Razvoj programske podrške za Raspberry Pi mobilnu platformu namijenjenu za kretanje u fizički omeđenom prostoru

Pitlović, Filip

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

Sveučilišni preddiplomski studij računarstva

Razvoj programske podrške za Raspberry Pi mobilnu platformu namijenjenu za kretanje u fizički omeđenom prostoru

Završni rad

Filip Pitlović

Osijek, 2021.

SADRŽAJ

1. UVOD

Robotika je danas postala dijelom skoro svih djelatnosti kojima se čovjek bavi. Sve češće pronalaze se projekti koji su zasnovani na robotici. Pojavljuju se roboti namijenjeni svakodnevnoj upotrebi, primjerice samohodni usisavač. U industrijama koriste se roboti koji služe kao zamjena ljudima, obavljaju poslove koji su zahtjevni i opasni po ljude. Imaju veliku ulogu u poslovima u kojima je potrebna preciznost u radu, preciznost koju roboti zasigurno imaju bolju od ljudi.

U ovom radu opisana je izrada programske podrške za *Raspberry Pi* mobilnu platformu namijenjenu za kretanje u fizički omeđenom prostoru. Izrada je prikazana u šest poglavlja. U drugom poglavlju objašnjen je model robota *AlphaBot2*. U trećem poglavlju obrađena je detaljno *Raspberry Pi* mobilna platforma. U četvrtom poglavlju opisano je programsko rješenje za kretanje u fizički omeđenom prostoru te mapiranje prostora. U petom poglavlju, obavljeno je i opisano testiranje već objašnjenog pogramskog riješenja te opisana je izrada okruženja za testiranje. Na samom kraju priložen je zaključak.

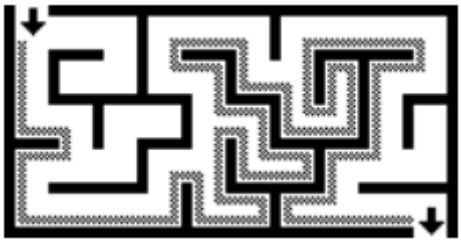
2. POSTOJEĆA RJEŠENJA I ALGORITMI

2.1. Postojeći algoritmi rješavanja labirinta

Prema [1], postoje razna rješenja koja omogućuju prolazak robota kroz labirint, odnosno *Maze runner* algoritmi. Neki od algoritama objašnjeni su u nastavku.

2.1.1. Wall follower

Najkorišteniji algoritam za rješavanje labirinta je *wall follower*, pravilo desnog odnosno lijevog zida. Prema [2], *wall follower* jednostavan je algoritam koji se temelji na tome da je jedna strana robota uvijek u vezi s jednom stranom zida što garantira uspješan izlazak iz labirinta. Algoritam se temelji na obilasku stabla po dubini. U ovoj metodi postoji ograničenje a ono je da labirint ne smije biti složeno povezan, odnosno labirint ne smije sadržavati petlje i putove koji se križaju. Ako labirint sadrži takvo što, robot neće doći do cilja. Za korištenje ovog algoritma, potrebni su bočni senzori odnosno jedan bočni senzor koji bi pratio jednu stranu zida do cilja (Sl. 2.1.).

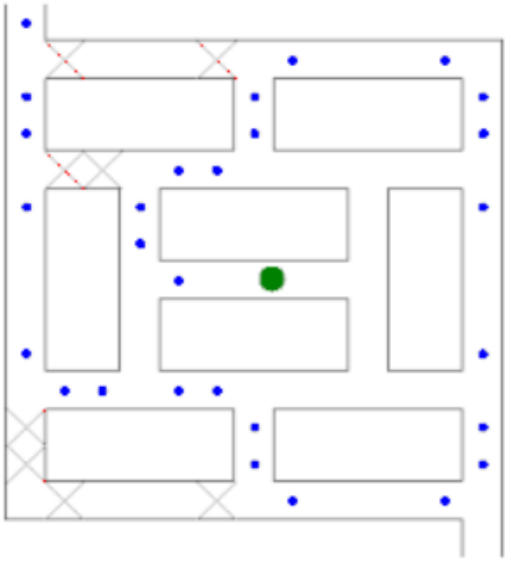


Sl. 2.1. *Wall follower* algoritam [1]

2.1.2. *Tremaux* algoritam

Malo složeniji i učinkovitiji algoritam prolaska kroz labirint je Tremaux algoritam. Prema [2], algoritam će biti učinkovit za sve dobro definirane labirinte, ali neće sa sigurnošću pronaći i najkraći put. Način na koji algoritam funkcionira je slijedeći. Prilikom prolaska robota kroz određeni puta, taj put je potrebno označiti. Oznake se mogu pohraniti ili fizički ili putem računalnog algoritma. Oznake se postavljaju na oba kraja staze. Pravilo je da robot ne prolazi kroz put koji je označen na oba kraja. Ako robot dođe do raskrižja na kojem nema oznaka, proizvoljno bira put te ga označava. U suprotnom, ako robot dođe do puta koji ima samo jednu oznaku, okreće se i vraća istom stazom označavajući ju ponovno. To je primjer slijepe ulice. Pravilo „okreni se i vrati“ učinkovito pretvara labirint s petljama u jednostavni povezani labirint. Kada god se pronađe

put koji je zatvorio petlju, gleda se kao slijepa ulica i ide se natrag. Na dolasku u cilj labirinta, staze koje su označene samo na jednom kraju pokazuju put do početka (Sl. 2.2.).



Sl. 2.2. *Tremaux* algoritam. [2]

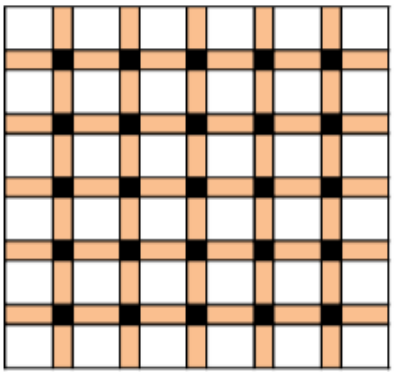
2.1.3. Dead - end filling

Prema [2], ovaj algoritam omogućuje rješavanje labirinta na način da ispunjava sve slijepe ulice a one ispravne ostavlja neispunjenima. Metoda radi na način da prvo pronađe sve slijepe ulice u labirintu a zatim ispuni putanju od svakog slijepog ugla sve dok se ne spoji sa prvim spojem. Ovaj algoritam ne može slučajno odvojiti početak od cilja jer svaki korak procesa čuva topologiju labirinta. Također, proces neće stati „prerano“ budući da krajnji rezultat ne može sadržavati slijepe ulice. Dakle, ako se slijepo punjenje vrši na savršenom labirintu odnosno labirintu bez petlji, tada će ostati samo rješenje. U suprotnom će ostati svako moguće rješenje.

2.2. Postojeća rješenja mapiranja prostora

2.2.1. *Flood - Fill* algoritam

Prema [2], jedan od kompleksnijih, ali učinkovitih algoritama za mapiranje prostora je *Flood - Fill* algoritam. Za rješavanje labirinta, robot mora znati veličinu labirinta zbog toga što radi na principu dijeljenja prostora na ćelije. Prema [3], ovaj algoritam se temelji na matrici. Na primjeru 6*6 matrice između dvije ćelije može biti zid. Dakle, u nizu od šest ćelija, između njih je pet zidova. Ukupno postoji 11 jedinica ćelija. To je pohranjeno u nizu 11 * 11 (Sl. 2.3.). Bijele ćelije su prostor u koje se robot može smjestiti a narančasto su lokacije za zidove.



