Razvoj programske podrške za Raspberry Pi mobilnu robotsku platformu

Miličić, Stjepan

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

Preddiplomski sveučilišni studij Računarstvo

RAZVOJ PROGRAMSKE PODRŠKE ZA RASPBERRY PI MOBILNU ROBOTSKU PLATFORMU

Završni rad

Stjepan Miličić

Osijek,2019.

SADRŽAJ

1. UVOD

Raspberry PI malo je računalo veličine kreditne kartice koje svoju primjenu pronalazi u brojnim tehnološkim granama ljudske djelatnosti. Glavna namjena Raspberry PI računala je edukacija, odnosno omogućiti pristup računalnoj tehnologiji i obrazovanju što širem broju ljudi. Raspberry PI Foundation s tim ciljem razvija svoj proizvod i 2012. godine izdaje prvu seriju ovakvog modela računala. Originalna verzija imala je jednojezgreni CPU jačine 700MHz i 256MB radne memorije. Ideja je da ni jedna serija Raspberry Pi-ja ne prelazi cijenu od 35 dolara, čineći ga tako objektivno pristupačnim svim društvenim slojevima.

Nije se očekivalo kako će ovakvo malo računalo na dlanu pronaći svoju primjenu, ne samo u educiranju ljudi o informatičkoj pismenosti, nego i u područjima kao što su robotika i automatizacija. Korisnici na Raspberry PI gledaju kao platformu na kojoj će naučiti programirati, a ako su već upoznati s programiranjem, tražeći nove izazove, započinju programirati elektroničke komponente za fizičke projekte.

Pokretan na Raspbianu, svom službenom operacijskom sustavu, Raspberry PI je programabilan pomoću programskih jezika kao što su Python i Scratch, a projekti i ideje koji svakodnevno nastaju pomoću navedene platforme govore nam kako je ljudska kreativnost zapravo jedina granica u ostvarivanju novih ideja koje izravno utječu na potrebu za razvoj i distribuciju novijih i jačih modela ovakvog računala.

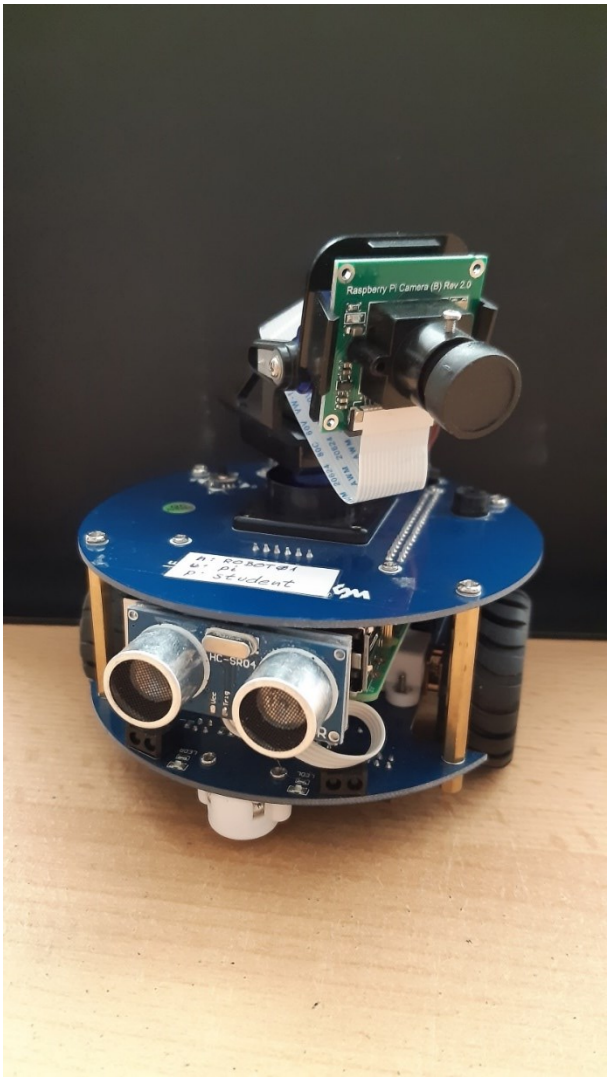
Robotika koja koristi ovakvu platformu zahtjeva razvoj dodatnih senzora i nadogradnji na originalni model Raspberry PI računala pa tako nastaju brojni priključci i setovi koji će automatizaciju robota ili uređaja na Raspberry PI platformi učiniti naprednijom i složenijom. Komponente kao što su toplinski senzori, senzori pokreta, bežična uspostava veze, motori, zasloni neophodni su za razvoj robotike na ovoj platformi.

1.1.

U ovom završnom radu prikazat ćemo moguća rješenja za razvoj programske podrške za samostalno kretanje mobilne robotske platforme zasnovane na ultrazvučnom senzoru i senzoru slike. Koristimo AlphaBot2-PI mobilnu robotsku platformu s ugrađenim navedenim senzorima.

2. MOBILNA PLATFORMA

Osnova mobilne robotske platforme je Raspberry PI 3 model B računalo koje se nalazi unutar zaštitnog kućišta. Kako bi se ono moglo kretati, na računalo su priključeni motori te kotačići, a za realizaciju samostalnog kretanja na računalo su priključeni senzori od kojih glavnu koncentraciju stavljamo na ultrazvučni senzor HC-SR04 te ST188 infracrvene senzore. Potrebno je upoznati se sa svakim od senzora i njihovim principom rada kako bismo uspješno implementirali odgovarajuće programsko rješenje i ostvarili funkcionalno samostalno kretanje AlphaBot2-PI mobilne platforme.

