

Univerzitet u Tuzli
Fakultet elektrotehnike
Automatika i robotika
Projektovanje sistema na čipu

TEMA: WiFi upravljeni mobilni robot sa autonomnim sistemima za sigurnost

Profesorica:

dr.sc. Lejla Banjanović – Mehmedović,
vanredni prof.

Studenti:

Amar Kešetović
Dženana Sadiković
Dženita Moranjkić
Mahir Suljić

Januar, 2026.

SADRŽAJ

1. UVOD	1
2. OPIS PROJEKTNOG ZADATKA	2
2.1. ZADACI PROJEKTA	3
2.2. CILJEVI PROJEKTA	3
2.3. PRIMJENA	4
3. SPECIFIKACIJA MATERIJALA	4
4. OPISI KORIŠTENIH KOMPONENTI	7
4.1. FPGA CYCLONE II	7
4.2. RASPBERRY PI 4	8
4.3. ESP – 01S	9
4.4. ULAZNE KOMPONENTE	10
4.4.1. Ultrasonični senzor HC-SR04P	10
4.4.2. Kamera Logitech C512	10
4.4.3. Infracrveni sensor	11
4.5. IZLAZNE KOMPONENTE	11
4.5.1. DC motor	11
4.5.2. Aktivni piezo buzzer	12
4.6. DC MOTOR DRIVER L298N	12
5. UPRAVLJAČKA LOGIKA I SISTEMSKA INTEGRACIJA	13
5.2. ULAZNI I IZLAZNI SIGNALI ZA FPGA	15
5.3. ULAZNI I IZLAZNI SIGNALI ZA RASPBERRY PI	17
5.4. LOGIKA PRIORITETA KRETANJA	17
5.5. ZVUČNA SIGNALIZACIJA I SENZORI	18
5.6. MEHANIZMI KONTROLE I ZAŠTITE	18
6. 3D MODEL I FINALNI IZGLED PROJEKTA	19
7. MOBILNA ANDROID APLIKACIJA	21
8. KOD	23
8.1. VERILOG KOD ZA KONTROLU DC MOTORA NA FPGA	23
8.2. VERILOG KODA ZA IMPLEMENTACIJU KONTROLE ZVUČNE SIGNALIZACIJE PARKING SENZORA NA FPGA	25
8.3. VERILOG KOD ZA INTEGRICIU SENZORA I MOTORA	26
8.4. KOD NA RASPBERRY PI ZA DETEKCIJU SAOBRAĆAJNIH ZNAKOVA KORIŠTENJEM KAMERE	28
8.5. KOD ZA KOMUNIKACIJU IZMEĐU MOBILNE APLIKACIJE I RASPBERRY PI	35
8.6. KOD ZA SIMULACIJU RADA SEMAFORA	39
8.7. KOTLIN KOD ZA ANDROID MOBILNU APLIKACIJU	40
8.7.1. data/models/RobotStatus.kt	40
8.7.2. data/RobotRepository.kt	41
8.7.3. network/RetrofitClient.kt	43
8.7.4. network/Robot ApiService.kt	44
8.7.5. ui/components/ActionButton.kt	45
8.7.6. ui/components/ControlPad.kt	46
8.7.7. ui/screens/MainActivity.kt	48
8.7.8. ui/screens/RobotControlScreen.kt	49

8.7.9..viewmodel/RobotViewModel.kt	54
8.7.10. Ostala podešavanja za rad aplikacije	58
REFERENCE	60

POPIS TABELA

Tabela 3.1. - Upravljačke ploče	4
Tabela 3.2. – Ulazne komponente	5
Tabela 3.3. - Dodatne komponente	5
Tabela 3.4. - Izlazne komponente	6

POPIS SLIKA

Slika 1. - Šema povezivanja komponenti sa FPGA i Raspberry Pi	14
Slika 2. - Lista pinova za FPGA	15
Slika 3. – Raspoloživo pinova za FPGA	16
Slika 4. - 3D model robota izrađen u FreeCAD-u	19
Slika 5. - Finalni izgled projektnog zadatka	20
Slika 6. - Maketa znaka STOP	21
Slika 7. - Model semafora	21
Slika 8. - Glavni upravljački ekran	22
Slika 9. - Prikaz greške u konekciji	23
Slika 10. - Postavke mrežnih parametara	23

1. UVOD

Savremeni razvoj transportnih sistema i logistike sve se više oslanja na integraciju autonomnih vozila i pametnih robotskih platformi. Glavni izazovi u ovom domenu obuhvataju postizanje visoke preciznosti u upravljanju, osiguranje maksimalne sigurnosti putem sistema za asistenciju vozaču, te uspostavljanje pouzdane komunikacije u realnom vremenu. Cilj ovog projekta je razvoj WiFi upravljanog mobilnog robota sa autonomnim sistemima za sigurnost, koji kombinuje fleksibilnost ručnog upravljanja putem mobilne aplikacije sa naprednim funkcijama automatske prevencije nezgoda.

Sistem je zasnovan na sinergiji Raspberry Pi platforme, koja vrši obradu slike i AI detekciju saobraćajne signalizacije, te FPGA kontrolera, koji osigurava determinističko upravljanje motorima i obradu senzorskih podataka u realnom vremenu. Ovakva arhitektura omogućava implementaciju kompleksnih funkcija poput detekcije linija, automatskog kočenja pri prepoznavanju "STOP" znaka ili crvenog svjetla na semaforu, te inteligentnog sistema za parkiranje.

Primjena ovakvog sistema nudi širok spektar mogućnosti, od edukacijskih platformi za testiranje algoritama autonomne vožnje do primjene u pametnim skladištima gdje je potrebna visoka sigurnost pri kretanju. Integracija senzora za blizinu, infracrvenih senzora za praćenje trake i vizuelne povratne informacije putem (near) real-time video streama, direktno doprinosi minimizaciji ljudske greške i povećanju operativne sigurnosti.

Kroz realizaciju ovog projekta, demonstrira se kako se moderni komunikacioni protokoli, vještačka inteligencija i namjenska hardverska arhitektura mogu ujediniti u jedinstven, skalabilan sistem. Ovakva rješenja predstavljaju ključni korak ka modernizaciji transporta i razvoja inteligentnih mašina sposobnih da autonomno reaguju na nepredviđene situacije u okruženju, čime se postavljaju novi standardi u sigurnosti i efikasnosti mobilnih robotskih sistema.

2. OPIS PROJEKTNOG ZADATKA

Tema projekta je automobil koji se kontroliše putem mobilne aplikacije. Automobil posjeduje 4 točka, kameru, parking senzore, zvučnu signalizaciju, LED svjetla, te sposobnost detektovanja svjetla na semaforu, saobraćajnih znakova i prevenciju prelazka preko linije.

Komunikacija između Raspberry Pi i FPGA ploče odvija se putem digitalnih pinova. Svaki pin je zadužen za jednu od sljedećih akcija: kretanje naprijed, kretanje nazad, kretanje desno, kretanje lijevo, zvučni signal.

Mobilna aplikacija je napisana u Kotlin programskom jeziku koristeći Android Studio i Jetpack Compose. Kamera se nalazi na prednjoj strani automobila, a aplikacija omogućava (near) real-time prikaz onoga što kamera automobila vidi. Na jednoj strani ekrana nalaze se komande za kretanje automobila, dok je na drugoj strani komanda za zvučnu signalizaciju. Komunikacija između mobilne aplikacije i Raspberry Pi mikroračunara odvija se putem HTTP protokola.

FPGA preko drivera za motore (H-mosta) kontroliše motore pričvršćene na točkove. Jedan driver može pokretati dva motora, pa su u sistem integrisana dva drivera – jedan za lijevi i jedan za desni par točkova. Ovakav raspored omogućava korištenje diferencijalnog upravljanja, gdje se za skretanje udesno snaga lijevom paru točkova povećava, a desnom smanjuje, i analogno za skretanje ulijevo.

Kada kamera detektuje crveno svjetlo na semaforu ili znak stop, Raspberry Pi automatski šalje signal za kočenje. Ako ultrazvučni senzori detektuju približavanje prepreki, FPGA pomoću aktivnog piezo buzzera generiše zvuk, slično parking senzorima u automobile. Sa strana automobila nalaze se infracrveni senzori za detekciju linija. Ako senzor detektuje liniju, šalje se signal za blago skretanje u suprotnom smjeru kako bi se vozilo vratio unutar linija.

Odvojeno od glavnog projekta realizovan je semafor. Semafor se sastoji od tri LED (crvena, žuta i zelena) koje kontroliše ESP-01S. Sistem simulira ponašanje pravog semafora sa smjenama crvenog, žutog i zelenog svjetla.

2.1. ZADACI PROJEKTA

Za uspjesnu realizaciju projekta potrebno je ispuniti sljedeće zadatke:

1. Programiranje FPGA kontrolera
 - Implementacija modula za precizno upravljanje motorima.
 - Razvoj logike za obradu signala sa ultrazvučnih senzora (parking senzori) i infracrvenih senzora (praćenje linije).
 - Logička kontrola svjetlosne i zvučne signalizacije (LED i buzzer).
2. Programiranje na Raspberry Pi platformi
 - Implementacija HTTP servera u Pythonu za potrebe mrežne komunikacije i razmjene podataka sa kontrolnom aplikacijom.
 - Razvoj AI detektora za prepoznavanje boja na semaforu i "Stop" znakova koristeći kameru.
3. Razvoj kontrolne android aplikacije
 - Izrada korisničkog interfejsa u Kotlinu koji omogućava upravljanje robotom.
4. Realizacija eksternog semafora
 - Programiranje ESP-01S modula za kontrolu LED dioda koje simuliraju rad pravog semafora.
5. Hardverska realizacija vozila
 - Sklapanje vozila sa četiri točka i integracija DC motora sa H-most driverima.
 - Povezivanje Raspberry Pi mikroračunara i FPGA ploče putem GPIO pinova za potrebe komunikacije.
 - Povezivanje senzorskog sistema (ultrazvučni, infracrveni i kamera).
6. Izrada makete

2.2. CILJEVI PROJEKTA

Glavni cilj projekta je dizajn i implementacija mobilnog robota koji kombinuje daljinsko upravljanje putem WiFi mreže i autonomne sigurnosne sisteme. Fokus je na korištenju Raspberry Pi i FPGA kako bi se postigao visok stepen pouzdanosti. Specifični ciljevi uključuju:

- **Povećanje sigurnosti:** Implementacija sistema koji procesiranjem informacija sa senzora i kamere sprječava vozača da namjerno ili nenamjerno ošteti robota ili načini neku vrstu prekršaja.
- **Istraživanje mašinskog učenja:** Primjena AI modela za prepoznavanje vizuelnih simbola u realnom vremenu.
- **Determinizam upravljanja:** Korištenje FPGA za trenutnu reakciju na prepreke i korekciju putanja, čime se eliminišu kašnjenja softvera.

2.3. PRIMJENA

Projektni sistem ima širok potencijal primjene u modernim industrijskim područjima:

- **Autonomna transportna vozila:** U pametnim fabrikama i skladištima za siguran prevoz robe uz poštovanje interne signalizacije.
- **Sistemi asistencije vozaču:** Prototip rješenja za prevenciju sudara i automatsko kočenje u putničkim vozilima.
- **Edukacija i istraživanje:** Platforma za testiranje algoritama autonomnog kretanja i proučavanje komunikacije između različitih hardverskih nivoa.
- **Rad u opasnim okruženjima:** Daljinski upravljana vozila sa ugrađenim "instinktima" za samoočuvanje u slučaju gubitka veze sa operaterom.

3. SPECIFIKACIJA MATERIJALA

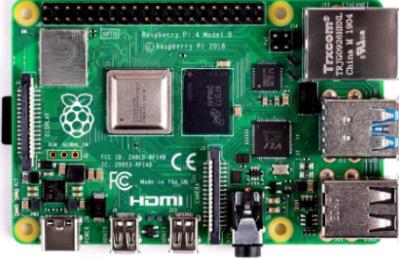
Upravljačke ploča	Slika
Raspberry Pi 4	
Altera Cyclone 2 (FPGA)	
ESP-01S	

Tabela 3.1. - Upravljačke ploče

Ulazi	Upravljačka ploča	Namjena	Slika
Ultrasonični senzor (HC-SR04P) x2	FPGA	Detekcija prepreka	
Kamera (Logitech C512)	Raspberry PI	Detekcija znakova i video stream	
Infracrveni senzor x2	FPGA	Detekcija linija	

Tabela 3.2. – Ulazne komponente

Dodatno	Upravljačka ploča	Namjena	Slika
DC motor driver L298N x2	FPGA	Kontrola motora	
LED semafor	ESP-01S	Svjetlosna signalizacija	

Tabela 3.3. - Dodatne komponente

Izlazi	Upravljačka ploča	Namjena	Slika
DC motori x4	FPGA	Kontrola kretanja	
Aktivni piezo buzzer	FPGA	Zvučna singnalizacija	

Tabela 3.4. - Izlazne komponente

4. OPISI KORIŠTENIH KOMPONENTI

4.1. FPGA CYCLONE II

FPGA (Field-Programmable Gate Array) je integrisano kolo koje korisnici mogu programirati nakon proizvodnje da bi realizovali specifične digitalne logičke funkcije. Cyclone II je serija FPGA uređaja razvijena od strane kompanije Intel (nekadašnja Altera), dizajnirana da pruži visoku gustinu logike po niskoj cijeni. Za razliku od procesora sa fiksnim instrukcijama, ovaj uređaj omogućava hardversku implementaciju specifičnih digitalnih funkcija putem konfigurabilnih logičkih blokova, optimizovanih za aplikacije gdje su bitni troškovi proizvodnje.

