slika 3.4. *Prikaz infracrvenog PIR senzora HC-SR501* [11]

- Servo motori za kretanje – Servo motori *MG90S* (*metal gear*, 360°)

Dva servo motora s mogućnošću kontinuirane rotacije omogućuju diferencijalni pogon. Omogućuju naprijed, natrag, te zakretanje u mjestu promjenom smjera i brzine okretanja.

- Servo motor za radar – Servo motor *MG90S* (*metal gear*, 180°)

Koristi se za zakretanje ultrazvučnog senzora u rasponu od 0° do 180°. Time se omogućuje prostorno skeniranje ispred robota i preciznija detekcija kuta pod kojim se nalazi prepreka ili objekt.

Slika 3.5. *Korišteni servo motori* [12]

- Nosiva konstrukcija

Fizikalna osnova robota na kojoj su postavljene sve komponente. Omogućuje stabilnost pri kretanju, raspodjelu mase i zaštitu komponenti. Konstrukcija je izrađena tako da se senzor i kamera nalaze u prednjem dijelu za maksimalnu vidljivost.

- Kondenzatori (100 nF, 470 µF)

Manji kondenzatori smanjuju visoko-frekventne smetnje, dok elektrolitski kondenzator pomaže pri stabilizaciji napona i smanjenju naponskih padova pri pokretanju motora.

- Schottky dioda (*1N5819*)

Postavljena na liniju napajanja radi zaštite od povratnih struja i obrnutog polariteta. Zbog niskog naponskog pada pogodna je za zaštitu osjetljivih sklopova poput *ESP32*

Alatni softver i programska okruženja:

- *Arduino IDE*

Glavno razvojno okruženje korišteno za pisanje i učitavanje programskog koda na *ESP32* i *ESP32-CAM* module. Omogućuje korištenje dodatnih knjižnica, integraciju s mikrokontrolerom i serijsku dijagnostiku tijekom razvoja.

- *ESPAsyncWebServer* knjižnica

Služi za uspostavu lokalnog *web* poslužitelja te omogućuje dvosmjernu komunikaciju putem *WebSocket* protokola. Korištena za prijenos podataka i slike između robota i korisnika.

- *TaskScheduler* knjižnica

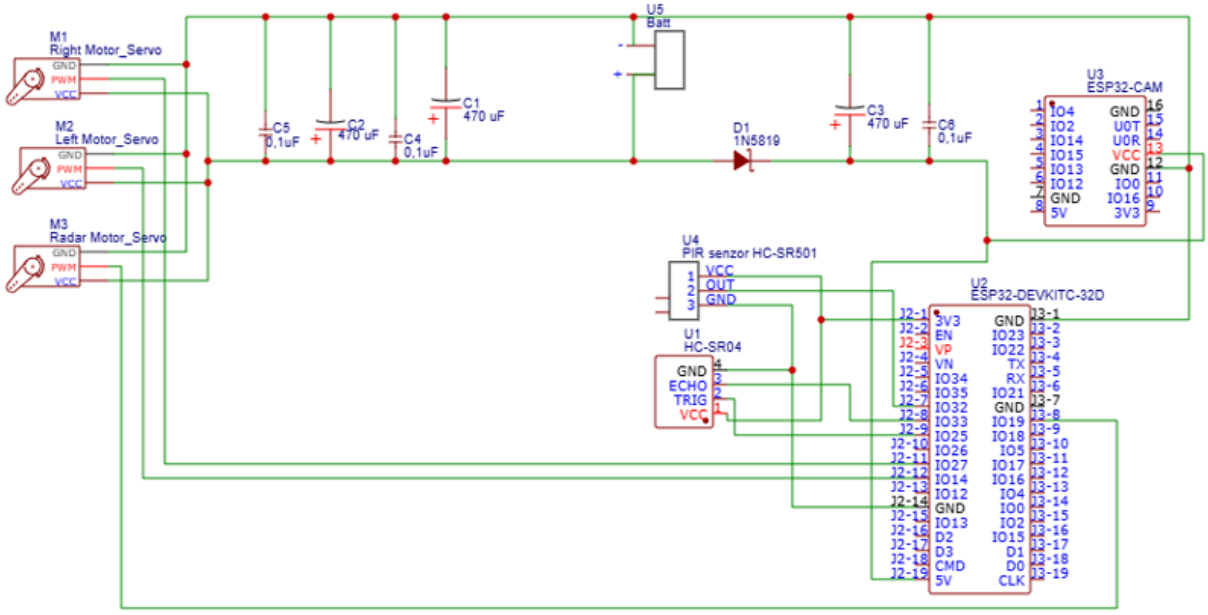
Omogućuje paralelno izvršavanje više funkcionalnih zadataka u pozadini, kao što su mjerenje udaljenosti, kretanje i obrada senzorskih ulaza, čime se osigurava glatko i sinkronizirano ponašanje robota.

- *HTML/JavaScript* sučelje

Razvijeno za prikaz statusa robota u *web* pregledniku. Omogućuje korisniku da vidi udaljenost, kut radara, status PIR senzora, kao i da upravlja početkom ili resetiranjem sustava.

3.2. Realizacija konstrukcijskog i sklopovskog rješenja

U ovom potpoglavlju prikazana je električna shema sustava te konstrukcijska rješenja s tehničkim nacrtima i fotografijama gotovog robota. Shema je temelj za reproducibilnost, dok fotografije i nacrti prikazuju stvarni raspored komponenti i način kabliranja.



Slika 3.6. *Električna shema sustava robota za detekciju objekata i vizualnu potvrdu cilja*

Slika 3.7. *Prikaz gotovog mobilnog robota – bočni pogled*

Vidi se raspored kotača, prednja kamera (*ESP32-CAM*) i ultrazvučni senzor montiran na servo motoru za radarsko skeniranje. Konstrukcija je izrađena od laganih materijala, a komponente su raspoređene radi stabilnosti i funkcionalnosti.

***Slika 3.8**. Prikaz gotovog mobilnog robota – prednji pogled*

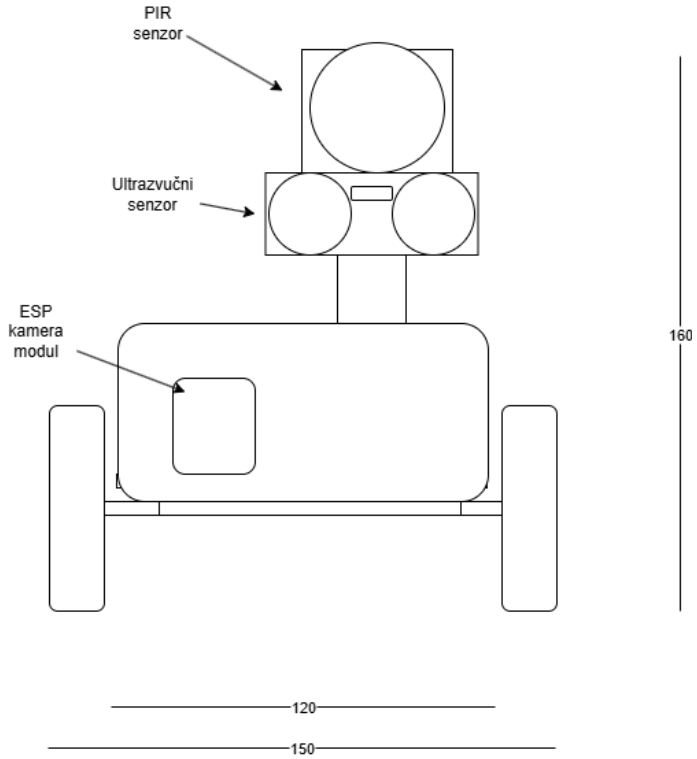
U prvom planu uočava se kamera *(ESP32-CAM*) za vizualnu potvrdu cilja te ultrazvučni senzor *HC-SR04* na pokretnoj platformi. Na ovom prikazu jasno se vidi pogon s dva pogonska kotača.