Sl. 2.3. Matrica 6*6 koja predstavlja labirint. [3]

Slijedi ažuriranje podataka o zidovima. Na principu jednog od prolaska kroz labirint koji je obašnjen u poglavlju ranije, robot ažurira podatke o zidovima. Nakon ažuriranja informacija o zidu u trenutnoj ćeliji, robot počinje preplavljivati matricu kako bi pronašao cilj. Algoritam dodjeljuje vrijednost svakoj ćeliji na temelju toga koliko je ona udaljena od odredišne ćelije odnosno cilja. Na temelju toga, ćelija cilja dobiva vrijednost 1. Ako robot stoji na ćeliji s vrijednosti 4, to znači da će robotu trebati 3 koraka do ciljne ćelije. Algoritam pretpostavlja da se robot ne može kretati dijagonalno i može napraviti samo okrete od 90 stupnjeva. Nakon što je robot odlučio u kojem će smjeru krenuti, vraća količinu stupnjeva koje je potrebno okrenuti da bi otišli na predviđenu ćeliju. Nakon okreta robota ažurira se nova orijentacija robota, odnosno je li robot okrenut na sjever, jug, istok ili zapad. Nakon što robot označi sva potrebna polja u matrici, time je mapirao prostor.

3. MODEL ROBOTA

3.1. *AlphaBot2-Pi*

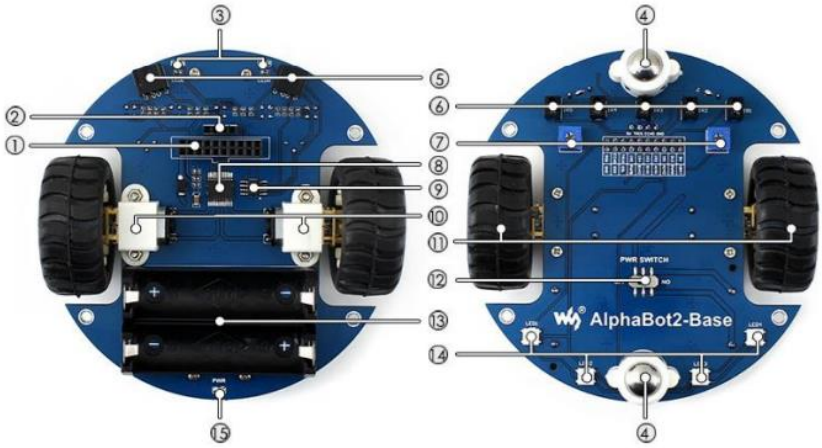
Model robota za izradu završnog rada je *AlphaBot2*. Prema [4], postoji više vrsta *AlphaBot2* robota od kojih su neki: *AlphaBot2-Pi*, *AlphaBot2-PiZero*, *AlphaBot2-Ar*. U ovom radu korišten je robot *AlphaBot2-Pi* čiji je glavni dio Raspberry Pi 3 mobilna platforma koja je opisana u idućem poglavlju (Sl. 3.1.).

Sl. 3.1. *AlphaBot2-Pi* [4]

3.2. Dijelovi robota

Glavni dijelovi robota su kućište i adapterska ploča.

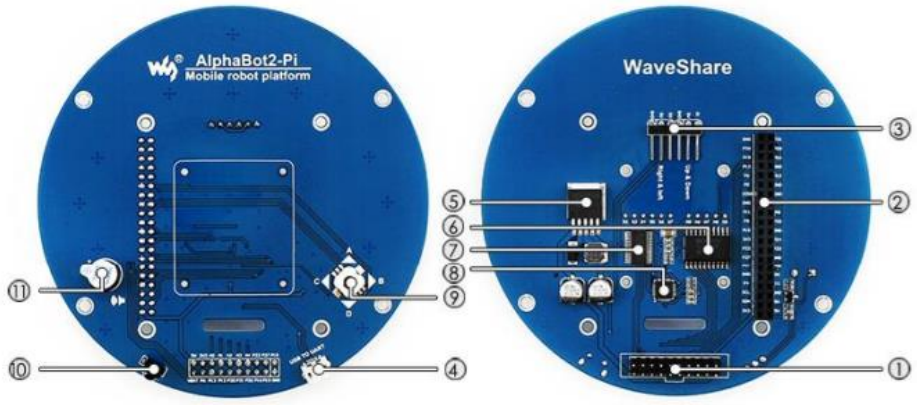
Prema [4], dijelovi kućišta *AlphaBot2-Pi su* (Sl. 3.2.):



Sl. 3.2. Kućište *AlphaBot2-Pi* [5]

1. *AlphaBot2* kontrolno sučelje: za povezivanje vrsta adapterske ploče regulatora,
2. sučelje ultrazvučnog modula,
3. pokazatelji za izbjegavanje prepreka,
4. višenamjenski kotač,
5. ST188: reflektirajući infracrveni fotoelektrični senzor za izbjegavanje prepreka,
6. ITR20001/T: reflektirajući infracrveni fotoelektrični senzor, za praćenje linija,
7. potenciometar za podešavanje dometa izbjegavanja prepreka,
8. TB6612FNG pokretač dvostrukog H-mosta,
9. LM393 usporednik napona,
10. brzina smanjenja motora N20 mikro zupčanika 1:30, 6V / 600 o / min,
11. gumeni kotači promjera 42 mm, širine 19 mm,
12. prekidač za napajanje,
13. držač baterije: podržava 14500 baterija,
14. WS2812B: RGB LED diode u boji,
15. indikator snage.

Prema [4], dijelovi adapterske ploče *AlphaBot2-Pi su* (Sl. 3.3.):



Sl. 3.3. Adapterska ploča *AlphaBot2-Pi* [6]

1. *AlphaBot2* kontrolno sučelje: za povezivanje *AlphaBot2-Base,*
2. sučelje Raspberry Pi: za povezivanje *Raspberry Pi* 3 Model B,
3. servo sučelje,
4. USB UART: jednostavan za upravljanje Pi putem UART-a,
5. LM2596: 5V regulator napona,

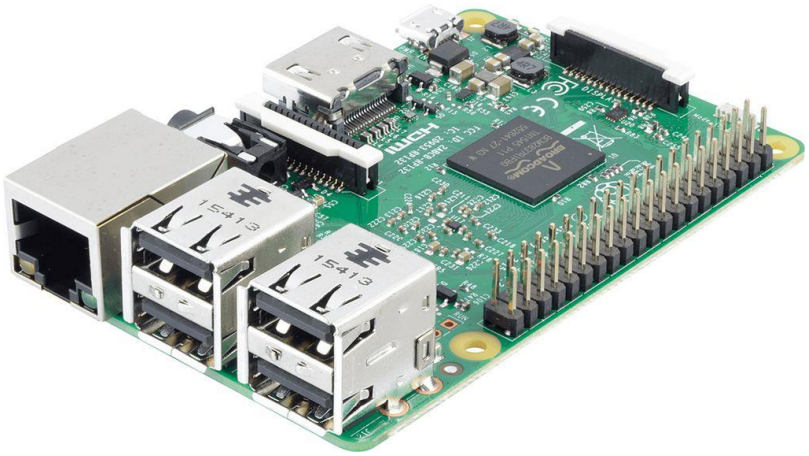
6. TLC1543: 10-bitni čip za prikupljanje AD-a,
7. PCA9685: servo upravljač,
8. CP2102: USB TO UART pretvarač,
9. *joystick*,
10. IR prijamnik,
11. zujalica.