Arhitekturu Cyclone II čipa čine sljedeći ključni podsistemi:

- **Logički elementi (LE):** Predstavljaju osnovne jedinice za obradu. Svaki LE sadrži četvoroulaznu tabelu pretraživanja (LUT), programabilni registar i prenosni lanac (carry chain) za brze aritmetičke operacije. Ovi elementi su organizovani u Logic Array Blocks (LAB), koji omogućavaju efikasno grupisanje logike i lokalno rutiranje signala.
- **Memorijski resursi (Embedded Memory):** Interna memorija je realizovana kroz M4K blokove. Svaki blok ima kapacitet od 4 kilobita i može se konfigurisati kao RAM, ROM ili FIFO memorija. Ovi blokovi podržavaju rad u režimu istovremenog čitanja i pisanja (dual-port RAM) i rade na frekvencijama do 250 MHz.
- **Namjenski množitelji (Embedded Multipliers):** Za razliku od novijih DSP blokova, Cyclone II posjeduje namjenske hardverske množitelje (18x18 bitova). Oni su optimizovani za operacije množenja koje su osnova digitalne obrade signala, čime se štede logički resursi koji bi inače bili utrošeni na softversku implementaciju množenja.
- **Ulazno-izlazni elementi (IOEs):** Podržavaju brojne I/O standarde (kao što su LVTTL, LVCMOS, SSTL) i diferencijalne signale (LVDS). Svaki IOE sadrži bidirekpcioni bafer i registre koji omogućavaju precizno tempiranje signala na pinovima čipa.
- **Sistem povezivanja (Interconnect):** Koristi se MultiTrack struktura usmjerenjavanja. To je mreža bakarnih linija različitih dužina koje omogućavaju povezivanje logičkih blokova uz predvidljivo i minimalno kašnjenje signala (propagation delay).
- **Konfiguracija:** Arhitektura je bazirana na SRAM tehnologiji, što znači da je konfiguracija volatilna i gubi se nakon isključenja napajanja. Čip se konfiguriše prilikom svakog pokretanja (power-up) putem eksternih konfiguracionih memorija (poput EPCS serije) ili direktno putem JTAG interfejsa.

4.2. RASPBERRY PI 4

Raspberry Pi je **računar na jednoj ploči (SBC)** zasnovan na **ARM arhitekturi**. On integriše procesor, memoriju, grafiku i ulazno/izlazne portove na jednoj štampanoj ploči (PCB). Zbog svoje robusnosti i malih dimenzija, koristi se za IoT aplikacije, robotiku i obradu podataka u stvarnom vremenu.

Hardverske komponente

- **Procesor (CPU):** Četverojezreni procesor (1.5 GHz) koji služi kao „mozak“ uređaja za obradu podataka.
- **Grafički procesor (GPU):** Broadcom VideoCore, omogućava 3D renderovanje i 4K prikaz na dva monitora.
- **Memorija i skladištenje:** Koristi MicroSD karticu za operativni sistem i pohranu podataka.
- **Mrežna povezanost:** Gigabit Ethernet za žičanu mrežu, uz ugrađeni Wi-Fi i Bluetooth (BLE).
- **GPIO pinovi:** 40 pinova za direktno upravljanje eksternim hardverom (senzori, motori, LED).

Interfejsi i konekcija

- **HDMI:** Prenos slike i zvuka visoke definicije.
- **CSI (Camera Serial Interface):** Namjenski port za povezivanje kamera.
- **DSI (Display Serial Interface):** 15-pinski konektor za LCD ekrane.
- **UART:** Protokol za serijsku komunikaciju sa drugim mikrokontrolerima ili modulima.
- **USB portovi:** Za povezivanje periferije (tastature, miševi, eksterni diskovi).
- **Napajanje:** USB-C konektor (kod novijih modela) za dovod električne energije.

Modeli i indikatori

- **Model A vs. Model B:** Model A je energetski efikasniji i nema Ethernet port, dok je Model B snažniji i opremljen mrežnim portom.
- **Statusne LED diode:**
 - **Crvena (PWR):** Indikator stabilnog napajanja.
 - **Zelena (ACT):** Indikator aktivnosti čitanja/pisanja na SD kartici.

4.3. ESP – 01S

ESP-01S je ultra-kompaktni Wi-Fi modul zasnovan na **ESP8266** mikrokontroleru. Dizajniran je da omogući uređajima pristup internetu ili da služi kao samostalni kontroler za jednostavne IoT projekte. Predstavlja unapredenu verziju originalnog ESP-01 modula, sa boljim performansama antene i optimizovanim hardverom.

Hardverske komponente

- **Procesor (CPU):** L106 32-bitni RISC mikroprocesor sa radnim taktom od 80 MHz (može se povećati do 160 MHz).
- **Memorija (Flash):** Najčešće dolazi sa **1 MB** flash memorije za skladištenje koda i podataka.
- **Wi-Fi modul:** Podržava 802.11 b/g/n standarde, omogućavajući Wi-Fi Direct (P2P) i soft-AP režime.
- **Integrirana antena:** Posjeduje PCB antenu „na ploči“, što eliminiše potrebu za vanjskim dodacima za osnovnu komunikaciju.

Pinovi i povezivanje

ESP-01S ima **8 pinova** raspoređenih u 2x4 zaglavlje:

- **VCC:** Napajanje od tačno **3.3V** (nije tolerantan na 5V).
- **GND:** Uzemljenje.
- **TX/RX:** Pinovi za serijsku komunikaciju (UART) za programiranje i slanje podataka.
- **GPIO 0 i GPIO 2:** Digitalni ulazno/izlazni pinovi (GPIO 0 se koristi i za ulazak u mod za programiranje).
- **CH_PD (EN):** Pin za omogućavanje čipa; mora biti spojen na visoki napon (3.3V) da bi modul radio.
- **RST:** Reset pin.

Ključne karakteristike i prednosti

- **Mala potrošnja energije:** Dizajniran za mobilne i baterijske uređaje; posjeduje "deep sleep" mod.
- **Samostalni rad:** Može se koristiti kao most za Arduino (preko AT komandi) ili se programirati direktno putem Arduino IDE ili MicroPython-a.
- **Kompaktnost:** Zahvaljujući malim dimenzijama, idealan je za ugradnju u pametne prekidače, senzore temperature i druge male uređaje.
- **Unapređenja (u odnosu na ESP-01):** Bolji dizajn štampane ploče i izbačeni nepotrebni otpornici, što olakšava direktno povezivanje.

4.4. ULAZNE KOMPONENTE

4.4.1. Ultrasonični senzor HC-SR04P

Ultrasonični senzor HC-SR04 je elektronski uređaj koji mjeri udaljenost do objekta emitovanjem ultrazvučnih talasa i pretvaranjem reflektovanog zvuka u električni signal. Sastoji se od dvije glavne komponente: **odašiljača**, koji emitiše zvuk pomoću piezoelektričnih kristala, i **prijemnika**, koji registruje povratni talas. Udaljenost D se izračunava mjerom vremena T potrebnog da talas ode do cilja i vrati se nazad, koristeći brzinu zvuka C. Ovi senzori su otporniji na smetnje izazvane dimom i gasovima u odnosu na infracrvene senzore, a primjenjuju se u sistemima za parkiranje, robotici i detekciji prepreka. Senzor radi na naponu od **5V DC** uz radnu struju od **15mA**. Njegov radni frekvencijski opseg je **40kHz**. Minimalni domet mjerjenja iznosi **2 cm**, dok je maksimalni domet do **400 cm**. Preciznost mjerjenja (tačnost) je **3 mm**, a senzor pokriva ugao manji od **15°**. Dimenzije samog uređaja su **45 x 20 x 15 mm**.

Raspored pinova:

- **Vcc:** Napajanje (+5V).
- **Trigger:** Pin koji inicira slanje ultrazvučnog impulsa.
- **Echo:** Pin koji šalje signal proporcionalan trajanju povratka talasa.
- **GND:** Uzemljenje.

4.4.2. Kamera Logitech C512

Logitech C512 je kompaktna, prenosiva HD web kamera dizajnirana za video pozive i snimanje u visokoj rezoluciji. Koristi stakleni objektiv i tehnologiju autofokusa kako bi osigurala oštru sliku čak i u krupnom planu, dok ugrađeni mikrofon sa redukcijom šuma omogućava jasan prenos glasa bez pozadinske buke. Zahvaljujući "fold-and-go" dizajnu, kamera se lako sklapa i štiti objektiv tokom transporta, što je čini idealnom za korisnike u pokretu. Povezuje se putem standardnog USB interfejsa (Plug-and-Play) i automatski korigira osvjetljenje kako bi pružila kvalitetan video u različitim uslovima ambijentalnog svjetla.

Tehničke specifikacije i povezivanje

- **Maksimalna rezolucija:** 720p HD video pozivi i snimanje.
- **Fokus:** Napredni sistem autofokusa.
- **Mikrofon:** Ugrađen sa tehnologijom za suzbijanje buke.
- **Povezivanje:** USB 2.0 (standardni tip A).
- **Dizajn:** Rotacija od 360 stepeni i sklopiva konstrukcija.

4.4.3. Infracrveni sensor

Infracrveni senzor je elektronski uređaj koji detektuje prisustvo objekata ili mjeri toplotu emitovanjem ili primanjem infracrvenog zračenja. Najčešće se koristi kao **senzor blizine** koji radi na principu refleksije: IR dioda (predajnik) emituje svjetlost nevidljivu ljudskom oku, a ukoliko se ispred senzora nađe objekat, svjetlost se odbija nazad do fotodiode (prijemnika). Za razliku od ultrasoničnih senzora, infracrveni senzori su brži u detekciji, ali mogu biti osjetljivi na ambijentalno svjetlo i boju objekta (npr. crne površine apsorbuju IR zrake, što otežava detekciju).

Tehničke karakteristike i povezivanje

- **Radni napon:** Tipično 3.3V do 5V DC.
- **Domet detekcije:** Podesiv putem potenciometra (obično od 2 cm do 30 cm).
- **Ugao detekcije:** Oko 35°.
- **Izlazni signal:** Digitalni (0 ili 1) – logička nula kod detekcije prepreke.

Opis pinova

- **VCC:** Pozitivno napajanje (3.3V - 5V).
- **GND:** Uzemljenje.
- **OUT:** Digitalni izlazni signal koji se povezuje na mikrokontroler.

4.5. IZLAZNE KOMPONENTE

4.5.1. DC motor

DC motor je elektromehanički uređaj koji pretvara jednosmjernu električnu energiju u mehaničku rotaciju. U projektima sa mikrokontrolerima, DC motori se koriste za pokretanje točkova, propelera ili drugih mehanizama. S obzirom na to da motori troše više struje nego što mikrokontroler može pružiti, oni se obično povezuju preko **H-mosta** ili tranzistora koji omogućavaju kontrolu brzine (pomoću PWM signala) i smjera rotacije.

Tehničke karakteristike i povezivanje:

- **Radni napon:** Najčešće 3V do 12V DC (zavisno od modela).
- **Struja:** Zahtijeva eksterne drijevere (npr. L298N) zbog velike startne struje.
- **Brzina:** Definisana u RPM (broj okretaja u minuti).
- **Povezivanje:** Posjeduje dva terminala; zamjena polariteta mijenja smjer okretanja.

4.5.2. Aktivni piezo buzzer

Aktivni piezo buzzer je zvučni indikator koji generiše ton fiksne frekvencije čim se na njega dovede napon. Za razliku od pasivnog buzzera, aktivni ima ugrađeno oscilatorno kolo, pa mu nije potreban kompleksan signal (PWM) za proizvodnju zvuka, već samo običan digitalni signal (High/Low). Idealni su za alarne, potvrde pritiska tastera i sistemska upozorenja.

Tehničke karakteristike i povezivanje:

- **Radni napon:** Obično 3.3V do 5V DC.
- **Frekvencija:** Fiksna, najčešće oko 2.3 kHz.
- **Potrošnja:** Veoma niska, obično ispod 30mA.
- **Povezivanje:** Posjeduje duži pin (anoda/+) koji ide na digitalni izlaz i kraći pin (katoda/-) koji ide na uzemljenje (GND).

4.6. DC MOTOR DRIVER L298N

L298N je modul zasnovan na dvostrukom **H-mostu** koji omogućava istovremenu kontrolu brzine i smjera rotacije dva DC motora ili jednog koračnog (stepper) motora. On služi kao posrednik između mikrokontrolera (poput Raspberry Pi ili Arduina) i motora, jer mikrokontroleri

ne mogu isporučiti dovoljno struje za pokretanje motora. Modul posjeduje ugrađeni regulator napona od **5V**, koji se može koristiti za napajanje same kontrolne logike ili vanjskih komponenti, pod uslovom da je ulazni napon motora veći od 7V.

Tehničke specifikacije

- **Radni napon (logika):** 5V DC.
- **Napon za motore (Drive voltage):** 5V do 35V DC.
- **Maksimalna struja po kanalu:** 2A (vršno do 3A).
- **Maksimalna snaga:** 25W.
- **Broj kanala:** 2 (omogućava kontrolu dva nezavisna DC motora).
- **Dodatne funkcije:** Zaštita od pregrijavanja i ugrađene diode za zaštitu od povratnog napona.