Konstrukcija mobilnog robota sastoji se od dviju nosivih ploha spojenih vijcima, čime je ostvarena stabilnost i čvrstoća cijele izvedbe. U prostoru između donje i gornje plohe smješteni su servo motori za pokretanje kotača, čime je postignut kompaktan raspored i zaštićena osovina motora od vanjskih oštećenja.

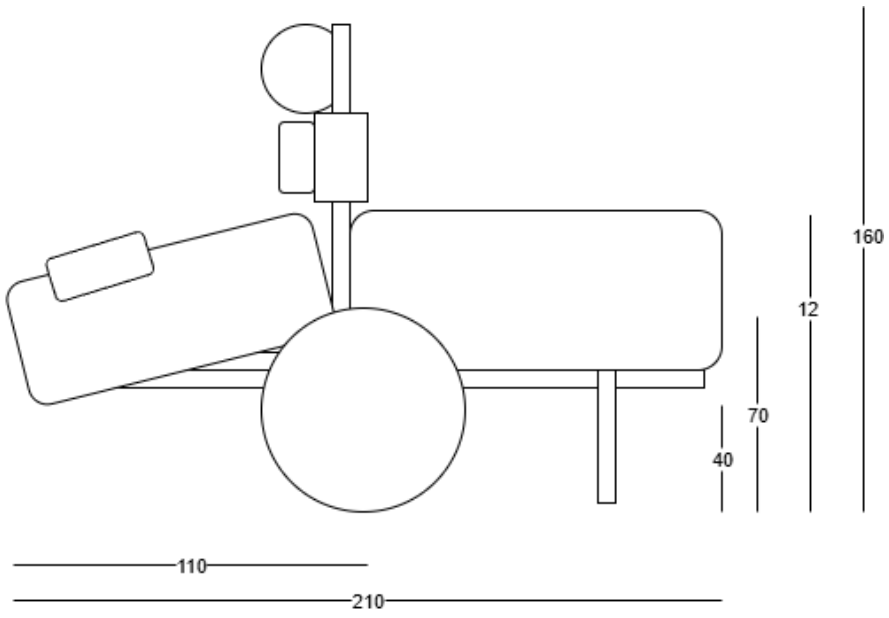
Na gornjoj plohi konstrukcije nalaze se svi ostali elektronički moduli. Njihova montaža izvedena je korištenjem vrućeg ljepila, čime je postignuta stabilnost pri kretanju robota. Kao dodatna mehanička zaštita i osiguranje, pojedini moduli učvršćeni su izolacijskom trakom koja omogućava povezivanje u funkcionalnu cjelinu bez opasnosti od pomicanja ili oštećenja tijekom rada.

Za *PIR* senzor korištena je dodatna nadogradnja u obliku aluminijske folije s malim otvorom u sredini. Na taj način senzor detektira toplinsko zračenje samo iz smjera interesa, čime se smanjuje mogućnost lažnih detekcija iz drugih kutova.

Prednji dio konstrukcije pokriven je kartonskim pokrovom izrađenim po mjeri, obojenim u crnu boju radi estetskog dojma i dodatne zaštite komponenti. Kamera je smještena u prednjem dijelu konstrukcije radi optimalnog vidnog polja i točne percepcije prostora.



Slika 3.9. *Pogled sprijeda konstrukcije mobilnog robota s naznačenim dimenzijama u mm*



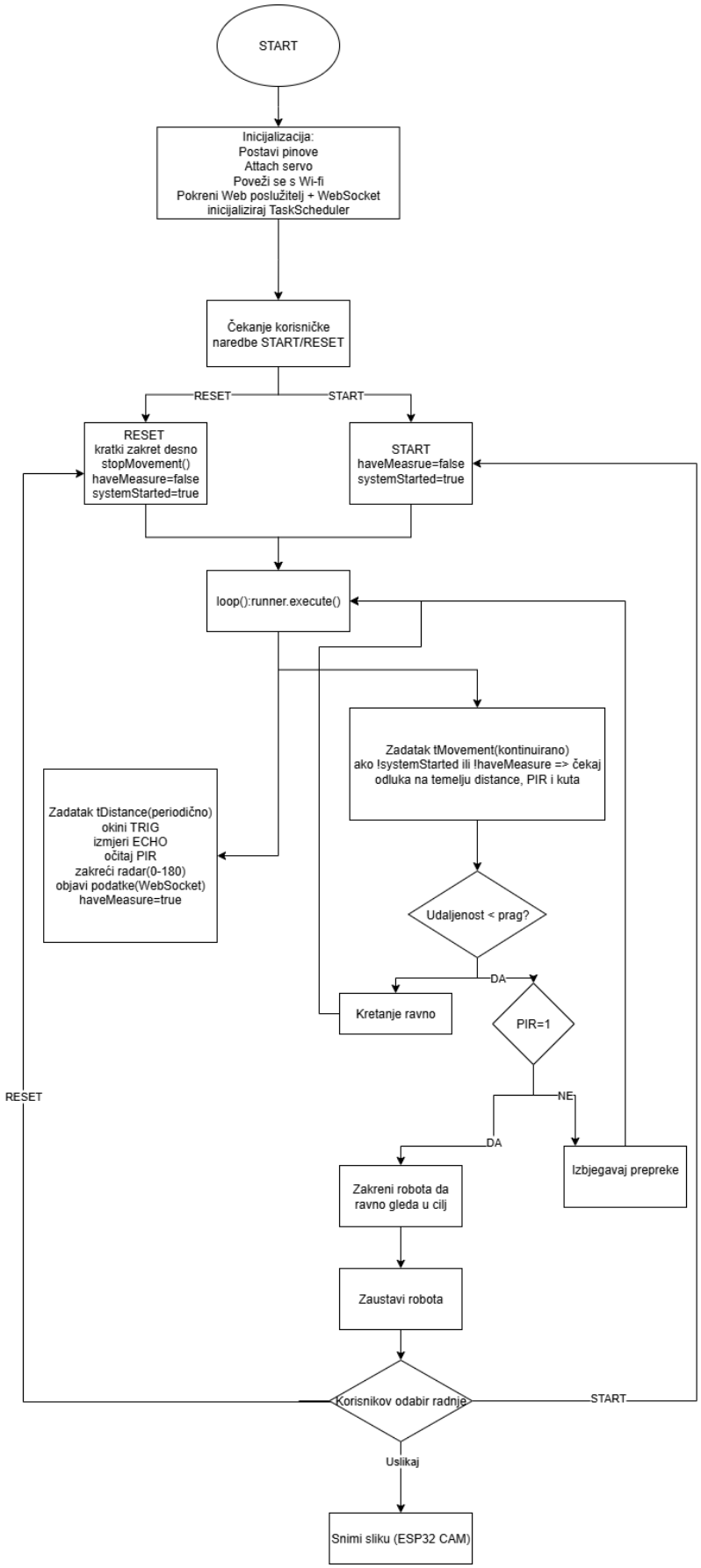
Slika 3.10. *Pogled s boka konstrukcije mobilnog robota s naznačenim dimenzijama u mm*

Prikazani su tehnički nacrti mobilnog robota gledani sprijeda i s boka. Na nacrtima su ucrtane osnovne dimenzije konstrukcije (duljina, širina i visina), kao i raspored ključnih komponenti sustava. Iz pogleda sprijeda vidljiv je položaj kotača, gornjeg pokrova, ultrazvučnog senzora, *PIR* senzora i kamere, dok bočni pogled prikazuje raspored konstrukcijskih ploča, položaj

servo motora, nosače senzora te ukupnu visinu sustava. Nacrti odgovaraju stvarnom fizičkom izgledu sustava i služe za jasniji prikaz konstrukcijskog rješenja.

3.3. Realizacija programskog rješenja

Programsko rješenje razvijeno je u razvojnom okruženju *Arduino IDE* za *ESP32* platformu. Logika rada temelji se na beskonačnoj petlji koja upravlja ponašanjem robota u stvarnom vremenu na osnovu podataka prikupljenih sa senzora i naredbi korisnika. Sustav je organiziran u funkcionalne cjeline: inicijalizaciju, obradu senzorskih podataka, donošenje odluka, upravljanje aktuatorima te komunikaciju s korisnikom putem *web*-poslužitelja.

