RAZVOJ PROGRAMSKE PODRŠKE ZA RASPBERRY PI MOBILNU PLATFORMU NAMIJENJENU SIMULACIJI KRETANJA AUTONOMNOG VOZILA

Petrik, Krunoslav

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

RAZVOJ PROGRAMSKE PODRŠKE ZA RASPBERRY PI MOBILNU PLATFORMU NAMIJENJENU SIMULACIJI KRETANJA AUTONOMNOG VOZILA

Završni rad

Krunoslav Petrik

Osijek, 2021.

SADRŽAJ

1. UVOD 1

1.1. Zadatak Završnog rada 1

2. PREGLED PODRUČJA TEME 2

2.1. Raspberry Pi 2

2.2. Alphabot mobilna platforma 3

2.2.1. Dijelovi mobilne platforme AlphaBot 3

2.2.2. Glavni modul za kontrolu 4

2.2.3. Modul za kontrolu motora 4

2.1. Detekcija kolizije 5

2.2. Računanje udaljenosti do prepoznatog objekta 5

2.3. Haarove značajke 6

2.4. Klasifikacija autonomnih vozila 7

2.5. Opisi algoritama 8

2.5.1. Algoritam mobilne platforme 8

2.5.2. Algoritam udaljenog računala 9

3. IZRADA PROGRAMSKE PODRŠKA MODELA 10

3.1. Spajanje na mobilnu platformu 10

3.1.1. Konfiguracija mobilne platforme 10

3.1.2. Priprema Python okruženja na mobilnoj platformi 11

3.2. Prijenos slike Pi kamere na mrežu 12

3.2.1. Instalacija paketa za prijenos slike na mrežu 12

3.3. Detekcija kolizije 14

3.4. Prepoznavanje objekata 14

3.4.1. Programska implementacija prepoznavanja objekta 14

3.4.2. Programska implementacija izračuna udaljenosti 15

3.5. Udaljeno spajanje na mobilnu platformu putem socketa 16

3.6. Kretanje mobilne platforme **17**
3.6.1. Praktično rješenje s udaljenog računala 18
3.7. Testiranje rada mobilne platforme **20**
4. ZAKLJUČAK **24**
5. LITERATURA **25**
SAŽETAK **26**
ABSTRACT **27**
ŽIVOTOPIS **28**
PRILOZI **29**

1. UVOD

Industrija autonomnih vozila se napretkom računalne tehnologije, pogotovo malih ugrađenih računala, naglo razvija. Taj napredak je omogućio da ugrađena računala veličine kreditne kartice imaju mogućnost adekvatno odigrati ulogu centralnog sustava za kontrolu autonomnog računala. Da bi se vozilo smartalo autonomnim drugog nivoa po klasifikaciji *SAE (*engl. *Society of Automotive Engineers)*, ono mora imati mogućnost praćenja puta (u ovom slučaju ceste) te izvršavanja najosnovnijih radnji vozila: stajanjem na znak stop, izbjegavanje sudara s objektom te regulaciju brzine s obzirom na maksimalnu dozvoljenu.

U drugom poglavlju opisane su značajke *Raspberry Pi* i *AlphaBot* mobilna platforme te njhovi pripadajući djelovi i značajke. Također, se navodi kratko područje teme te opis teoretskih rješenja predmeta rada. U trećem poglavlju opisano je načelo spajanja s mobilnom platformom, programski kod te testiranje autonomnosti mobilne platforme, a u četvrtom poglavlju iznesen je zaključak na temelju rezultata.

1.1. Zadatak Završnog rada

Cilj zadatka je obrađivati videozapis koju mobilna platforma šalje u stvarnom vremenu, te pomoću *OpenCV* (eng. Open Computer Vision) biblioteke otvorenog koda za računalni vid, izraditi algoritam za praćenje puta i prepoznavanja odgovarajućih objekata. Na osnovu obrađene slike, poslati odgovarajuću povratnu informaciju s račnuala na mobilnu platformu i time kontrolirati ponašanje mobilne platforme.

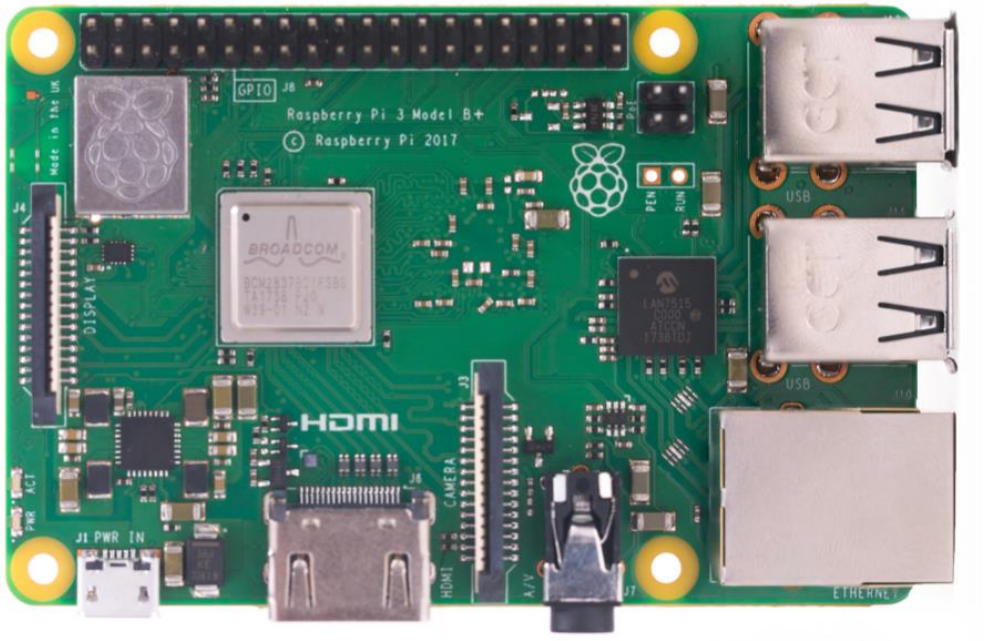
2. PREGLED PODRUČJA TEME

Da bi se zadatak mogao potpuno izvršiti, potrebno je koristiti sistem na čipu *Raspberry Pi 3* udruge *Pi foundation* na čije *GPIO* pinove je spojena mobilna platforma *AlphaBot*.

2.1. Raspberry Pi

Raspberry Pi je računalo veličine kreditne kartice koje sadrži brojna sučelja za vanjsko spajanje i komunikaciju.

AlphaBot2 mobilna platforma na prednjem dijelu ima namontiran ultrazvučni senzor objekata. Sastoji se od odašiljača ultrazvučnog signala te prijemnika istog ultrazvučnog signala. Slika 2.1. prikazuje izgled Raspberry Pi računala.



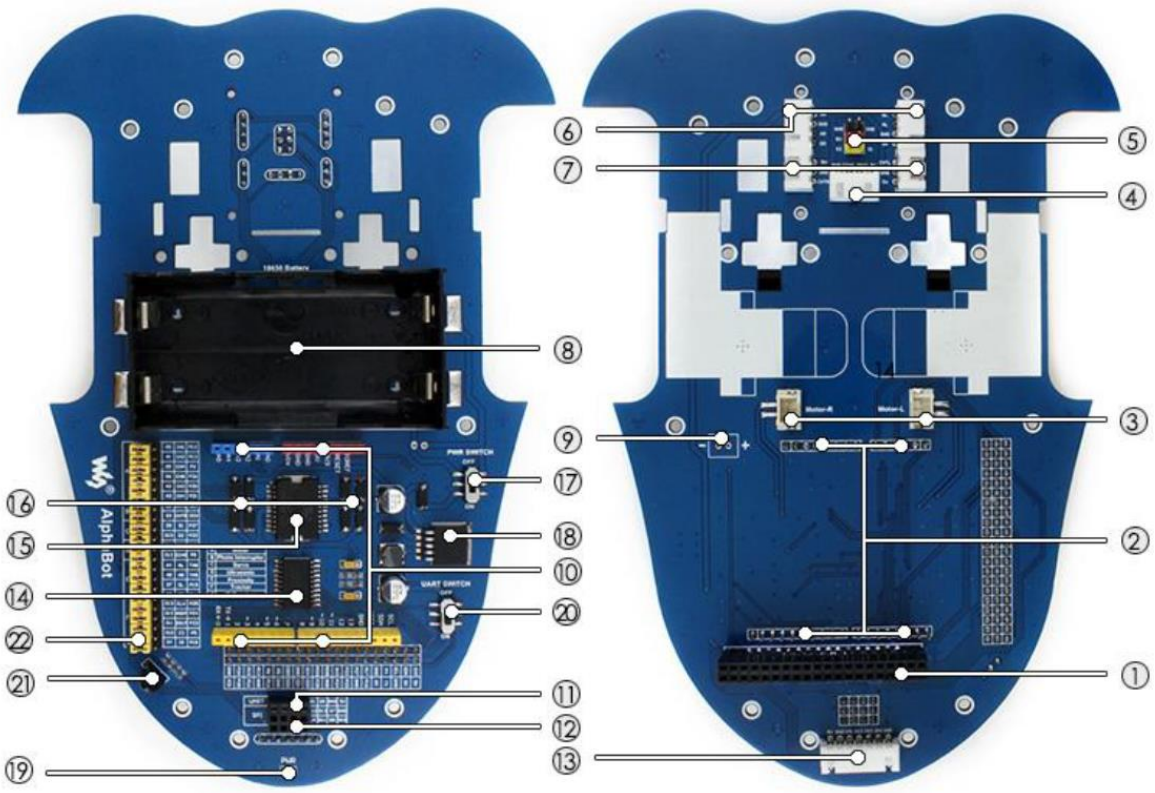
Slika 2.1 Raspberry Pi 3B+ [1]

Računalo *Raspberry Pi 3B+* posjeduje 64-bitni četverojezgreni procesor takta 1.4 gigaherza, dvopojasni 2.4 GHz i 5GHz bežični LAN prijamnik, ethernet priključak te GPIO naglavlje s 40 pinova. [1]

2.2. Alphabot mobilna platforma

AlphaBot je mobilna platforma namjenjena za razvoj te je kompatibilna s Raspberry Pi/Arduino računalima. Sastoji se od AlphaBot matične ploče, pokretne šasije te svega ostalog potrebnog za kretanje modela. [2]

2.2.1. Dijelovi mobilne platforme AlphaBot



Slika 2.2 Presjek AlphaBot 2 mobilne platofrme [2]

Dijelovi mobilne platforme prema slici 2.2. su:

1. sučelje za Rapsberry Pi,
2. sučelje za Arduino,
3. sučelje za motor,
4. sučelje za ultrazvučni modul,
5. sučelje za servo modul,
6. sučelje za modul izbjegavanja prepreka,
7. sučelje za detektiranje brzine modela,
8. držač za baterije,
9. ulaz za napajanje,

10. naglavlje za Arduino proširenja,
11. UART sučelje,
12. SPI sučelje,
13. sučelje za modul praćenja linija,
14. TLC1534 : čip koji omogućava Pi-u da pristupi analognim senzorima,
15. L298P ,
16. schottkyjeva dioda,
17. prekidač za napon,
18. LM2596,
19. indikator napona,
20. UART prekidač,
21. IR prijemnik,
22. Raspberry Pi/Arduino izbornik.