Opis pinova i povezivanje

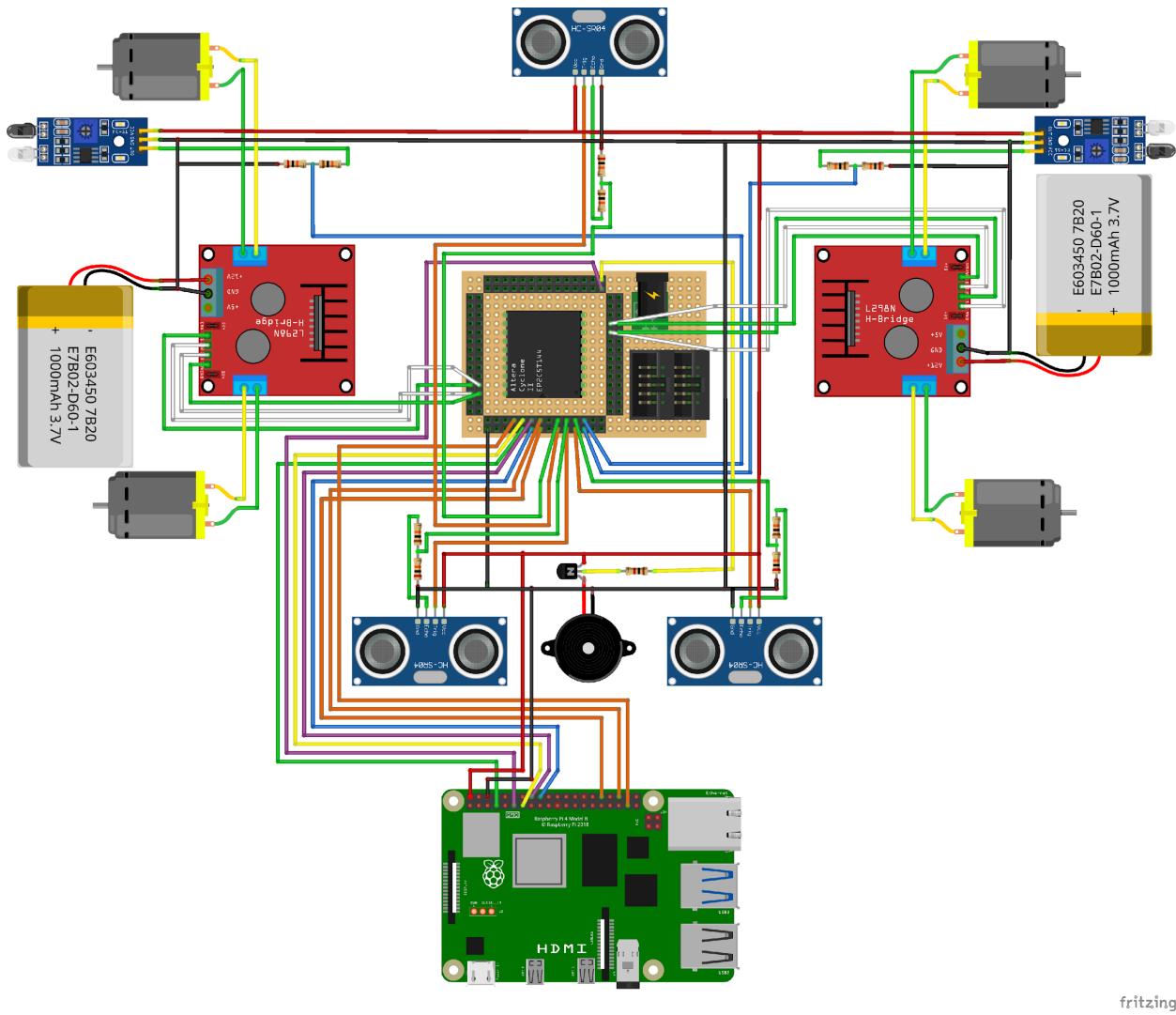
Modul je podijeljen na tri glavna dijela za povezivanje:

1. **Terminali za napajanje**
 - **VCC:** Ulaz za napajanje motora (npr. 12V).
 - **GND:** Zajedničko uzemljenje za motor i mikrokontroler.
 - **5V:** Izlazni/ulazni pin (daje 5V ako je aktivan interni regulator).
2. **Izlazi za motore (A i B)**
 - **OUT1 i OUT2:** Priklučci za prvi motor.
 - **OUT3 i OUT4:** Priklučci za drugi motor.
3. **Kontrolni pinovi (Logika)**
 - **ENA i ENB:** Pinovi za omogućavanje motora (koriste se za kontrolu brzine putem PWM-a).
 - **IN1, IN2, IN3, IN4:** Digitalni ulazi koji određuju smjer okretanja (npr. IN1 High / IN2 Low okreće motor naprijed).

5. UPRAVLJAČKA LOGIKA I SISTEMSKA INTEGRACIJA

Upravljanje sistemom podijeljeno je između **Raspberry Pi 4** računara i **Altera Cyclone II FPGA** čipa. Raspberry Pi 4 služi za procesiranje slike putem OpenCV biblioteke i upravljanje

mrežnim saobraćajem preko Flask servera. FPGA vrši obradu signala sa senzora u realnom vremenu, upravlja radom motora i mehanizam kontrole.



Slika 1. - Šema povezivanja komponenti sa FPGA i Raspberry Pi

fritzing

5.2. ULAZNI I IZLAZNI SIGNALI ZA FPGA

Ulagani signali

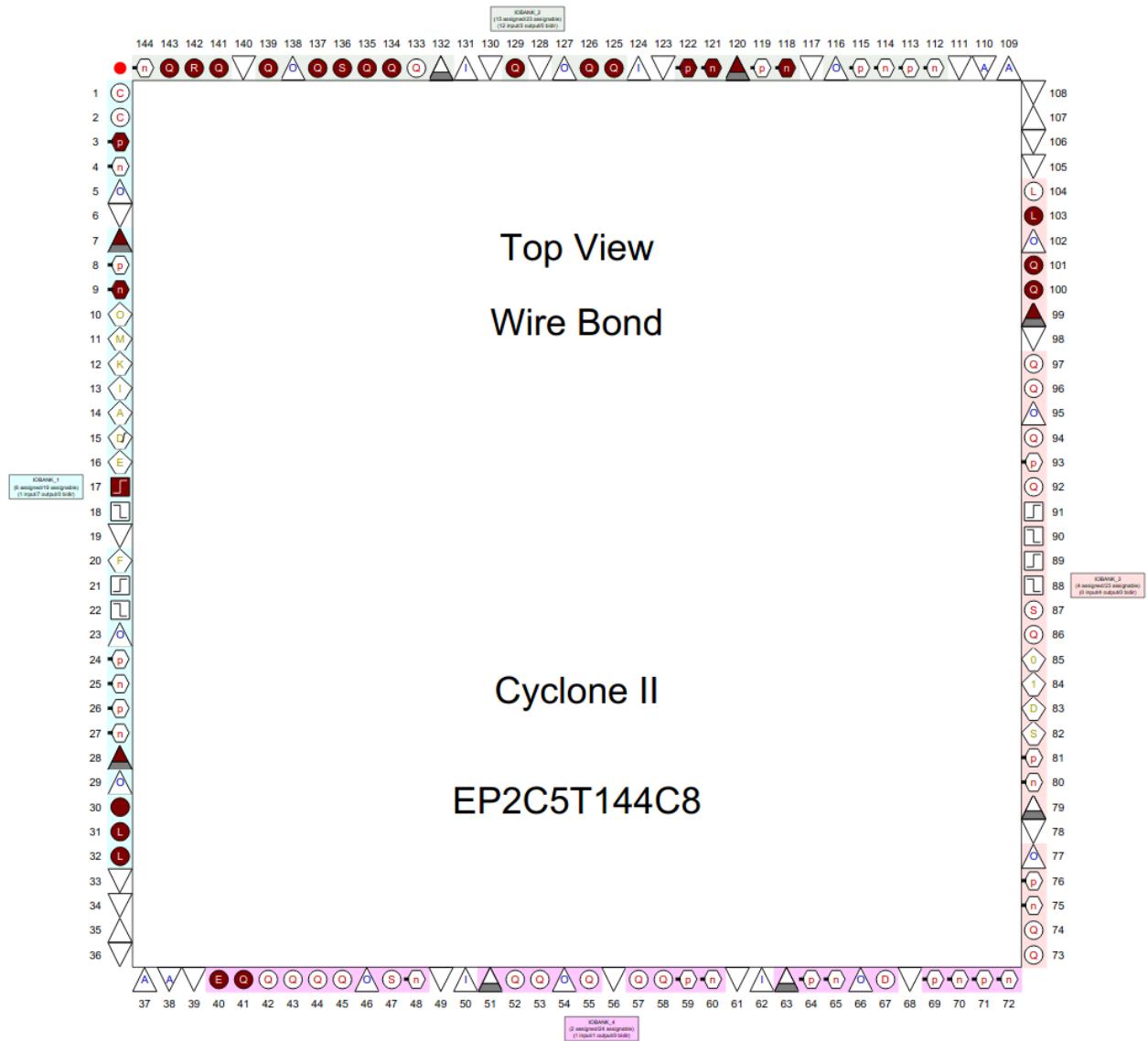
- Raspberry Pi (GPIO):** Prima 7 digitalnih signala koji definišu kretanje (fwd_in, bwd_in, left_in, right_in), detekciju saobraćajnih znakova (stopsign_in, stoplight_in) i sigurnosni prekid (failsafe_in).
- Infracrveni (IR) senzori:** Dva digitalna ulaza (ld_left, ld_right) koji šalju logičku jedinicu pri detekciji linije.
- Ultrazvučni senzori (HC-SR04):** Tri echo signala koji prenose informaciju o udaljenosti od prepreke.
- Aplikacija:** buzzer_in signal za manuelno aktiviranje sirene.

Izlazni signali

- Motor drajveri (L298N):** Dva porta sa po 4 bita (m1_out, m2_out) koji kontrolisu smjer i rad četiri elektromotora.
- Ultrazvučni senzori:** Tri trig signala za inicijalizaciju mjerena.
- Zvučna signalizacija:** Jedan izlazni pin za kontrolu buzzera.

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength	Differential Pair
out_buzzer	Output	PIN_40	4	B4_N1	PIN_40	3.3-V LVTTL (default)		24mA (default)	
in_buzzer_in	Input	PIN_41	4	B4_N1	PIN_41	3.3-V LVTTL (default)		24mA (default)	
in_bwd_in	Input	PIN_121	2	B2_N0	PIN_121	3.3-V LVTTL (default)		24mA (default)	
in_clk	Input	PIN_17	1	B1_N0	PIN_17	3.3-V LVTTL (default)		24mA (default)	
in_echo_bwd1	Input	PIN_141	2	B2_N1	PIN_141	3.3-V LVTTL (default)		24mA (default)	
in_echo_bwd2	Input	PIN_137	2	B2_N1	PIN_137	3.3-V LVTTL (default)		24mA (default)	
in_echo_fwd	Input	PIN_135	2	B2_N1	PIN_135	3.3-V LVTTL (default)		24mA (default)	
infailsafe_in	Input	PIN_118	2	B2_N0	PIN_118	3.3-V LVTTL (default)		24mA (default)	
in_fwd_in	Input	PIN_120	2	B2_N0	PIN_120	3.3-V LVTTL (default)		24mA (default)	
in_ld_left	Input	PIN_143	2	B2_N1	PIN_143	3.3-V LVTTL (default)		24mA (default)	
in_ld_right	Input	PIN_142	2	B2_N1	PIN_142	3.3-V LVTTL (default)		24mA (default)	
in_left_in	Input	PIN_125	2	B2_N0	PIN_125	3.3-V LVTTL (default)		24mA (default)	
out_m1_out[3]	Output	PIN_103	3	B3_N0	PIN_103	3.3-V LVTTL (default)		24mA (default)	
out_m1_out[2]	Output	PIN_101	3	B3_N0	PIN_101	3.3-V LVTTL (default)		24mA (default)	
out_m1_out[1]	Output	PIN_100	3	B3_N0	PIN_100	3.3-V LVTTL (default)		24mA (default)	
out_m1_out[0]	Output	PIN_99	3	B3_N0	PIN_99	3.3-V LVTTL (default)		24mA (default)	
out_m2_out[3]	Output	PIN_32	1	B1_N1	PIN_32	3.3-V LVTTL (default)		24mA (default)	
out_m2_out[2]	Output	PIN_31	1	B1_N1	PIN_31	3.3-V LVTTL (default)		24mA (default)	
out_m2_out[1]	Output	PIN_30	1	B1_N1	PIN_30	3.3-V LVTTL (default)		24mA (default)	
out_m2_out[0]	Output	PIN_28	1	B1_N1	PIN_28	3.3-V LVTTL (default)		24mA (default)	
out_md_state[2]	Output	PIN_9	1	B1_N0	PIN_9	3.3-V LVTTL (default)		24mA (default)	
out_md_state[1]	Output	PIN_7	1	B1_N0	PIN_7	3.3-V LVTTL (default)		24mA (default)	
out_md_state[0]	Output	PIN_3	1	B1_N0	PIN_3	3.3-V LVTTL (default)		24mA (default)	
in_right_in	Input	PIN_122	2	B2_N0	PIN_122	3.3-V LVTTL (default)		24mA (default)	
in_stoplight_in	Input	PIN_126	2	B2_N0	PIN_126	3.3-V LVTTL (default)		24mA (default)	
in_stopsign_in	Input	PIN_129	2	B2_N1	PIN_129	3.3-V LVTTL (default)		24mA (default)	
out_trig_bwd1	Output	PIN_139	2	B2_N1	PIN_139	3.3-V LVTTL (default)		24mA (default)	
out_trig_bwd2	Output	PIN_136	2	B2_N1	PIN_136	3.3-V LVTTL (default)		24mA (default)	
out_trig_fwd	Output	PIN_134	2	B2_N1	PIN_134	3.3-V LVTTL (default)		24mA (default)	

Slika 2. - Lista pinova za FPGA



Slika 3. – Raspored pinova za FPGA

5.3. ULAZNI I IZLAZNI SIGNALI ZA RASPBERRY PI

Raspberry Pi 4 obrađuje podatke sa kamere i upravlja komunikacijom sa korisničkom aplikacijom. On pretvara vizuelne informacije i mrežne komande u digitalne signale koje FPGA može razumjeti.