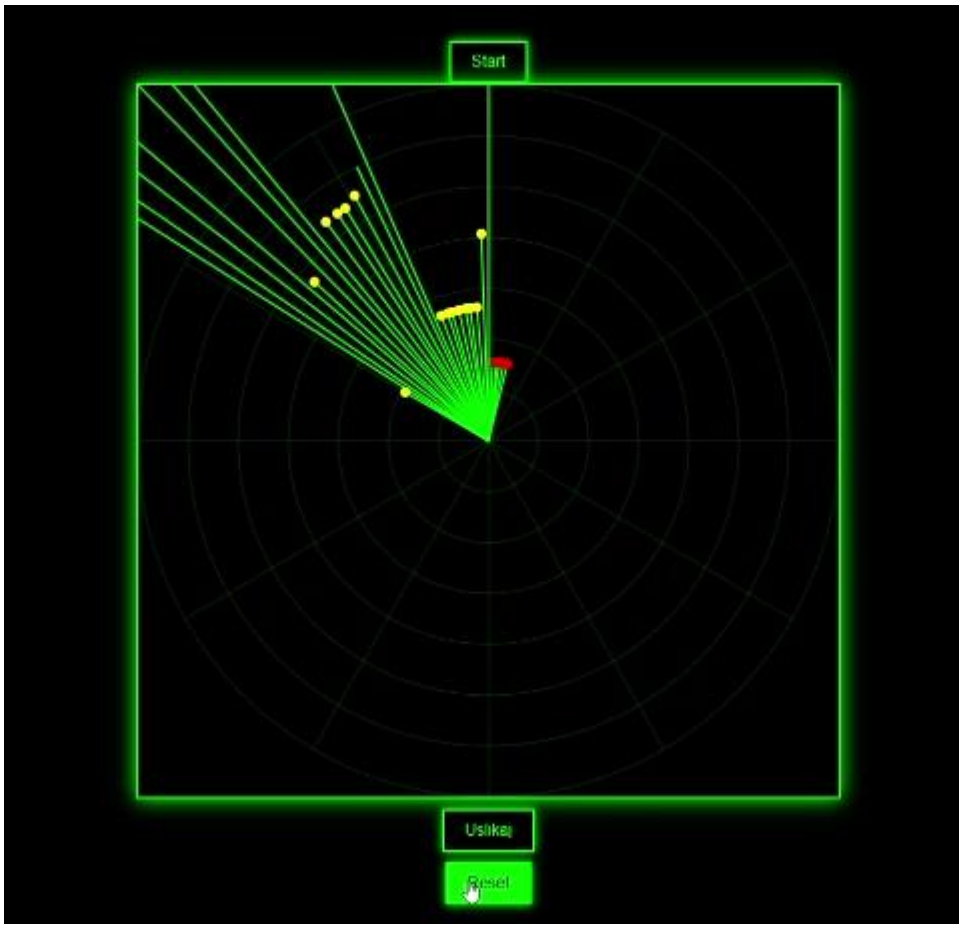


***Slika 3.11**. Detaljni blok dijagram toka programskog rješenja mobilnog robota*

Programski kod implementira logiku autonomnog kretanja mobilnog robota, obradu podataka sa senzora i komunikaciju s korisnikom putem *web* sučelja. Na samom početku rada, u funkciji *setup*(), pokreće se serijska komunikacija i ispisuje razlog posljednjeg reseta mikrokontrolera, čime se olakšava dijagnostika sustava. Zatim se rezerviraju *PWM* timere i za sve servo motore postavlja frekvencija rada od 50 Hz. U ovom koraku inicijaliziraju se servo motori povezani na tri različita pina: lijevi kotač (*leftWheelPin*=14), desni kotač (*rightWheelPin*=27) i radar servo (*radarServoPin*=19). Paralelno se definira i smjer rada pinova koji pripadaju senzorima – *TRIG* pin ultrazvučnog senzora (*trigPin*=25) konfigurira se kao izlazni, dok su *ECHO* pin ultrazvučnog senzora (*echoPin*=33) i izlaz *PIR* senzora (*pirPin*=32) postavljeni kao ulazni. Budući da je *ECHO* pin na *ESP32* ulaznog tipa i radi s logikom 3.3 V, u kodu se eksplicitno naglašava da je potrebno smanjiti signal s 5 V na prihvatljivu razinu.

Nakon inicijalizacije pinova uspostavlja se *Wi-Fi* veza. Uređaj se povezuje s mrežom pomoću funkcije *WiFi.begin*, a nakon uspješnog spajanja ispisuje se lokalna *IP* adresa. Time se otvara mogućnost mrežne komunikacije između korisnika i robota. Za dvosmjernu razmjenu podataka koristi se *WebSocket* protokol, konfiguriran putem objekta *AsyncWebSocket* ws("/ws"). U okviru *WebSocket* događaja definirano je ponašanje sustava prilikom primitka poruka od korisnika. Ako se primi poruka "reset", robot se kratko zakrene udesno, zaustavi se, a sustav se pripremi za novo mjerenje postavljanjem varijabli *haveMeasure=false* i *systemStarted=true*. Ako se, pak, primi poruka "start", robot započinje rad s novim mjerenjem, pri čemu se također postavljaju vrijednosti *haveMeasure=false* i *systemStarted=true,* čime se osigurava da robot ne reagira na zastarjele podatke prije nego što stigne prvo očitanje senzora.

Web poslužitelj (*AsyncWebServer*) dodatno omogućuje korisniku pristup jednostavnom *HTML* sučelju u *web* pregledniku. U ovom sučelju nalaze se gumbi za pokretanje, resetiranje i snimanje slike, a vizualizacija radarskih mjerenja prikazuje se u stvarnom vremenu. Na kraju inicijalizacije dodaju se i dvije glavne zadaće u raspored izvršavanja pomoću biblioteke *TaskScheduler* – zadatak *tDistance*, koji je zadužen za očitanje senzora i prikupljanje podataka, te zadatak *tMovement*, koji upravlja logikom kretanja. Oba zadatka se aktiviraju odmah na početku, a njihovo izvršavanje koordinira *runner.execute*() u beskonačnoj petlji (*loop*()).



Slika 3.12. *Web sučelje robota s prikazom radarskih mjerenja*

Korisničko sučelje dodatno sadrži i grafički prikaz radarskih mjerenja u stvarnom vremenu. Zelene linije prikazuju smjer skeniranja ultrazvučnog senzora, dok se detektirani objekti označavaju različitim bojama:

- crvena – prepreka detektirana ultrazvučnim senzorom,
- žuta – objekt istovremeno detektiran ultrazvučnim i *PIR* senzorom (potencijalno živo biće).

Na taj način korisnik dobiva jasan vizualni uvid u stanje okoline robota. Uz prikaz, i dalje su dostupni gumbi *Start*, *Reset* i Uslikaj za kontrolu sustava.

Zadatak *measureDistance*() periodički generira ultrazvučni impuls, pri čemu se *TRIG* pin aktivira logičkom jedinicom u trajanju od 10 mikrosekundi, a zatim deaktivira. Vrijeme trajanja povratnog *ECHO* impulsa mjeri se funkcijom pulseIn, pri čemu se uvodi maksimalni timeout od 25 ms. Ako je mjerenje uspješno, trajanje impulsa pretvara se u udaljenost u centimetrima, dok se istovremeno čita i status *PIR* senzora. U tom slučaju varijabla *haveMeasure* postavlja se na true, čime se omogućava izvršavanje logike kretanja. Ako mjerenje nije uspjelo, *haveMeasure* se postavlja na false, čime se sprječava reakcija sustava na nevaljane podatke. Uz mjerenje

udaljenosti, zadatak obavlja i radarsko skeniranje prostora. Servo motor koji pokreće ultrazvučni senzor zakreće se postupno između 0 i 180 stupnjeva, čime se omogućava kontinuirano skeniranje prostora ispred robota. Svakih nekoliko stupnjeva rada (točnije pri svakom trećem stupnju) svim povezanim klijentima putem *WebSocket*-a šalju se podaci o trenutnom kutu, izmjerenoj udaljenosti i statusu *PIR* senzora. Na ovaj način korisnik u *web* sučelju dobiva ažurnu radarsku sliku prostora.