2.2.2. Glavni modul za kontrolu

Glavni modul za kontrolu je ključni dio pametnog robota. AlphaBot daje sučelja i za Arduino i za Raspberry Pi. Može se birati između te dvije opcije ili se mogu koristiti zajedno.

2.2.3. Modul za kontrolu motora

AlphaBot platforma za kontrolu motora koristi L298P chip od tvrtke *ST Electronics* te je ono ključ za kontrolu mobilne platforme.

Tablica 2.1 Pinovi AlphaBot 2 mobilne platforme [2]

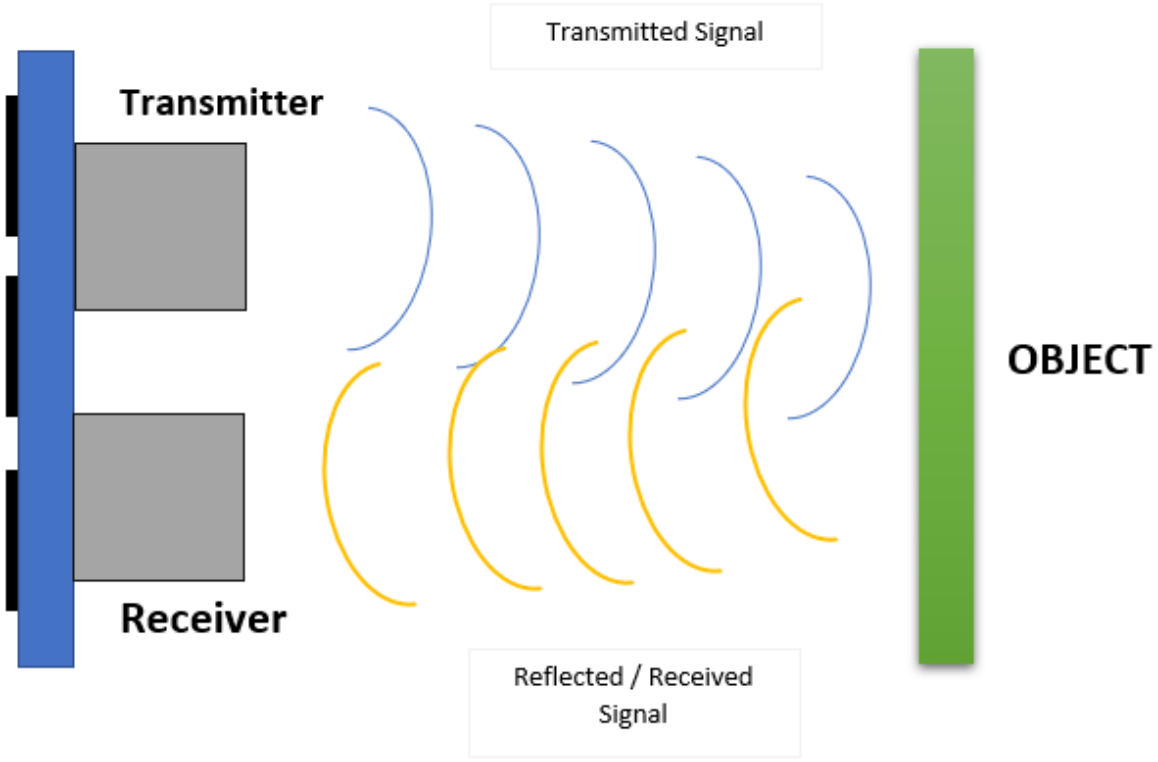
Interfaces	Raspberry Pi	Arudino
IN1	P12	A0
IN2	P13	A1
ENA	P6	D5
IN3	P20	A2
IN4	P21	A3
ENB	P26	D6

Tablica 2.1 Pinovi AlphaBot 2 mobilne platforme predstavlja pinove te sučelja na koja su spojena. IN1 i IN2 su spojeni na lijevi motor, a IN3 i IN4 su spojeni na desni motor. ENA i ENB su *output*

enable pinovi, aktivni na visoki napon. Ako je do njih doveden visoki napon, PWM puls izlazi iz IN1, IN2, IN3, IN4 da bi se mogla kontrolirati brzina motora robota.

2.1. Detekcija kolizije

Za detekciju kolizije koristit će se ultrazvučni senzor ugrađen na mobilnoj platformi s prednje strane robota kao što je prikazano na sliciSlika 2.3.

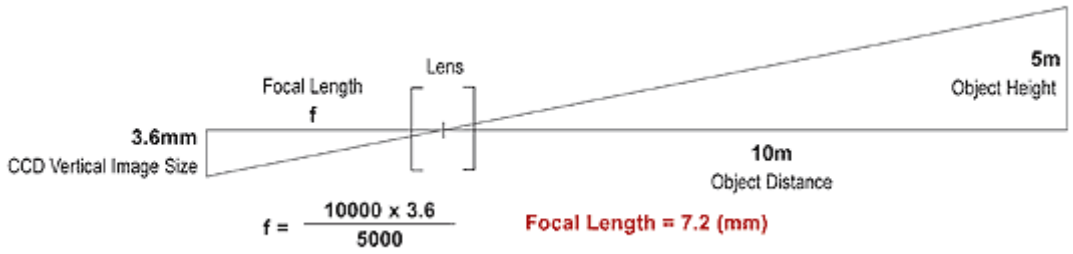


Slika 2.3 Vizualizacija rada ultrazvučnog senzora [3]

Da bi se dobila udaljenost objekta od prijemnika, odašiljač šalje ultrazvučni signal koji se odbija od objekt te prijemnik prima odbijeni val. Mjerenjem razlike vremena od odašiljanja do primanja ultrazvučnog vala, može se izračunadi udaljenost do objekta jer je poznata brzina kretanja ultrazvučnog vala koja iznosi ~34000 cm/s [4].

2.2. Računanje udaljenosti do prepoznatog objekta

Običnom kamerom je moguće izračunati udaljenost do objekta na slici ako je već poznata stvarna širina ili visina objekta te žarišna udaljenost leće kamere [5]. U slučaju da žarišna udaljenost leće nije poznata, i nju je moguće izračunati zbog poučka o sličnosti trokuta kako je na slici 2.4. prikazano



Slika 2.4 Vizualizacija leće kamere i objekta [6]

Iz priložene slike Slika 2.4. može se izvesti formula za izračun žarišne udaljenosti leće:

$$f = \frac{D * w'}{w} \ [mm] \tag{2.1.}$$

gdje je:

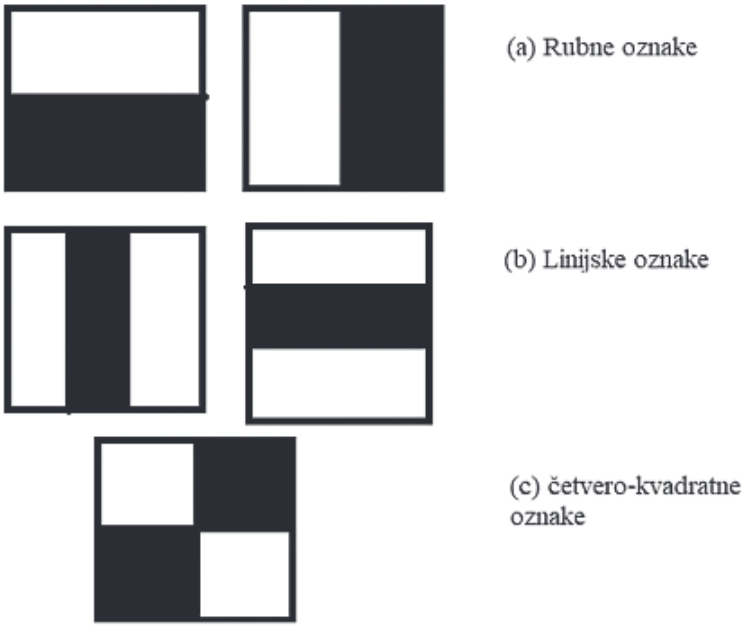
- f – žarišna udaljenost izražena u milimetrima,
- D – udaljenost objekta od kamere u milimetrima,
- w' – širina objekta na slici izražena u pikselima,
- w – stvarna širina objekta izražena u milimetrima.

Ako je već poznata žarišna dulijna, iz formule 2.1. se može dobiti izraz za udaljenost od kamere

$$D = \frac{f * w}{w'} \ [mm] \tag{2.2.}$$

2.3. Haarove značajke

Prvo objavljene 2001. godine [7], haarove značajke predstavljaju alternativu ostalim algoritmima za prepoznavanje objekata. Detektiranje objekata pomoću haarovih značajki znatno je brža od prijašnih algoritama te dovoljno brza za detekciju u stvarnom vremenu. Osnovni princip rada jest obrađivanje slike tako da se uspoređuju elementi slike sa značajkama na slici. Haarove značajke su prikazane slikom Slika 2.5.