Ulazni signali

- **Kamera:** Video signal koji ulazi preko **CSI interfejsa** (poseban utor na ploči za kameru). OpenCV biblioteka analizira ove slike kako bi prepoznala STOP znak i crveno svjetlo na semaforu.
- **Mrežni saobraćaj (Wi-Fi):** HTTP zahtjevi koji pristižu na Flask server, sadrže komande za smjer kretanja i "heartbeat" potvrdu od mobilne aplikacije.

Izlazni signali

- **Upravljački GPIO pinovi:** Sedam digitalnih izlaza koji su direktno povezani na ulaze FPGA čipa (GPIO 4, 27, 23, 24, 16, 12 i 26). Ovi signali prenose komande kretanja, signale detekcije objekata i stanje sigurnosne veze (failsafe).
- **Statusni odgovor:** HTTP odgovor koji se šalje nazad aplikaciji kao informacija o tome da li se robot trenutno kreće.

5.4. LOGIKA PRIORITETA KRETANJA

Unutar FPGA modula za upravljanje motorima (`motor_driver`) implementirana je stroga hijerarhija signala. Ovo osigurava da robot reaguje ispravno u situacijama kada se pojavi više komandi istovremeno.

Redoslijed prioriteta je sljedeći:

1. **Potpuno zaustavljanje:** Ako su aktivni signali za gubitak veze, crveno svjetlo na semaforu ili STOP znak, svi ostali ulazi se ignoriraju i motori se gase.
2. **Praćenje putanje:** Ukoliko nema signala za zaustavljanje, a robot se kreće naprijed, prioritet preuzimaju infracrveni senzori. Ako lijevi senzor detektuje liniju, robot automatski skreće udesno, a ako desni detektuje, skreće uljevo.
3. **Korisničko upravljanje:** Tek kada su ispunjeni svi sigurnosni uslovi i nema detekcije linija, FPGA izvršava komande kretanja poslane sa Raspberry Pi uređaja.

5.5. ZVUČNA SIGNALIZACIJA I SENZORI

Sistem koristi zvučnu signalizaciju kao metodu upozorenja na prepreke. Ova logika je odvojena od modula za motore kako bi radila neovisno i bez kašnjenja.

Tri ultrazvučna senzora (prednji i dva zadnja) šalju podatke FPGA čipu. Unutar modula parking_sensor, trajanje povratnog zvučnog talasa se preračunava u udaljenost. Na osnovu te udaljenosti, FPGA kontroliše zvučnik (buzzer):

- Ako je prepreka blizu (između 15 cm i 20 cm), zvučnik pišti sporije.
- Kako se prepreka približava (između 10 cm i 15 cm), pištanje postaje brže.
- Na udaljenosti manjoj od 10 cm, zvuk postaje konstantan.

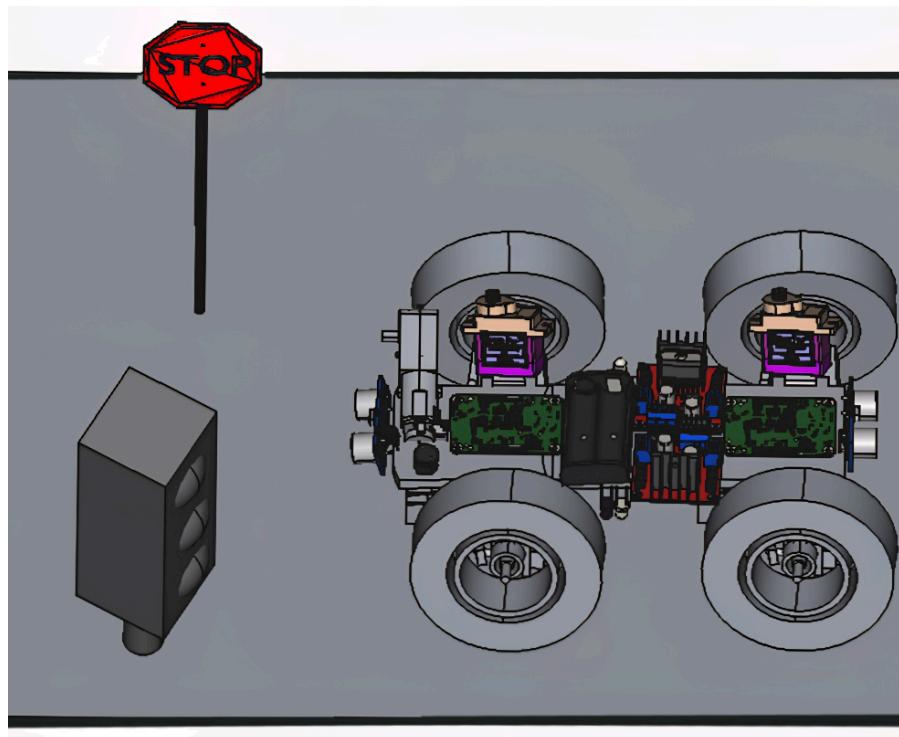
Finalni signal za zvučnik dobija se sabiranjem (logička OR operacija) signala sa sva tri senzora i signala svirene sa aplikacije. To znači da će se zvučnik aktivirati ako bilo koji senzor osjeti blizinu ili ako korisnik manuelno pritisne svirenu u kontrolnom interfejsu.

5.6. MEHANIZMI KONTROLE I ZAŠTITE

Ovaj dio sistema osigurava da robot ne ostane u pokretu ako nastane problem u komunikaciji ili softveru. Glavni mehanizam je "heartbeat" provjera. Ako aplikacija prestane slati potvrdu u roku od 300 ms (npr. zbog gubitka Wi-Fi signala), Raspberry Pi postavlja failsafe signal na nulu. FPGA to odmah prepoznaje i momentalno isključuje napajanje motora.

Dodatno, kod detekcije STOP znaka, softver putem stopsign_in pina blokira kretanje na tačno 3 sekunde. Tokom ovog perioda, FPGA drži motore u zakočenom stanju, čime se simulira obavezno zaustavljanje u saobraćaju, nakon čega se upravljanje ponovo predaje korisniku.

6. 3D MODEL I FINALNI IZGLEĐ PROJEKTA



Slika 4. - 3D model robota izrađen u FreeCAD-u



Slika 5. - Finalni izgled projektnog zadatka



Slika 6. - Maketa znaka STOP



Slika 7. - Model semafora

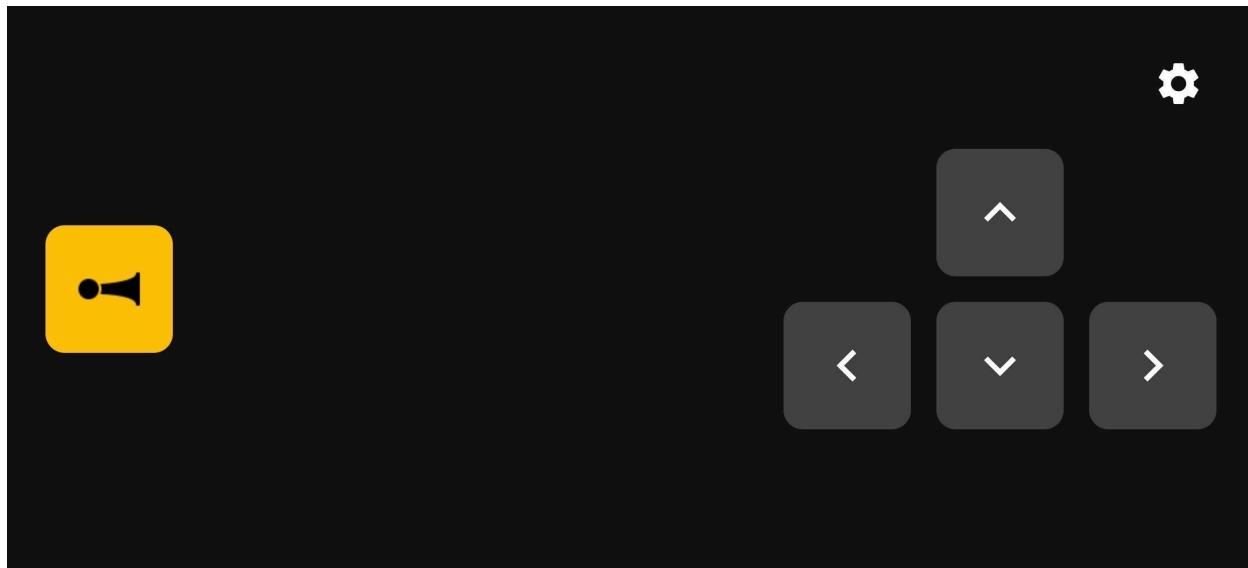
7. MOBILNA ANDROID APLIKACIJA

Kontrolni interfejs projekta realizovan je u obliku mobilne aplikacije za Android platformu. Aplikacija je razvijena u razvojnom okruženju **Android Studio** koristeći programski jezik **Kotlin** i moderan deklarativni okvir **Jetpack Compose**.

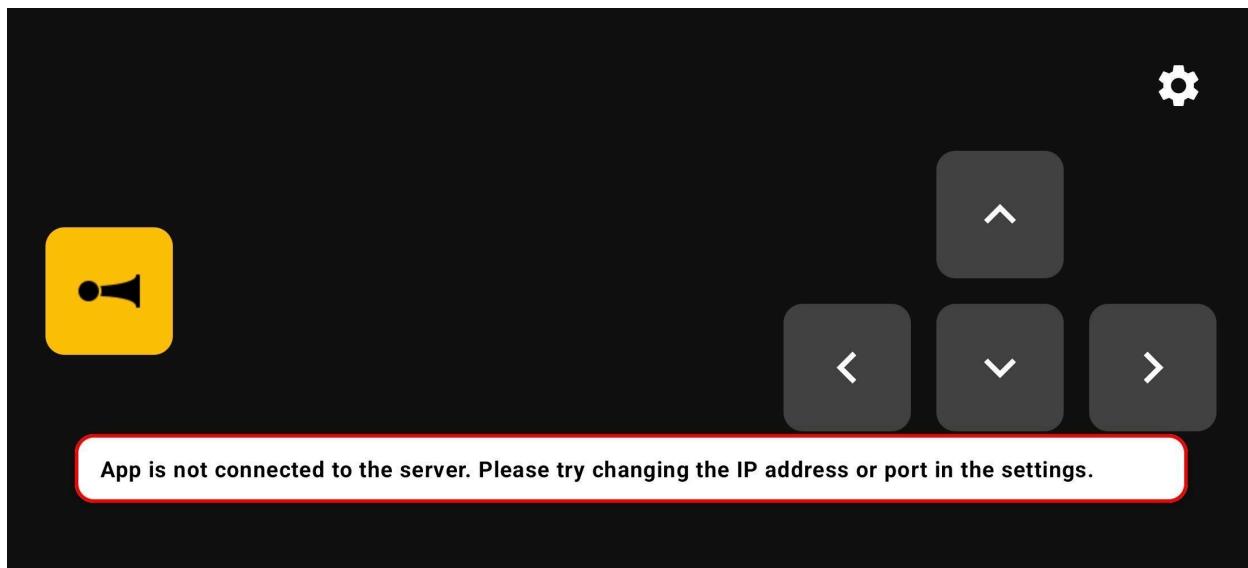
Glavne funkcionalnosti aplikacije obuhvataju:

- **Upravljanje u realnom vremenu:** Intuitivni korisnički interfejs sa komandama za navigaciju (naprijed, nazad, lijevo, desno) i kontrolu sigurnosnih funkcija (zvučni signal).
- **Mrežna komunikacija:** Slanje upravljačkih instrukcija putem HTTP protokola ka serveru na Raspberry Pi platformi unutar lokalne WiFi mreže.

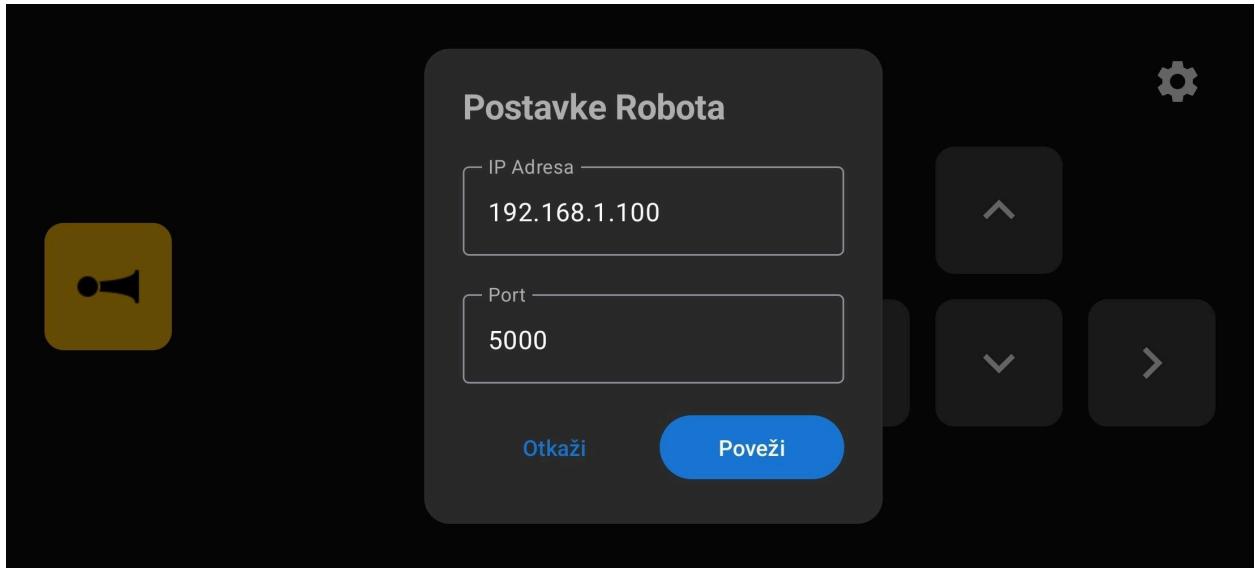
Na sljedećim slikama prikazan je izgled ekrana mobilne aplikacije kada je aplikacija spremna za upotrebu, tačnije kada je aplikacija povezana sa serverom, kada aplikacija nije spojena sa serverom i prozor za postavku IP adrese i porta kako bi se aplikacija spojila na odgovarajući server.