Drugi zadatak, *movementLogic*(), implementira stroj stanja koji definira ponašanje robota tijekom kretanja. Sustav razlikuje tri osnovna stanja: mirovanje (*IDLE*), skretanje (*TURNING*) i kretanje naprijed (*MOVING_FORWARD*). Ako je robot u fazi skretanja, provjerava se je li prošlo predviđeno vrijeme zakreta (*TURN_TIME*=300 ms). Nakon što vrijeme istekne, robot se kratko zaustavlja, ponovno pokreće naprijed i prelazi u stanje kretanja ravno. Dodatno, postavlja se zaštitno vrijeme (*FWD_LOCK_MS*=400 ms) tijekom kojeg se ignoriraju nova očitanja senzora kako bi se izbjegla osciliranja nakon skretanja.

Kada robot identificira metu, logika kombinira očitanja ultrazvučnog i *PIR* senzora. Ako *PIR* senzor signalizira prisutnost izvora topline i ako je udaljenost manja od 50 cm, robot se postupno usmjerava prema cilju. Ako je cilj detektiran pod manjim kutom (<80°), robot se zakreće udesno; ako je kut veći od 100°, zakreće se ulijevo, a u međuvremenu nastavlja ravno kada se nalazi u prihvatljivom rasponu. Kada udaljenost do cilja padne ispod 20 cm, robot se zaustavlja, šalje se posljednja poruka s očitanim kutem i udaljenošću, a varijabla systemStarted postavlja se na false. Time se sprječava daljnje kretanje i robot prelazi u stanje čekanja korisničke naredbe. Korisnik tada može odabrati novu akciju – resetiranje sustava, ponovno pokretanje ili snimanje slike pomoću *ESP32-CAM* modula.

Ako *PIR* senzor ne detektira toplinu, ali se ispred robota nalazi prepreka unutar 20 cm, robot provodi jednostavnu logiku izbjegavanja – zakreće se udesno na unaprijed određeno vrijeme, prelazi u stanje *TURNING*, te nakon isteka vremena nastavlja kretanje naprijed. U slučaju kada ispred nema prepreka, a robot nije u stanju *MOVING_FORWARD*, poziva se funkcija *moveForwards*(), čime se održava osnovna logika kretanja naprijed.

Korisničko sučelje omogućuje jednostavno i intuitivno upravljanje sustavom. Putem *web* stranice dostupna su tri osnovna gumba: *Start*, *Reset* i Uslikaj. Pritiskom na gumbe *Start* i *Reset* šalju se poruke "start" i "reset" preko *WebSocket*-a, čime se izravno upravlja stanjem sustava. Gumb Uslikaj otvara *IP* adresu *ESP32-CAM* modula, omogućujući korisniku snimanje i pregled slike cilja u trenutku kada je robot zaustavljen. Posebno je važno naglasiti da nakon zaustavljanja

zbog detekcije mete, robot ne nastavlja s kretanjem sve dok korisnik izričito ne pošalje novu naredbu. Time se postiže sigurnost i predvidljivost rada sustava.

Cjelokupno programsko rješenje osigurava paralelno izvršavanje mjerenja i logike kretanja, pri čemu se koriste biblioteke *TaskScheduler* i *ESP32Servo*. Varijabla *systemStarted* omogućava ili blokira automatizirani rad, dok *haveMeasure* signalizira ispravnost očitanja senzora. Konačno, pragovi od 50 cm i 20 cm koriste se za definiranje prilaska cilju i konačnog zaustavljanja, dok stanja stroja kretanja osiguravaju glatko i stabilno gibanje. Izvorni kod s potpunim komentarima priložen je u zasebnom odjeljku dokumentacije.

4. TESTIRANJE I REZULTATI

U ovom poglavlju prikazani su postupci i nalazi ispitivanja funkcionalnosti robota, s naglaskom na pouzdanost senzora, stabilnost komunikacije i ponašanje pri kretanju. Testiranja su provedena u kontroliranim uvjetima, na ravnoj podlozi i uz stabilizirano napajanje, kako bi se osigurala ponovljivost rezultata. Za svaki podsustav definirani su mjerljivi kriteriji (točnost, latencija, broj lažnih detekcija, stabilnost gibanja) te je provedeno više ponavljanja mjerenja. Rezultati su prikazani tablično i grafički uz kratko tumačenje, a na kraju su istaknute ključne prednosti i uočena ograničenja sustava.

4.1. Metodologija testiranja

Testiranje sustava provedeno je s ciljem potvrde funkcionalnosti pojedinih komponenti, ispravnosti interakcije između modula te provjere cjelokupnog ponašanja robota u realnim uvjetima rada. S obzirom na to da sustav sadrži više međusobno povezanih funkcionalnih cjelina, testiranje je provedeno postupno, modularno i u kontroliranim uvjetima.

Prvo je provjeren rad mikrokontrolera i komunikacija preko *Wi-Fi* mreže. Povezivanje s lokalnom mrežom testirano je putem serijskog monitora, dok je ispravnost *web* sučelja i WebSocket komunikacije ispitivana iz preglednika (Chrome/Firefox), gdje se pratilo ažuriranje podataka u stvarnom vremenu.

Nakon toga, testiran je ultrazvučni senzor (HC-SR04) u kombinaciji s rotirajućim servo motorom. Testiranja su uključivala mjerenje udaljenosti do prepreka na različitim udaljenostima (10 cm – 100 cm), uz bilježenje stabilnosti i točnosti očitanja. Paralelno se provjeravala i mogućnost radarskog skeniranja pomoću rotirajućeg mehanizma te prikaz očitanja u *web* sučelju.

Zasebno se testirao PIR senzor (HC-SR501) u uvjetima prisutnosti i odsutnosti toplinskog izvora (npr. ruka, lice, grijač). Cilj je bio provjeriti pouzdanost otkrivanja toplinskog zračenja i brzinu reakcije senzora nakon detekcije udaljenosti. Testiranje se odvijalo u prostoriji sobne temperature, uz minimalno ambijentalno kretanje kako bi se izbjegli lažni pozitivni signali.

Segment kretanja robota testiran je na ravnoj, čvrstoj površini dimenzija otprilike 2x2 metra. Promatrano je ponašanje pri različitim vrstama prepreka (statične kutije, ruka, osoba), uključujući:

- ispravno zaustavljanje pri detekciji osobe,
- pokretanje kamere,
- reakciju na „neživu“ prepreku (izbjegavanje),

- oporavak iz situacija zaglavljenosti.

Za svaki slučaj vođena je evidencija stvarnih reakcija robota i uspoređivana s očekivanim ponašanjem prema algoritmu. Također, kamera (*ESP32-CAM*) testirana je u trenucima kada PIR detektira toplinu, s ciljem potvrde da korisnik može uspješno dobiti vizualni uvid u situaciju.

Na kraju, sustav je testiran u kontinuiranom radu tijekom 10 minuta, bez prekida napajanja, kako bi se detektirali eventualni problemi s memorijom, kašnjenjem, stabilnošću senzora i komunikacije.

Poseban izazov tijekom testiranja pokazao se PIR senzor (HC-SR501). Iako je senzor načelno funkcionirao prema očekivanjima, uočena je sklonost povremenim lažno pozitivnim detekcijama, osobito u uvjetima promjena temperature u prostoriji ili u prisutnosti reflektirajućih površina. Tijekom razvoja i testiranja uloženo je mnogo vremena i truda kako bi se pronašlo optimalno rješenje za filtriranje ovakvih pogrešnih signala, no unatoč različitim pokušajima (promjena pozicije, podešavanje osjetljivosti, vremenski filteri), nije pronađeno pouzdano rješenje koje bi u potpunosti eliminiralo ovaj problem.

Zbog toga je važno napomenuti da i dalje postoji značajna mogućnost lažno pozitivnih detekcija toplinskih izvora, što može rezultirati neželjenim zaustavljanjem robota i aktivacijom kamere u situacijama kada u stvarnosti nije prisutna osoba ili drugo živo biće.