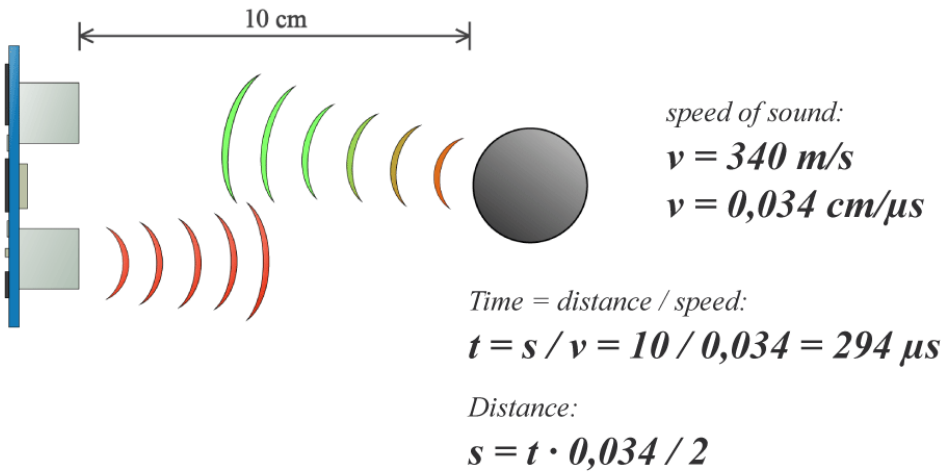


Slika 2.1 AlphaBot2 PI

Slika 2.2 Raspberry PI 3 model B računalo

2.1. Ultrazvučni senzor HC-SR04

HC-SR04 ultrazvučni senzor koristi ultrazvučne valove kako bi odredio udaljenost od predmeta. Karakteriziraju ga dobre tehničke mogućnosti jer ne zbunjuju ga crni materijali te sunčeva svjetlost. Princip rada zasniva se na dva osnovna dijela ovog modula, a to su prekidač (trig) i refleksija (echo). Udaljenost objekta ultrazvučni senzor računa preko formule gdje se kao vrijednosti uzimaju brzina zvuka (v = 340 m/s) i vrijeme refleksije vala.



Slika 2.3 Princip rada ultrazvučnog senzora

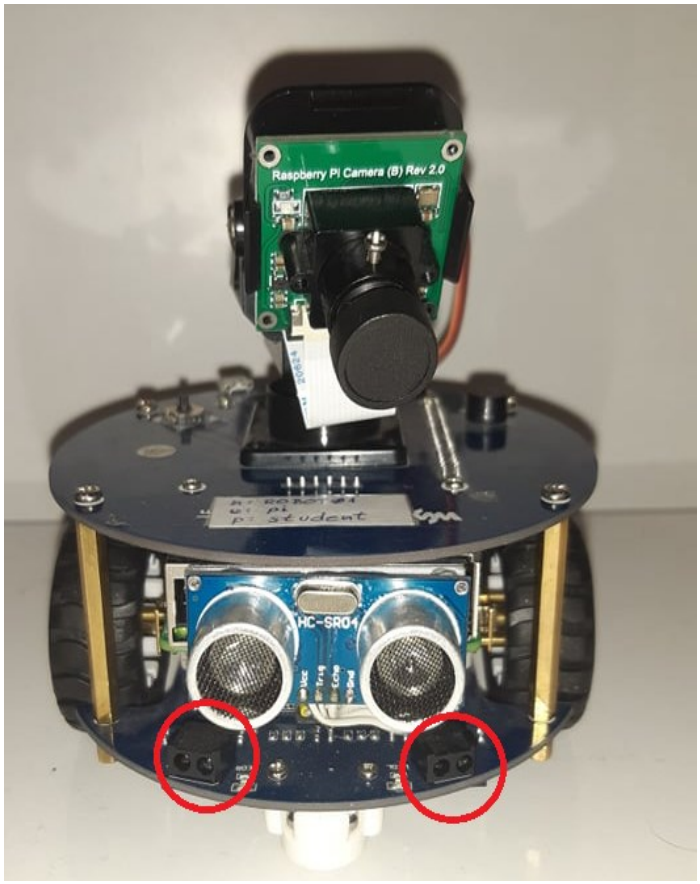
Slika 2.4 HC-SR04 senzor

Slika 2.5 Položaj senzora na mobilnoj platformi

2.2. ST188 senzor

ST188 infracrveni senzor radi na principu samorefleksije, sastoji se od LED diode i visokoosjetljivog Darlingtonovog fototranzistora. Princip rada mu je sličan kao i kod HC-SR04 senzora. Dioda šalje svjetlost te ako se ona odbije od objekt, fototranzistor ju detektira. Može odašiljati svjetlost u rasponu od 4 do 13 milimetara što mu daje prednost u uočavanju i reagiranju na prepreku koja se nalazi ispred mobilne platforme.

Slika 2.6 ST188 senzor



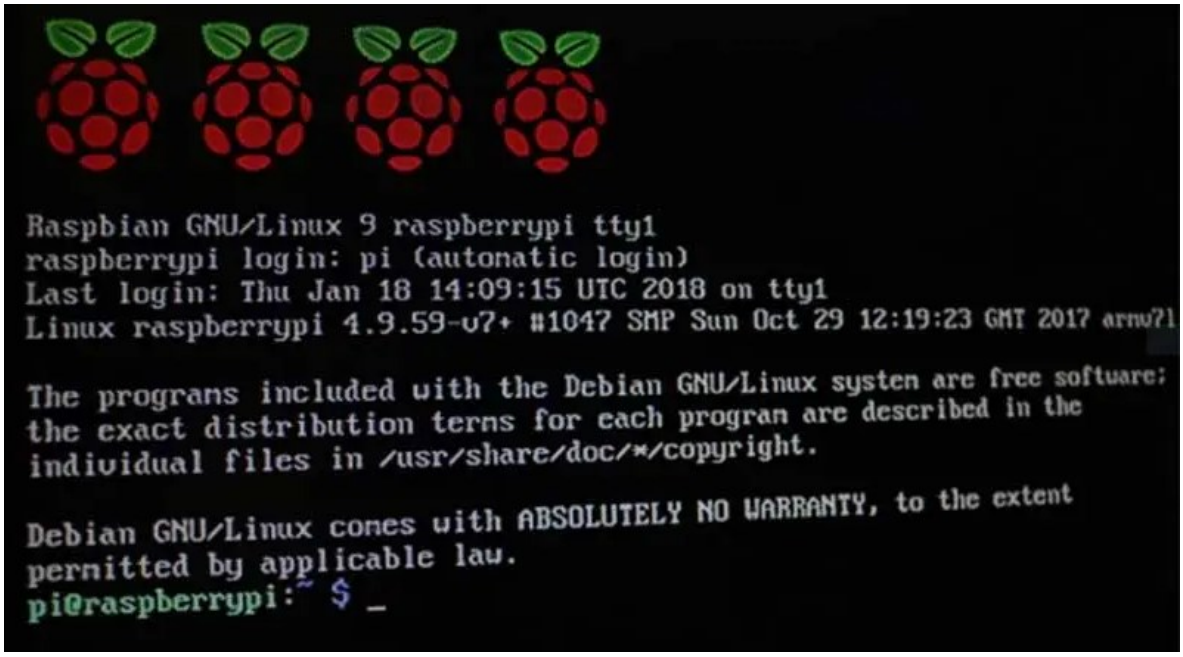
Slika 2.7 Položaj ST188 senzora na mobilnoj platformi