Slika 8. - Glavni upravljački ekran



Slika 9. - Prikaz greške u konekciji



Slika 10. - Postavke mrežnih parametara

8. KOD

8.1. VERILOG KOD ZA KONTROLU DC MOTORA NA FPGA

```
module motor_driver (
    input wire clk,
    // movement controls from backend
    input wire fwd_in,
    input wire bwd_in,
    input wire left_in,
    input wire right_in,
    input wire stoplight_in,
    input wire stopsign_in,
    input wire failsafe_in,
    // line detectors
    input wire ld_left,
    input wire ld_right,
    // motors (A0 A1 B0 B1)
    output reg [3:0] m1_out,  // left
    output reg [3:0] m2_out,  // right
    output reg [2:0] state
);
    // states
```

```

localparam STOP = 0, FORWARD = 1, BACKWARD = 2, LEFT = 3, RIGHT = 4;

always @(posedge clk) begin
    // prioritize stop
    if (failsafe_in == 1 || stoplight_in == 1 || stopsign_in == 1) begin
        state = STOP;
    end else if (fwd_in == 1) begin
        // when moving forward check for lines
        if (ld_left == 1) begin
            state = RIGHT;
        end else if (ld_right == 1) begin
            state = LEFT;
        end else begin
            state = FORWARD;
        end
    end else if (bwd_in == 1) begin
        state = BACKWARD;
    end else if (right_in == 1) begin
        state = RIGHT;
    end else if (left_in == 1) begin
        state = LEFT;
    end else begin
        state = STOP;
    end
end

always @(state) begin
    case (state)
        FORWARD: begin
            m1_out = 4'b0110;
            m2_out = 4'b1001;
        end

        BACKWARD: begin
            m1_out = 4'b1001;
            m2_out = 4'b0110;
        end

        LEFT: begin
            m1_out = 4'b0000;
            m2_out = 4'b1001;
        end

        RIGHT: begin
            m1_out = 4'b0110;
            m2_out = 4'b0000;
        end

        STOP: begin
            m1_out = 4'b0000;
            m2_out = 4'b0000;
        end
    endcase
end

```

```

    default: begin
        m1_out = 4'b0000;
        m2_out = 4'b0000;
    end
    endcase
end
endmodule

```

8.2. VERILOG KODA ZA IMPLEMENTACIJU KONTROLE ZVUČNE SIGNALIZACIJE PARKING SENZORA NA FPGA

```

module parking_sensor (
    input wire clk,
    input wire echo,
    output reg trig,
    output reg signal
);

localparam CLK_FREQ = 50000000;
localparam CYCLES_PER_CM = 2915;

localparam DIST_CONST = 10 * CYCLES_PER_CM;
localparam DIST_FAST = 15 * CYCLES_PER_CM;
localparam DIST_SLOW = 20 * CYCLES_PER_CM;

localparam TIME_500MS = CLK_FREQ / 2;
localparam TIME_250MS = CLK_FREQ / 4;

reg [21:0] trig_timer = 0;
reg [21:0] echo_width = 0;
reg [21:0] last_dist = 0;
reg [25:0] toggle_timer = 0;

```

```

always @(posedge clk) begin
    // TRIGGER
    if (trig_timer < 4000000) trig_timer <= trig_timer + 1;
    else trig_timer <= 0;
    trig <= (trig_timer > 0 && trig_timer < 500);

    // ECHO
    if (echo == 1) echo_width <= echo_width + 1;
    else if (echo == 0 && echo_width > 0) begin
        last_dist <= echo_width;
        echo_width <= 0;
    end

    // TOGGLE TIMER
    if (toggle_timer < CLK_FREQ) toggle_timer <= toggle_timer + 1;
    else toggle_timer <= 0;

    // PROXIMITY LOGIC
    if (last_dist > DIST_SLOW || last_dist == 0) begin
        signal <= 0;
    end else if (last_dist > DIST_FAST) begin
        signal <= (toggle_timer < TIME_500MS) ? 1 : 0;
    end else if (last_dist > DIST_CONST) begin
        signal <= (toggle_timer % (TIME_250MS * 2) < TIME_250MS) ? 1 : 0;
    end else begin
        signal <= 1;
    end
end
endmodule

```

8.3. VERILOG KOD ZA INTEGRICIU SENZORA I MOTORA

```

module main (
    input wire clk,

    // movement controls
    input wire fwd_in,
    input wire bwd_in,
    input wire left_in,
    input wire right_in,
    input wire stoplight_in,
    input wire stopsign_in,
    input wire failsafe_in,

```

```

// motors
output wire [3:0] m1_out,
output wire [3:0] m2_out,
output wire [2:0] md_state,

// parking sensors
input  wire echo_fwd,
output wire trig_fwd,
input  wire echo_bwd1,
output wire trig_bwd1,
input  wire echo_bwd2,
output wire trig_bwd2,

// buzzer
input  wire buzzer_in,
output wire buzzer,

// line detectors
input wire ld_left,
input wire ld_right
);
motor_driver md (
    .clk(clk),
    .fwd_in(fwd_in),
    .bwd_in(bwd_in),
    .left_in(left_in),
    .right_in(right_in),
    .stoplight_in(stoplight_in),
    .stopsign_in(stopsign_in),
    .failsafe_in(failsafe_in),
    .ld_left(ld_left),
    .ld_right(ld_right),
    .m1_out(m1_out),
    .m2_out(m2_out),
    .state(md_state)
);
wire ps_buzzer_fwd, ps_buzzer_bwd1, ps_buzzer_bwd2;
assign buzzer = ps_buzzer_fwd | ps_buzzer_bwd1 | ps_buzzer_bwd2 | buzzer_in;

parking_sensor ps_fwd (
    .clk(clk),
    .echo(echo_fwd),

```

```

    .trig(trig_fwd),
    .signal(ps_buzzer_fwd)
);

parking_sensor ps_bwd1 (
    .clk(clk),
    .echo(echo_bwd1),
    .trig(trig_bwd1),
    .signal(ps_buzzer_bwd1)
);

parking_sensor ps_bwd2 (
    .clk(clk),
    .echo(echo_bwd2),
    .trig(trig_bwd2),
    .signal(ps_buzzer_bwd2)
);
endmodule

```

8.4. KOD NA RASPBERRY PI ZA DETEKCIJU SAOBRAĆAJNIH ZNAKOVA KORIŠTENJEM KAMERE

```

"""
detector.py

Detektor STOP znaka i crvenog svjetla na semaforu.

"""

import cv2
import numpy as np
import time
import threading
import collections
import argparse
import sys
import requests # <-- dodano za pitati Flask server da li se auto kreće

try:
    import RPi.GPIO as GPIO
    GPIO.setmode(GPIO.BCM)
    HAVE_GPIO = True
except Exception:
    HAVE_GPIO = False

```

```

# ----- CONFIG -----
CASCADE_PATH = "stop_sign.xml" # putanja do Haar Cascade modela za prepoznavanje
STOP znaka
CAM_INDEX = 0

WORK_WIDTH = 640
WORK_HEIGHT = 480

STOP_HOLD_ACTIVE = False
STOP_HOLD_END = 0

USE_ROI = False # za detekciju samo unutar odredjene regije
ROI = (200, 50, 440, 380) # ima smisla samo ako je gornje True

# STOP znak
STOP_MIN_AREA = 1500 # minimalan povrsina u pixelima koju znak mora imat da bi
bio detektovan
STOP_FRAMES_NEEDED = 3 # minimaln broj uzastopnih frame-ova da bi znak bio
prepoznat
STOP_HOLD_SECONDS = 3.0 # koliko dugo zaustavljamo robot-a
STOP_COOLDOWN_SECONDS = 8.0 # koliko dugo imamo za pokrenut se

# konfiguracija detekcije crvenog svjetla
LOWER_RED1 = np.array([0, 120, 120])
UPPER_RED1 = np.array([10, 255, 255])
LOWER_RED2 = np.array([160, 120, 120])
UPPER_RED2 = np.array([179, 255, 255])

RED_MIN_AREA = 5
RED_MAX_AREA = 5000
RED_MIN_BRIGHTNESS = 30

LED_MAX_SIZE = 80
CIRCULARITY_MIN = 0.3

RED_FRAMES_ON_NEEDED = 10
RED_FRAMES_OFF_NEEDED = 10

GPIO_RED_PIN = 12 # pin na koji saljemo signal crvenog svjetla
GPIO_STOP_PIN = 16 # pin na koji saljemo signal nakon detekcije stop znaka

CASCADE_SCALE_FACTOR = 1.1
CASCADE_MIN_NEIGHBORS = 5
CASCADE_MIN_SIZE = (40, 40)

```

```

MORPH_KERNEL = np.ones((3, 3), np.uint8)
# ----- CONFIG -----

if HAVE_GPIO:
    try:
        GPIO.setup(GPIO_RED_PIN, GPIO.OUT, initial=GPIO.LOW)
        GPIO.setup(GPIO_STOP_PIN, GPIO.OUT, initial=GPIO.LOW)
    except Exception:
        HAVE_GPIO = False

def gpio_write(pin, value):
    if HAVE_GPIO:
        try:
            GPIO.output(pin, GPIO.HIGH if value else GPIO.LOW)
        except Exception:
            pass

class VideoCaptureThread:
    def __init__(self, src, width, height):
        self.cap = cv2.VideoCapture(src)
        self.cap.set(cv2.CAP_PROP_FRAME_WIDTH, width)
        self.cap.set(cv2.CAP_PROP_FRAME_HEIGHT, height)
        self.frame = None
        self.lock = threading.Lock()
        self.stop_flag = False
        threading.Thread(target=self._loop, daemon=True).start()

    def _loop(self):
        while not self.stop_flag:
            ret, f = self.cap.read()
            if ret:
                with self.lock:
                    self.frame = f

    def read(self):
        with self.lock:
            if self.frame is None:
                return None
            return self.frame.copy()

    def stop(self):
        self.stop_flag = True
        try:
            self.cap.release()
        except:

```

```

    pass

def apply_roi(img):
    if not USE_ROI:
        return img, (0, 0, img.shape[1], img.shape[0])
    x1, y1, x2, y2 = ROI
    roi_img = img[y1:y2, x1:x2]
    return roi_img, (x1, y1, x2, y2)

# ucitavanje Cascade modela za detekciju stop znaka
cascade = cv2.CascadeClassifier(CASCADE_PATH)
if cascade.empty():
    print("ERROR loading cascade")
    sys.exit(1)

# historija prepoznatih objekata
stop_history = collections.deque(maxlen=STOP_FRAMES_NEEDED)
red_on_history = collections.deque(maxlen=RED_FRAMES_ON_NEEDED)
red_off_history = collections.deque(maxlen=RED_FRAMES_OFF_NEEDED)
stop_cooldown_end = 0

last_stop_time = 0
red_state = False
stop_detected_once = False # flag da STOP znak ne detektuje više puta dok auto ne krene

FLASK_URL = "http://192.168.1.132:5000" # <--- postaviti na static ip kad bude bio

def main():
    global last_stop_time, red_state, stop_detected_once, stop_cooldown_end

    parser = argparse.ArgumentParser()
    parser.add_argument("--no-gui", action="store_true")
    parser.add_argument("--camera", type=int, default=CAM_INDEX)
    args = parser.parse_args()

    cap = VideoCaptureThread(args.camera, WORK_WIDTH, WORK_HEIGHT)
    print("Running...")

    try:
        while True:
            frame = cap.read()
            if frame is None:
                time.sleep(0.01)

```

```

        continue

frame = cv2.resize(frame, (WORK_WIDTH, WORK_HEIGHT))
roi_frame, (rx1, ry1, rx2, ry2) = apply_roi(frame)

# detekcija stop znaka
gray = cv2.cvtColor(roi_frame, cv2.COLOR_BGR2GRAY)
stops = cascade.detectMultiScale(
    gray,
    scaleFactor=CASCADE_SCALE_FACTOR,
    minNeighbors=CASCADE_MIN_NEIGHBORS,
    minSize=CASCADE_MIN_SIZE
)

best_stop = None
best_area = 0
for (x, y, w, h) in stops:
    area = w * h
    ar = w / float(h)
    if area >= STOP_MIN_AREA and 0.6 <= ar <= 1.6:
        if area > best_area:
            best_area = area
            best_stop = (x, y, w, h)

stop_history.append(1 if best_stop else 0)
stop_confirmed = len(stop_history) == STOP_FRAMES_NEEDED and
all(stop_history)

# kad se auto pocne kretat poslije stop znaka, resetuj flag
try:
    r = requests.get(f"{FLASK_URL}/is_moving", timeout=0.1)
    moving = r.json().get("moving", False)
except:
    moving = False

if moving:
    stop_detected_once = False

# detekcija stop znaka
# 1. Modify the STOP detection condition
# Added check: time.time() > stop_cooldown_end
global STOP_HOLD_ACTIVE, STOP_HOLD_END
if stop_confirmed and not stop_detected_once and time.time() >
stop_cooldown_end:
    last_stop_time = time.time()