Cilj cijelog testiranja bio je provjeriti:

- da svi senzori daju vjerodostojna i ponovljiva očitanja,
- da je logika odlučivanja konzistentna,
- da je krajnje ponašanje robota u skladu s očekivanim scenarijima.

Svi testovi provedeni su u zatvorenom prostoru, bez značajnih vanjskih utjecaja, uz napajanje preko stabiliziranog izvora napona (naponski adapter 5V/2A). Time su stvoreni uvjeti za vjerodostojno i ponovljivo ispitivanje, čime se osigurava da drugi korisnici mogu provesti jednako testiranje i usporediti rezultate.

4.2. Rezultati testiranja

Tijekom testiranja sustava bilježeni su rezultati koji se odnose na ispravnost rada pojedinih komponenti, odziv na vanjske podražaje te cjelokupno ponašanje robota u stvarnim uvjetima.

Rezultati su analizirani kroz više scenarija, a evaluacija se temeljila na ponovljivosti i pouzdanosti reakcija robota u zadanim situacijama.

Mjerenja udaljenosti sa ultrazvučnim senzorom u većini slučajeva pokazivala su dobru točnost u rasponu od 5 cm do 50 cm, s odstupanjem manjim od ±10 cm do 30 cm. Međutim, povremeno su se javljali slučajevi u kojima senzor nije vraćao vjerodostojne podatke. Ipak, u prosjeku se ponašao stabilno. Rotacija senzora omogućila je radarsko skeniranje prostora ispred robota, a svi kutni i podaci o udaljenosti su se uspješno slali putem *WebSocket*-a — dok je protok bio stabilan.

Najveći broj problema tijekom testiranja uzrokovao je PIR senzor. Iako u idealnim uvjetima ispravno detektira prisutnost topline, vrlo često daje lažno pozitivne rezultate, posebno u prostorijama s promjenama temperature ili refleksijama. Unatoč velikom trudu, podešavanju potenciometara i testiranju raznih pozicija, dodavanjem aluminijske zaštite kako bi se smanjio kut detekcije, nije pronađeno rješenje koje bi osiguralo potpunu pouzdanost detekcije. Zbog toga postoji ozbiljna šansa da robot zaustavi rad bez stvarnog razloga.

Sam mehanički sustav kretanja pokazao se stabilnim, ali je bilo teško i osjetljivo kalibrirati servo motore da bi se robot kretao ravno i precizno. Svako odstupanje u brzini motora moglo je uzrokovati skretanje s pravca. Ipak, konačno rješenje omogućava relativno stabilno gibanje s podrškom za osnovne manevre: naprijed, nazad, skretanje i zaokret u mjestu. Logika izbjegavanja prepreka i oporavka kod zaglavljenosti funkcionira kako je predviđeno.

Web sučelje generalno funkcionira, ali se *WebSocket* kanal pokazao kao najproblematičniji segment sustava. U trenutku kada se šalje previše podataka (npr. kada radar brzo skenira prostor i šalje velike količine informacija), *WebSocket* se vrlo lako preoptereti, što dovodi do gubitka podataka i izostanka točnog mapiranja prostora u *web* pregledniku. Ovaj nedostatak utječe na korisničko iskustvo i smanjuje upotrebljivost vizualizacije u stvarnom vremenu.

Sustav je pokazao funkcionalnost i sposobnost obavljanja osnovnih zadataka u kontroliranim uvjetima. Integracija svih komponenti u jednu radnu cjelinu uspješno je ostvarena. No, sustav još nije u potpunosti robustan: lažno pozitivni signali PIR senzora, osjetljivost kalibracije motora, te posebno nestabilnost *WebSocket* komunikacije predstavljaju glavne nedostatke. Unatoč tome, projekt u cjelini ispunjava definirane ciljeve i predstavlja dobar temelj za daljnju nadogradnju i optimizaciju.

5. ZAKLJUČAK

Projekt mobilnog robota temeljenog na *ESP32* platformi uspješno je realiziran u skladu s definiranim ciljevima i funkcionalnostima. Glavna ideja bila je izraditi autonomni sustav sposoban za detekciju i identifikaciju objekata u okolini korištenjem ultrazvučnog senzora, PIR senzora i kamere, uz osnovnu mogućnost samostalnog kretanja i interakcije s korisnikom putem bežičnog sučelja.

Razvoj je uključivao spajanje više različitih tehnoloških područja: elektronike, mikrokontrolera, obrade podataka sa senzora, upravljanja pokretom i *web* komunikacije u stvarnom vremenu. Sustav je funkcionalno integriran te uspješno reagira na prepreke, razlikuje objekte na temelju toplinskog zračenja i omogućuje korisniku uvid u situaciju putem slike s kamere. Rad motora, senzora i *web* sučelja koordiniran je putem višezadaćnog programiranja korištenjem TaskScheduler biblioteke.

Tijekom testiranja, sustav je pokazao stabilnost u većini funkcionalnosti. Međutim, identificirani su i određeni nedostaci, osobito vezani uz pouzdanost PIR senzora, koji je pokazivao sklonost lažno pozitivnim detekcijama, te ograničenja *WebSocket* komunikacije pri prijenosu većih količina podataka. Dodatno, mehanička kalibracija motora zahtijevala je značajan trud kako bi se postiglo skladno kretanje.

Unatoč navedenim ograničenjima, projekt ispunjava sve temeljne funkcijske zahtjeve i predstavlja dobar primjer integracije osnovnih komponenti u autonomni mobilni sustav. Nadalje, ovaj sustav može poslužiti kao početna točka za daljnji razvoj, kao što su dodavanje naprednijih algoritama za prepoznavanje objekata, korištenje pouzdanijih senzora za detekciju prisutnosti te optimizacija komunikacijskih mehanizama.

LITERATURA

[1] Acrome. *ESP32 Mobile Robot Teleoperation with Acrome SMD Red: Wi-Fi Control and Real-Time Video Streaming*. Acrome.net, objavljeno 02.07.2025. Dostupno na: https://acrome.net/post/esp32-mobile-robot-teleoperation-with-acrome-smd-red-wi-fi-control-and-real-time-video-streaming, pristup: 19.09.2025.

[2] Dev-Bhandari. *ESP32 Radar System (GitHub repository)*. Dostupno na: https://github.com/Dev-Bhandari/ESP32-Radar-System, pristup: 19.09.2025.

[3] Cirkit Designer. *ESP32-Controlled Obstacle Avoidance Robot with Ultrasonic Sensor and Servo Motor*. Dostupno na: https://docs.cirkitdesigner.com/project/published/d2baf5a1-3db3-41f0-bdcc-253318b8bb7d/esp32-controlled-obstacle-avoidance-robot-with-ultrasonic-sensor-and-servo-motor, pristup: 19.09.2025.

[4] *STM32 with HC-SR04 Ultrasonic Sensor*. The Embedded Things, Hackster.io. Dostupno na: https://www.hackster.io/theembeddedthings/stm32-with-hc-sr04-ultrasonic-sensor-c58329 pristup: 20. 9. 2025.

[5] *Hardware – IOT notes by Parita – WordPress.com.* Dostupno na: https://iotnotesbyparita.wordpress.com/hardware/ pristup: 20. 9. 2025.

[6] Espressif Systems. *ESP32 Technical Reference Manual.* Dostupno na: https://www.espressif.com, pristup: 19.09.2025.

[7] ESP32-CAM docs. Dostupno na: https://lastminuteengineers.com/getting-started-with-esp32-cam/, pristup: 19.09.2025.

[8] *Getting-Started With ESP32-CAM.* OceanLabz. Dostupno na: https://www.oceanlabz.in/getting-started-with-esp32-cam/ pristup: 20. 9. 2025.

[9] Ultrasonic sensor with arduino complete guide. Dostupno na: https://projecthub.arduino.cc/lucasfernando/ultrasonic-sensor-with-arduino-complete-guide-284faf, pristup: 20.09.2025

[10] *Ultrasonic Sensor Tutorial.* EmbetronicX. Dostupno na: https://embetronicx.com/tutorials/tech_devices/ultrasonic_sensor/ pristup: 20. 9. 2025.