3. PROGRAMSKO RJEŠENJE

Za razvoj programskog rješenja koristit ćemo Python programski jezik. Potrebno je razviti i implementirati kod za pokretanje motora te implementaciju navedenih senzora s kojima smo se pobliže upoznali u prethodnom poglavlju. Za početak, upoznajemo se s načelima rada Raspbian operacijskog sustava.

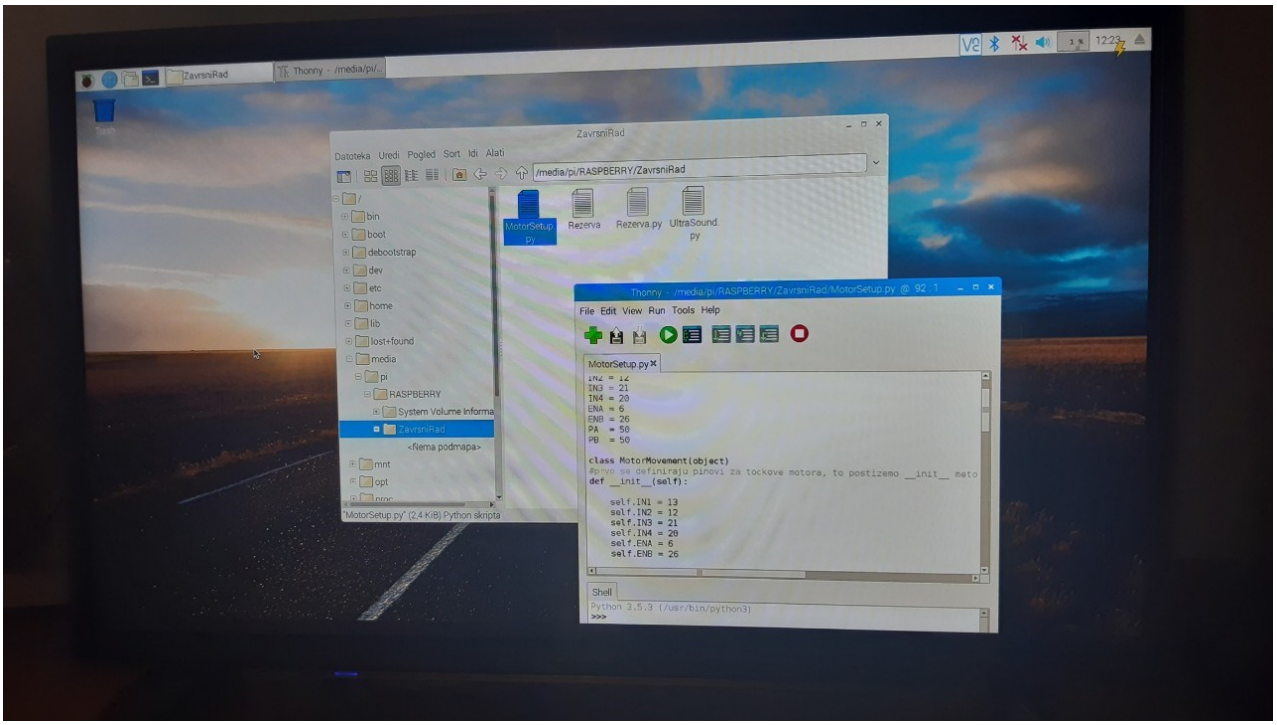
3.1. Povezivanje na Raspbian

Kako bismo se povezali na Raspbian potrebni su nam zaslon, HDMI kabel, miš i tipkovnica. HDMI kablom povezujemo mobilnu platformu s ekranom, a mišem i tipkovnicom se navigiramo Raspbianom. Nakon što upalimo Raspberry PI, Raspbian zahtjeva da unesemo host name i zaporku. Svi podaci navedeni su na robotu.



Slika 3.1 Raspbian operacijski sustav

Nakon unesenih potrebnih informacija imena hosta i zaporke uspješno smo se prijavili na Raspbian operacijski sustav. Potrebno je upoznati se s osnovnim funkcijama Raspbiana kako bismo mogli pozvati programska rješenja i povezati ih s našom mobilnom platformom. Nakon što smo pokrenuli Raspbian, prikazat će nam se Terminal u kojem pozivamo naredbe i upravljamo podacima. Kako bismo se vratili na grafičko sučelje operacijskog sustava, pozivamo naredbu *startx*.



Slika 3.2 GUI Raspbian operacijskog sustava

3.2. Biblioteke

Prije implementacije rješenja za naše senzore, potrebno je na računalo instalirati WiringPI i Python biblioteke koje omogućuju komunikaciju s pinovima, senzorima i općenitu vezu između platforme i programskog rješenja.

WiringPI je biblioteka koja omogućuje GPIO pristup pinovima na našem PI računalu, povezivanje i uspostavljanje veze sa senzorima te postavljanje vrijednosti na pinove i pristup pinovima u našem programskom kodu. Napisan je u C programskom jeziku i podržava analogno čitanje i pisanje.

Python biblioteka sadrži modul pomoću kojeg se uspostavlja komunikacija sa serijskim periferijskim uređajima(SPI) preko Raspbiana.