4. *RASPBERRY PI* MOBILNA PLATFORMA

4.1. *Raspberry Pi*

Raspberry Pi malo je računalo veličine 85 mm*56 mm*17 mm koje se priključuje na monitor koristi tipkovnicu i miš. Prema [5], ono je sposoban mali uređaj koji omogućava ljudima svih dobnih skupina da istražuju računarstvo i nauče kako programirati na jezicima kao što su Scratch i Python. Sposoban je raditi sve što i obično stolno računalo, od pregledavanja interneta i reprodukcije video zapisa visoke definicije, do izrade proračunskih tablica, obrade teksta i igranja igara (Sl. 4.1.).

Raspberry Pi izradila je *Raspberry Pi Foundation*, dobrotvorna organizacija u Velikoj Britaniji kojoj je cilj educirati ljude u računarstvu i stvoriti lakši pristup računalnom obrazovanju.



Sl. 4.1. *Raspberry Pi 3* [7]

4.2. *Raspberry Pi* 3 postavke

Prema [6], *Raspberry pi 3* karakteristike su:

- *Broadcom BCM2837 SoC (4 ARM Cortex-A53),*
- *1 GB RAM,*
- integrirani *WiFi,*
- *HDMI* priključak, Internet priključak i četiri USB priključka.

Prema [6], Pinovi koji se koriste:

- dva pina od 5 Volti,
- dva pina od 3.3 Volti,
- 8 *ground* pinova,
- 26 podatkovnih pinova,
- 1 pin PWM.

Prema [7], *Raspberry Pi 3* ima 26 *GPIO* lemljenih igala i 5 *GPIO* pinova koje trebate lemiti. Oni se mogu koristiti za kontrolu elektronike spojene na *Raspberry Pi*. Među elementima koji se mogu učiniti su:

- upaliti *LED* diode,
- postaviti gumbove,
- koristiti releje,
- kontrolirali motore.

5. PROGRAMSKO RJEŠENJE

Na samom početku realizacije zadatka potrebno je osposobiti robota. Prema [8], prvi je korak spajanje *Raspberry Pi*-a, koji je dio robota, zajedno s računalom. Za to je potrebno pronaći *IP* adresu *Raspberry Pi*-a. Pomoću *HDMI* kabla potrebno je povezati *Raspberry Pi* s monitorom. Također potrebno je povezati i tipkovnicu na *Raspberry Pi*. Zatim nakon unosa podataka za logiranje u operacijski sustav *Raspberry Pi*-a, potrebno je omogućiti *SSH*. *SSH* je mrežni protokol koji korisnicima omogućuje uspostavu sigurnog komunikacijskog kanala između 2 računala putem računalne mreže. Zatim, pomoću naredbe „hostname -i“ moguće je saznati *IP* adresu *Raspberry Pi*-a. Sada nakon pronalaska *IP* adrese moguće je spojiti *Raspberry Pi* s računalom. Spajanje je moguće pomoću programa *Putty* koji za ulogiranje traži *IP* adresu, korisničko ime i lozinku. Nakon uspješnog spajanja omogućena je interakcija računala i *Raspberry Pi*-a.

Programska rješenja mogu se pisati na računalu i zatim prenijeti na Raspberry pomoću programa *WinSCP*. Za ulogiranje putem *WinSCP* također je potrebna *IP* adresa, korisničko ime te lozinka. Također, moguće je pisati sam programski kod preko terminala. *WinSCP* omogućuje brzo i efikasno prebacivanje programske datoteke na *Raspberry* operacijski sustav što također nudi jednostavnije pisanje i brisanje koda u samom editoru nego u terminalu.

5.1. Python

Za realizaciju ovog zadatka upotrebljeno je znanje programiranja u *Pythonu*. Prema [9], *Python* je programski jezik opće namjene, jezik visoke razine i jezik otvorenog koda koji će se koristiti za rad s *Raspberry Pi*-om. Fleksibilan zato što omogućuje objektno orijentirano, strukturno i aspektno orijentirano programiranje. *Raspberry Pi* sadrži iglice (*inpute*) kojima se upravlja putem raznih biblioteka kao što su RPIO.GPIO, pigpio, gpiozeo, RPi.GPIO. Za izradu programskog riješenja korištena je Rpi.GPIO biblioteka.

Python je korišten za razvoj programskog rješenja u 3 razine: pokretanje i kontrola motora, kretanje kroz prostor izbjegavajući prepreke koristeći ultrazvučne senzore te mapiranje prostora. Kontrola motora odnosi se na manipuliranje izlaznim signalima *Raspberry Pi* računala pomoću *Python* programa, čime se aktiviraju motori koji omogućuju kretanje robota.

5.2. Pokretanje motora

Za početak potrebno je postaviti odgovarajuće pinove na kotače. Potrebno je definirati RPi.GPIO i time biblioteke. Prema [10,11], Rpi biblioteka sadrži klase koje kontroliraju GPIO na *Raspberry* operacijskom sustavu i omogućuje da pinove definiramo kao ulaze odnono izlaze. Time bibilioteka omogućuje uzimanje i vraćanje vremena kao varijable. Dakle, prvo slijedi definiranje pinova i povezivanje istih s motorima (Sl. 5.1.).

```
 1 import RPi.GPIO as GPIO
 2 import time
 3
 4 class AlphaBot2(object):
 5
 6         def __init__(self,ain1=12,ain2=13,ena=6,bin1=20,bin2=21,enb=26):
 7                 self.AIN1 = ain1
 8                 self.AIN2 = ain2
 9                 self.BIN1 = bin1
10                 self.BIN2 = bin2
11                 self.ENA = ena
12                 self.ENB = enb
13                 self.PA  = 50
14                 self.PB  = 50
15
16                 GPIO.setmode(GPIO.BCM)
17                 GPIO.setwarnings(False)
18                 GPIO.setup(self.AIN1,GPIO.OUT)
19                 GPIO.setup(self.AIN2,GPIO.OUT)
20                 GPIO.setup(self.BIN1,GPIO.OUT)
21                 GPIO.setup(self.BIN2,GPIO.OUT)
22                 GPIO.setup(self.ENA,GPIO.OUT)
23                 GPIO.setup(self.ENB,GPIO.OUT)
24                 self.PWMA = GPIO.PWM(self.ENA,500)
25                 self.PWMB = GPIO.PWM(self.ENB,500)
26                 self.PWMA.start(self.PA)
27                 self.PWMB.start(self.PB)
28                 self.stop()
```

Sl. 5.1. Definiranje biblioteka i postavljanje pinova

Svakom pinu dodjeljuje se brojčana vrijednost te vrijednost ulaza ili izlaza. Varijable *self*.PA i *self*.PB služe za postavljanje vrijednosti brzine rada motora. U nastavku, primjerice metoda PWM.*start*(*self*.PA), omogućuje pokretanje prvog motora i to brzinom 50% koja je zadana u varijabli iznad. Također, potrebno je postaviti način rada, u našem slučaju GPIO.BCM. GPIO BCM odnosi se na pin prema „*Broadcom SOC kanalu*“. Ovaj način rada govori koji će se sustav za numeriranje pin -ova namjeravati koristiti. BOARD označava korištenje fizičkih brojeva pinova na RPi P1 konektoru.