Slika 2.5 Osnovne haar-ove značajke [8]

Testiranje svih značajki na svakom potprozoru bi dugo trajalo te ne bi bilo učikovito stoga se testiranja provode na skupu značajki isfiltrirane procesom treniranja, čime se kreiraju klasifikatori kaskade. [8]

2.4. Klasifikacija autonomnih vozila

Udruga SAE predstavlja profesionalnu udrugu i organizaciju za razvoj standarda u inžinjerstvu. Godine 2018. predlaže standard za klasifikaciju autonomnih vozila podjeljenje u razrede:

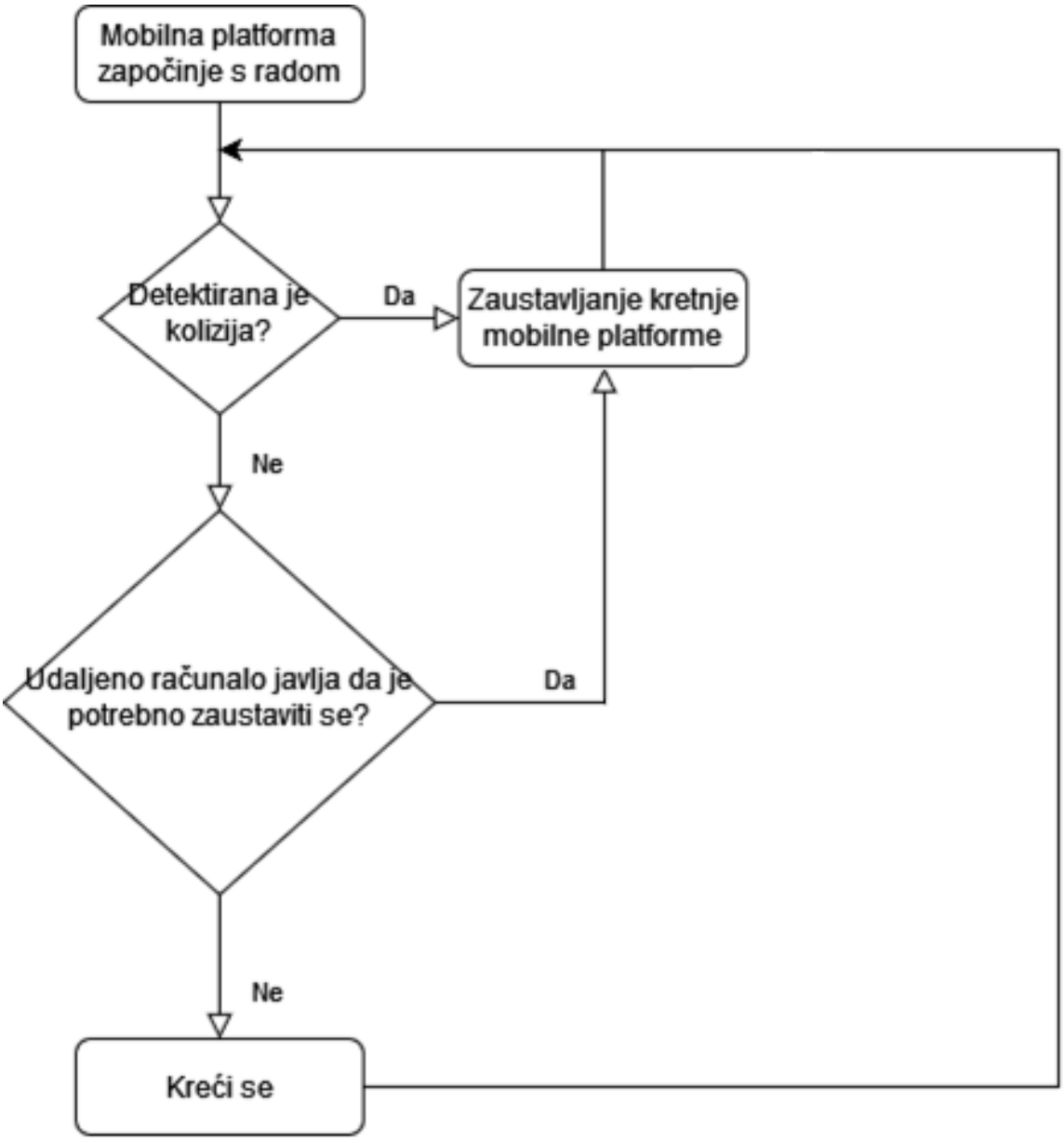
1. Nulta razina podrazumijeva vozila bez automatizacije. Čitava odgovornost je na ljudskom vozaču na svim dinamičkim aspektima vožnje, čak i ako postoje sustavi za upozoravanje.
2. Prva razina predstavlja pomoć vozaču. Sustav pomaže vozaču tako da automatski regulira brzinu vožne i/ili pomaže pri skretanju ili održavanju pravca.
3. Druga razina znači djelomičnu automatizaciju upravljanja automobilom. Sustav ima mogućnost automatskog skretanja i reguliranja brzine kretanja, a ljudski vozač promatra okruženje vožnje te preuzima kontrolu nad vozilom ako dođe do greške.
4. Treća razina predstavlja uvjetovanu automatizaciju. Svi dinamički elementi vožnje su automatizirani, sustav analizira okoliš kretanja te propisno reagira, od ljudskog vozača se očekuje da intervenira ako dođe do te potrebe
5. Četvrta razina podrazumijeva visoku automatizaciju. Za razliku od treće razine, ako ljudski vozač ne reagira na zahtjev za intervenciju, sustav može zaustaviti vozilo na siguran način

6. Peta razina znači potpunu automatizaciju. Svi načini vožnje i uvjeti okruženja su podržani, sve elemente vožnje odrađuje sustav za autonomno upravljanje.

2.5. Opisi algoritama

2.5.1. Algoritam mobilne platforme

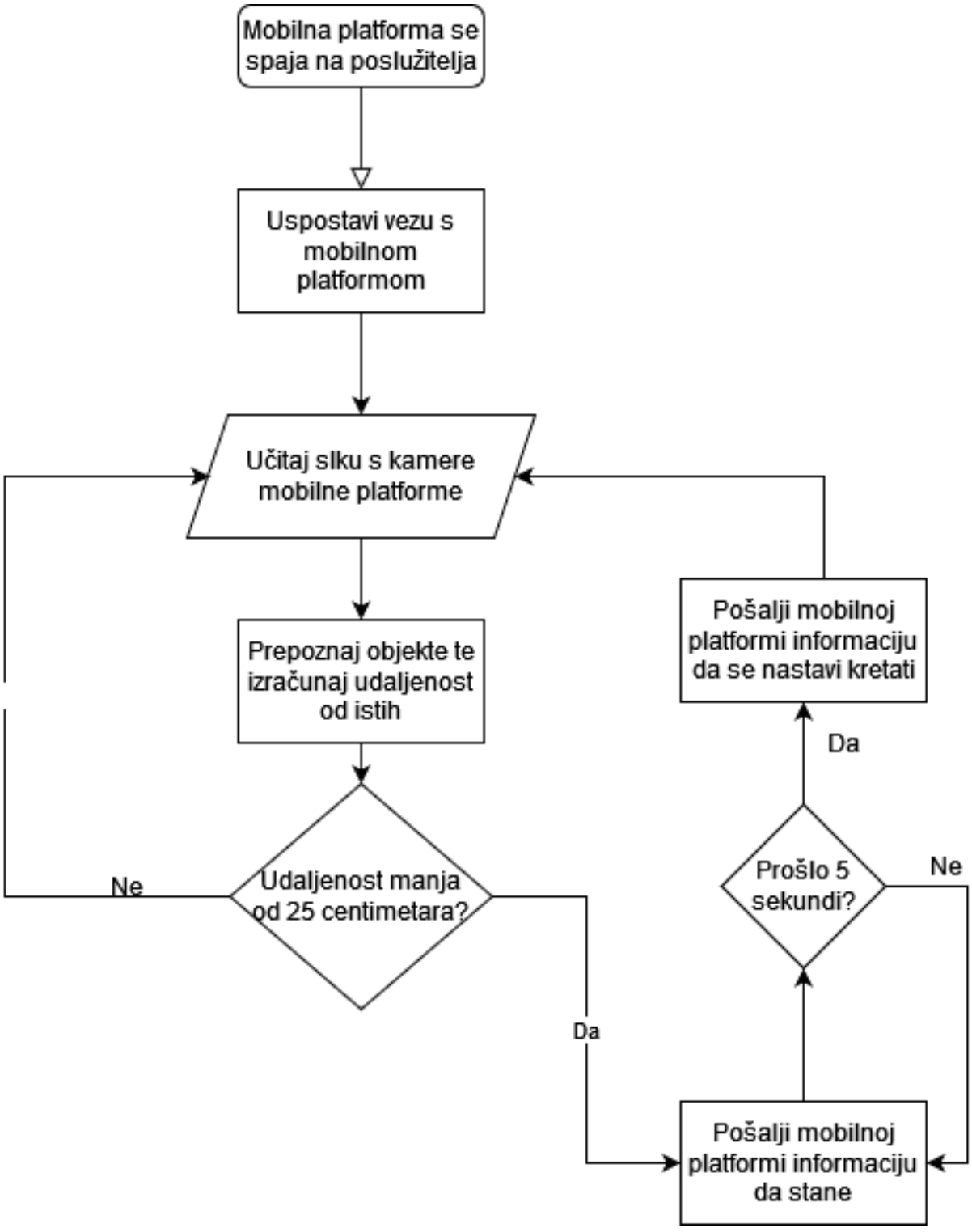
Na slici 2.6. je prikazan dijagram toka algoritma koji će se izvršavati na mobilnoj platformi. Mobilna platforma će se gibati ovisno o detektiranoj koliziji ultrazvučnog senzora te povratnoj informaciji s udaljenog računala.



Slika 2.6 Dijagram toka algoritma mobilne platforme

2.5.2. Algoritam udaljenog računala

Na slici 2.7. je prikazan algoritam koji će se izvršavati na udaljenom računalu. Udaljeno računalo uspotavlja vezu s mobilnom platformu te obrađuje sliku i ovisno o rezultatu te obrade šalje određenu povratnu informaciju mobilnoj platformi.



Slika 2.7 Dijagram toka algoritma izvođenog na udaljenom računalu

3. IZRADA PROGRAMSKE PODRŠKA MODELA

Povezivanje na model ostvaruje se programskim paketom *Putty* koji pruža sučelje korisniku za korištenje SSH protokola. Moguće je udaljeno povezivanje ako su klijent i poslužitelj na istoj lokalnoj mreži.

Za kontroliranje motora i pristup optičkom senzoru koristit ćemo službenu knjižnicu *AlphaBot* mobilne platforme otvorenog koda u programskom jeziku *Python*.