P1: The Main GPIO connector							
WiringPi Pin	BCM GPIO	Name	Header		Name	BCM GPIO	WiringPi Pin
		3.3v	1	2	5v		
8	Rv1:0 - Rv2:2	SDA	3	4	5v		
9	Rv1:1 - Rv2:3	SCL	5	6	0v		
7	4	GPIO7	7	8	TxD	14	15
		0v	9	10	RxD	15	16
0	17	GPIO0	11	12	GPIO1	18	1
2	Rv1:21 - Rv2:27	GPIO2	13	14	0v		
3	22	GPIO3	15	16	GPIO4	23	4
		3.3v	17	18	GPIO5	24	5
12	10	MOSI	19	20	0v		
13	9	MISO	21	22	GPIO6	25	6
14	11	SCLK	23	24	CE0	8	10
		0v	25	26	CE1	7	11
WiringPi Pin	BCM GPIO	Name	Header		Name	BCM GPIO	WiringPi Pin

Slika 3.3 Brojčane vrijednosti pinova na PI računalu

3.3. Implementacija motora

Za uspješnu implementaciju motora potrebno je definirati odgovarajuće pinove preko kojih su kotačići povezani na Raspberry PI računalo te smjerove kretanja. Koristit ćemo softver Brackets na kojem ćemo izraditi programsko rješenje za pojedini senzor AlphaBot2-PI platforme. Potrebno je definirati biblioteke *Rpi.GPIO* i *time* biblioteku. *Rpi.GPIO* biblioteka sadrži klase koje kontroliraju GPIO na Raspberry PI-ju. GPIO kratica je za General Purpose Input/Output, gdje pinove možemo definirati kao ulaz, odnosno izlaz. *Time* biblioteka sadrži brojne funkcije koje uzimaju i vraćaju vrijeme kao varijablu. Nakon definiranja biblioteka, definiramo pinove na kojima su povezani motori jer ćemo im tako moći pristupiti i, pomoću poziva odgovarajućih funkcija, pokrenuti, odnosno zaustaviti.

```
import RPi.GPIO as GPIO
import time

class MotorMovement(object):

        def __init__(self,L1=12,L2=13,R1=20,R2=21,LW=6,RW=26):
            self.Left1 = L1
            self.Left2 = L2
            self.Right1 = R1
            self.Right2 = R2
            self.LWheel = LW
            self.RWheel = RW

        #nakon definiranih pinova, moramo im postaviti gpio vrijednost, odnosno IN ili OUT. svi pinovi su vrijednost GPIO.OUT
            GPIO.setmode(GPIO.BCM)
            GPIO.setwarnings(False)
            GPIO.setup(self.Left1,GPIO.OUT)
            GPIO.setup(self.Left2,GPIO.OUT)
            GPIO.setup(self.Right1,GPIO.OUT)
            GPIO.setup(self.Right2,GPIO.OUT)
            GPIO.setup(self.LWheel,GPIO.OUT)
            GPIO.setup(self.RWheel,GPIO.OUT)
            self.PWMOne = GPIO.PWM(self.LWheel,500)
            self.PWMTwo = GPIO.PWM(self.RWheel,500)
            self.PWMOne.start(20)
            self.PWMTwo.start(20)
            self.stop()

```

Slika 3.4 Definiranje biblioteka i pinova potrebnih za osposobljavanje rada motora

Pinove definiramo i postavljamo im vrijednost pomoću __init__ metode koja se pozivanjem instance klase MotorMovement prva izvršava. Nakon brojčane vrijednosti pinova postavljamo im GPIO vrijednost, odnosno vrijednost IN ili OUT. Svim pinovima postavljamo OUT vrijednost. GPIO.BCM sugerira kako se brojke pinova iščitaju prema "Broadcom SOC channel" numeriranju, standardnom numeriranju za PI uređaje. PWM (Pulse Width Modulation) metoda je pomoću koje kontroliramo brzinu i smjer kretanja motora, a kao ulazne parameter uzima pin i frekvenciju rada koju želimo koristiti.

Nakon definiranja biblioteka i pinova, započinjemo s pisanjem funkcija za pokretanje, zaustavljanje i smjerove kretanja kotačića, odnosno motora. Princip rada ovih funkcija jest pozvati Boolean varijablu, varijablu True ili False na output motora, ovisno o tome u kojem smjeru želimo da se motor kreće. ChangeDutyCycle naredbom manipuliramo brzinu kretanja kotačića koju ćemo, ovisno o dobivenim rezultatima testiranja, postaviti na optimalnu vrijednost.

```
30          def forward(self):
31              self.PWMOne.ChangeDutyCycle(20)
32              self.PWMTwo.ChangeDutyCycle(20)
33              GPIO.output(self.Left1,GPIO.LOW)
34              GPIO.output(self.Left2,GPIO.HIGH)
35              GPIO.output(self.Right1,GPIO.LOW)
36              GPIO.output(self.Right2,GPIO.HIGH)
37
38          def backward(self):
39              self.PWMOne.ChangeDutyCycle(20)
40              self.PWMTwo.ChangeDutyCycle(20)
41              GPIO.output(self.Left1,GPIO.HIGH)
42              GPIO.output(self.Left2,GPIO.LOW)
43              GPIO.output(self.Right1,GPIO.HIGH)
44              GPIO.output(self.Right2,GPIO.LOW)
45
46          def left(self):
47              self.PWMOne.ChangeDutyCycle(12)
48              self.PWMTwo.ChangeDutyCycle(12)
49              GPIO.output(self.Left1,GPIO.HIGH)
50              GPIO.output(self.Left2,GPIO.LOW)
51              GPIO.output(self.Right1,GPIO.LOW)
52              GPIO.output(self.Right2,GPIO.HIGH)
53
54          def right(self):
55              self.PWMOne.ChangeDutyCycle(12)
56              self.PWMTwo.ChangeDutyCycle(12)
57              GPIO.output(self.Left1,GPIO.LOW)
58              GPIO.output(self.Left2,GPIO.HIGH)
59              GPIO.output(self.Right1,GPIO.HIGH)
60              GPIO.output(self.Right2,GPIO.LOW)
61
62          def stop(self):
63              self.PWMOne.ChangeDutyCycle(0)
64              self.PWMTwo.ChangeDutyCycle(0)
65              GPIO.output(self.Left1,GPIO.LOW)
66              GPIO.output(self.Left2,GPIO.LOW)
67              GPIO.output(self.Right1,GPIO.LOW)
68              GPIO.output(self.Right2,GPIO.LOW)
69
70
```