Nakon definiranja i postavljanja svih dijelova potrebnih za pokretanje motora, slijedi pisanje metoda za kretanje robota naprijed, nazad, lijevo i desno. Najvažnija naredba je *ChangeDutyCycle* koja kao parametar prima brzinu rada motora odnosno kretanja kotača. Ukoliko je potrebno da se robot kreće prema naprijed, potrebno je postaviti oba motora na *HIGH* koja omogućuju da se oba kotača kreću prema naprijed. Za kretanje u desno, postavlja se da lijevi motor ide naprijed (*HIGH*) a desni nazad (*LOW*) što omogućuje rotiranje robota u desno. Analogno tome, postavlja se rad motora za preostale 2 funkcije kretanja, nazad i lijevo. Ukoliko je potrebno da se robot zaustavi, svi motori postavljaju se na *LOW* (Sl. 5.2.).

```
        def forward(self):
                self.PWMA.ChangeDutyCycle(self.PA)
                self.PWMB.ChangeDutyCycle(self.PB)
                GPIO.output(self.AIN1,GPIO.LOW)
                GPIO.output(self.AIN2,GPIO.HIGH)
                GPIO.output(self.BIN1,GPIO.LOW)
                GPIO.output(self.BIN2,GPIO.HIGH)

        def stop(self):
                self.PWMA.ChangeDutyCycle(0)
                self.PWMB.ChangeDutyCycle(0)
                GPIO.output(self.AIN1,GPIO.LOW)
                GPIO.output(self.AIN2,GPIO.LOW)
                GPIO.output(self.BIN1,GPIO.LOW)
                GPIO.output(self.BIN2,GPIO.LOW)

        def backward(self):
                self.PWMA.ChangeDutyCycle(self.PA)
                self.PWMB.ChangeDutyCycle(self.PB)
                GPIO.output(self.AIN1,GPIO.HIGH)
                GPIO.output(self.AIN2,GPIO.LOW)
                GPIO.output(self.BIN1,GPIO.HIGH)
                GPIO.output(self.BIN2,GPIO.LOW)

        def left(self):
                self.PWMA.ChangeDutyCycle(30)
                self.PWMB.ChangeDutyCycle(30)
                GPIO.output(self.AIN1,GPIO.HIGH)
                GPIO.output(self.AIN2,GPIO.LOW)
                GPIO.output(self.BIN1,GPIO.LOW)
                GPIO.output(self.BIN2,GPIO.HIGH)

        def right(self):
                self.PWMA.ChangeDutyCycle(30)
                self.PWMB.ChangeDutyCycle(30)
                GPIO.output(self.AIN1,GPIO.LOW)
                GPIO.output(self.AIN2,GPIO.HIGH)
                GPIO.output(self.BIN1,GPIO.HIGH)
                GPIO.output(self.BIN2,GPIO.LOW)

        def setPWMA(self,value):
                self.PA = value
```

```
43                 self.PWMA.ChangeDutyCycle(self.PA)
44
45         def setPWMB(self,value):
46                 self.PB = value
47                 self.PWMB.ChangeDutyCycle(self.PB)
```

Sl. 5.2. Funkcije za kretanje motora

5.3. Rješavanje labirinta

Vodeći se kroz algoritme i razna već postojeća rješenja bilo je potrebno osmisliti i razviti algoritam za samostalno kretanje robota kroz prostor. Algoritam koji je korišten za izradu rada temelji se na rješenju *'Wall follower'* algoritma, ali je prilagođen za robota koji sadrži samo prednje senzore. Također, mapiranje prostora odrađeno je na ideju *'Flood-fill'* algoritma. Dijelovi robota potrebni za izvršavanje algoritma su 2 ultrazvučna senzora koja se nalaze s prednje strane robota a njihova uloga objašnjena je u nastavku.

5.3.1. Ultrazvučni senzori

AlphaBot2 robot sadrži 2 ultrazvučna senzora s prednje strane robota. Ona omogućuju prepoznavanje prepreka ispred sebe tako što šalju ultrazvučne valove koji se odbijaju i šalju signale natrag senzorima. Time omogućuju i računanje udaljenosti od prepreke ispred što će biti objašnjeno u nastavku. Ultrazvučnim senzorima zvanim *'TRIG'* i *'ECHO'* potrebno je postaviti vrijednosti (Sl. 5.3.).

```
1  #ultrazvučni senzori
2  TRIG = 22
3  ECHO = 27
4  GPIO.setmode(GPIO.BCM)
5  GPIO.setwarnings(False)
6  GPIO.setup(TRIG,GPIO.OUT,initial=GPIO.LOW)
7  GPIO.setup(ECHO,GPIO.IN)
```

Sl. 5.3. Definiranje ultrazvučnih senzora

5.3.2. Računanje udaljenosti

Kao što je već spomenuto, ultrazvučni senzori omogućuju računanje udaljenosti do prepreke koja se nalazi na njihovoj putanji. Metoda za računanje udaljenosti radi na način da se ispituje stanje *ECHO* varijable mijenjanje *TRIG* vrijednosti iz *HIGH* u *LOW*, odnosno vrijeme koje je potrebno da varijabla *ECHO* pređe iz stanja *LOW* u stanje *HIGH* predstavlja udaljenost. Naravno, potrebno je pretvoriti vrijeme u mjernu jedinicu za udaljenost. Udaljenost se mjeri na slijedećem principu. Odašiljač šalje signal, a zatim *while* petlja provjerava je li se signal vratio. Ukoliko se signal nije

vratio izvršava se prva petlja koja u varijablu *'time'* zabilježava vrijeme. Ukoliko se signal vratio izvršava se druga petlja te se zabilježava vrijeme povratnog signala. Zatim se oduzimaju vrijeme dolaska signala i vrijeme poslanja signala. Ta razlika se množi sa 34000 što predstavlja brzinu zvuka u centimentrima po sekundi i još se podijeli s dva (Sl. 5.4.).

```
def Distance():
        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
```

Sl. 5.4. Računanje udaljenosti putem ultrazvučnih senzora

5.3.3. Samostalno kretanje kroz labirint do cilja

Nakon pripreme ultrazvučnih senzora i napisane metode za računanje udaljenosti slijedi obrada algoritma za samostalno kretanje kroz prostor. Algoritam se temelji na idejama koje su opisane u poglavlju 2.2. koje se odnose na već postojeća rješenja samostalnog kretanja kroz prostor i stizanja na cilj labirinta.

Robot je na početku labirinta postavljen gledajući prema sjeveru. Algoritam je osmišljen na način da se robot svakih 20 centimetara zaustavi. Nakon prvih pređenih 20 centimetara robot staje. Pomoću ultrazvučnih senzora očitava udaljenost do prepreke ispred i sprema ju u varijablu koja se zove *'Distance_Front'*. Ukoliko je *'Distance_Front'* veći od 200 centimetara znači da je robot došao do izlaza i algoritam završava. Ukoliko nije, robot se okreće za 90 stupnjeva desno, prema istoku. Robot pomoću ultrazvučnih senzora očitava postoji li prepreka pri okretu u desno. Udaljenost se sprema u varijablu *'Distance_Right'*. Ukoliko ultrazvučni senzori ne očitavaju udaljenost od zida manju od 10 centimetara znači da se pred njim ne nalazi zid i moguć je prolaz desno, prolaz prema istoku. Temeljno pravilo je pravilo desne strane te zbog toga robot nastavlja ravno u smjeru kojem je okrenut, prema istoku. Ukoliko je pri mjerenju *'Distance_Right'* manji od 10 centimetara znači da se ispred robota s desne strane nalazi zid i da robot tu ne može proći. Slijedi provjera varijable *'Distance_Front'*. Ukoliko je udaljenost prepreke ispred koja je izmjerena na početku između 20 i 200 centimetara znači da robot može proći ravno. Zato, pravilo koje vrijedi poslije pravila desne strane je pravilo prednje strane. Ukoliko robot nema prepreku, okreće se za 90 stupnjeva u desno u prvobitni položaj i kreće naprijed 20 centimetara. U slučaju da je