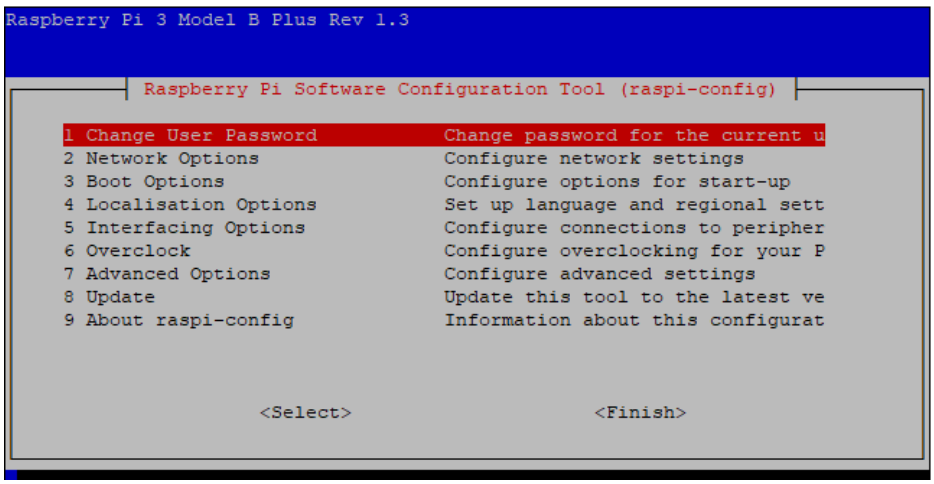
Za prepoznavanje objekata koristit će se programska podrška *Open CV* te pripadajući algoritam za pronalaženje Haar-ovih značajki.

3.1. Spajanje na mobilnu platformu

Da bi bilo moguće udaljeno se spojiti na mobilnu platformu s računala, prvo je potrebno konfigurirati mobilnu platformu za spajanje na lokalnu mrežu. Da bi se postiglo navedeno, treba se izabrati način na koji ćemo pristupiti konfiguracijskim podatcima mobilne platforme. Za potrebe ovog rada, koristit će se spajanje na izlazni uređaj pomoću HDMI izlaza na mobilnoj platformi.

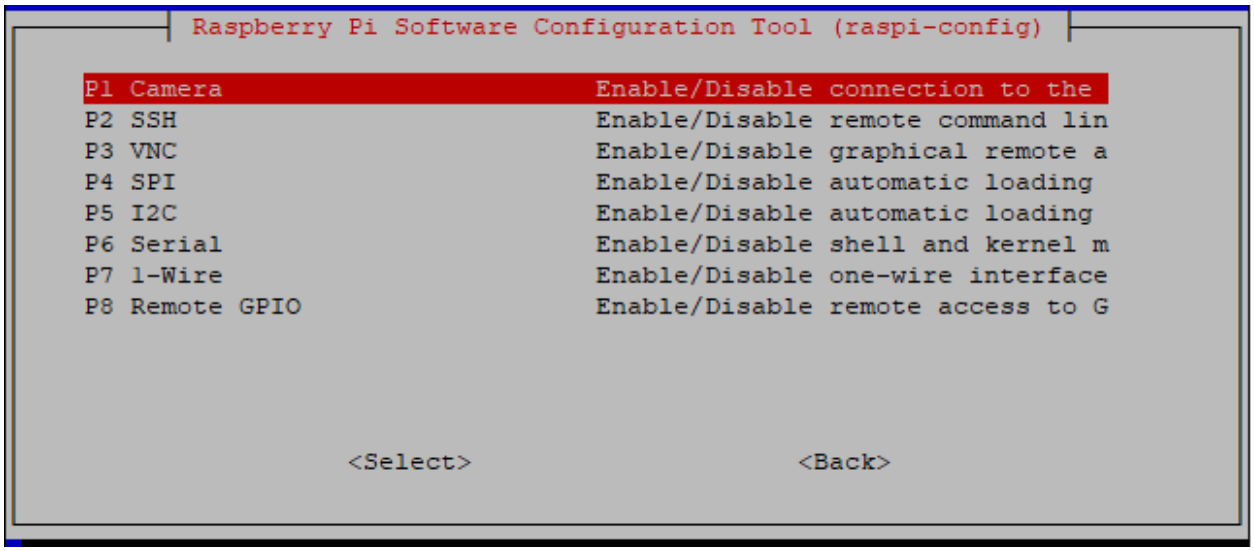
3.1.1. Konfiguracija mobilne platforme

Nakon što se *Raspberry Pi OS* upali u potpunosti, ovisno o konfiguraciji operacijskog sustava može se pojaviti na izlaznom uređaju ili potpuno grafičko sučelje ili samo bash terminal. Ako je prvi slučaj, potrebno je otvorit terminal korištenjem prečaca CTRL+ALT+T. U terminalu pokretanjem naredbe *raspi-config* dobija se izlaz prikazan na slici Slika 3.1.



Slika 3.1 Raspi-config izbornik

Prvo je potrebno izabrati *Network Options* te unijeti ispravne podatke za mrežu na koju se spaja, SSID te lozinku. Ako je povezivanje na mrežu bilo uspješno, potrebno je u izborniku izabrati opciju *Interfacing Options* kako bismo omogućili sučelja koja su potrebna za komunikaciju mobilne platforme i udaljenog računala. Slika 3.2. predstavlja izbornik sučelja.



Slika 3.2 Izbornik sučelja

U izborniku je potrebno omogućiti opcije *Camera* kako bi se mogao koristiti *Pi-Camera* modul te *SSH* da bi se moglo pristupiti mobilnoj platformi udaljeno pomoću SSH protokola za komunikaciju.

3.1.2. Priprema Python okruženja na mobilnoj platformi

Za omogućavanje potpune kontrole i pristupa mobilnoj platformi, tvrtka Waveshare u svojoj službenoj dokumentaciji [2] navodi potrebne Python biblioteke koje je potrebno instalirati pomoću sljedećih komandi u terminalu:

```
sudo pip3 install pillow
sudo pip3 install numpy
sudo apt-get install libopenjp2-7
sudo apt install libtiff
sudo apt install libtiff5
sudo apt-get install libatlas-base-dev
sudo apt-get update
sudo apt-get install python3-pip
sudo pip3 install RPi.GPIO
sudo pip3 install smbus
```

Python 3 dolazi predinstaliran s *Raspbian* operacijskim sustavom tako da nema potrebe isti zasebno instalirati.

3.2. Prijenos slike Pi kamere na mrežu

Pomoću alata za prijenos slike (eng. *mjpg-streamer)* može se prenositi izlaz kamere mobilne platforme na lokalnu mrežu u obliku IP-kamere. Ostala računala koja pristupaju toj IP-kameri izlaz vide kao niz JPEG slika koje se mijenjaju u stvarnom vremenu.

3.2.1. Instalacija paketa za prijenos slike na mrežu

Za prijenos izlaza *Pi-camera* modula, bit će potrebno koristiti poseban *fork mjpg-streamer* programskog alata. U proizvoljni direktorij klonirat će se zadani repozitorij sljedećom naredbom:

```
git clone https://github.com/jacksonliam/mjpg-streamer.git
```

Nakon toga potrebno je instalirati razvojnu verziju *libjpeg* biblioteke

```
sudo apt-get install cmake libjpeg8-dev
```

Ako nije instaliran *gcc* ili *g++*, oba je moguće instalirati sljedećom naredbom:

```
sudo apt-get install gcc g++
```

Za kraj je potrebno kompajlirati sve te datoteke kako bi se mogla pokrenuti izvršna datoteka;

```
cd mjpg-streamer-experimental
make
sudo make install
```

Za pokretanje *mjpg-streamer* paketa, potrebno je pokrenuti skriptu *start.sh* koja se nalazi u *mjpg-experimental* direktoriju. Ako je sve u redu, dočekat će nas sljedeći izlaz prikazan slikom 3.3.

```
pi@robot04:~/AlphaBot2/mjpg-streamer/mjpg-streamer-experimental $ ./start.sh
MJPG Streamer Version: git rev: 310b29f4a94c46652b20c4b7b6e5cf24e532af39
 i: Using V4L2 device.: /dev/video0
 i: Desired Resolution: 640 x 480
 i: Frames Per Second.: -1
 i: Format............: JPEG
 i: TV-Norm...........: DEFAULT
UVCIOC_CTRL_ADD - Error at Pan (relative): Inappropriate ioctl for device (25)
UVCIOC_CTRL_ADD - Error at Tilt (relative): Inappropriate ioctl for device (25)
UVCIOC_CTRL_ADD - Error at Pan Reset: Inappropriate ioctl for device (25)
UVCIOC_CTRL_ADD - Error at Tilt Reset: Inappropriate ioctl for device (25)
UVCIOC_CTRL_ADD - Error at Pan/tilt Reset: Inappropriate ioctl for device (25)
UVCIOC_CTRL_ADD - Error at Focus (absolute): Inappropriate ioctl for device (25)
UVCIOC_CTRL_MAP - Error at Pan (relative): Inappropriate ioctl for device (25)
UVCIOC_CTRL_MAP - Error at Tilt (relative): Inappropriate ioctl for device (25)
UVCIOC_CTRL_MAP - Error at Pan Reset: Inappropriate ioctl for device (25)
UVCIOC_CTRL_MAP - Error at Tilt Reset: Inappropriate ioctl for device (25)
UVCIOC_CTRL_MAP - Error at Pan/tilt Reset: Inappropriate ioctl for device (25)
UVCIOC_CTRL_MAP - Error at Focus (absolute): Inappropriate ioctl for device (25)
UVCIOC_CTRL_MAP - Error at LED1 Mode: Inappropriate ioctl for device (25)
UVCIOC_CTRL_MAP - Error at LED1 Frequency: Inappropriate ioctl for device (25)
UVCIOC_CTRL_MAP - Error at Disable video processing: Inappropriate ioctl for device (25)
UVCIOC_CTRL_MAP - Error at Raw bits per pixel: Inappropriate ioctl for device (25)
 o: www-folder-path......: ./www/
 o: HTTP TCP port........: 8080
 o: HTTP Listen Address..: (null)
 o: username:password....: disabled
 o: commands.............: enabled
```