[11] Components101. *HC-SR501 PIR Sensor Documentation.* Dostupno na: https://components101.com/sensors/hc-sr501-pir-sensor, pristup: 19.09.2025

[12] *MG90S 9g Metal-Gear Micro Servo Motor.* AliExpress. Dostupno na: https://www.aliexpress.com/item/1005007592969369.html (pristupljeno: 20. 9. 2025.)

[13] Arduino.cc. *Obstacle Avoiding Robot – Arduino Project Hub.* Dostupno na: https://create.arduino.cc/projecthub, pristup: 19.09.2025.

[14] GitHub. *ESPAsyncWebServer dokumentacija.* Dostupno na: https://github.com/me-no-dev/ESPAsyncWebServer, pristup: 19.09.2025.

[15] GitHub. *TaskScheduler biblioteka za Arduino.* Dostupno na: https://github.com/arkhipenko/TaskScheduler, pristup: 19.09.2025.

[16] FERIT. *Upute za pisanje projektne dokumentacije,* verzija 1.3, 09/2025.

PRILOZI I DODACI

```
// === LIBRARIES ===
//
#include <TaskScheduler.h>
#include <ESP32Servo.h>
#include <WiFi.h>
#include <AsyncTCP.h>
#include <ESPAsyncWebServer.h>
#include <esp_system.h>

//
// === WIFI CONFIG ===
//  (Maskirati vjerodajnice u objavljenoj verziji)
//
const char* ssid = "<SSID>";
const char* password = "<PASSWORD>";
bool systemStarted = false;   // Kapija: automatika aktivna/stop

//
// === SERVER / SCHEDULER ===
//
AsyncWebServer server(80);
AsyncWebSocket ws("/ws");     // WebSocket za dvosmjernu poruku "kut,udaljenost,PIR" i komande
Scheduler runner;             // TaskScheduler za kooperativno izvršavanje

//
// === PINS (ESP32) ===
// Mapiranje pinova (ESP32 DevKit v1):
// - Kotači (kontinuirani servoi, 50 Hz): lijevi=GPIO14, desni=GPIO27
// - Radar servo (0–180°): GPIO19
// - HC-SR04: TRIG=GPIO25 (OUTPUT), ECHO=GPIO33 (INPUT-ONLY, 3V3 tolerant)
// - PIR (HC-SR501): GPIO32 (INPUT-ONLY)
//
const int leftWheelPin  = 14;
const int rightWheelPin = 27;
```

```
const int radarServoPin = 19;

const int trigPin = 25;
const int echoPin = 33; // INPUT-ONLY (spustiti 5 V ECHO na 3V3!)
const int pirPin  = 32; // INPUT-ONLY

//
// === SERVOS ===
//
Servo leftWheel, rightWheel, radar;

//
// === STATE & PRAGOVI ===
//
bool  motionDetected = false;  // PIR status
float distanceCm     = 0;      // zadnje izmjerena udaljenost (cm)
int   radarAngle     = 0;      // 0..180 (servo kut)
int   actualAngle    = 0;      // 10..170 (za UI prikaz)
int   radarDir       = 1;      // +1 ili -1 (smjer rada skeniranja)

// mjerenje spremno? (štiti logiku od reakcije bez valjanih podataka)
bool  haveMeasure = false;

// TURNING & post-turn lock (sprječava "drhtanje" odmah nakon skretanja)
unsigned long turnStartTime = 0;
const unsigned long TURN_TIME   = 300;  // ms trajanje skretanja (burst)
const unsigned long FWD_LOCK_MS = 400;  // ms forsiraj ravno nakon skretanja
unsigned long afterTurnForwardUntil = 0;

//
// === HTML UI ===
// Ugrađeno sučelje: radar canvas + gumbi (Start/Reset/Uslikaj).
// "Uslikaj" otvara ESP32-CAM IP (po potrebi promijeniti IP adresu).
//
const char index_html[] PROGMEM = R"HTML(
```

```
<!DOCTYPE html><html><head><meta charset="UTF-8"><title>Radar</title>

<meta name="viewport" content="width=device-width,initial-scale=1" />

<style>

 body{background:#000;margin:0;display:flex;flex-direction:column;align-items:center;justify-
content:center;min-height:100vh;font-family:Arial,Helvetica,sans-serif}

 canvas{background:#000;border:2px solid lime;box-shadow:0 0 20px lime;max-
width:92vw;height:auto}

 .row{display:flex;gap:10px;margin:10px 0}

 button{padding:10px 20px;background:#000;color:lime;border:2px solid lime;font-
size:16px;cursor:pointer;box-shadow:0 0 10px lime}

 button:hover{background:lime;color:#000}

</style></head><body>

<div class="row">

  <button id="start-btn">Start</button>

  <button id="get-still">Uslikaj</button>

  <button id="reset-btn">Reset</button>

</div>

<canvas id="radar" width="700" height="700"></canvas>

<script>

let canvas=document.getElementById("radar"),ctx=canvas.getContext("2d");

let trail=document.createElement("canvas"); trail.width=canvas.width;
trail.height=canvas.height;

let tctx=trail.getContext("2d");

let lastAngle=null, sweepDir=1, socket;


function connect(){

  socket=new WebSocket("ws://"+location.host+"/ws");

  socket.onmessage=(e)=>{

    let[a,d,m]=e.data.split(","),A=parseInt(a),D=parseInt(d),M=(m==="1");

    drawRadar(A,D,M);

  };

  socket.onclose=()=>setTimeout(connect,2000);

}

connect();


function drawGrid(){
```

```
  ctx.save(); ctx.translate(canvas.width/2,canvas.height/2);

  ctx.strokeStyle="rgba(0,255,0,0.2)"; ctx.lineWidth=1;

  for(let
r=50;r<=canvas.width/2;r+=50){ctx.beginPath();ctx.arc(0,0,r,0,2*Math.PI);ctx.stroke();}

  for(let i=0;i<360;i+=30){

    let rad=i*Math.PI/180, x=Math.cos(rad)*(canvas.width/2),
y=Math.sin(rad)*(canvas.height/2);

    ctx.beginPath();ctx.moveTo(0,0);ctx.lineTo(x,y);ctx.stroke();

  }

  ctx.restore();

}

function drawRadar(angle,dist,motion){

  if(lastAngle!==null){let nd=angle>lastAngle?1:-1; if(nd!==sweepDir){sweepDir=nd;
tctx.clearRect(0,0,canvas.width,canvas.height);}}

  lastAngle=angle;

  tctx.save(); tctx.translate(canvas.width/2,canvas.height/2); tctx.strokeStyle="lime";
tctx.lineWidth=2;

  let rad=-angle*Math.PI/180, scale=6, x=dist*scale*Math.cos(rad), y=dist*scale*Math.sin(rad);

  tctx.beginPath(); tctx.moveTo(0,0); tctx.lineTo(x,y); tctx.stroke();

  if(dist>0 && dist<50){

    tctx.fillStyle=motion?"yellow":"red";

    tctx.beginPath(); tctx.arc(x,y,5,0,2*Math.PI); tctx.fill();

  }

  tctx.restore();

  ctx.clearRect(0,0,canvas.width,canvas.height); ctx.drawImage(trail,0,0); drawGrid();

}

document.getElementById("get-still").onclick=()=>window.open("http://192.168.1.90/","_blank");
// TODO: prilagodi IP

document.getElementById("reset-
btn").onclick=()=>socket&&socket.readyState===1&&socket.send("reset");

document.getElementById("start-
btn").onclick=()=>socket&&socket.readyState===1&&socket.send("start");

</script></body></html>

)HTML";


//

// === TASKOVI (TaskScheduler) ===
```

```
// Dva zadatka u pozadini (kooperativno):
// - tDistance (~120 ms): TRIG/ECHO mjerenje, PIR očitanje, sweep radara, slanje WebSocket poruke "kut,udaljenost,PIR"
// - tMovement (~100 ms): stroj stanja kretanja (FWD/TURNING) + odluke prema distance/PIR/kutu
//
void measureDistance();
Task tDistance(120, TASK_FOREVER, &measureDistance);
void movementLogic();
Task tMovement(100, TASK_FOREVER, &movementLogic);

//
// === KRETANJE: pomoćne funkcije ===
// Servo write(°) kao "gas/kočnica" za kontinuirane servoe.
// Vrijednosti (70..110) su kalibrirane za tvoj model.