'Distance_Front' manji od 20 centimetara, robot se okreće za 180 stupnjeva u lijevo i provjerava postoji li prepreka s lijeve strane. Udaljenost s lijeve strane sprema se u varijablu *'Distance_Left'*. Ako je *'Distance_Left'* veći od 20 centimetara robot nastavlja naprijed u tom smjeru, smjeru zapada. Ukoliko je *'Distance_Left'* manji od 20 centimetara znači da se sa sve 3 strane nalazi zid. Slijedi *if* uvjet koji provjerava nalaze li se zidovi sa sve 3 strane. Ukoliko da, robot se okreće za 90 stupnjeva lijevo i vraća se u položaj za povratak nazad. Nakon toga, ponavlja se algoritam i kreće ispočetka zajedno sa pravilom desne strane pa tako dalje. Robot dolazi na cilj kada ultrazvučni valovi očitaju udaljenost *'Distance_Front'* veću od 200 centimetara i tu se algoritam prekida. Programski kod dan je u nastavku zajedno s komentarima koji objašnjavaju pojedine linije (Sl. 5.5.).

```
def MazeSolver():
        if zastavica = 1
                element_out_of_stack = stack.pop()
        time.sleep(1)
        print("20 cm naprijed")
        Ab.forward(
        time.sleep(1.5) #1.5 sekundu odnosno 20 cm robot ide naprijed
        Ab.stop()
        time.sleep(1) #1 sekundu robot stoji
        Distance_Front = Distance() #racunanje udaljenosti do prepreke
                                    ispred
        print("Distance_Front = %0.2f cm" %Distance_Front)
        if Distance_Front > 200: #ako je udaljenost do prepeke ispred veca
                                  od 00, znaci da je kraj labirinta
                Ab.stop()
                print("Izlazak iz labirinta je uspjesan!!")
                return 0
        Ab.right() #robot se okrece za 90 stupnjeva desno zbog potrebe
                   senzora
        time.sleep(0.4)
        Ab.stop()
        time.sleep(2)
        Distance_Right = Distance() #racunanje udaljenosti do prepreke s
                                     desne strane
        print("Distance_Right = %0.2f cm" %Distance_Right)
        if Distance_Right >=20: #ako nema desno zida nastavi u tom smjeru,
                                 prema istoku
                last_element = Stack.peek()
                if zastavica = 1 and element_out_of_stack = 2
                        time.sleep(1)
                        Stack.push(1)
                        zastavica = 0
                        print("Nema zida desno, skretanje desno")
                        return MazeSolver()
                if zastavica = 1:
                        time.sleep(1)
                        Stack.push(3)
                        zastavica = 0
                        print("Nema zida desno, skretanje desno")
```

```
                        return MazeSolver()
                elif zastavica = 0:
                        time.sleep(1)
                        Stack.push(2)
                        print("Nema zida desno, skretanje desno")
                        return MazeSolver()
        if (20 <= Distance_Front <= 200):
                if zastavica = 0:
                        Ab.left() #robot se vraća za 90 stupnjeva prema
                                  lijevo, na sjever
                        time.sleep(0.4)
                        print("Vozi ravno")
                        Stack.push(1)
                        return MazeSolver()
        Ab.left()
        time.sleep(1.05) #robot se okrece za 180 stupnjeva na lijevo
        Ab.stop()
        time.sleep(2)
        Distance_Left = Distance()#racunanje udaljenosti do prepreke s
                                  lijeva
        print("Distance_Left = %0.2f cm" %Distance_Left)
        elif Distance_Left >= 20: #ako s lijeve strane nema zida, idi
                                  lijevo
                if zastavica = 0:
                        Stack.push(3)
                        time.sleep(1)
                        print("Nema zida lijevo, skretanje lijevo")
                        return MazeSolver()
        elif ((Distance_Left <= 20 ) and (Distance_Front <= 20) and
             (Distance_Right <=20)): #ako je sa sve 3 strane zid, okreni
                                      se
               zastavica = 1
                print("Okreni se nazad")
               Ab.left()
                time.sleep(0.5)
               return MazeSolver()
```

Sl. 5.5. Algoritam za samostalno kretanje kroz labirint do cilja

5.4. Mapiranje prostora

Kako robot prilikom prolaska prvi puta kroz labirint prolazi kroz sve dijelove labirinta, također I slijepe ulice, potrebno je osmisliti način mapiranja prostora kako bi robot iz drugog prolaska kroz labirint prošao samo onim putem koji direktno vodi do izlaza. Za takvo nešto potrebno je mapiranje prostora odnosno labirinta.

U prošlom poglavlju, u dijelu u kojem se nalazi kod, mogu se pronaći dijelovi koda koji su vezani za stog, a nemaju veze sa algoritmom samostalnog kretanja kroz labirint. Upravo stog, temelj je algoritma mapiranja prostora danog u nastavku. Prvo je potrebno implementirati stog. Funkcije *push* i *pop* omogućuju ubacivanje i izbacivanje elemenata iz stoga (Sl. 5.6.).

```
class Stack:
    def __init__(self):
        self.elements = []
    def push(self, data):
        self.elements.append(data)
        return data
    def pop(self):
        return self.elements.pop()
    def peek(self):
        return self.elements[-1]
    def get_stack(self):
        return self.elements
    def is_empty(self):
        return self.size == 0
```

Sl. 5.6. Implementacija stoga

Način na koji je algoritam osmišljen je slijedeći. Robot nakon svakog stajanja i ponovnog kretanja svakih 20 centimetara, što je objašnjeno u poglavlju prije, ubacuje na stog smjer kretanja. Na početku, odmah je dodan smjer 'naprijed' kao startno kretanje. Broj 1 na stogu označava naprijed, 2 označava desno, a 3 označava kretanje lijevo. U nastavku algoritma, ukoliko robot na slijedećem stajalištu skreće desno na stog se dodaje broj 2 i tako se popunjava stog do kraja.