Slika 3.5 Definiranje funkcija za smjer kretanja motora

3.4. Implementacija Ultrazvučnog senzora

S principom rada ultrazvučnog senzora upoznali smo se kada smo definirali hardversko sklopovlje i naveli njegov princip rada te varijable koje senzor koristi pri izračunavanju udaljenosti od predmeta. Ideja implementacije ultrazvučnog senzora je da reagira na prepreku koja se nalazi na putanji mobilne platforme tako da promjeni smjer kretanja ovisno veličini i udaljenosti prepreke. Najprije je potrebno definirati neophodne biblioteke i varijable te im postaviti odgovarajuću vrijednost.

```
import RPi.GPIO as GPIO
import time
from MotorSetup import MotorMovement

TRIG = 22
ECHO = 27

Mm = MotorMovement()

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

def DistanceMeasure():
    GPIO.output(TRIG,GPIO.HIGH)
    time.sleep(0.00001)
    GPIO.output(TRIG,GPIO.LOW)

    while GPIO.input(ECHO)==0:
        pass
    time1=time.time()
    while GPIO.input(ECHO)==1:
        pass
    time2=time.time()
    time3=time2-time1
    return (34000*time3)/2

try:
    while True:
        Distance = DistanceMesure()
        print("%0.2f cm"%Distance)

        if Dist <= 25:
            Mm.left()
            Mm.right()

        elif Dist <12:
            Mm.backward()
        else:
            Mm.forward()
        time.sleep(0.05)

except KeyboardInterrupt:
    GPIO.cleanup();
```

Slika 3.6 Programsko rješenje za funkcionalni rad ultrazvučnog senzora

Nakon postavljanja vrijednosti varijablama TRIG i ECHO potrebno je uvesti klasu MotorMovement i napraviti njenu instancu kako bismo omogućili kretanje platforme korištenjem ovog programskog koda. Prema načelima rada ultrazvučnog senzora, kreiramo algoritam koji na osnovi varijable vremena omogućuje računanje udaljenosti između mobilne platforme i prepreke. To ostvarujemo tako da mijenjamo TRIG vrijednost iz HIGH u LOW u vrlo kratkom vremenskom intervalu i ispitujemo stanje ECHO varijable. Vrijeme prelaska ECHO varijable iz stanja LOW u stanje HIGH evidentiramo te je njihova razlika, uz potrebno konvertiranje, izmjerena udaljenost. Kako bi platforma reagirala na prepreku, potrebno je napisati metodu za prepoznavanje kritične udaljenosti, odnosno udaljenosti na kojoj će mobilna platforma reagirati na prepreku i promijeniti svoj smjer kretanja. To postižemo jednostavnim postavljanjem uvjeta koji udaljenost od predmeta uspoređuje sa zadanom vrijednosti kritične udaljenosti.

3.5. Implementacija senzora slike

Pristup razvijanju programskog rješenja za senzor slike bit će sličan kao i onaj koji smo primijenili na ultrazvučni senzor. Dva su ST188 senzora na mobilnoj platformi te na osnovi refleksije svjetlosti, odnosno nailaska na predmet, oni će promijeniti stanje. I u ovom slučaju potrebno je definirati *GPIO* i *time* biblioteke te postaviti GPIO vrijednosti svim varijablama koje se koriste u funkcijama za upravljanje senzora. Trebamo napraviti instance klase MotorMovement kako bismo mogli uvesti funkcije za pokretanje motora.

ST188 senzor može prepoznati objekt maksimalno do 13 milimetara udaljenosti, što nam daje do znanja da kritičnu udaljenost ne možemo mijenjati nego programsko rješenje formiramo na temelju promjene stanja senzora. Ako se prepreka detektira na lijevom senzoru, robot će promijeniti smjer kretanja udesno, a ako je detektirana na desnom senzoru, robot se kreće ulijevo.

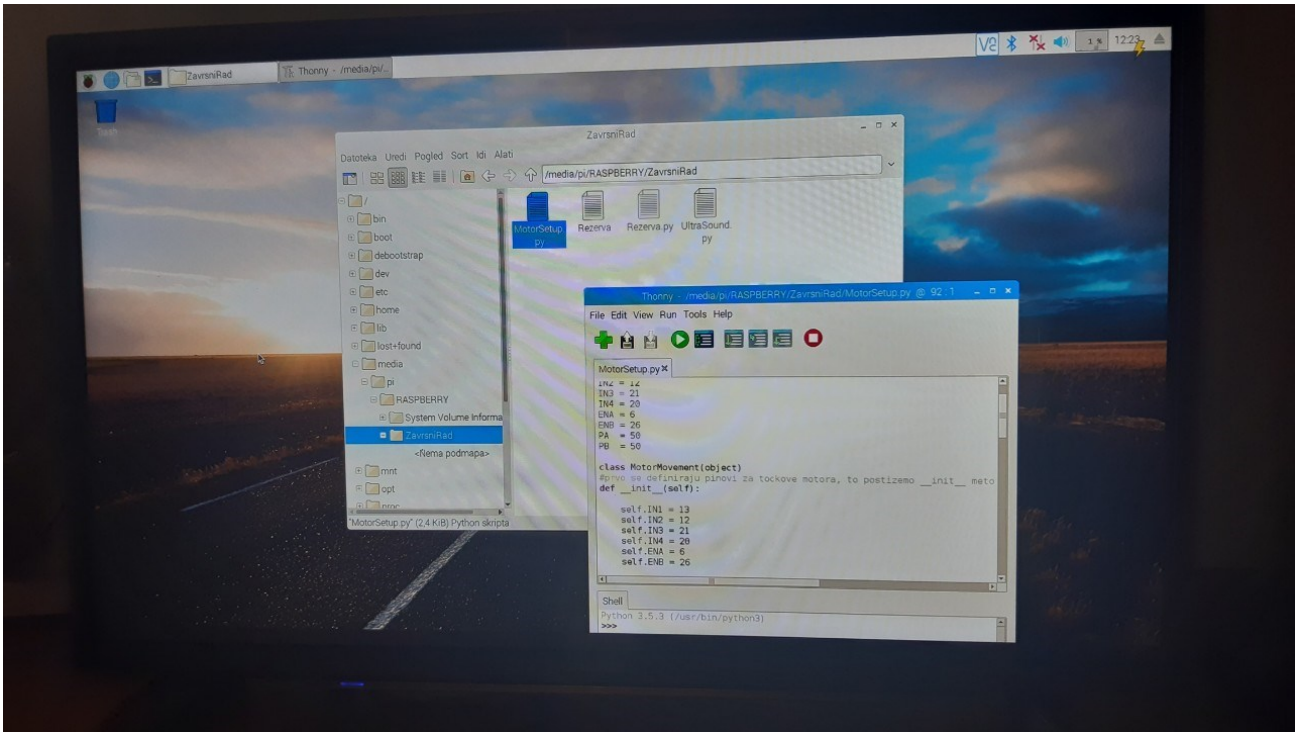
```
import RPi.GPIO as GPIO
import time
from MotorSetup import MotorMovement

LeftSensor = 19
RightSensor = 16


Mm=MotorMovement()

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(RightSensor,GPIO.IN,GPIO.PUD_DOWN)
GPIO.setup(LeftSensor,GPIO.IN,GPIO.PUD_DOWN)



try:
    while True:
        LeftSensorState = GPIO.input(LeftSensor)
        RightSensorState = GPIO.input(RightSensor)


        if(LeftSensorState == 1):
            Mm.right()
            time.sleep(0.005)
            Mm.stop()
        elif(RightSensorState==1):
            Mm.left()
            time.sleep(0.005)
            Mm.stop()

        else:
            Mm.forward()


except KeyboardInterrupt:
    GPIO.cleanup();


    
```