Slika 3.3 Izlaz *mjpg-streamer* alata

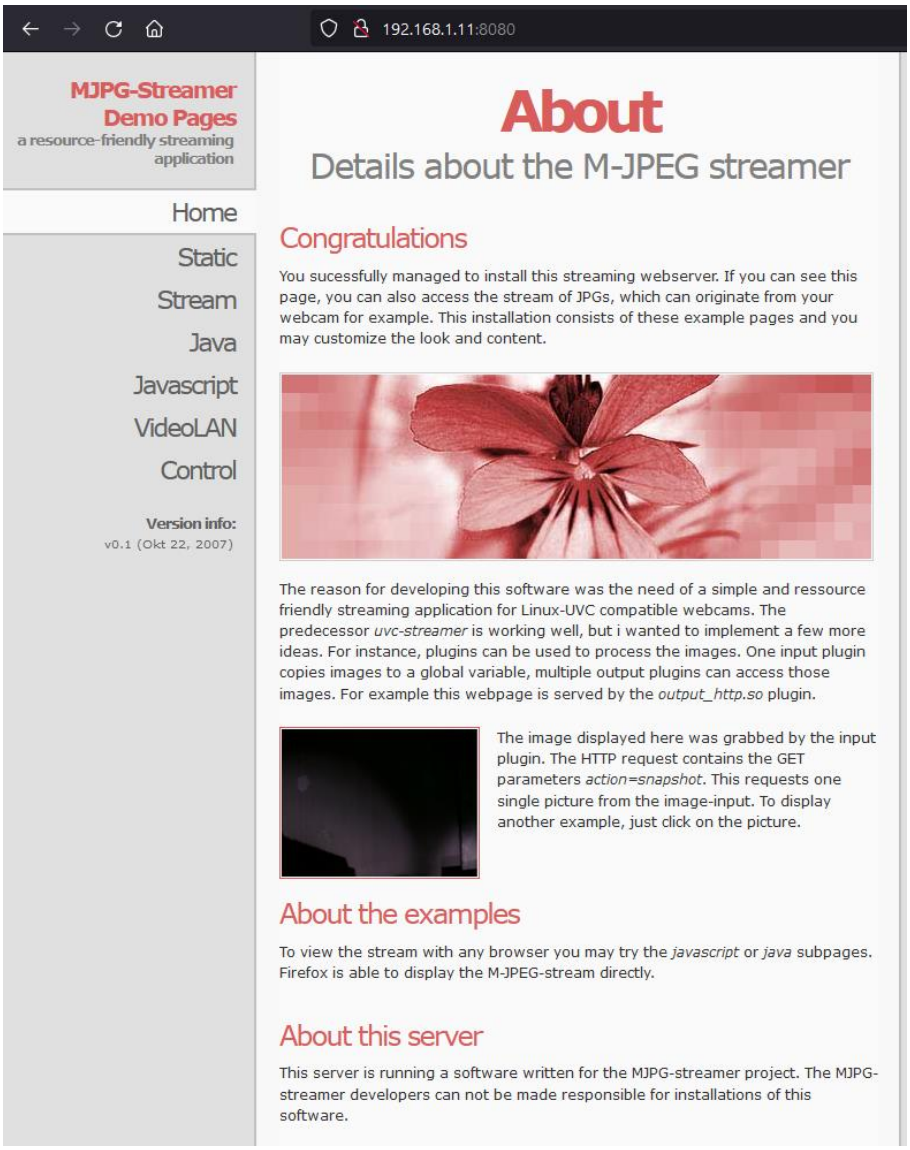
Posebno treba obratiti pažnju na izlistani *HTTP TCP port* jer će se preko njega spajati na IP-kameru. U svrhu testiranja, moguće je pokrenuti komandu u terminalu koja izlistava lokalnu IP adresu mobilne platforme;

```
hostname -I
```

te u željenom internet pregledniku upisati lokalnu adresu mobilne platforme praćenu s portom koji je prethodno bio izlistan, za primjer u ovom slučaju će se pokrenuti adresa

```
192.168.1.11:8080
```

Ako je sve pravilno postavljeno, otvorit će se sljedeća stranica kao na slici Slika 3.4 Testna web-stranica generirana od strane *mjpg-streamer* alata



Slika 3.4 Testna web-stranica generirana od strane *mjpg-streamer* alata

3.3. Detekcija kolizije

Za programsku izvedbu izračuna udaljenosti, koristit će se Rpi.GPIO biblioteka koja omogućava lakše korištenje GPIO pinova mobilne platforme.

```
TRIG = 22
ECHO = 27

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(TRIG, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(ECHO, GPIO.IN)
```

Slika 3.5 Postavljanje pinova senzora

Za početak je potrebno definirati koji pin je odašiljač, a koji transmiter, u ovom slučaju pin 22 je odašiljač, a pin 27 je prijemnik.

```
def dist():
    GPIO.output(TRIG, GPIO.HIGH)
    time.sleep(0.000015)
    GPIO.output(TRIG, GPIO.LOW)
    while not GPIO.input(ECHO):
        pass
    t1 = time.time()
    while GPIO.input(ECHO):
        pass
    t2 = time.time()
    return (t2 - t1) * 34000 / 2
```

Slika 3.6 Funkcija za izračun udaljenosti

Funkcija *dist()* za ulogu ima računanje udaljenosti objekta korištenjem ultrazvučnog senzora. Prvo se na pin koji predstavlja odašiljač postavlja visok napon te nakon jako kratkog vremena postavlja se niski napon. Zabilježava se vrijeme kada se pojavljuje logička nula na pinu odašiljača, te nakon toga se zabilježava vrijeme kada se na pinu odašiljača pojavi logička jedinica. Polovina razlike između ta dva vremena pomnožena s 34000 predstavlja udaljenost između objekta od kojeg se odbio ultrazvučni val u centimetrima te je ista povratna vrijednost ove funkcije.

3.4. Prepoznavanje objekata

3.4.1. Programska implementacija prepoznavanja objekta

Korištenjem detekcije haarovih značajki objekta na slici te kaskadnih klasifikatora detektirat će se objekti na predanoj slici. Funkcija *detectAndDisplay* omogućava detekciju i prikaz prepoznatih objekata.

```
def detectAndDisplay(frame, cascade):
    frame_gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
    frame_gray = cv.equalizeHist(frame_gray)

    objects_detected = cascade.detectMultiScale(frame_gray, 1.05, 3)
    for (x, y, w, h) in objects_detected:
        center = (x + w // 2, y + h // 2)
        frame = cv.ellipse(frame, center, (w // 2, h // 2), 0, 0, 360, (255,
0, 255), 4)
    cv.imshow('Capture - Stop sign detection', frame)
    return objects_detected
```

Slika 3.7 Funkcija prepoznavanja i prikazivanja objekata

Za paremetre uzima sliku na kojoj se vrši detekcija te odgovarajući kaskadni klasifikator. Slika se pretvara u *grayscale* format zbog kompatibilnosti s algoritmom te se izjednači histogram slike. Nakon toga, varijabla *objects_detected* preuzima vrijednosti koje funkcija *detectMultiScale* klase predanog kaskadnog klasifikatora vraća. Te vrijednosti su u obliku *numpy* niza koje se za svaki prepoznati objekt sastoje od četiri elementa: x-koordinate i y-koordinate gornjeg lijevog vrha prepoznatog objekta na slici te dužine i visine prepoznatog objekta. Radi bolje zornosti prikaza, u petlji koja prolazi kroz sve detektirane objekte se pomoću ugrađene *OpenCV* funkcije za crtanje elipsa ista kreira oko prepoznatog objekta. Na kraju, pomoću naredbe *imshow* stvara se prozor na kojem će se prikazivati okviri te elipse oko prepoznatih objekata.

3.4.2. Programska implementacija izračuna udaljenosti

U datoteci *functions.py* prikazane slikom 3.8. opisane su funckije koje su potrebne za uspješno prepoznavanje i prikazivanje slike, ujedno i funkcija *distanceToObject*

```
def distanceToObject(object, widthOfObject, focalLength, frame):
    perceivedWidth = object[2]
    distance = (widthOfObject * focalLength) / perceivedWidth
    cv.putText(frame, '{:05.2f}'.format(distance)+" mm", (object[0],
object[1]), cv.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2, cv.LINE_AA)
    cv.imshow('Capture - Stop sign detection', frame)
    return distance
```

Slika 3.8 Izračun udaljenosti od objekta

Funkcija prima četiri parametra: prepoznati objekt, stvarnu širinu objekta, žarišnu duljinu leće te sliku koju obrađujemo. Parametar prepoznatog objekta se sastoji od četiri vrijednosti: x-koordinate i y-koordinate gornjeg lijevog vrha prepoznatog objekta na slici te dužine i visine prepoznatog objekta. Širina objekta je treća vrijednost u nizu detektiranog objekta, stoga je ona uzeta kao varijabla za računanje udaljenosti objekta od kamere. Varijabla *distance* predstavlja vrijednost funkcije (2.2.) te je njen iznos jednak udaljenosti objekta od kamere u milimetrima. Također, u svrhu bolje ilustracije, koristi se *putText* funkcija *OpenCV* biblioteke koja omogućava dodavanje teksta na sliku, u ovom slučaju iznos udaljenosti formatiran tako da se sastoji od maksimalno pet

znamenaka od kojih su dvije iza decimalne točke. Na samom kraju je potrebno vratiti izračunatu udaljenost kako bi se ona mogla koristiti u ostalim dijelovima programa.

3.5. Udaljeno spajanje na mobilnu platformu putem socketa

Mobilna platforma nema dovoljnu računalnu snagu da u stvarnom vremenu obrađuje slike pomoću haarovih značajki stoga će se ona izvršavati na udaljenom računalo koje će povratnu informaciju slati mobilnoj platformi. Za implementaciju TCP protokola preko kojeg će se slati povratna informacija koristit će se biblioteka *socket*. Udaljeno računalo će predstavljati poslužitelj na koji će se spojiti mobilna platforma u obliku klijenta. Izgled koda za server prikazan je slikom 3.9.

```
s = socket.socket()
port = 12345
s.bind(('', port))
s.listen(5)
c, address = s.accept()
print("Socket Up and running with a connection from", address)
```

Slika 3.9 Otvaranje poslužitelja

Varijablom *s* postavlja se pokazivač na inicijaliziranu klasu *socket* tipa. Potrebno je odrediti proizvoljan port za spajanje imajući u vidu da nije već zauzet na računalu ili mreži. Naredbom *s.bind* postavlja se socket poslužitelj lokalnog posluživača na portu koji je unaprijed određen te naredbom *s.listen(5)* započinje osluhivanje porta za veze koje se pokušavaju spojiti na poslužitelja. Poslužitelj prihvaća sve veze koje zatraže pristup, do pet mreža istovremeno, te varijablom *c* postavlja pokazivač na klijenta koji se spaja.