Javlja se problem slijepe ulice. Što kada robot ubaci smjerove na stog, ali se mora vratiti nazad? To je riješeno na način da, ukoliko robot dođe do kraja slijepe ulice i potrebno je vratiti se nazad, on izbacuje sa stoga element po element na način da prilikom očitavanja slijepe ulice postavlja se zastavica na 1. Sve dok je zastavica jednaka 1 u ponovnom pokretanju algoritma se izbacuje jedan element sa stoga. Izbacuje se sve dok se zastavica ne postavi na 0. Zastavica se postavlja na nula tek onda kada robot skrene prema desno, što mu označava da je krenuo novim putem i tada se u stogu nalaze samo elementi koji označavaju pravi put do cilja. Bitno je prije promjene zastavice na 0 u *if* uvjetu, koja je vezana za skretanje desno, provjeriti je li zastavica na 1. Ukoliko je zastavica = 1 i zadnji element izbacen sa stoga je desno, na stog se push-a smjer naprijed. Ukoliko je zastavica = 1, ali zadnji element nije naprijed, na stog se neće *push*-ati skretanje desno već skretanje lijevo, jer u ponovnom prolasku robota on zapravo skreće lijevo a ne desno. Potpun kod dan je u nastavku, a komentarima su sada označeni dijelovi vezani za mapiranje prostora pomoću stoga (Sl. 5.7.).

```
# 1 - naprijed
# 2 - desno
# 3 - lijevo
Stack = Stack()
Stack.push(1) #prvi element je uvijek naprijed (1)
def MazeSolver():
        if zastavica = 1 #znaci da se vraca nazad
                element_out_of_stack = stack.pop() #vrijednost izbačenog
                                                    elementa
        time.sleep(1)
        print("20 cm naprijed")
        Ab.forward(
        time.sleep(1.5)
        Ab.stop()
        time.sleep(1)
        Distance_Front = Distance()
        print("Distance_Front = %0.2f cm" %Distance_Front)
        if Distance_Front > 200:
                Ab.stop()
                print("Izlazak iz labirinta je uspjesan!!")
                return 0
        Ab.right()
        time.sleep(0.4)
        Ab.stop()
        time.sleep(2)
        Distance_Right = Distance()
        print("Distance_Right = %0.2f cm" %Distance_Right)
        if Distance_Right >=20:
                last_element = Stack.peek()
                if zastavica = 1 and element_out_of_stack = 2:# ako se
                   robot vraca nazad i element izbacen sa stoga je
                  'desno', neka pusha opet 'naprijed'
                        time.sleep(1)
                        Stack.push(1) #push-a se smjer 'naprijed'
                        zastavica = 0 #ponistava zastavicu, znaci da vise
                                       ne ide istim putem
                        print("Nema zida desno, skretanje desno")
                        return MazeSolver()
                if zastavica = 1: #ako je zastavica 1, neka pusha
                                   'lijevo'
                        time.sleep(1)
                        Stack.push(3)
                        zastavica = 0 #ponistava zastavicu, znaci da vise
                                       ne ide istim putem
                        print("Nema zida desno, skretanje desno")
                        return MazeSolver()
                elif zastavica = 0:
                        time.sleep(1)
                        Stack.push(2)
                        print("Nema zida desno, skretanje desno")
                        return MazeSolver()
        if (20 <= Distance_Front <= 200):
                if zastavica = 0:
                        Ab.left()
                        time.sleep(0.4)
                        print("Vozi ravno")
```

```
57                          Stack.push(1)
58                          return MazeSolver()
59          Ab.left()
60          time.sleep(1.05)
61         Ab.stop()
62         time.sleep(2)
63          Distance_Left = Distance()
64         print("Distance_Left = %0.2f cm" %Distance_Left)
65         elif Distance_Left >= 20:
66                  if zastavica = 0:
67                          Stack.push(3)
68                          time.sleep(1)
69                          print("Nema zida lijevo, skretanje lijevo")
70                          return MazeSolver()
71         elif ((Distance_Left <= 20 ) and (Distance_Front <= 20) and
72 (Distance_Right <=20)):
73                  zastavica = 1
74                  print("Okreni se nazad")
75                  Ab.left()
76                  time.sleep(0.5)
77                  return MazeSolver()
```

Sl. 5.7. Mapiranje prostora pomoću stoga

Nakon što robot dođe do izlaska, stog je popunjen skretanjima koja označuju pravilan i direktan put do cilja bez ulaska u petlje. Poznato je kako stog radi na način *First in last out*, zato je potrebno prvo stog prebaciti na novi stog. Tada zadnji element postaje prvi element u novom stogu, i omogućuje jednostavniji način algoritma za ponovni prolazak (Sl. 5.8.).

```
1  def New_stack():
2          Stack_new = Stack()
3         while Stack.is_empty() != 0:
4                  direction = Stack.pop()
5                  Stack_new.push(direction)
6         print("Smjerovi za ponovni prolazak labirinta su: ")
7         print(Stack_new.get_stack())
```

Sl. 5.8. Stvaranje novog stoga s putanjama

Algoritam prolazi kroz novi stog, odnosno nazvan *Stack_new*. Prvo se provjerava je li stog prazan, ukoliko nije izbacuje se zadnji element iz stoga koji predstvalja prvi smjer kretanja koji je zapisan s početka prvog prolaska labirintom. Ukoliko je smjer jednak broju jedan, robot se kreće 20 centimetara naprijed. Ukoliko je smjer jednak broju 2, robot se okreće desno za 90 stupnjeva i vozi 20 centimetara ravno u smjeru istoka. Ukoliko je smjer jednak broju 3, robot se okreće lijevo za 90 stupnjeva i vozi 20 centimetara ravno u smjeru zapada. Algoritam se analogno nastavlja dalje. Kada uvjet da je stog prazan bude istinit znači da je robot stigao do cilja i ispisuje se poruka (Sl. 5.9.).

```
def MazeSolverSecondTurn():
        if Stack.notEmpty() != 0:
                direction = Stack_new.pop()
                print("Preostali smjerovi za ponovni prolazak labirinta
                       su: ")
                print(Stack_new.get_stack())
                if smjer = 1:
                        Ab.forward()
                        time.sleep(1.5)
                elif direction = 2:
                        Ab.right()
                        time.sleep(0.4)
                        Ab.stop()
                        time.sleep(2)
                        Ab.forward()
                        time.sleep(1.5)
                elif direction = 3:
                        Ab.left()
                        time.sleep(0.4)
                        Ab.stop()
                        time.sleep(2)
                        Ab.forward()
                        time.sleep(1.5)
        elif:
                print("Cilj! Izlazak iz labirinta je uspješan!")
```

Sl. 5.9. Drugi prolazak kroz labirint uz poznate putanje

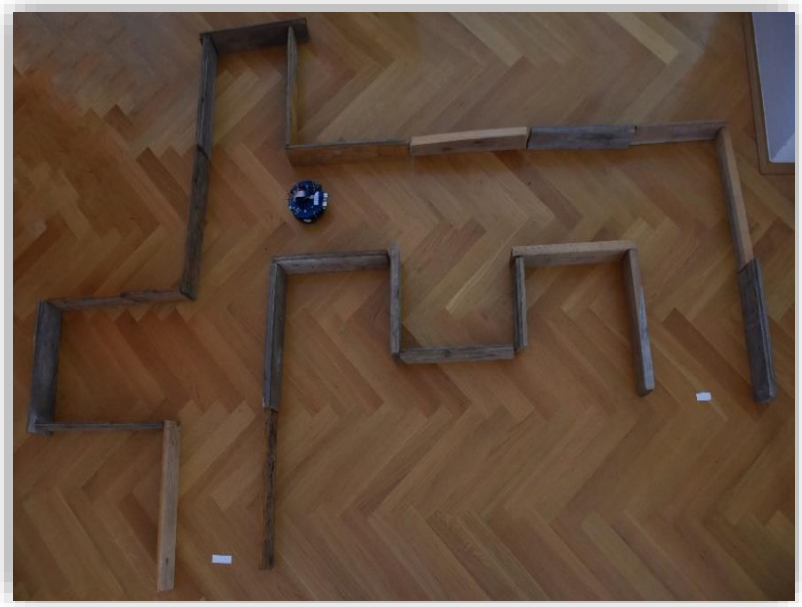
6. TESTIRANJE

Naposlijetku slijedi testiranje učinjenog. Provodi se testiranje ranije objašnjenog algoritma za slobodno kretanje kroz labirint, koji je opisan u poglavlju 5.3. , te algoritma mapiranja prostora pomoću stoga koji je opisani ranije u 5.4. poglavlju. Za testiranje programske podrške bilo je potrebno napraviti poligon u obliku labirinta. Labirint je izrađen pomoću drvenih daščica koje su služile kao zid. Testiranje se temelji na uspješnosti prolaska robota od starta labirinta do kraja bez vraćanja na mjesta na kojima je već bio.

6.1. Izrada labirinta

Labirint je izrađen pomoću drvenih daščica. Labirint je rasprostanjen na 3 kvadratna metra. Test je proveden na 7 različitih labirinata (Sl. 6.1., Sl. 6.2., Sl 6.3., Sl. 6.4., Sl. 6.5., Sl. 6.6., Sl. 6.7.).