Slika 3.7 Programsko rješenje za funkcionalni rad senzora slike

Nakon što smo postavili vrijednosti pinova za desni i lijevi senzor, postavljamo im i GPIO vrijednost. Oba senzora ćemo postaviti na PUD_DOWN stanje, odnosno stanje gdje senzori imaju vrijednost 0 sve dok ne reagiraju na prepreku te mijenjaju vrijednost u 1. Pomoću izmjene stanja možemo napraviti *if* uvjet, odnosno ako je jedan od senzora naišao na prepreku, robot mijenja smjer kretanja. Ukoliko nije naišao na prepreku, robot se nastavlja kretati prema naprijed.

3.6. Prebacivanje programskog rješenja na AlphaBot2 PI

Nakon što smo napisali programsko rješenje za svaki od navedenih senzora, pohranjujemo ih sve kao Python datoteku i pomoću USB sticka prebacujemo na Raspbian operacijski sustav, koji je u GUI (grafičko sučelje) stanju rada. Pohranjujemo ih među ostale datoteke i kako bi ih mogli pozvati, sistematiziramo ih po adekvatnim imenima koje opisuju njihovu namjenu. Na primjer, u Terminalu pozivamo *cd ~/ZavrsniRad* naredbu. Cd naredba služi za otvaranje datoteke u kojoj se nalazi programski kod. Te nakon nje naredbom *sudo python* upisujemo ime robotske funkcionalnosti koju želimo aktivirati. Želimo li ispitati ispravnost programskog rješenja za ultrazvučni senzor, program pozivamo naredbom *sudo python USOAvoidance.py.* Idealna situacija je ona u kojoj imamo bežični miš i tipkovnicu jer u svakom programskom kodu smo definirali iznimke koje zaustavljaju izvođenje programa pozivanjem naredbi *Ctrl+C* na tipkovnici.



Slika 3.8 Uređivanje programskih rješenja u GUI sučelju Raspbiana

Raspbian nam omogućuje otvaranje, uređivanje i provjeru ispravnosti python datoteka. To nam omogućuje ispravak svake greške u kodu kao i njegovo modificiranje i poboljšanje preko same robotske platforme, koja u ovom slučaju ima ulogu funkcionalnog računala.

4. TESTIRANJE

Kako bismo ustanovili uspješnost programskog rješenja, našu mobilnu platformu potrebno je testirati u kontroliranim laboratorijskim uvjetima. Osim provjere ispravnosti koda, testiranje nam omogućuje bolji uvid u našu programsku implementaciju i ukazuje na moguća poboljšanja i modifikacije sukladno o performansama mobilne platforme. Možemo dobiti povratnu informaciji o robotovoj brzini, brzini reakcije senzora i promjeni smjera kretanja. Daje nam i uvid u situacije koje prije ne bismo uzeli u obzir. Sve informacije dobivene iz testiranja utječu na poboljšanje kvalitete i ispravnosti našeg programskog rješenja, a što je više situacija koje možemo realizirati, bolju povratnu informaciju možemo dobiti. Pošto je glavna svrha programskog rješenja robotova mogućnost izbjegavanja prepreka, staza će se sastojati od prepreka

4.1. Testiranje ultrazvučnog senzora

Prvo ćemo testirati ultrazvučni senzor i autonomno kretanje mobilne platforme pomoću njega. Testiranje se temelji na robotovoj mogućnosti da uspješno izbjegne prepreke i dođe s jednog kraja staze na drugu. Tri su prepreke na koje robot može naići i na koje pravovremeno treba reagirati.

Slika 4.1 Staza za testiranje ultrazvučnog senzora

Testiranje ćemo provesti 10 puta i bilježiti uspješnost mobilne platforme. Sve podatke prikazat ćemo u tablici:

Pokušaj	Uspješnost
1.	NE
2.	NE
3.	DA
4.	NE
5.	NE
6.	DA
7.	DA
8.	DA
9.	NE
10.	DA

Tablica 4.1 Rezultati testiranja uspješnosti robota za ultrazvučni senzor

Uspješnost prolaza staze je 50%. Nakon testiranja shvatili smo kako se robot kreće prebrzo i kako ne stigne reagirati na prepreku na vrijeme. Problem smo rješili smanjenjem brzine kotačića i dodavanjem slučaja da, ukoliko se robot nađe u situaciji da se zabio u prepreku ili je prešao kritičnu udaljenost, započinje se kretati unazad dok ne stekne minimalnu dozvoljenu udaljenost od prepreke. Testiranje ćemo ponoviti.