//
enum MovementState { IDLE, TURNING, MOVING_FORWARD };
MovementState movementState = IDLE;

void moveForwards(){ leftWheel.write(110); rightWheel.write(70);  Serial.println("[MOVE] FWD"); }
void moveBackwards(){leftWheel.write(70);  rightWheel.write(110); Serial.println("[MOVE] BACK");}
void moveLeft(){     leftWheel.write(75);  rightWheel.write(75);  Serial.println("[MOVE] LEFT");}
void moveRight(){    leftWheel.write(105); rightWheel.write(105); Serial.println("[MOVE] RIGHT");}
void stopMovement(){ leftWheel.write(90);  rightWheel.write(90);  Serial.println("[MOVE] STOP"); }

//
// === measureDistance() ===
// - Ako systemStarted=false → preskoči
// - Okidanje TRIG (10 µs), mjerenje ECHO s timeout=25 ms (~4.3 m)
// - Ako duration>0 → distanceCm = duration * 0.0343 / 2; PIR očitanje; haveMeasure=true
// - Sweep radara 0↔180°, zapis servo kuta; svakih 3° pošalji "kut,udaljenost,PIR"
//
void measureDistance() {
```

```
  if (!systemStarted) return;

  digitalWrite(trigPin, LOW); delayMicroseconds(2);
  digitalWrite(trigPin, HIGH); delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  long duration = pulseIn(echoPin, HIGH, 25000); // max 25 ms ~ 4.3 m

  if (duration > 0) {                 // valjano mjerenje
    haveMeasure   = true;
    distanceCm    = duration * 0.0343f / 2.0f;  // 0..~400 cm (idealno)
    motionDetected= digitalRead(pirPin);
  } else {
    haveMeasure = false;              // spriječi reakcije bez svježeg mjerenja
  }

  // radarski sweep 0⇔180°
  radarAngle += radarDir;
  if (radarAngle >= 180) { radarAngle = 180; radarDir = -1; }
  else if (radarAngle <= 0) { radarAngle = 0; radarDir = 1; }

  radar.write(radarAngle);
  actualAngle = map(radarAngle, 0, 180, 0, 160) + 10; // prikaz 10..170° u UI

  // throttling slanja prema klijentima (svaka ~3°)
  if (radarAngle % 3 == 0) {
    String msg = String(actualAngle) + "," + String((int)distanceCm) + "," +
String(motionDetected);
    ws.textAll(msg);
  }
}

//
// === movementLogic() ===
// - State machine: IDLE / TURNING / MOVING_FORWARD
```

```
// - TURNING traje TURN_TIME, zatim kratki lock FWD_LOCK_MS (izbjegava titranje)
// - Ako PIR=1 i distance<50 → poravnaj (L/D prema kutu), stop na <=20 cm i
//   postavi systemStarted=false (korisnička kapija: Uslikaj/Start/Reset)
// - Ako PIR=0 i vrlo blizu prepreka (<=20 cm) → jedan "burst" udesno (osnovno izbjegavanje)
// - Inače: default naprijed
//
void movementLogic() {
  if (!systemStarted) return;

  // Dok skrećemo – ništa drugo
  if (movementState == TURNING) {
    if (millis() - turnStartTime >= TURN_TIME) {
      stopMovement(); delay(100);
      moveForwards();
      movementState = MOVING_FORWARD;
      afterTurnForwardUntil = millis() + FWD_LOCK_MS; // kratko forsiraj ravno
    }
    return;
  }

  // Bez prvog mjerenja -> ne reagiraj (sprječava "desno" nakon start/reset)
  if (!haveMeasure) return;

  // Odmah nakon skretanja, ignoriraj prepreke kratko (lock)
  if (millis() < afterTurnForwardUntil) return;

  // META → prilazi i dotjeraj smjer
  if (motionDetected && distanceCm < 50) {
    if (distanceCm <= 20) {
      stopMovement();
      int sendAngle = map(radarAngle, 0, 180, 10, 170);
      String lastMsg = String(sendAngle) + "," + String((int)distanceCm) + ",1";
      ws.textAll(lastMsg);
      systemStarted = false;  // Kapija: UI (Uslikaj/Start/Reset)
      Serial.println("[LOGIC] Target reached → waiting for user");
```

```
      return;
    }
    if (radarAngle < 80)  { moveRight(); movementState = TURNING; turnStartTime = millis();
return; }
    if (radarAngle > 100) { moveLeft();  movementState = TURNING; turnStartTime = millis();
return; }
    moveForwards(); movementState = MOVING_FORWARD; return;
  }

  // PREPREKA blizu (bez PIR) → jedno skretanje desno
  if (!motionDetected && distanceCm > 0 && distanceCm <= 20) {
    moveRight(); movementState = TURNING; turnStartTime = millis();
    Serial.println("[LOGIC] Obstacle → right");
    return;
  }

  // DEFAULT → naprijed
  if (movementState != MOVING_FORWARD) {
    moveForwards();
    movementState = MOVING_FORWARD;
  }
}

//
// === setup() ===
// - Serijska dijagnostika + ispis reset razloga
// - PWM timere (4×) i servo period 50 Hz; attach servo pinova
// - PinMode za TRIG/ECHO/PIR
// - Wi-Fi spajanje, WebSocket onEvent (obrada "start"/"reset")
// - Pokretanje WebServera (HTML UI)
// - Inicijalizacija TaskScheduler-a i enable zadataka
//
void setup() {
  Serial.begin(115200);
  Serial.printf("[BOOT] Reset reason: %d\n", esp_reset_reason());
```

```
ESP32PWM::allocateTimer(0);
ESP32PWM::allocateTimer(1);
ESP32PWM::allocateTimer(2);
ESP32PWM::allocateTimer(3);
leftWheel.setPeriodHertz(50);
rightWheel.setPeriodHertz(50);
radar.setPeriodHertz(50);
leftWheel.attach(leftWheelPin);
rightWheel.attach(rightWheelPin);
radar.attach(radarServoPin);

pinMode(trigPin, OUTPUT);
pinMode(echoPin, INPUT);
pinMode(pirPin, INPUT);

WiFi.begin(ssid, password);
Serial.print("[WiFi] Connecting");
while (WiFi.status() != WL_CONNECTED) { delay(400); Serial.print("."); }
Serial.print("\n[WiFi] IP: "); Serial.println(WiFi.localIP());

// WebSocket event handler: "reset"/"start" komande
ws.onEvent([](AsyncWebSocket * server, AsyncWebSocketClient * client, AwsEventType type,
              void * arg, uint8_t * data, size_t len) {
  if (type == WS_EVT_CONNECT) {
    Serial.println("[WS] client connected");
  } else if (type == WS_EVT_DISCONNECT) {
    Serial.println("[WS] client disconnected");
  } else if (type == WS_EVT_DATA) {
    String msg; msg.reserve(len);
    for (size_t i=0;i<len;i++) msg += (char)data[i];
    Serial.printf("[WS] msg: %s\n", msg.c_str());
    if (msg == "reset") {
      moveRight(); delay(600); stopMovement();
      haveMeasure = false;            // čekaj prvo mjerenje poslije reseta
```

```
        systemStarted = true;
        Serial.println("[WS] reset done");
      } else if (msg == "start") {
        haveMeasure = false;            // spriječi desno prije prvog mjerenja
        systemStarted = true;
        Serial.println("[WS] start");
      }
    }
  });

  server.addHandler(&ws);
  server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send_P(200, "text/html", index_html);
  });
  server.begin();

  stopMovement();
  runner.init();
  runner.addTask(tDistance);  tDistance.enable();
  runner.addTask(tMovement);  tMovement.enable();
}

//
// === loop() ===
// Kooperativno izvršavanje zadataka (TaskScheduler).
//
void loop() {
  runner.execute();
}
```