Sl. 6.1. Labirint broj 1



Sl. 6.2. Labirint broj 2

Sl. 6.3. Labirint broj 3

Sl. 6.4. Labirint broj 4

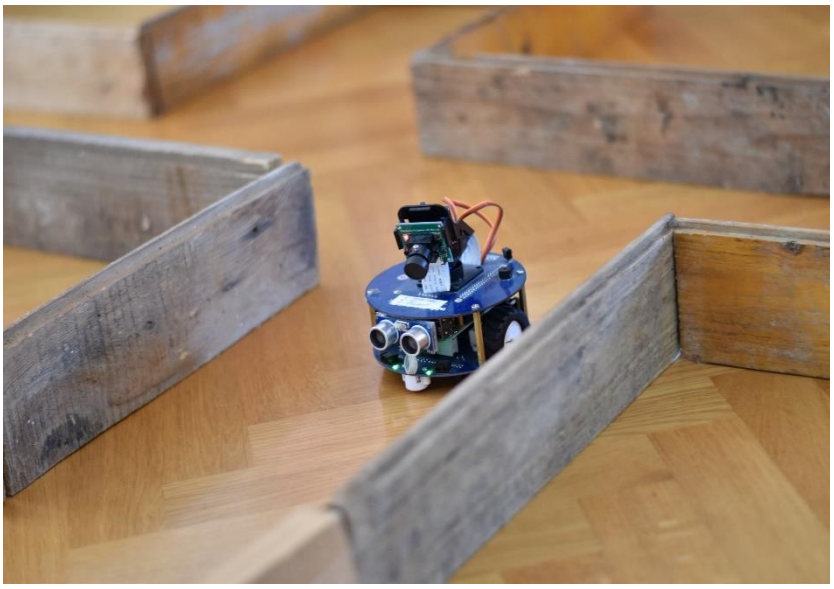
Sl. 6.5. Labirint broj 5

Sl. 6.6. Labirint broj 6

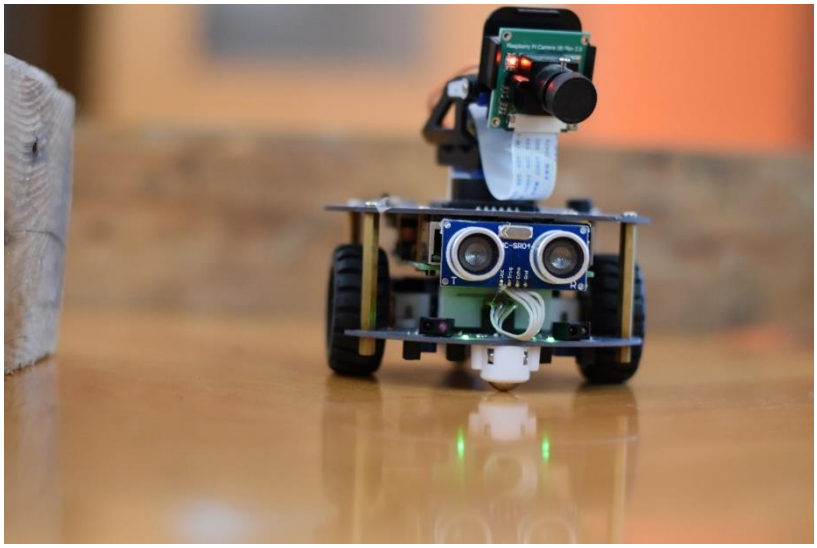
Sl. 6.7. Labirint broj 7

6.2. Obavljanje testa

Test je obavljen na 7 različito složenih labirinata. Kroz svih 7 labirinata robot je prošao uspješno i došao do cilja. Test je obavljan uz pomoć *AlphaBot2* robota koji je opisan u poglavljima ranije. (Sl. 6.8 i Sl. 6.9.) Primjer testiranja labirinta broj 1 opisan je u nastavku.



Sl. 6.8. *AlphaBot2* robot



Sl. 6.9. *AlphaBot2* robot

6.2.1. Prvi prolazak kroz labirint

Prvi prvom prolasku labirinta pokreće se funkcija *MazeSolver*. U nastavku je dan prikaz terminala u kojem se ispisuje sve što se događa prilikom prolaska kroz labirint (Sl.6.10.).

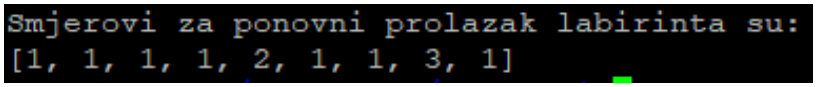
```
20cm naprijed
Distance_Front = 102.65 cm
Distance_Right = 9.12 cm
Vozi ravno
20cm naprijed
Distance_Front = 82.24 cm
Distance_Right = 8.76 cm
Vozi ravno
20cm naprijed
Distance_Front = 62.56 cm
Distance_Right = 8.89 cm
Vozi ravno
20cm naprijed
Distance_Front = 42.11 cm
Distance_Right = 120.25 cm
Nema zida desno, skretanje desno
20cm naprijed
Distance_Front = 99.78 cm
Distance_Right = 9.89 cm
Vozi ravno
20cm naprijed
Distance_Front = 80.21 cm
Distance_Right = 23.7 cm
Nema zida desno, skretanje desno
20cm naprijed
Distance_Front = 12.51 cm
Distance_Right = 9.21 cm
Distance_Left = 7.17 cm
Okreni se nazad
20cm naprijed
Distance_Front = 30.12 cm
Distance_Right = 60.90 cm
Nema zida desno, skretanje desno
```

```
20cm naprijed
Distance_Front = 12.51 cm
Distance_Right = 9.21 cm
Distance_Left = 7.17 cm
Okreni se nazad
20cm naprijed
Distance_Front = 30.12 cm
Distance_Right = 60.90 cm
Nema zida desno, skretanje desno
20cm naprijed
Distance_Front = 41.87 cm
Distance_Right = 8.22 cm
Vozi ravno
Distance_Front = 22.25 cm
Distance_Right = 31.52 cm
Nema zida desno, skretanje desno
20cm naprijed
Distance_Front = 18.22 cm
Distance_Right = 9.50 cm
Distance_Left = 7.57 cm
Okreni se nazad
20cm naprijed
Distance_Front = 37.67 cm
Distance_Right = 7.82 cm
Vozi ravno
20cm naprijed
Distance_Front = 14.57 cm
Distance_Right = 8.12 cm
Distance_Left = 101.17 cm
Nema zida lijevo, skretanje lijevo
20cm naprijed
Distance_Front = 82.52 cm
Distance_Right = 8.42 cm
```

```
Vozi ravno
20cm naprijed
Distance_Front = 61.22 cm
Distance_Right = 280.42 cm
Nema zida desno, skretanje desno
20cm naprijed
Distance_Front = 260.22 cm
Izlazak iz labirinta je uspjesan!!
```