```

```

        print("[EVENT] STOP SIGN detected")
        gpio_write(GPIO_STOP_PIN, True)
        STOP_HOLD_ACTIVE = True
        STOP_HOLD_END = time.time() + STOP_HOLD_SECONDS
        stop_detected_once = True

    # 2. Modify the STOP HOLD release logic
    if STOP_HOLD_ACTIVE and time.time() >= STOP_HOLD_END:
        gpio_write(GPIO_STOP_PIN, False)
        STOP_HOLD_ACTIVE = False
        # Set the cooldown to start NOW for 5 seconds
        stop_cooldown_end = time.time() + STOP_COOLDOWN_SECONDS
        print(f"[EVENT] STOP HOLD ended. Cooldown active for
{STOP_COOLDOWN_SECONDS}s")

    # 3. Optional: Clean up the moving reset
    # You might want to keep stop_detected_once = True until the cooldown
ends
    # to be doubly sure it doesn't flicker.
    try:
        r = requests.get(f"{FLASK_URL}/is_moving", timeout=0.1)
        moving = r.json().get("moving", False)
    except:
        moving = False

    # Only reset the flag if the car is moving AND the cooldown has
passed
    if moving and time.time() > stop_cooldown_end:
        stop_detected_once = False

    # detekcija crvenog svjetla
    hsv = cv2.cvtColor(roi_frame, cv2.COLOR_BGR2HSV)
    m1 = cv2.inRange(hsv, LOWER_RED1, UPPER_RED1)
    m2 = cv2.inRange(hsv, LOWER_RED2, UPPER_RED2)
    mask = cv2.bitwise_or(m1, m2)

    mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, MORPH_KERNEL)
    mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, MORPH_KERNEL)

    contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

    detected_red = False
    red_bbox = None

```

```

if contours:
    c = max(contours, key=cv2.contourArea)
    area = cv2.contourArea(c)

    if RED_MIN_AREA <= area <= RED_MAX_AREA:
        x, y, w, h = cv2.boundingRect(c)
        if w <= LED_MAX_SIZE and h <= LED_MAX_SIZE:
            v = hsv[y:y+h, x:x+w, 2]
            mean_v = int(np.mean(v)) if v.size else 0
            if mean_v >= RED_MIN_BRIGHTNESS:
                per = cv2.arcLength(c, True)
                if per > 0:
                    circularity = 4 * np.pi * area / (per * per)
                else:
                    circularity = 0
            if circularity >= CIRCULARITY_MIN:
                detected_red = True
                red_bbox = (x, y, w, h)

red_on_history.append(1 if detected_red else 0)
if detected_red:
    red_off_history.clear()
else:
    red_off_history.append(1)

if not red_state:
    if len(red_on_history) == RED_FRAMES_ON_NEEDED and
all(red_on_history):
        red_state = True
        print("[EVENT] RED LED ON")
        #gpio_write(GPIO_RED_PIN, True)
    else:
        if len(red_off_history) == RED_FRAMES_OFF_NEEDED and
all(red_off_history):
            red_state = False
            print("[EVENT] RED LED OFF")
            #gpio_write(GPIO_RED_PIN, False)

display = frame.copy()
if USE_ROI:
    cv2.rectangle(display, (rx1, ry1), (rx2, ry2), (255, 255, 0), 1)

if best_stop:
    x, y, w, h = best_stop
    x += rx1; y += ry1

```

```

        cv2.rectangle(display, (x, y), (x+w, y+h), (0, 255, 0), 2)
        cv2.putText(display, "STOP", (x, y-6), cv2.FONT_HERSHEY_SIMPLEX,
0.7, (0, 255, 0), 2)

    if red_bbox:
        x, y, w, h = red_bbox
        x += rx1; y += ry1
        cv2.rectangle(display, (x, y), (x+w, y+h), (0, 0, 255), 2)
        cv2.putText(display, "RED", (x, y-6), cv2.FONT_HERSHEY_SIMPLEX,
0.7, (0, 0, 255), 2)

    status = "RED: ON" if red_state else "RED: OFF"
    cv2.putText(display, status, (10, display.shape[0]-10),
                cv2.FONT_HERSHEY_SIMPLEX, 0.8,
                (0, 0, 255) if red_state else (0, 255, 0), 2)

if not args.no_gui:
    cv2.imshow("Detection", display)
    cv2.imshow("Mask", mask)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

time.sleep(0.001)

except KeyboardInterrupt:
    print("Stopping...")

finally:
    cap.stop()
    if not args.no_gui:
        cv2.destroyAllWindows()
    if HAVE_GPIO:
        GPIO.cleanup()

if __name__ == "__main__":
    main()

```

stop_sign.xml je moguće pogledati klikom na link [13] koji se nalazi unutar naslova [reference](#).

8.5. KOD ZA KOMUNIKACIJU IZMEĐU MOBILNE APLIKACIJE I RASPBERRY PI

```
from flask import Flask, jsonify, request
```

```

from picamera2 import Picamera2
import RPi.GPIO as GPIO
import time
import threading

app = Flask(__name__)

# PINOUT
PIN_FWD = 4
PIN_BWD = 27
PIN_STOP = 22
PIN_LEFT = 23
PIN_RIGHT = 24
FAILSAFE = 26
PIN_HORN = 17

PINS = [PIN_FWD, PIN_BWD, PIN_STOP, PIN_LEFT, PIN_RIGHT]

GPIO.setmode(GPIO.BCM)
GPIO.setup(PINS, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(FAILSAFE, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(PIN_HORN, GPIO.OUT, initial = GPIO.HIGH)

motor_state = {"fwd": False, "bwd": False, "left": False, "right": False, "stop": False}
state_lock = threading.Lock()
last_move_time = time.time()

last_heartbeat_time = time.time() # vrijeme posljednjeg heartbeat-a
HEARTBEAT_TIMEOUT = 0.3 # 300 ms
last_seq = -1

def handle_header():
    global last_seq
    seq = int(request.headers.get("sequence"))
    if(seq > last_seq):
        last_seq = seq
        print(seq)
        return True
    return False

picam = Picamera2()
picam.configure(picam.create_video_configuration())
picam.start()

```

```

def gen():
    while True:
        frame = picam.capture_array()
        _, jpeg = cv2.imencode('.jpg', frame)
        yield (b'--frame\r\n'
               b'Content-Type: image/jpeg\r\n\r\n' +
               jpeg.tobytes() + b'\r\n')

def set_motor(cmd, value=True):
    with state_lock:
        motor_state[cmd] = value

# ----- Fail-safe heartbeat checker -----
def heartbeat_loop():
    global last_heartbeat_time
    while True:
        now = time.time()
        if now - last_heartbeat_time > HEARTBEAT_TIMEOUT:
            # ako je > timeout, posalji na failsafe pin signal, zaustavi sve
            with state_lock:
                for key in motor_state:
                    motor_state[key] = False
            GPIO.output(FAILSAFE, GPIO.LOW)
        time.sleep(0.01)
threading.Thread(target=heartbeat_loop, daemon=True).start()

@app.route("/heartbeat", methods=["POST"])
def heartbeat():
    global last_heartbeat_time
    last_heartbeat_time = time.time()
    return jsonify({"status": "ok"})

@app.route("/forward/on")
def forward_on():
    global last_move_time
    last_move_time = time.time()
    if(handle_header()):
        GPIO.output(PIN_FWD, GPIO.HIGH)
    return jsonify({"status": "forward ON"})

@app.route("/is_moving")
def is_moving():
    moving = (time.time() - last_move_time) < 0.5
    return jsonify({"moving": moving})

```

```

@app.route("/forward/off")
def forward_off():
    if(handle_header()):
        GPIO.output(PIN_FWD, GPIO.LOW)
    return jsonify({"status": "forward OFF"})

@app.route("/backward/on")
def backward_on():
    if(handle_header()):
        GPIO.output(PIN_BWD, GPIO.HIGH)
    return jsonify({"status": "backward ON"})

@app.route("/backward/off")
def backward_off():
    if(handle_header()):
        GPIO.output(PIN_BWD, GPIO.LOW)
    return jsonify({"status": "backward OFF"})

@app.route("/left/on")
def left_on():
    global last_move_time
    last_move_time = time.time()
    if(handle_header()):
        GPIO.output(PIN_LEFT, GPIO.HIGH)
    return jsonify({"status": "left ON"})

@app.route("/left/off")
def left_off():
    if(handle_header()):
        GPIO.output(PIN_LEFT, GPIO.LOW)
    return jsonify({"status": "left OFF"})

@app.route("/right/on")
def right_on():
    global last_move_time
    last_move_time = time.time()
    if(handle_header()):
        GPIO.output(PIN_RIGHT, GPIO.HIGH)
    return jsonify({"status": "right ON"})

@app.route("/right/off")
def right_off():
    if(handle_header()):
        GPIO.output(PIN_RIGHT, GPIO.LOW)
    return jsonify({"status": "right OFF"})

```

```

@app.route("/horn/on")
def horn_on():
    if(handle_header()):
        GPIO.output(PIN_HORN, GPIO.HIGH)
    return jsonify({"status": "horn ON"})

@app.route("/horn/off")
def horn_off():
    if(handle_header()):
        GPIO.output(PIN_HORN, GPIO.LOW)
    return jsonify({"status" : "horn OFF"})

@app.route("/stop")
def stop_all():
    GPIO.output(PIN_FWD, GPIO.LOW)
    GPIO.output(PIN_BWD, GPIO.LOW)
    GPIO.output(PIN_LEFT, GPIO.LOW)
    GPIO.output(PIN_RIGHT, GPIO.LOW)
    return jsonify({"status": "all stopped"})

@app.route("/")
def index():
    return "Backend running"

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000)

```

8.6. KOD ZA SIMULACIJU RADA SEMAFORA

```

const static byte Rp = 1;
const static byte Yp = 0;
const static byte Gp = 2;
const static int BASE_DELAY = 2000;
const static int RED_DELAY = 3 * BASE_DELAY;
const static int RED_YELLOW_DELAY = 2 * BASE_DELAY;
const static int GREEN_DELAY = 2 * BASE_DELAY;
const static int GREEN_YELLOW_DELAY = 1 * BASE_DELAY;

void setup() {
    pinMode(Rp, OUTPUT);
    pinMode(Yp, OUTPUT);
    pinMode(Gp, OUTPUT);
}

void loop() {

```

```

digitalWrite(Rp, HIGH);
digitalWrite(Yp, LOW);
digitalWrite(Gp, LOW);

delay(RED_DELAY);

digitalWrite(Rp, LOW);
digitalWrite(Yp, HIGH);
digitalWrite(Gp, LOW);

delay(RED_YELLOW_DELAY);

digitalWrite(Rp, LOW);
digitalWrite(Yp, LOW);
digitalWrite(Gp, HIGH);

delay(GREEN_DELAY);

digitalWrite(Rp, LOW);
digitalWrite(Yp, HIGH);
digitalWrite(Gp, LOW);

delay(GREEN_YELLOW_DELAY);}
```

8.7. KOTLIN KOD ZA ANDROID MOBILNU APLIKACIJU

Kod mobilne aplikacije je raspoređen u sljedećim fajlovima.

8.7.1. data/models/RobotStatus.kt

```

package com.example.robotcontrolapp.data.models

data class RobotStatus(
    val connected: Boolean = false,
    val moving: Boolean = false,
    val lastCommand: String = "none"
)

data class ControlResponse(
    val status: String
)

enum class Direction(val value: String) {
    FORWARD("forward"),
    BACKWARD("backward"),
```

```
    LEFT("left"),
    RIGHT("right"),
    STOP("stop")
}
```