Pokušaj	Uspješnost
1.	DA
2.	DA
3.	NE
4.	DA
5.	DA
6.	DA
7.	NE
8.	DA
9.	DA
10.	NE

Tablica 4.2 Rezultati ponovljenog testiranja uspješnosti robota za ultrazvučni senzor

Uspješnost prolaza staze je 70%. Drugo testiranje pokazuje nam poboljšanje uspješnosti kretanja mobilne platforme što nam daje do znanja da je ispravljena implementacija povećala efektivnost autonomnog kretanja.

4.2. Testiranje infracrvenog senzora

Slijedi testiranje senzora slike. Pošto fotosenzori imaju malu udaljenost detekcije, stavit ćemo manje prepreke kako bi ih mobilna platforma mogla pravovremeno izbjeći. Uzeći u obzir kako smo već u prethodnom testiranju optimizirali robotovu brzinu, očekujemo veću uspješnost robotovog autonomnog kretanja u ovom slučaju.

Slika 4.2 Staza za testiranje senzora slike

Testiranje ćemo provesti 10 puta i bilježiti uspješnost mobilne platforme. Sve podatke prikazat ćemo u tablici:

Pokušaj	Uspješnost
1.	DA
2.	DA
3.	DA
4.	NE
5.	DA
6.	DA
7.	NE
8.	DA
9.	DA
10.	NE

Tablica 4.3 Rezultati testiranja uspješnosti robota za senzor slike

Uspješnost prolaza staze je 70%. Mobilna platforma se uspješno kretala stazom no nekada prekasno reagira na prepreku te robot izgubi putanju. Fotosenzor pokazuje se kao zahvalno rješenje pri autonomnom kretanju robota, ali mala udaljenost na kojoj senzor može detektirati prepreku predstavlja problem za veće prepreke koje robot tako ne stigne izbjeći.

5. ZAKLJUČAK

Raspberry PI platforma je koja omogućuje velike mogućnosti u području robotike i razvoju jednostavnih i složenih robotskih ideja počevši od Raspberry PI računala kao temelja. Naučili smo kako, uz poznavanje načela rada računala i njegovih senzora, razviti i realizirati ideju autonomnog kretanja robota uz implementaciju odgovarajuće programske podrške.

AlphaBot2-Pi idealan je primjer robotskog rješenja na ovakvoj platformi te uz mnogobrojne ugrađene senzore i module daje dobru podlogu za implementiranje početničkih i naprednih programskih rješenja u Python programskom jeziku.

Upoznali smo se sa senzorima koji se nalaze na AlphaBot PI mobilnom rješenju te naučili njihove principe rada i pomoću prikupljenih informacija došli do najpovoljnijeg puta za izgradnju programskog rješenja za zadanu platformu. Započeli smo s implementiranjem motora i definiranjem pinova te smjerova kretanja. Pošto je robot autonoman, ostatak njegove orijentacije smo riješili implementacijom programskog rješenja za korištene senzore. Ultrazvučni senzor, kojemu su potrebne varijable brzine zvuka i vremena refleksija vala, prepoznaje prepreku te reagira na nju mijenjanjem smjera robota i time smo započeli implementaciju autonomnog kretanja naše platforme. Senzor slike, na temelju samorefleksije, također omogućuje autonomno kretanje robota uz detektiranje prepreke i reagiranje na nju promjenom smjera kretanja.

LITERATURA

[1] AlphaBot2-PI https://www.mouser.com/pdfdocs/Alphabot2-user-manual-en.pdf [12.6.2019.]

[2]AlphaBot2-PI https://www.waveshare.com/wiki/AlphaBot2-Pi [12.6.2019.]

[3] ST188 sensor https://www.electrodragon.com/product/reflective-infrared-opto-electrical-st188-sensor/ [7.7.2019.]

[4] HC-SR04 sensor https://howtomechatronics.com/tutorials/arduino/ultrasonic-sensor-hc-sr04/ [7.7.2019.]

[5] Infrared obstacle https://www.waveshare.com/wiki/AlphaBot2-Pi#Infrared_obstacle_avoidance [5.8.2019.]

[6] GPIO https://sourceforge.net/p/raspberry-gpio-python/wiki/PWM/ [12.8.2019.]

[7] Sensors https://www.raspberrypi.org/forums/viewforum.php?f=37 [12.6.2019.]

[8] Raspberry PI https://www.raspberrypi.org/help/what-%20is-a-raspberry-pi/[12.6.2019.]

[9] Raspbian https://www.raspberrypi.org/downloads/raspbian/[12.6.2019.]

[10] WiringPI https://projects.drogon.net/raspberry-pi/wiringpi/download-and-install/ [7.7.2019.]

[11] Python library https://pypi.org/project/spidev/ [7.7.2019.]

[12] Python programming https://pythonprogramming.net/robot-remote-control-car-with-the-raspberry-pi/ [10.7.2019.]

[13] GPIO programming https://sourceforge.net/p/raspberry-gpio-python/wiki/Inputs/ [25.8.2019.]

SAŽETAK

Ovaj završni rad prikazuje primjere razvoja programske podrške za AlphaBot2-Pi mobilnu robotsku platformu gdje se ona autonomno kreće korištenjem ultrazvučnog senzora i senzora slike. Neophodno je upoznati se s modulima i komponentama koje sadrži sam robot te načinom na koji svaki od njih funkcionira kako bismo uspješno razvili i implementirali programsko rješenje. Robot se samostalno kreće i ako nailazi na prepreku, pomoću ultrazvučnog senzora i senzora slike ju prepoznaje i reagira na nju. Nakon implementiranja programskog rješenja, radi provjere uspješnosti zadatka, vršimo testiranje u kontroliranim uvjetima gdje vidimo može li se robot samostalno kretati uz ispunjenje zadanih kriterija.