Sl. 6.10. Prvi prolazak labirinta – ispis u terminalu

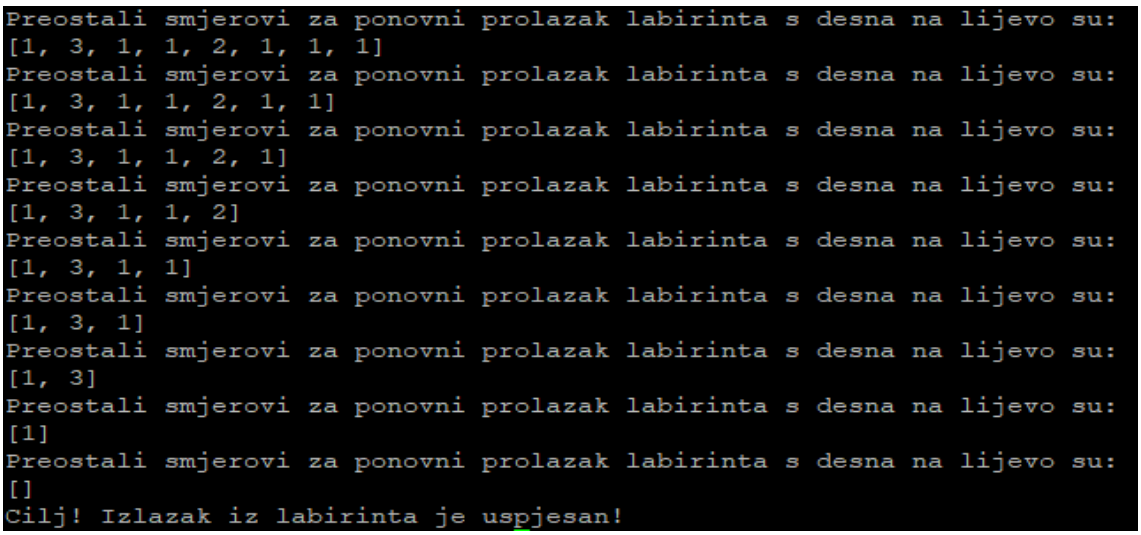
Nakon što je robot prvi puta prošao kroz labirint, putevi za ponovni prolazak su spremljeni na stog. Kao što je objašnjeno u nastavku sve elemente stoga prvog prolaska potrebno je prebaciti u novi stog, kako bi prvi element postao zadnji i da bi se moglo lakše koristiti prilikom prolaska drugi put kroz labirint. Pokreće se funkcija *new_stack*() (Sl. 6.11.).



Slika 6.11. Funkcija new_stack() – ispis u terminalu

6.2.2. Drugi prolazak kroz labirint

Za drugi prolazak labirinta postavljamo robota ponovno na početak labirinta. Pokreće se funkcija *MazeSolverSecondTurn*(). Na ekran se redom ispisuju preostale putanje potrebne za izlazak iz labirinta. Putanje su označene brojevima 1-ravno, 2-desno, 3-lijevo (Slika 6.12.).



Sl. 6.12. Funkcija MazeSolverSecondTurn() – ispis u terminalu

7. ZAKLJUČAK

U priloženom završnom radu obrazložena je tema izrade programske podrške za *Raspberry Pi* mobilnu platformu namijenjenu za kretanje u fizički omeđenom prostoru. Glavni uvjeti za izradu su bili samostalno kretanje robota kroz prostor i usputno mapiranje prostora koje kasnije omogućuje ponovni prolazak kroz labirint direktno do cilja, bez ulaska u petlje ili slijepe ulice. U radu je objašnjen sam *AlphaBot2* robot koji se služio za testiranje algoritma vezanog za temu. Objašnjena je njegova unutrašnjost i dijelovi kao što su ultrazvučni senzori koji su najvažniji dio za uspješan rad algoritma. Također objašnjen je *Raspberry Pi* operacijski sustav i njegove značajke. U radu su navedena već postojeća rješenja koja su imala utjecaj na izradu mog algoritma. Već postojeća rješenja koriste senzore sa svake strane robota, dok je algoritam objašnjen u radu vezan posebno za robote koji imaju samo prednje senzore, kao što je naš. U radu su dalje detaljno objašnjeni algoritmi za samostalno kretanje kroz labirint te algoritam za mapiranje prostora. Na samom kraju obavljeno je testiranje koje je prikazano u vidu fotografija i pokazuje ispise rada svih funkcija u terminalu prilikom izvršavanja algoritma.

LITERATURA

[1] Jamis Buck, *Mazes for Programmers*, 2011., str. 212. [1.9.2021.]

[2] Martin Montes Rivera , *Maze Solver Algorithm for a NAO Humanoid Robot* , 2015., str 18., [25.7.2021.]

[3] *Flood-fill algoritam*, https://research.ijcaonline.org/volume56/number5/pxc3882882.pdf, [20.8.2021.]

[4] *AlphaBot2*, https://www.mouser.com/pdfdocs/Alphabot2-user-manual-en.pdf [25.7.2021.]

[5] *Raspberry Pi*, https://www.raspberrypi.org/help/what-%20is-a-raspberry-pi/ [25.7.2021.]

[6] Carol Fairchild,Thomas Harman , *ROS Robotics By Example*, 2016., [1.9.2021.]

[7] *Raspberry Pi*, https://opensource.com/resources/raspberry-pi [25.7.2021.]

[8] Carol Fairchild,Thomas Harman , *ROS Robotics By Example*, 2016., [1.9.2021.]

[9] Lutz Mark, *Learning Python, 5th Edition*, 2001., str. 112-140, [25.7.2021.]

[10] *Stuart J. Russell and Peter Norvig, „Artificial Intelligence”,* [25.7.2021.]

[11] Jefferies, Margaret E., Yeap, *Robotics and Cognitive Aproaches to Spatial Mapping*, 2008., str. 34-80, [25.7.2021.]

[12] *Maze Solving*, https://en.m.wikipedia.org/wiki/Maze-solving_algorithm [16.8.2021.]

Izvori slika:

[1] *Wall follower* algoritam, https://en.m.wikipedia.org/wiki/Maze-solving_algorithm [16.8.2021.]

[2] *Tremaux* algoritam, https://en.m.wikipedia.org/wiki/Maze-solving_algorithm [16.8.2021.]

[3] *Flood*-fill algoritam, https://research.ijcaonline.org/volume56/number5/pxc3882882.pdf

[20.8.2021.]

[4] *AlphaBot2* presjek, https://www.waveshare.com/wiki/AlphaBot2-Pi [25.7.2021.]

[5] Kućište *AlphaBot2-Pi*, https://www.waveshare.com/wiki/AlphaBot2-Pi [25.7.2021.]

[6] Adapterska ploča *AlphaBot2-Pi*, https://www.waveshare.com/wiki/AlphaBot2-Pi [25.7.2021.]

[7] *Raspberry Pi* 3, https://www.raspberrypi.org/products/raspberry-pi-3-model-b/ [25.7.2021.]

SAŽETAK

Cilj ovog završnog rada je istražiti i ući u svijet robotike. Tema je izraditi programsku podršku za *Raspberry Pi* mobilnu platformu namijenjenu za kretanje u fizički omeđenom prostoru. Za izradu se koriste svojstva *AlphaBot2* robota koji sadrži *Raspberry Pi* operacijski sustav. Najvažniji korišteni elementi robota koji služe za samostalno kretanje u prostoru su ultrazvučni senzori koji omogućuju očitavanje prepreka. Za izradu i pisanje algoritama slobodnog kretanja kroz prostor i mapiranja prostora koristi se programski jezik *Python*.

Ključne riječi: *AlphaBot2*, *Raspberry Pi*, labirint, mapiranje prostora, *maze solver*, *python*

ABSTRACT

The aim of this final work is to explore and enter the world of robotics. The theme is to develop software for the Raspberry Pi mobile platform designed to move in physically confined space. The properties of the AlphaBot2 robot containing the Raspberry Pi operating system are used for the creation. The most important used elements of the robot that are used for independent movement in space are ultrasonic sensors which enable the reading of the obstacles. The Python programming language is used to create and write an independent space moving algorithms and space mapping.

Signal words: AlphaBot2, Raspberry Pi, maze, space mapping, maze solver, python

ŽIVOTOPIS

Filip Pitlović, rođen 14. svibnja 1999. godine u Slavonskom Brodu. Pohađao je osnovnu školu Đuro Pilar u Brodskom Vinogorju. Srednjoškolsko obrazovanje nastavio je u Općoj Gimnaziji u Slavonskom Brodu. Nakon završetka srednjoškolskog školovanja, 2018. godine upisuje Fakultet elektrotehnike, računarstva i informacijskih tehnologija u Osijeku kojeg trenutno pohađa.