8.7.2. data/RobotRepository.kt

```
package com.example.robotcontrolapp.data

import com.example.robotcontrolapp.data.models.ControlResponse
import com.example.robotcontrolapp.data.models.RobotStatus
import com.example.robotcontrolapp.network.RetrofitClient
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.delay
import kotlinx.coroutines.Flow
import kotlinx.coroutines.flow.flow
import kotlinx.coroutines.flow.flowOn
import kotlinx.coroutines.withContext
import retrofit2.Response

class RobotRepository {
    private val apiService = RetrofitClient.apiService

    fun startHeartbeat(): Flow<Result<Unit>> = flow {
        while (true) {
            try {
                val response = apiService.sendHeartbeat()
                if (response.isSuccessful) {
                    emit(Result.success(Unit))
                } else {
                    emit(Result.failure(Exception("Heartbeat failed")))
                }
            } catch (e: Exception) {
                emit(Result.failure(e))
            }
            delay(1000)
        }
    }
}
```

```

    }
}.flowOn(Dispatchers.IO)

suspend fun forwardOn(seq: Int): Result<ControlResponse> {
    return executeRequest { apiService.forwardOn(seq) !!}
}
suspend fun forwardOff(seq: Int): Result<ControlResponse> {
    return executeRequest { apiService.forwardOff(seq) !!}
}
suspend fun backwardOn(seq: Int): Result<ControlResponse> {
    return executeRequest { apiService.backwardOn(seq) !!}
}
suspend fun backwardOff(seq: Int): Result<ControlResponse> {
    return executeRequest { apiService.backwardOff(seq) !!}
}
suspend fun leftOn(seq: Int): Result<ControlResponse> {
    return executeRequest { apiService.leftOn(seq) !!}
}
suspend fun leftOff(seq: Int): Result<ControlResponse> {
    return executeRequest { apiService.leftOff(seq) !!}
}
suspend fun rightOn(seq: Int): Result<ControlResponse> {
    return executeRequest { apiService.rightOn(seq) !!}
}
suspend fun rightOff(seq: Int): Result<ControlResponse> {
    return executeRequest { apiService.rightOff(seq) !!}
}
suspend fun stopAll(seq: Int): Result<ControlResponse> {
    return executeRequest { apiService.stopAll(seq) !!}
}
suspend fun hornOn(seq: Int): Result<ControlResponse> {
    return executeRequest { apiService.hornOn(seq) !!}
}
}
suspend fun hornOff(seq: Int): Result<ControlResponse> {
    return executeRequest { apiService.hornOff(seq) !!}
}

fun getRobotStatusFlow(): Flow<Result<RobotStatus>> = flow {
    while (true) {
        try {
            val response = apiService.isMoving()
            if (response.isSuccessful && response.body() != null) {
                emit(Result.success(response.body()!!))
            } else {
                emit(Result.failure(Exception("Failed to get status")))
            }
        } catch (e: Exception) {
            emit(Result.failure(e))
        }
        delay(500)
    }
}.flowOn(Dispatchers.IO)

private suspend fun executeRequest(
    request: suspend () -> Response<ControlResponse>
): Result<ControlResponse> {
    return withContext(Dispatchers.IO) {
        try {
            val response = request()
            if (response.isSuccessful && response.body() != null) {
                Result.success(response.body()!!)
            }
        } catch (e: Exception) {
            Result.failure(e)
        }
    }
}

```

```

        } else {
            Result.failure(Exception("Request failed:
${response.code()}"))
        }
    } catch (e: Exception) {
    Result.failure(e)
}
}

fun updateRobotIp(ip: String, port: Int = 5000) {
    RetrofitClient.updateBaseUrl(ip, port)
}
}

```

8.7.3. network/RetrofitClient.kt

```

package com.example.robotcontrolapp.network

import okhttp3.OkHttpClient
import okhttp3.logging.HttpLoggingInterceptor
import retrofit2.Retrofit
import retrofit2.converter.gson.GsonConverterFactory
import java.util.concurrent.TimeUnit

object RetrofitClient {
    private const val DEFAULT_IP = "192.168.1.132"
    private const val DEFAULT_PORT = 5000
    private var currentBaseUrl = "http://$DEFAULT_IP:$DEFAULT_PORT/"
    private val loggingInterceptor = HttpLoggingInterceptor().apply {
        level = HttpLoggingInterceptor.Level.BODY
    }

    private val okHttpClient = OkHttpClient.Builder()
        .addInterceptor(loggingInterceptor)
        .connectTimeout(5, TimeUnit.SECONDS)
        .readTimeout(5, TimeUnit.SECONDS)
        .writeTimeout(5, TimeUnit.SECONDS)
        .build()

    private fun createRetrofit(baseUrl: String): Retrofit {
        return Retrofit.Builder()
            .baseUrl(baseUrl)
            .client(okHttpClient)
            .addConverterFactory(GsonConverterFactory.create())
    }
}

```

```

        .build()
    }

private var retrofit = createRetrofit(currentBaseUrl)
val apiService: Robot ApiService
    get() = retrofit.create(Robot ApiService::class.java)

fun updateBaseUrl(ip: String, port: Int = DEFAULT_PORT) {
    currentBaseUrl = "http://$ip:$port/"
    retrofit = createRetrofit(currentBaseUrl)
}
}

```

8.7.4. network/Robot ApiService.kt

```

package com.example.robotcontrolapp.network

import com.example.robotcontrolapp.data.models.ControlResponse
import com.example.robotcontrolapp.data.models.RobotStatus
import retrofit2.Response
import retrofit2.http.GET
import retrofit2.http.Header
import retrofit2.http.POST

interface Robot ApiService {

    @POST("heartbeat")
    suspend fun sendHeartbeat(@Header("sequence") seq : Int):
    Response<ControlResponse>
    @GET("forward/on")
    suspend fun forwardOn(@Header("sequence") seq : Int):
    Response<ControlResponse>
    @GET("forward/off")
    suspend fun forwardOff(@Header("sequence") seq : Int):
    Response<ControlResponse>
    @GET("backward/on")
    suspend fun backwardOn(@Header("sequence") seq : Int):
    Response<ControlResponse>
    @GET("backward/off")
}

```

```

        suspend fun backwardOff(@Header("sequence") seq : Int):
Response<ControlResponse>
    @GET("left/on")
    suspend fun leftOn(@Header("sequence") seq : Int): Response<ControlResponse>
    @GET("left/off")
    suspend fun leftOff(@Header("sequence") seq : Int): Response<ControlResponse>
    @GET("right/on")
    suspend fun rightOn(@Header("sequence") seq : Int): Response<ControlResponse>
    @GET("right/off")
    suspend fun rightOff(@Header("sequence") seq : Int):
Response<ControlResponse>
    @GET("stop")
    suspend fun stopAll(): Response<ControlResponse>
    @GET("horn/on")
    suspend fun hornOn(@Header("sequence") seq : Int): Response<ControlResponse>
    @GET("horn/off")
    suspend fun hornOff(@Header("sequence") seq : Int): Response<ControlResponse>
    @GET("is_moving")
    suspend fun isMoving(): Response<RobotStatus>
}

```

8.7.5. ui/components/ActionButton.kt

```

package com.example.robotcontrolapp.ui.components

import androidx.compose.foundation.Image
import androidx.compose.foundation.background
import androidx.compose.foundation.gestures.detectTapGestures
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material3.MaterialTheme
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.input.pointer.pointerInput
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.unit.dp

@Composable
fun ActionButton(
    imageId: Int,
    labelId: Int,
    isActive: Boolean,
    color: Color,
    onPress: () -> Unit,
    onRelease: () -> Unit,

```

```

        modifier: Modifier = Modifier
) {
    Box(
        modifier = modifier
        .size(80.dp)
        .clip(RoundedCornerShape(12.dp))
        .background(color)
        .pointerInput(Unit) {
            detectTapGestures(
                onPress = {
                    onPress()
                    tryAwaitRelease()
                    onRelease()
                }
            ),
            contentAlignment = Alignment.Center
        } {
            Image(
                painter = painterResource(imageId),
                contentDescription = stringResource(labelId),
                modifier = Modifier.size(40.dp),
                contentScale = ContentScale.FillBounds)
        }
    )
}

```

8.7.6. ui/components/ControlPad.kt

```

package com.example.robotcontrolapp.ui.components

import androidx.compose.foundation.background
import androidx.compose.foundation.gestures.detectTapGestures
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material.icons(Icons)
import androidx.compose.material.icons.automirrored.filled.KeyboardArrowRight
import androidx.compose.material.icons.filled.KeyboardArrowDown
import androidx.compose.material.icons.filled.KeyboardArrowLeft
import androidx.compose.material.icons.filled.KeyboardArrowRight
import androidx.compose.material.icons.filled.KeyboardArrowUp
import androidx.compose.material3.Icon
import androidx.compose.material3.MaterialTheme
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.graphics.vector.ImageVector
import androidx.compose.ui.input.pointer.pointerInput
import androidx.compose.ui.unit.dp
import com.example.robotcontrolapp.data.models.Direction
import com.example.robotcontrolapp.ui.theme.ControlButtonActive
import com.example.robotcontrolapp.ui.theme.ControlButtonInactive

@Composable
fun ControlButton(

```

```

        icon: ImageVector,
        direction: Direction,
        isPressed: Boolean,
        onPress: () -> Unit,
        onRelease: () -> Unit,
        modifier: Modifier = Modifier
    ) {
        Box(
            modifier = modifier
                .size(80.dp)
                .clip(RoundedCornerShape(12.dp))
                .background(
                    if (isPressed) ControlButtonActive
                    else ControlButtonInactive
                )
                .pointerInput(Unit) {
                    detectTapGestures(
                        onPress = {
                            onPress()
                            tryAwaitRelease()
                            onRelease()
                        }
                    )
                },
            contentAlignment = Alignment.Center
        ) {
            Icon(
                imageVector = icon,
                contentDescription = direction.value,
                tint = MaterialTheme.colorScheme.onPrimary,
                modifier = Modifier.size(40.dp)
            )
        }
    }
}

@Composable
fun ControlPad(
    currentDirection: Direction,
    onDirectionChange: (Direction) -> Unit,
    onStop: () -> Unit,
    modifier: Modifier = Modifier
) {
    Column(
        modifier = modifier,
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.spacedBy(16.dp)
    ) {
        ControlButton(
            icon = Icons.Default.KeyboardArrowUp,
            direction = Direction.FORWARD,
            isPressed = currentDirection == Direction.FORWARD,
            onPress = { onDirectionChange(Direction.FORWARD) },

```

```

        onRelease = onStop
    )

Row(
    horizontalArrangement = Arrangement.spacedBy(16.dp),
    verticalAlignment = Alignment.CenterVertically
) {
    ControlButton(
        icon = Icons.Default.KeyboardArrowLeft,
        direction = Direction.LEFT,
        isPressed = currentDirection == Direction.LEFT,
        onPress = { onDirectionChange(Direction.LEFT) },
        onRelease = onStop
    )

    ControlButton(
        icon = Icons.Default.KeyboardArrowDown,
        direction = Direction.BACKWARD,
        isPressed = currentDirection == Direction.BACKWARD,
        onPress = { onDirectionChange(Direction.BACKWARD) },
        onRelease = onStop
    )

    ControlButton(
        icon = Icons.AutoMirrored.Filled.KeyboardArrowRight,
        direction = Direction.RIGHT,
        isPressed = currentDirection == Direction.RIGHT,
        onPress = { onDirectionChange(Direction.RIGHT) },
        onRelease = onStop
    )
}
}

```

8.7.7. ui/screens/MainActivity.kt

```

package com.example.robotcontrolapp.ui.screens

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.activity.enableEdgeToEdge
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Surface
import androidx.compose.ui.Modifier
import androidx.lifecycle.viewmodel.compose.viewModel
import com.example.robotcontrolapp.ui.theme.RobotControlAppTheme
import com.example.robotcontrolapp.viewmodel.RobotViewModel

```

```

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()

        setContent {
            RobotControlAppTheme {
                Scaffold(modifier = Modifier.fillMaxSize()) { innerPadding ->
                    Surface(
                        modifier = Modifier
                            .fillMaxSize()
                            .padding(innerPadding),
                        color = MaterialTheme.colorScheme.background
                    ) {
                        val viewModel: RobotViewModel = viewModel()
                        RobotControlScreen(viewModel = viewModel)
                    }
                }
            }
        }
    }
}

```

8.7.8. ui/screens/RobotControlScreen.kt

```

package com.example.robotcontrolapp.ui.screens

import androidx.compose.foundation.background
import androidx.compose.foundation.border
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material.icons(Icons
import androidx.compose.material.icons.filled.Settings
import androidx.compose.material3.*
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.compose.ui.window.Dialog
import androidx.lifecycle.compose.collectAsStateWithLifecycle
import com.example.robotcontrolapp.ui.componentsActionButton
import com.example.robotcontrolapp.ui.components.ControlPad
import com.example.robotcontrolapp.ui.theme.EmergencyRed
import com.example.robotcontrolapp.ui.theme.StatusWarning
import com.example.robotcontrolapp.viewmodel.RobotViewModel
import com.example.robotcontrolapp.R
import com.example.robotcontrolapp.ui.theme.EmergencyRedActive
import com.example.robotcontrolapp.ui.theme.StatusWarningActive

```

```

import com.example.robotcontrolapp.viewmodel.StopReason

@Composable
fun RobotControlScreen(
    viewModel: RobotViewModel,
    modifier: Modifier = Modifier
) {
    val uiState by viewModel.uiState.collectAsStateWithLifecycle()

    Box(
        modifier = modifier
            .fillMaxSize()
            .background(MaterialTheme.colorScheme.background)
            .padding(24.dp)
    ) {
        IconButton(
            onClick = { viewModel.toggleSettings() },
            modifier = Modifier.align(Alignment.TopEnd)
        ) {
            Icon(
                imageVector = Icons.Default.Settings,
                contentDescription = "Settings",
                tint = MaterialTheme.colorScheme.onBackground,
                modifier = Modifier.size(32.dp)
            )
        }
    }

    Row(
        modifier = Modifier.align(Alignment.Center).fillMaxWidth(),
        horizontalArrangement = Arrangement.SpaceBetween,
        verticalAlignment = Alignment.CenterVertically
    ) {
        Column(verticalArrangement = Arrangement.spacedBy(24.dp)) {
            ActionButton(
                imageId = R.drawable.horn,
                labelId = R.string.horn,
                color = if (uiState.isHornActive) StatusWarningActive else
StatusWarning,
                isActive = uiState.isHornActive,
                onPress = { viewModel.activateHorn() },
                onRelease = { viewModel.releaseHorn() }
            )
        }

        ControlPad(
            currentDirection = uiState.currentDirection,
            onDirectionChange = { viewModel.sendCommand(it) },
            onStop = { viewModel.stopCommand() }
        )
    }
}

Column(