***Kod 7.1**. Cjeloviti Arduino kod za ESP32*

```
// === BIBLIOTEKE ===
#include "esp_camera.h"      // ESP32-CAM biblioteka za upravljanje kamerom
#include <WiFi.h>            // Wi-Fi podrška
#include <WebServer.h>       // Jednostavni HTTP server
```

```
// === Wi-Fi PODACI ===
const char* ssid = "pp";          // SSID tvoje Wi-Fi mreže
const char* password = "dinajs123"; // lozinka tvoje Wi-Fi mreže

// === Web server ===
WebServer server(80);  // HTTP server na portu 80

// === Pin konfiguracija za AI-Thinker ESP32-CAM modul ===
#define CAMERA_MODEL_AI_THINKER
#if defined(CAMERA_MODEL_AI_THINKER)
#define PWDN_GPIO_NUM     32   // Power down pin
#define RESET_GPIO_NUM    -1   // Reset pin (nije korišten)
#define XCLK_GPIO_NUM      0   // XCLK – eksterni clock
#define SIOD_GPIO_NUM     26   // SDA za I2C
#define SIOC_GPIO_NUM     27   // SCL za I2C

// Data linije (kamera -> ESP32)
#define Y9_GPIO_NUM       35
#define Y8_GPIO_NUM       34
#define Y7_GPIO_NUM       39
#define Y6_GPIO_NUM       36
#define Y5_GPIO_NUM       21
#define Y4_GPIO_NUM       19
#define Y3_GPIO_NUM       18
#define Y2_GPIO_NUM        5

// Sinkronizacija
#define VSYNC_GPIO_NUM    25
#define HREF_GPIO_NUM     23
#define PCLK_GPIO_NUM     22
#endif

// === Funkcija za inicijalizaciju kamere ===
void setupCamera() {
```

```
camera_config_t config;
config.ledc_channel = LEDC_CHANNEL_0;   // PWM kanal za clock
config.ledc_timer   = LEDC_TIMER_0;     // Timer za clock

// Mapiranje pinova
config.pin_d0       = Y2_GPIO_NUM;
config.pin_d1       = Y3_GPIO_NUM;
config.pin_d2       = Y4_GPIO_NUM;
config.pin_d3       = Y5_GPIO_NUM;
config.pin_d4       = Y6_GPIO_NUM;
config.pin_d5       = Y7_GPIO_NUM;
config.pin_d6       = Y8_GPIO_NUM;
config.pin_d7       = Y9_GPIO_NUM;
config.pin_xclk     = XCLK_GPIO_NUM;
config.pin_pclk     = PCLK_GPIO_NUM;
config.pin_vsync    = VSYNC_GPIO_NUM;
config.pin_href     = HREF_GPIO_NUM;
config.pin_sscb_sda = SIOD_GPIO_NUM;
config.pin_sscb_scl = SIOC_GPIO_NUM;
config.pin_pwdn     = PWDN_GPIO_NUM;
config.pin_reset    = RESET_GPIO_NUM;

// Parametri kamere
config.xclk_freq_hz = 20000000;           // 20 MHz clock
config.pixel_format = PIXFORMAT_RGB565;   // RGB565 format (ne JPEG)

config.frame_size   = FRAMESIZE_VGA;      // 640x480 rezolucija
config.jpeg_quality = 12;                 // ne koristi se za RGB565
config.fb_count     = 2;                  // 2 frame buffera
config.grab_mode    = CAMERA_GRAB_LATEST; // uzmi najnoviji frame

// Inicijalizacija kamere
esp_err_t err = esp_camera_init(&config);
if (err != ESP_OK) {
  Serial.printf("Camera init failed: 0x%x", err);
```

```
    while (true) delay(1000); // Ako ne uspije, vječno čekaj
  }
}

// === HTTP handler za početnu stranicu ===
void handleRoot() {
  // HTML + JavaScript kod koji se šalje korisniku
  // Canvas služi za prikaz slike
  String html = R"rawliteral(
  <!DOCTYPE html>
  <html>
  <head>
    <title>ESP32-CAM Raw Image</title>
    <style>
      body { font-family: sans-serif; text-align: center; }
    </style>
  </head>
  <body>
    <h1>ESP32-CAM Raw Image (RGB565 - VGA)</h1>
    <canvas id="canvas" width="640" height="480"></canvas>
    <script>
      const canvas = document.getElementById('canvas');
      const ctx = canvas.getContext('2d');

      // Fetch zahtjev prema /capture endpointu
      fetch('/capture')
        .then(res => res.arrayBuffer())
        .then(buf => {
          const u8 = new Uint8Array(buf);
          const imageData = ctx.createImageData(640, 480);

          // Pretvaranje RGB565 piksela u RGBA format za canvas
          for (let i = 0; i < 640 * 480; i++) {
            const hi = u8[i * 2];
            const lo = u8[i * 2 + 1];
```

```
            const val = (hi << 8) | lo;

            const r = ((val >> 11) & 0x1F) << 3;
            const g = ((val >> 5) & 0x3F) << 2;
            const b = (val & 0x1F) << 3;

            const idx = i * 4;
            imageData.data[idx]     = r;
            imageData.data[idx + 1] = g;
            imageData.data[idx + 2] = b;
            imageData.data[idx + 3] = 255;
          }

          // Iscrtavanje slike
          ctx.putImageData(imageData, 0, 0);
        });
    </script>
  </body>
  </html>
  )rawliteral";

  server.send(200, "text/html", html);
}

// === HTTP handler za dohvat slike ===
void handleCapture() {
  camera_fb_t *fb = esp_camera_fb_get();   // Dohvati frame buffer
  if (!fb) {
    server.send(500, "text/plain", "Camera capture failed");
    return;
  }

  if (fb->format != PIXFORMAT_RGB565) {
    esp_camera_fb_return(fb);
    server.send(500, "text/plain", "Unexpected pixel format");
```

```
    return;
  }

  // Slanje slike klijentu kao binarni tok podataka
  WiFiClient client = server.client();
  client.println("HTTP/1.1 200 OK");
  client.println("Content-Type: application/octet-stream");
  client.print("Content-Length: ");
  client.println(fb->len);
  client.println("Connection: close");
  client.println();
  client.write(fb->buf, fb->len);

  // Oslobodi frame buffer
  esp_camera_fb_return(fb);
}

// === Setup funkcija ===
void setup() {
  Serial.begin(115200);
  setupCamera();  // Inicijalizacija kamere

  // Spajanje na Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println(WiFi.localIP()); // Ispis IP adrese

  // Definiraj HTTP rute
  server.on("/", handleRoot);
```

```
  server.on("/capture", handleCapture);
  server.begin();
}

// === Glavna petlja ===
void loop() {
  server.handleClient();  // Obrada HTTP zahtjeva
}
```

***Kod 7.2**. Cjeloviti kod ESP32-Cam modula*

Test	Mjerena udaljenost (mm)	Dohvaćena udaljenost (mm)
1	708	383
2	750	411
3	737	397
4	706	466
5	819	391
6	788	384
7	687	391
8	735	409
9	809	457
10	686	451
11	730	471
12	819	433
13	736	437
14	830	415
15	681	477
16	720	469
17	788	423
18	751	399

Test	Mjerena udaljenost (mm)	Dohvaćena udaljenost (mm)
19	735	477
20	766	393

Tablica 7.1. *Rezultati mjerenja ultrazvučnog senzora*

Test	Uvjet	Očekivani izlaz	Izmjereni izlaz
1	osoba prisutna	1	0
2	osoba prisutna	1	1
3	osobe nema	0	1
4	osobe nema	0	0
5	osobe nema	0	0
6	osobe nema	0	0
7	osobe nema	0	0
8	osobe nema	0	0
9	osoba prisutna	1	1
10	osoba prisutna	1	0
11	osobe nema	0	0
12	osoba prisutna	1	0
13	osobe nema	0	1
14	osobe nema	0	0
15	osobe nema	0	1
16	osobe nema	0	0
17	osobe nema	0	0
18	osoba prisutna	1	1
19	osoba prisutna	1	1
20	osoba prisutna	1	1

Tablica 7.2*. Rezultati mjerenja PIR senzora*