Mobilna platforma se spaja na poslužitelja također preko biblioteke *socket* i TCP protokola. Da bi isto bilo moguće potrebno se povezati na lokalnu IP adresu poslužitelja te odgovarajući port kako je prikazano na slici 3.10

```
s = socket.socket()
s.connect(('192.168.1.7', 12345))
```

Slika 3.10 Spajanje na poslužitelj

U ovom slučaju, lokalna adresa je *192.168.1.7* te pripadajući port *12345*. Za uspješno primanje podataka s poslužitelja, definirat će se funkcija *getServerInput()* sa slike

```
def getServerInput():
    while True:
        global str_input
        str_input = s.recv(1024).decode()
```

Slika 3.11 Primanje informacije od poslužitelja

Funkcija predstavlja beskonačnu petlju koja mijenja globalnu varijablu *str_input* vrijednošću koja je zaprimljena putem TCP protokola s poslužitelja. Tu vrijednost je prije obrade potrebno

dekodirati u *UTF-8* format jer poslužitelj šalje niz bajtova. Pošto je potrebno izvoditi osluhivanje paralelno s izvođenjem glavnog dijela programa, koristit će se niti. Python u sebi ima ugrađenu biblioteku *threading* koja pojednostavljuje baratanje s nitima korisniku te omogućava stvaranje istih što je prikazano kodom na slici 3.12.

```
thread1 = threading.Thread(target=getServerInput)
thread1.setDaemon(True)
thread1.start()
```

Slika 3.12 Postavljanje niti

Postavlja se pokazivač na nit koja se definira naredbom *Thread*. Za metu izvršavanja niti postavlja se funkcija *getServerInput* koja stalno osluhuje podatke koje šalje poslužitelj. Nit se postavlja kao *daemon* tip jer završavanjem rada glavnog programa, završava se i rad niti. Pokretanjem niti započinje se izvršavanje funkcije za osluhivanje.

3.6. Kretanje mobilne platforme

Kretanje mobilne platforme određeno je izbjegavnjem kolizija s objektima u stvarnom svijetu te interakcijom s definiranim objektima, u ovom slučaju, sa znakom *STOP*. Mobilna platforma će se kretati naprijed, a ako se preko ultrazvučnog senzora detektira moguća kolizija, mobilna platforma će se prestati kretati dok se kolizija ne otkloni. Također, na pojavu znaka *STOP*, mobilna platforma prestaje s kretanjem sve dok udaljeno računalo koje vrši prepoznavanje objekata pošalje povratnu informaciju mobilnoj platformi da se može nastaviti slobodno kretati. Logiku rada mobilne platforme može se vidjeti na slici koda 3.13.

```
str_input = "w"
...
try:
    while True:
        print("Moving to: " + str_input)
        if dist() < 25:
            Ab.stop()
            print("Obstacle detected, stopping.")
            time.sleep(0.3)
        elif str_input == "s":
            Ab.stop()
            print("Stop sign detected, stopping.")
            time.sleep(0.3)
        elif str_input == "w":
            Ab.forward()
            print("Going forward.")
            time.sleep(0.02)
        else:
            Ab.stop()
            time.sleep(0.3)
except KeyboardInterrupt:
    GPIO.cleanup()
    thread1.join()
```

```
    s.close()
```

Slika 3.13 Logika kretanja platforme

Prije spomenuta nit koja se izvršava u pozadini osluhuje informacije koje šalje poslužitelj, udaljeno računalo koje vrši prepoznavanje objekta. Globalna varijabla *str_input* predstavlja vrijednost koja je poslana s udaljenog računala te ona može biti jednaka ili *w* ili *s*. Ako je varijabla jednaka *w*, mobilna platforma smatra da je slobodna za kretanje naprijed. U tom slučaju, to kretanje će biti jedino sprijećeno ako je udaljenost detektirana na ultrazvučnom senzoru manja od 25 centimetara, čime će se mobilna platforma zaustaviti te pričekati 300 milisekundi za ponovno računanje udaljenosti. U slučaju da je mobilna platforma zaprimila podatak *s* s udaljenog računala, smatra se da je naišla na znak *STOP* u neposrednoj blizini te zaustavlja svoje kretnje sve dok ne zaprimi podatak *w* koji signalizira da je moguć nastavak slobodne kretnje. U slučaju da se prekine izvođenje petlje tipkom za prekid na tipkovnici, potrebno je počistiti GPIO pinove, ugasiti nit koja se izvodila u pozadini te zatvoriti konekciju s udaljenim računalom putem socketa.

3.6.1. Praktično rješenje s udaljenog računala

Udaljeno računalo, točnije poslužitelj, za cilj ima obrađivati sliku koju mobilna platforma šalje te u slučaju prepoznavanja objekta, poslati povratnu informaciju mobilnoj platformi. Izvršna datoteka *run.py* ovog projekta sadrži sve što je potrebno za spajanje i obrađivanje slike mobilne platforme. Prvo je potrebno preko *OpenCV* biblioteke inicijalizirat pokazivač na IP-kameru s koje se preuzima videozapis kako je prikazano na slici 3.14.

```
cap = cv.VideoCapture("http://192.168.1.11:8080/?action=stream")
cap.set(cv.CAP_PROP_FOURCC, cv.VideoWriter.fourcc('m','j','p','g'))
```

Slika 3.14 Postavljanje pokazivača na stream

Radi boljih performansi, postavlja se kao codec *MJPG*. Kaskadni filtri se inicijaliziraju kako je prikazano na slici 3.15.

```
stop_cascade = cv.CascadeClassifier(os.path.join(os.getcwd(),
"haarcascades/stop_sign.xml"))
```

Slika 3.15 Postavljanje kaskadnog klasifikatora

Bibliotekom *os* omogućava se nesmetan rad na različitim podatkovnim sustavima te se postiže agnostičnost platforme izvršnog programa. *XML* datoteka koja sadrži kaskadne klasifikatore potrebne za prepoznavanje znakova *STOP* se nalazi u *haarcascades* direktoriju pood imenom *stop_sign.xml* te se ista postavlja pomoću biblioteke *OpenCV* kao zadani kaskadni klasifikator. Nadalje, potrebno je pokrenuti petlju koja će se izvršavati sve dok se izvođenje programa ne prekine trećom stranom pokazano na slici 3.16.

```
lastStopTime = time.time() - 5
while True:
    ret, frame = cap.read()
    if frame is None:
        print('--(!) No captured frame -- Break!')
        break

    stop_signs = functions.detectAndDisplay(frame, stop_cascade)
    currentTime = time.time()
    num_of_detected_stop_signs = len(stop_signs)

    if num_of_detected_stop_signs > 0:
        for object in stop_signs:
            if functions.distanceToObject(object, 70, 900, frame) < 350 and
currentTime - lastStopTime >= 3:
                c.send("s".encode())
                time.sleep(5)
                lastStopTime = time.time()
                c.send("w".encode())
    if cv.waitKey(10) == 27:
        c.close()
        break
```

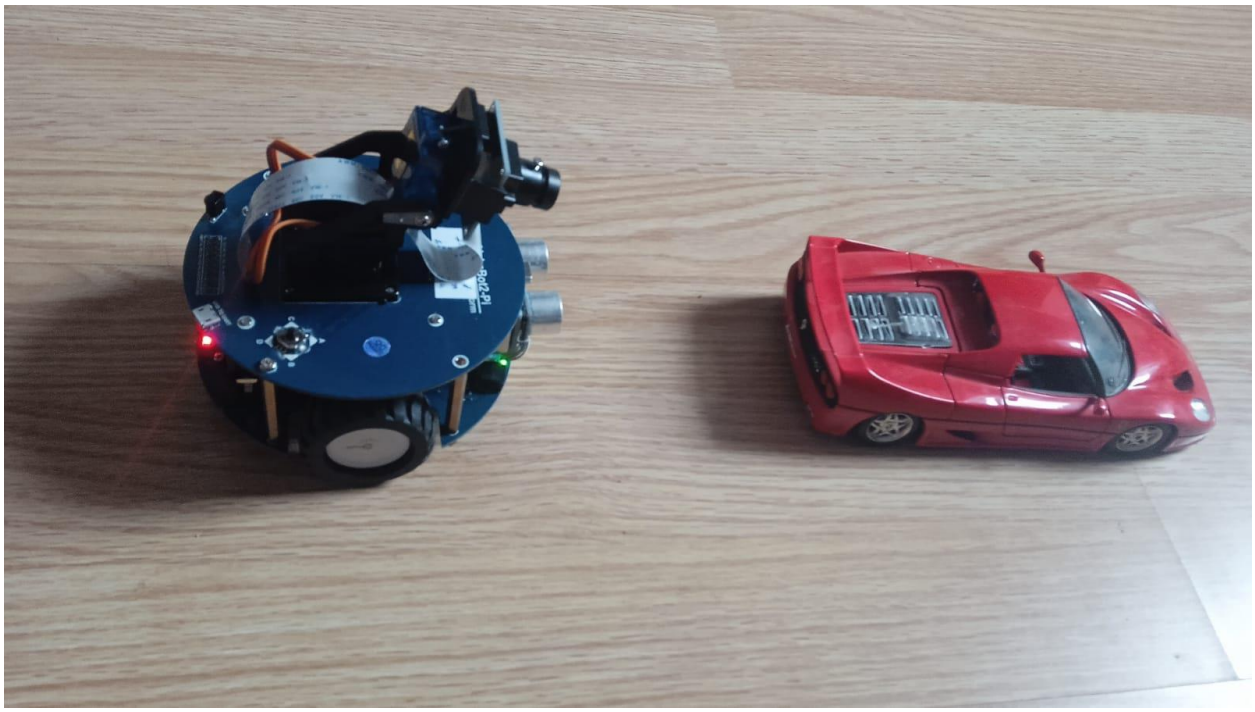
Slika 3.16 Glavna petlja na računalu

Na početku se inicijalizira varijabla *lastStopTime* koja će predstavljati vrijeme kada je mobilna platforma zadnji put stala na znaku stop. Nakon toga slijedi beskonačna petlja koja prvo išćitava okvire iz izvora IP-kamere, u slučaju da nema slike, petlja se prekida i kod se prestaje izvršavati. Varijabla *stop_signs* predstavlja listu *numpy* detektiranih objekata koji se sastoje od prije navedenih vrijednosti. Funkciji *detectAndDisplay* predajemo preuzeti okvir te kaskadni klasifikator objekta koji odgovara onome koji želimo detektirati, u ovom slučaju znakovi *STOP*. Varijabla *currentTime* se predstavlja kao trenutno vrijeme pomoću biblioteke *time* što će biti potrebno u kasnijem dijelu izvođenja. Varijabla *num_of_detected_stop_signs* predstavlja veličinu liste *stop_signs*, točnije broj objekata koji su detektirani u trenutnom okviru koji se obrađuje. Ako je taj broj veći od nule, točnije ako je detektiran minimalno jedan objekt, dolazi do pojavljivanja petlje koja prolazi kroz svaki detektirani objekt te računa je li udaljenost objekta od kamere manja od 350 milimetara te je li prošlo više od 3 sekunde od posljednjeg stajanja na znaku *STOP*. Ako su obje tvrdnje istinite, računalo mobilnoj platformi šalje znak *s*, što mobilna platforma interpretira kao pojavu znaka *STOP*. Bibliotekom *time*, čeka se pet sekundi te nakon toga se zabilježava trenutno vrijeme kao vrijeme zadnjeg stajanja na znak *STOP* te se šalje znak *w* mobilnoj platformi što ona interpretira kao da smije nastaviti kretanje. U slučaju pojave prekida tipkovnicom, zatvara se veza s mobilnom platformom te se petlja prekida.