```

```

        modifier = Modifier
            .align(Alignment.BottomCenter)
            .padding(bottom = 20.dp)
            .fillMaxWidth(),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.spacedBy(10.dp)
    ) {
        if (!uiState.isConnected && uiState.errorMessage != null) {
            InfoWhiteBox(
                message = uiState.errorMessage!!,
                isError = true
            )
        }

        if (uiState.stopReason != StopReason.NONE) {
            val stopMessage = when (uiState.stopReason) {
                StopReason.RED_LIGHT -> "Robot stopped: RED LIGHT DETECTED!"
                StopReason.STOP_SIGN -> "Robot stopped: STOP SIGN DETECTED!"
                else -> ""
            }
            if (stopMessage.isNotEmpty()) {
                InfoWhiteBox(
                    message = stopMessage,
                    isError = false,
                    onDismiss = { viewModel.handleStopReason(StopReason.NONE) }
                )
            }
        }

        if (uiState.isConnected) {
            Card(
                colors = CardDefaults.cardColors(
                    containerColor = MaterialTheme.colorScheme.surfaceVariant
                )
            ) {
                Text(
                    text = "Robot Status: ${if (uiState.status.moving) "Moving" else "Stopped"}",
                    modifier = Modifier.padding(16.dp),
                    style = MaterialTheme.typography.bodyLarge
                )
            }
        }
    }

    if (uiState.showSettings) {
        SettingsDialog(
            onDismiss = { viewModel.toggleSettings() },
            onConnect = { ip, port ->
                viewModel.connectToRobot(ip, port)
            }
        )
    }
}

```

```

        viewModel.toggleSettings()
    }
}
}

@Composable
fun InfoWhiteBox(
    message: String,
    isError: Boolean,
    onDismiss: (() -> Unit)? = null
) {
    Card(
        modifier = Modifier
            .fillMaxWidth(0.95f)
            .border(2.dp, if (isError) Color.Red else Color.Black,
RoundedCornerShape(12.dp)),
        shape = RoundedCornerShape(12.dp),
        colors = CardDefaults.cardColors(containerColor = Color.White),
        elevation = CardDefaults.cardElevation(8.dp)
    ) {
        Row(
            modifier = Modifier
                .padding(horizontal = 16.dp, vertical = 12.dp)
                .fillMaxWidth(),
            verticalAlignment = Alignment.CenterVertically,
            horizontalArrangement = Arrangement.SpaceBetween
        ){
            Text(
                text = message,
                color = Color.Black,
                fontSize = 14.sp,
                fontWeight = FontWeight.Bold,
                modifier = Modifier.weight(1f)
            )

            if (onDismiss != null) {
                TextButton(onClick = onDismiss, contentPadding =
PaddingValues(start = 8.dp)) {
                    Text("OK", color = Color.Blue, fontWeight = FontWeight.Black,
fontSize = 15.sp)
                }
            }
        }
    }
}

@Composable
fun SettingsDialog(
    onDismiss: () -> Unit,
    onConnect: (String, Int) -> Unit
) {

```

```

var ipAddress by remember { mutableStateOf("192.168.1.100") }
var port by remember { mutableStateOf("5000") }

Dialog(onDismissRequest = onDismiss) {
    Card(
        modifier = Modifier.fillMaxWidth().padding(16.dp),
        shape = RoundedCornerShape(16.dp)
    ) {
        Column(
            modifier = Modifier.padding(24.dp),
            verticalArrangement = Arrangement.spacedBy(16.dp)
        ) {
            Text(text = "Postavke Robota", style =
MaterialTheme.typography.titleLarge)

            OutlinedTextField(
                value = ipAddress,
                onValueChange = { ipAddress = it },
                label = { Text("IP Adresa") },
                modifier = Modifier.fillMaxWidth(),
                singleLine = true
            )

            OutlinedTextField(
                value = port,
                onValueChange = { port = it },
                label = { Text("Port") },
                modifier = Modifier.fillMaxWidth(),
                singleLine = true
            )

            Row(
                modifier = Modifier.fillMaxWidth(),
                horizontalArrangement = Arrangement.spacedBy(8.dp)
            ) {
                TextButton(onClick = onDismiss, modifier =
Modifier.weight(1f)) {
                    Text("Otkaži")
                }

                Button(
                    onClick = {
                        val portInt = port.toIntOrNull() ?: 5000
                        onConnect(ipAddress, portInt)
                    },
                    modifier = Modifier.weight(1f)
                ) {
                    Text("Poveži")
                }
            }
        }
    }
}

```

```
    }  
}
```

8.7.9. viewmodel/RobotViewModel.kt

```
package com.example.robotcontrolapp.viewmodel  
  
import androidx.lifecycle.ViewModel  
import androidx.lifecycle.viewModelScope  
import com.example.robotcontrolapp.data.RobotRepository  
import com.example.robotcontrolapp.data.models.Direction  
import com.example.robotcontrolapp.data.models.RobotStatus  
import kotlinx.coroutines.Job  
import kotlinx.coroutines.flow.MutableStateFlow  
import kotlinx.coroutines.flow.StateFlow  
import kotlinx.coroutines.flow.asStateFlow  
import kotlinx.coroutines.flow.catch  
import kotlinx.coroutines.flow.updateAndGet  
import kotlinx.coroutines.launch  
  
enum class StopReason {  
    NONE, RED_LIGHT, STOP_SIGN  
}  
  
data class RobotUiState(  
    val isConnected: Boolean = false,  
    val status: RobotStatus = RobotStatus(),  
    val currentDirection: Direction = Direction.STOP,  
    val isHornActive: Boolean = false,  
    val errorMessage: String? = "App is not connected to the server. Please try  
changing the IP address or port in the settings.",  
    val isLoading: Boolean = false,
```

```

    val showSettings: Boolean = false,
    val stopReason: StopReason = StopReason.NONE,
    val currentIp: String = "192.168.1.132",
    val currentPort: Int = 5000
    val sequence: Int = 0
)

class RobotViewModel : ViewModel() {

    private val repository = RobotRepository()
    private val _uiState = MutableStateFlow(RobotUiState())
    val uiState: StateFlow<RobotUiState> = _uiState.asStateFlow()

    private var statusJob: Job? = null
    private var heartbeatJob: Job? = null

    private fun setConnectionError() {
        val state = _uiState.value
        val message = "Failed connection with the server at IP:  

${state.currentIp} and Port: ${state.currentPort}. Please try changing the IP  

address or port in the settings."
        _uiState.value = _uiState.value.copy(
            errorMessage = message,
            isConnected = false,
            isLoading = false
        )
    }

    fun connectToRobot(ip: String, port: Int = 5000) {
        viewModelScope.launch {
            _uiState.value = _uiState.value.copy(isLoading = true, currentIp = ip, currentPort = port)

            try {
                repository.updateRobotIp(ip, port)
                startHeartbeat()
                startStatusUpdates()
                _uiState.value = _uiState.value.copy(
                    isConnected = true,
                    isLoading = false,
                    errorMessage = null
                )
            } catch (e: Exception) {
                setConnectionError()
            }
        }
    }

    fun sendCommand(direction: Direction) {
        viewModelScope.launch {
            val seq = _uiState.updateAndGet {
                it.copy(

```

```

        currentDirection = direction,
        stopReason = StopReason.NONE,
        sequence = it.sequence + 1
    )
}.sequence
val result = when (direction) {
    Direction.FORWARD -> repository.forwardOn(seq)
    Direction.BACKWARD -> repository.backwardOn(seq)
    Direction.LEFT -> repository.leftOn(seq)
    Direction.RIGHT -> repository.rightOn(seq)
    Direction.STOP -> repository.stopAll()
}
result.onSuccess {
    _uiState.value = _uiState.value.copy(errorMessage = null,
isConnected = true)
}.onFailure {
    setConnectionError()
}
}
}
}

fun stopCommand() {
    viewModelScope.launch {
        val currentDir = _uiState.value.currentDirection

        val seq = _uiState.updateAndGet {
            it.copy(
                currentDirection = Direction.STOP,
                sequence = it.sequence + 1
            )
        }.sequence

        val result = when (currentDir) {
            Direction.FORWARD -> repository.forwardOff(seq)
            Direction.BACKWARD -> repository.backwardOff(seq)
            Direction.LEFT -> repository.leftOff(seq)
            Direction.RIGHT -> repository.rightOff(seq)
            Direction.STOP -> repository.stopAll()
        }
        result.onFailure { setConnectionError() }
    }
}

fun activateHorn() {
    viewModelScope.launch {
        val seq = _uiState.updateAndGet {
            it.copy(
                isHornActive = true,
                sequence = it.sequence + 1
            )
        }.sequence
    }
}

```

```

        repository.hornOn(seq).onSuccess {
            _uiState.value = _uiState.value.copy(errorMessage = null,
isConnected = true)
        }. onFailure {
            setConnectionError()
        }
    }
}

fun releaseHorn() {
    viewModelScope.launch {
        val seq = _uiState.updateAndGet {
            it.copy(
                isHornActive = false,
                sequence = it.sequence + 1
            )
        }.sequence
        repository.hornOff(seq).onFailure { setConnectionError() }
    }
}

private fun startHeartbeat() {
    heartbeatJob?.cancel()
    heartbeatJob = viewModelScope.launch {
        repository.startHeartbeat()
        .catch { setConnectionError() }
        .collect { result ->
            result.onFailure { setConnectionError() }
        }
    }
}

private fun startStatusUpdates() {
    statusJob?.cancel()
    statusJob = viewModelScope.launch {
        repository.getRobotStatusFlow()
        .catch { setConnectionError() }
        .collect { result ->
            result.onSuccess { status ->
                _uiState.value = _uiState.value.copy(
                    status = status,
                    isConnected = true,
                    errorMessage = null
                )
            }.onFailure { setConnectionError() }
        }
    }
}

fun handleStopReason(reason: StopReason) {
    _uiState.value = _uiState.value.copy(stopReason = reason)
}

```

```

        fun toggleSettings() {
            _uiState.value = _uiState.value.copy(showSettings =
!_uiState.value.showSettings)
        }

        fun clearError() {
            _uiState.value = _uiState.value.copy(errorMessage = null)
        }
    }
}

```

8.7.10. Ostala podešavanja za rad aplikacije

Kako bi aplikacija komunicirala sa Raspberry PI, u AndroidManifest.xml file dodane su sljedeće permisije:

```

<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>

```

Zatim, u file-u build.gradle.kts su dependencies prošireni sa sljedećim kodom:

```

// =====
// CORE ANDROID
// =====
implementation("androidx.core:core-ktx:1.12.0")
implementation("androidx.lifecycle:lifecycle-runtime-ktx:2.7.0")
implementation("androidx.activity:activity-compose:1.8.2")

// =====
// JETPACK COMPOSE
// =====
implementation(platform("androidx.compose:compose-bom:2024.01.00"))
implementation("androidx.compose.ui:ui")
implementation("androidx.compose.ui:ui-graphics")
implementation("androidx.compose.ui:ui-tooling-preview")
implementation("androidx.compose.material3:material3")
implementation("androidx.compose.material:material-icons-extended")

// =====
// LIFECYCLE & VIEWMODEL
// =====
implementation("androidx.lifecycle:lifecycle-viewmodel-compose:2.7.0")
implementation("androidx.lifecycle:lifecycle-runtime-compose:2.7.0")
implementation("androidx.lifecycle:lifecycle-viewmodel-ktx:2.7.0")

// =====
// NAVIGATION (ako bude potrebno kasnije)
// =====
implementation("androidx.navigation:navigation-compose:2.7.6")

```

```
// =====
// NETWORKING - RETROFIT & OKHTTP
// =====
implementation("com.squareup.retrofit2:retrofit:2.9.0")
implementation("com.squareup.retrofit2:converter-gson:2.9.0")
implementation("com.squareup.okhttp3:okhttp:4.12.0")
implementation("com.squareup.okhttp3:logging-interceptor:4.12.0")

// Gson za JSON parsing
implementation("com.google.code.gson:gson:2.10.1")

// =====
// IMAGE LOADING - COIL
// =====
implementation("io.coil-kt:coil-compose:2.5.0")
implementation("io.coil-kt:coil-gif:2.5.0")

// =====
// COROUTINES
// =====
implementation("org.jetbrains.kotlinx:kotlinx-coroutines-android:1.7.3")
implementation("org.jetbrains.kotlinx:kotlinx-coroutines-core:1.7.3")
```

Vizuelni izgled aplikacije definisan je unutar paketa ui/theme, gdje datoteke Color.kt, Type.kt i Theme.kt služe za konfiguraciju palete boja, tipografije i opšte teme korisničkog interfejsa. Grafički resursi, poput ikona za kontrolne tipke (buttons), smješteni su u drawable folder i koriste se za vizuelnu reprezentaciju komandi za upravljanje robotom.

REFERENCE

[1] Cyclone II Device Handbook, Volume 1 – Architecture Chapter

https://cdrdv2-public.intel.com/654843/cyc2_cii51002.pdf

[2] Cyclone II Device Family Data Sheet

<https://home.etf.rs/~vm/os/vlsi/razno/izabrano%20iz%20Altera%20Cyclone%202%20datasheet.pdf>

[3] <https://www.raspberrypi.org/>

[4] <https://www.raspberrypi.com/documentation/>

[5] <https://www.espressif.com/en/products/socs/esp8266>

[6] <https://components101.com/sensors/ultrasonic-sensor-working-pinout-datasheet>

[7] <https://www.logitech.com/en-us/software>

[8] <https://components101.com/sensors/ir-sensor-module>

[9]

<https://www.electronicsforu.com/technology-trends/learn-electronics/ir-led-infrared-sensor-basics>

[10] <https://learn.sparkfun.com/tutorials/light-emitting-diodes-leds/all>

[11] <https://www.st.com/en/motor-drivers/l298.html>

[12] <https://components101.com/misc/buzzer-pinout-working-datasheet>

[13] <https://github.com/mahirsuljic-fet/PSoC-Projekat/tree/main>