ABSTRACT

This final assignment describes the examples of software development for the AlphaBot2-PI mobile robot platform where it moves autonomously using ultrasound and photosensors. It is necessary to learn about the modules and components of the robot, and their functionalities so we can successfully develop program solutions. The robot moves autonomously, and if it meets an obstacle, it recognizes it by using a photosensor or an ultrasound sensor, which reacts to it by changing the direction of movement. After the implementation of program solutions, we have to test the code in controlled laboratory conditions, where we are able to see if the robot can actually move autonomously and successfully finish its given tasks.

ŽIVOTOPIS

Stjepan Miličić rođen je 30.04.1998. u Zagrebu. Nakon završetka Osnovne Škole Mitnica Vukovar, upisuje opći smjer u Gimnaziji Vukovar. 2016. Godine upisuje Fakultet Elektrotehnike, Računarstva i Informacijskih Tehnologija Osijek te je trenutno 3. godina preddiplomskog studija računarstvo.

PRILOZI

```
#MotorSetup.py

#definiranje potrebnih biblioteka

import RPi.GPIO as GPIO

import time


#definranje klase za kretanje kotacica motora, trebamo definirati BCM numericku vrijednost pinovima

class MotorMovement(object):


    def __init__(self,L1=12,L2=13,R1=21,R2=20,LW=6,RW=26):

        self.Left1 = L1

        self.Left2 = L2

        self.Right1 = R1

        self.Right2 = R2

        self.LWheel = LW

        self.RWheel = RW


    #nakon definiranih pinova, moramo im postaviti gpio vrijednost, odnosno IN ili OUT. svi pinovi su vrijednost GPIO.OUT

        GPIO.setmode(GPIO.BCM)
```

```
    GPIO.setwarnings(False)
    GPIO.setup(self.Left1,GPIO.OUT)
    GPIO.setup(self.Left2,GPIO.OUT)
    GPIO.setup(self.Right1,GPIO.OUT)
    GPIO.setup(self.Right2,GPIO.OUT)
    GPIO.setup(self.LWheel,GPIO.OUT)
    GPIO.setup(self.RWheel,GPIO.OUT)
    self.PWMOne = GPIO.PWM(self.LWheel,500)
    self.PWMTwo = GPIO.PWM(self.RWheel,500)
    self.PWMOne.start(25)
    self.PWMTwo.start(25)
    self.stop()


def forward(self):
    self.PWMOne.ChangeDutyCycle(25)
    self.PWMTwo.ChangeDutyCycle(25)
    GPIO.output(self.Left1,GPIO.LOW)
    GPIO.output(self.Left2,GPIO.HIGH)
    GPIO.output(self.Right1,GPIO.LOW)
    GPIO.output(self.Right2,GPIO.HIGH)
```

```
def backward(self):
    self.PWMOne.ChangeDutyCycle(25)
    self.PWMTwo.ChangeDutyCycle(25)
    GPIO.output(self.Left1,GPIO.HIGH)
    GPIO.output(self.Left2,GPIO.LOW)
    GPIO.output(self.Right1,GPIO.HIGH)
    GPIO.output(self.Right2,GPIO.LOW)

def left(self):
    self.PWMOne.ChangeDutyCycle(17)
    self.PWMTwo.ChangeDutyCycle(17)
    GPIO.output(self.Left1,GPIO.HIGH)
    GPIO.output(self.Left2,GPIO.LOW)
    GPIO.output(self.Right1,GPIO.LOW)
    GPIO.output(self.Right2,GPIO.HIGH)

def right(self):
    self.PWMOne.ChangeDutyCycle(17)
    self.PWMTwo.ChangeDutyCycle(17)
    GPIO.output(self.Left1,GPIO.LOW)
```

```
        GPIO.output(self.Left2,GPIO.HIGH)
        GPIO.output(self.Right1,GPIO.HIGH)
        GPIO.output(self.Right2,GPIO.LOW)

    def stop(self):
        self.PWMOne.ChangeDutyCycle(0)
        self.PWMTwo.ChangeDutyCycle(0)
        GPIO.output(self.Left1,GPIO.LOW)
        GPIO.output(self.Left2,GPIO.LOW)
        GPIO.output(self.Right1,GPIO.LOW)
        GPIO.output(self.Right2,GPIO.LOW)
```

```
#USOAvoidance.py
import RPi.GPIO as GPIO
import time
from MotorSetup import MotorMovement

TRIG = 22
ECHO = 27

Mm = MotorMovement()
```

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


def DistanceMeasure():
        GPIO.output(TRIG,GPIO.HIGH)
        time.sleep(0.00001)
        GPIO.output(TRIG,GPIO.LOW)


   while GPIO.input(ECHO)==0:
                pass
        time1=time.time()
        while GPIO.input(ECHO)==1:
                pass
        time2=time.time()
        time3= time2-time1
        return (34000*time3)/2


try:
```

```
        while True:
                Distance = DistanceMesure()
                print("%0.2f cm"%Distance)

    if Dist <= 25:
                        Mm.left()
                        Mm.right()

    elif Dist <12:
                        Mm.backward()
        else:
                        Mm.forward()
                        time.sleep(0.05)

except KeyboardInterrupt:
        GPIO.cleanup();
```

```
#PSOAvoidance.py

import RPi.GPIO as GPIO

import time

from MotorSetup import MotorMovement


LeftSensor = 19

RightSensor = 16



Mm=MotorMovement()


GPIO.setmode(GPIO.BCM)

GPIO.setwarnings(False)

GPIO.setup(RightSensor,GPIO.IN,GPIO.PUD_DOWN)

GPIO.setup(LeftSensor,GPIO.IN,GPIO.PUD_DOWN)




try:

        while True:

    LeftSensorState = GPIO.input(LeftSensor)
```

```
                RightSensorState = GPIO.input(RightSensor)



                if(LeftSensorState == 1):
                        Mm.right()
                        time.sleep(0.005)
                        Mm.stop()
        elif(RightSensorState==1):
            Mm.left()
            time.sleep(0.005)
            Mm.stop()


                else:
                        Mm.forward()



except KeyboardInterrupt:
        GPIO.cleanup();
```