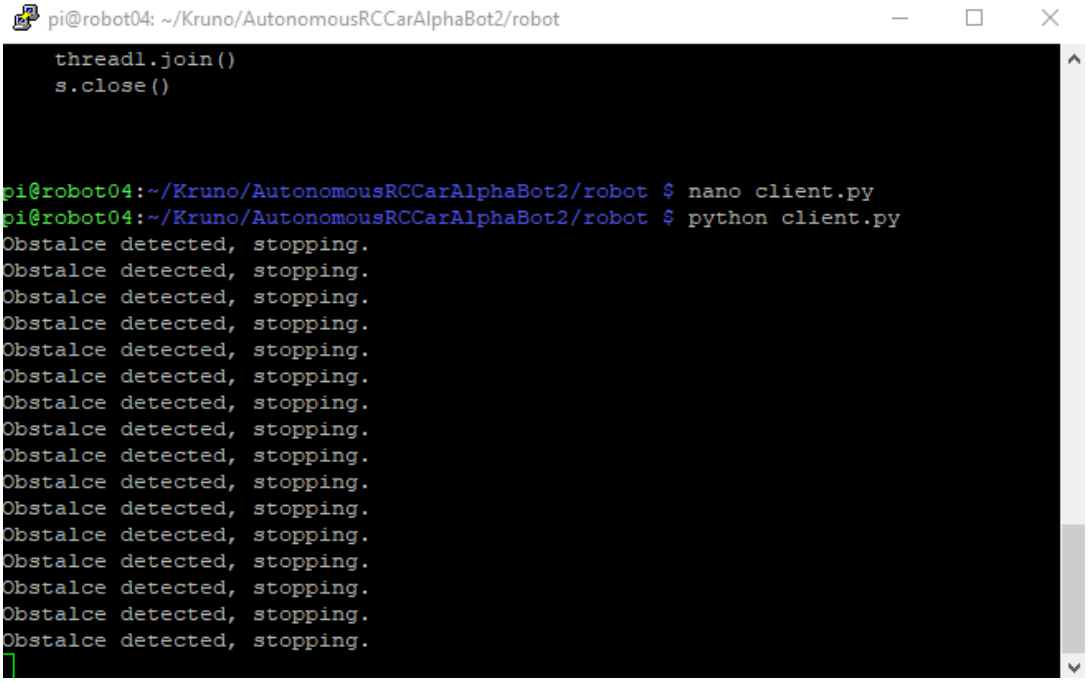
3.7. Testiranje rada mobilne platforme

Na udaljenom računalu pokreće se skripta *run.py* koja započinje komunikaciju s mobilnom platformom. S druge strane, na mobilnoj platformi je nužno pokrenuti skriptu *client.py* koja se spaja na poslužitelja te regulira rad motora.

Slika 3.17 Mobilna platforma započinje s kretanjem

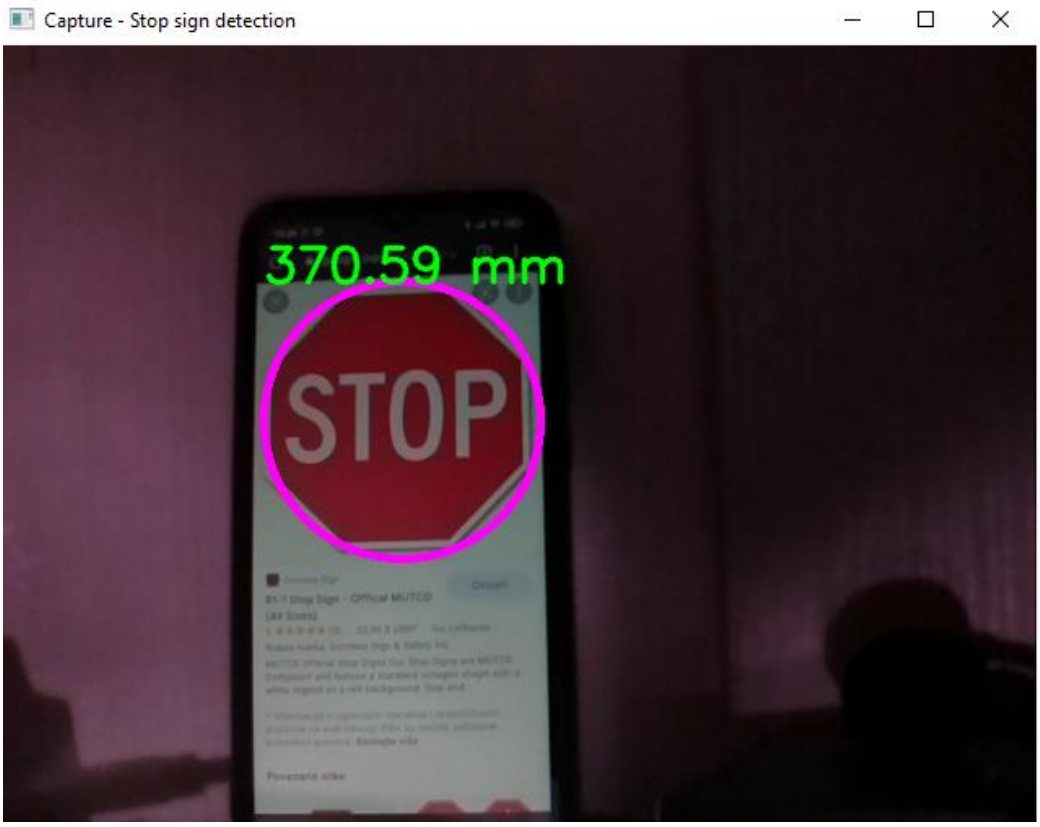


Slika 3.18 Mobilna platforma se zaustavlja prije kolizije



Slika 3.19 Izlaz mobilne platforme na pojavu kolizije

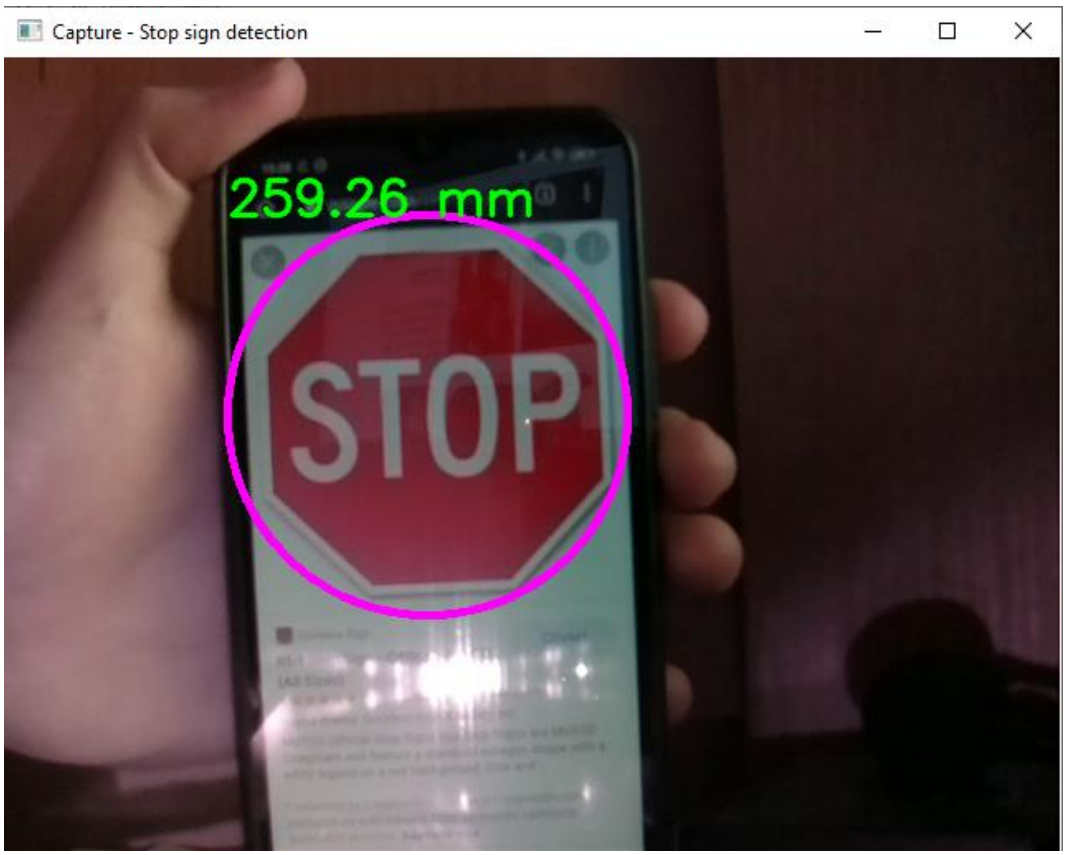
Slika 3.19. predstavlja izlaz mobilne platforme i udaljenog računala na pojavu objekta koji je bliži od 25 centimetara mobilnoj platformi. Mobilna platforma ispisuje na ekran da je detektirana prepreka te prestaje s kretanjem.



Slika 3.20 Obrađena slika gdje je prepoznat objekt te udaljenost od istog

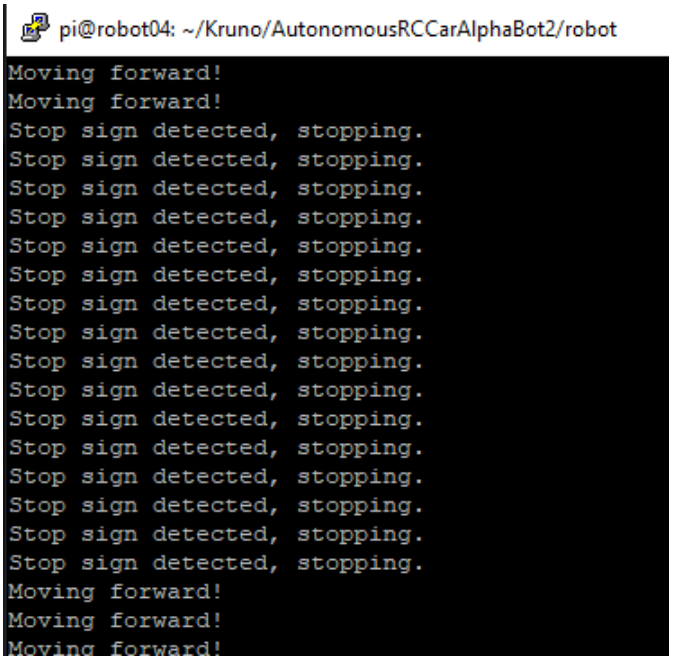
Slika 3.20 predstavlja prozor slike na udaljenom računalu pri pojavi detektiranog znaka *STOP* na udaljenosti od 370.59 milimetara. Pošto je udaljenost i dalje veća od prije definirane udaljenosti zaustavljivanja od 350 milimetara, mobilna platforma i dalje nastavlja slobodnu kretnju.

U slučaju da je ta udaljenost manja kao na slici 3.21.



Slika 3.21 Izgled obrađene slike u slučaju da se staje na znak STOP

mobilna platforma zaustavlja kretanje sve dok ne primi povratnu informaciju od poslužitelja da može nastaviti s kretanjem. Izlaz koji mobilna platforma ispisuje u tom slučaju izgledat će kao na slici 3.22.



Slika 3.22 Izlaz mobilne platforme na susret sa znakom *STOP*

4. ZAKLJUČAK

Izvršavanjem zadatka ovog rada, može se doći do zaključka o pozitivnim stranama predmeta rada. Autonomna vozila mogu jako olakšati ljudima sve naporne zadatke koje vožnja automobila zahtijeva te neke stvari koje su ljudskom oku teško vidljive kao što su procjena udaljenosti. Također, senzori svakako pomažu vozaču da primjeti opasnosti u prometu.

No, testiranjem su se pokazali i nedostatci. Prvi i najveći nedostatak je količina lažno-pozitivnih detekcija koje su nastale provođenjem algoritma detekcije. U ovom slučaju to nije bio preveliki problem jer su sve lažne-pozitivne detekcije nastale izvan predefinirane udaljenosti. Također, pri većim brzinama mobilne platforme, postaje sve teže pravilno detektirati objekt te se povećava mogućnost neuspješnosti pravilne detekcije. Također, ultrazvučni senzor sadrži takozvanu *mrtvu točku* na neposrednoj udaljenosti ispred platforme što onemogućava pravilnu detekciju jako malih udaljenosti.

Autonomna vozila sigurno imaju budućnost, ali uvidjevši sve točke kod kojih može nastati problem i u stvarnom svijetu izazvati kobne greške, autor ovog rada smatra da je autonomnim vozilima još uvijek potrebna ljudska ruka za preuzimanje kontrole kod kritičnih situacija.

LITERATURA

[1] Raspberry Pi Foundation, »Raspberry Pi 3B+ Datasheet,« Raspberry Pi Foundation, 2017.. [Mrežno]. Available: https://static.raspberrypi.org/files/product-briefs/Raspberry-Pi-Model-Bplus-Product-Brief.pdf.

[2] Waveshare INC., »AlphaBot2-Pi Wiki,« 2019. [Mrežno]. Available: https://www.waveshare.com/wiki/AlphaBot2-Pi. [Pokušaj pristupa 17 September 2021].

[3] A. Aquib, »Ultrasonic Sensor and Arduino tutorial,« 29 February 2020. [Mrežno]. Available: https://medium.com/@aquibansari12377/ultrasonic-sensor-and-arduino-tutorial-89c38c81f103.

[4] F. R. Pereira, Ultrasonic wave speed measurement using the time-delay profile of rf-backscattered signals: Simulation and experimental results, Rio de Janeiro: Federal University of Rio de Janeiro, 2001.

[5] A. Rosebrock, »PyImageSearch,« PyImageSearch, 19 January 2015. [Mrežno]. Available: https://www.pyimagesearch.com/2015/01/19/find-distance-camera-objectmarker-using-python-opencv/.

[6] Eagle CCTV, »Calculation of focal length,« [Mrežno]. Available: https://www.eaglecctv.co.za/02-calculation-of-focal-length.

[7] P. Viola i M. Jones, Rapid Object Detection using a Boosted Cascade of Simple, Cambridge: University of Cambridge, 2001.

[8] M. Hrga, Računalni vid, Šibenik: Veleučilište u Šibeniku, 2018..

[9] P. Marwedel, Embedded System Design - Embedded Systems Foundations of Cyber-Physical Systems, Springer Netherlands, 2011.

[10] Society of Automotive Engineers, »LEVELS OF DRIVING AUTOMATION ARE DEFINED IN NEW SAE INTERNATIONAL STANDARD J3016,« 1 July 2018. [Mrežno]. Available: https://web.archive.org/web/20180701034327/https://cdn.oemoffhighway.com/files/base/acbm/ooh/document/2016/03/automated_driving.pdf.

SAŽETAK

U ovom radu je obrađena tema modeliranja autonomnog vozila pomoću AlphaBot 2 mobilne platforme. Opisano je Raspberry Pi računalo te sama AlphaBot 2 mobilna platforma, njeni djelovi te način uspostave komunikacije udaljenog računala i mobilne platforme. Na temelju teorijske podloge rješenja, napisan je programski kod za rješavanje zadatka završnog rada. Udaljeno računalo prepoznaje objekte te računa udaljenost do istih nakon čega vraća povratnu informaciju mobilnoj platformi, a mobilna platforma interpretira tu povratnu informaciju na način koji je adekvatan. Također, mobilna platforma u sebi ima ugrađen ultrazvučni senzor pomoću kojeg može detektirati hoće li doći do kolizije s nekim objektom. Na samom kraju, dokumentirano je testiranje koje služi za donošenje zaključka.

Ključne riječi: AlphaBot2, autonomno vozilo, detekcija kolizije, prepoznavanje objekata, Raspberry Pi

ABSTRACT

The motive of this undergraduate thesis is modelling an autonomous vehicle using the technology of a Raspberry Pi computer and AlphaBot 2 mobile platform for development with the help of a remote computer for more powerful computing. Both the Raspberry Pi computer and AlphaBot2 mobile platform are thoroughly described and the communication between both the remote computer and the mobile platform are also explained. Programming code written in Python neccesarry for achieving the goal of this finals article is also displayed and its function explained. In the end, the testings of the mobile platfrom were documented with explainations and example images for the goal of reaching a conclusion.

Keywords: AlphaBot2, autonomous car, collision detection, object detection, Raspberry Pi

ŽIVOTOPIS

Autor ovog predloška, Krunoslav Petrik, student je treće godine preddiplomskog studija računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija. Završio je srednju školu „Gimnazija Matija Mesić“ u Slavonskome Brodu smjer prirodoslovna-matematička te trenutno živi u Slavonskome Brodu.

_____

Potpis autora

PRILOZI

```
import cv2 as cv
import os
import functions
import socket
import time

# Grab the video stream from bot
cap = cv.VideoCapture("http://192.168.1.11:8080/?action=stream")
cap.set(cv.CAP_PROP_FOURCC, cv.VideoWriter.fourcc('m','j','p','g'))

stop_cascade = cv.CascadeClassifier(os.path.join(os.getcwd(), "haarcascades/s
top_sign.xml"))
traffic_cascade = cv.CascadeClassifier(os.path.join(os.getcwd(), "haarcascade
s/traffic_light.xml"))

# Connect to the robot through a socket
s = socket.socket()
port = 12345
s.bind(('', port))
s.listen(5)
c, address = s.accept()
print("Socket Up and running with a connection from", address)

lastStopTime = time.time() - 5
while True:
    ret, frame = cap.read()
    if frame is None:
        print('--(!) No captured frame -- Break!')
        break

    stop_signs = functions.detectAndDisplay(frame, stop_cascade)
    currentTime = time.time()
    num_of_detected_stop_signs = len(stop_signs)

    if num_of_detected_stop_signs > 0:
        for object in stop_signs:
            if functions.distanceToObject(object, 70, 900, frame) < 350 and c
urrentTime - lastStopTime >= 3:
                c.send("s".encode())
                time.sleep(5)
                lastStopTime = time.time()
                c.send("w".encode())
    if cv.waitKey(10) == 27:
        c.close()
        break
```

P1 run.py datoteka

```
import cv2 as cv


def detectAndDisplay(frame, cascade):
    frame_gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
    frame_gray = cv.equalizeHist(frame_gray)

    objects_detected = cascade.detectMultiScale(frame_gray)
    for (x, y, w, h) in objects_detected:
        center = (x + w // 2, y + h // 2)
```

```
        frame = cv.ellipse(frame, center, (w // 2, h // 2), 0, 0, 360, (255,
0, 255), 4)
    cv.imshow('Capture - Stop sign detection', frame)
    return objects_detected


def distanceToObject(object, widthOfObject, focalLength, frame):
    perceivedWidth = object[3]
    distance = (widthOfObject * focalLength) / perceivedWidth
    # focalLength=(perceivedWidth*300)/widthOfObject
    cv.putText(frame, '{:05.2f}'.format(distance)+" mm", (object[0], object[1
]), cv.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2, cv.LINE_AA)
    cv.imshow('Capture - Stop sign detection', frame)
    return distance
```

P2 functions.py datoteka

```
import Rpi.GPIO as GPIO
import time
from AlphaBot2 import AlphaBot2
import socket
import threading

Ab = AlphaBot2()

TRIG = 22
ECHO = 27

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(TRIG, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(ECHO, GPIO.IN)

Ab.setPWMA(30)
Ab.setPWMB(30)

s = socket.socket()
s.connect(('192.168.1.7', 12345))
str_input = „w“


def dist():
    GPIO.output(TRIG, GPIO.HIGH)
    time.sleep(0.000015)
    GPIO.output(TRIG, GPIO.LOW)
    while not GPIO.input(ECHO):
        pass
    t1 = time.time()
    while GPIO.input(ECHO):
        pass
    t2 = time.time()
    return (t2 - t1) * 34000 / 2


def getServerInput():
    while True:
        global str_input
        str_input = s.recv(1024).decode()
```

```
thread1 = threading.Thread(target=getServerInput)
thread1.setDaemon(True)
thread1.start()

try:
    while True:
        print(„Moving to: „ + str_input)
        if dist() < 25:
            Ab.stop()
            print(„Obstacle detected, stopping.“)
            time.sleep(0.3)
        elif str_input == „s“:
            Ab.stop()
            print(„Stop sign detected, stopping.“)
            time.sleep(0.3)
        elif str_input == „w“:
            Ab.forward()
            print(„Going forward.“)
            time.sleep(0.02)
        else:
            Ab.stop()
            time.sleep(0.3)
except KeyboardInterrupt:
    GPIO.cleanup()
    thread1.join()
    s.close()
```

P3 client.py daatoteka