Izrada inteligentnog agenta na mobilnoj robotskoj platform

Čordašić, Ivan

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

Preddiplomski sveučilišni studij Računarstvo

IZRADA INTELIGENTNOG AGENTA NA MOBILNOJ ROBOTSKOJ PLATFORMI

Završni rad

Ivan Čordašić

Osijek, 2021.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac Z1P - Obrazac za ocjenu završnog rada na preddiplomskom sveučilišnom studiju

Osijek, 09.09.2021.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada na preddiplomskom sveučilišnom studiju

Ime i prezime studenta:	Ivan Čordašić
Studij, smjer:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	4501, 23.07.2018.
OIB studenta:	88705497149
Mentor:	Izv. prof. dr. sc. Damir Blažević
Sumentor:	
Sumentor iz tvrtke:	
Naslov završnog rada:	Izrada inteligentnog agenta na mobilnoj robotskoj platform
Znanstvena grana rada:	**Procesno računarstvo (zn. polje računarstvo)**
Predložena ocjena završnog rada:	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 2 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 2 bod/boda Jasnoća pismenog izražavanja: 2 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	09.09.2021.
Datum potvrde ocjene Odbora:	22.09.2021.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 23.09.2021.

Ime i prezime studenta:	Ivan Čordašić
Studij:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	4501, 23.07.2018.
Turnitin podudaranje [%]:	4

Ovom izjavom izjavljujem da je rad pod nazivom**: Izrada inteligentnog agenta na mobilnoj robotskoj platform**

izrađen pod vodstvom mentora Izv. prof. dr. sc. Damir Blažević

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD

Raspberry Pi je računalo malenih dimenzija koje na tržištu košta oko trideset dolara. Razvijeno je kako bi pomoglo mladim i inovativnim ljudima naučiti vještinu programiranja te ima mnogo primjena, od znanosti pa čak sve do glazbene industrije. Pokreće se na Linuxu, besplatnom operativnom sustavu kojega je razvio Linus Torvalds u prošlome stoljeću. Razlozi zbog kojega je korišten u većoj mjeri su malena veličina, prenosivost, niska cijena, programabilnost te laka povezivost sa ostalom periferijom poput tipkovnice, zaslona i priključcima poput USB i HDMI. Postoje brojne verzije Raspberry Pi uređaja i svi su kompaktibilni. S obzirom da posjeduje vlastiti operativni sustav, programiranje se može vršiti u bilo kojem programskom jeziku uz uvjet da se isti može prevesti i pokrenuti na Linux OS-u. Unatoč tomu, najčešće se koristi Python programski jezik zbog lakoće pisanja koda i široke primjene.

U posljednje vrijeme, sve više i više se razvija umjetna inteligencija kako bi poboljšala i olakšala svakodnevni život. Upravo u tome svoje mjesto je našao Raspberry Pi, koji je veoma moćan alat kada pričamo u terminima umjetne inteligencije, robotike te strojnog učenja. Njegova brza sposobnost procesuiranja informacija je odličan izbor za početnike koji žele naučiti nešto više o robotici. Također, i na Internetu postoji velika zajednica programera koji su spremni podijeliti svoje znanje.

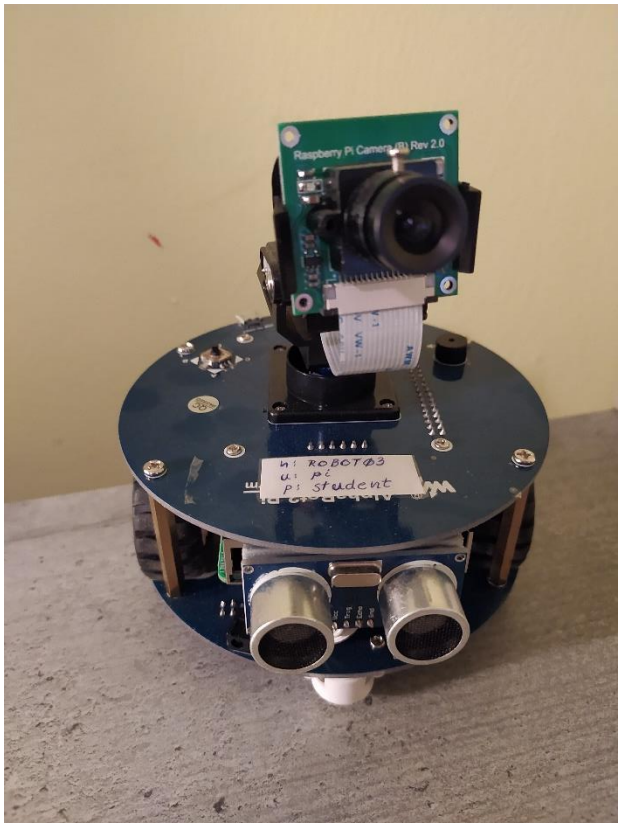
Jedan od primjera spoja Raspberry Pi sa robotikom jesu inteligentni agenti, računalni softveri koji samostalno bez pomoći korisnika obavljaju nekakav zadatak. Spomenuti agent je u stalnoj komunikaciji sa okolinom kako bi što točnije obavio zadatak. Riječ inteligentni u nazivu inteligentni agent predstavlja sposobnost procesuiranja informacija i generiranja novog znanja. Agenti djeluju u ime korisnika da obrade informaciju, imaju interakciju s ostalim mogućim agentima u okolini te tako riješe postavljeni cilj.

1.1 ZADATAK ZAVRŠNOG RADA

U ovom završnom radu definirati ćemo inteligentnog agenta te ga implementirati na mobilnoj robotskoj platformi. Zadatak agenta biti će rješavanje labirinta koji je postavljenim crnom linijom na bijeloj podlozi koristeći nekoliko različitih proizvoljno odabranih algoritama čiju ćemo efikasnost naknadno usporediti. Koristiti ćemo AlphaBot2-Pi platformu sa pripadajućim ugrađenim senzorima.

2. ALPHABOT2-PI PLATFORMA

Temelj korištene robotske platforme je Raspberry Pi 3 model B računalo koje je smješteno unutar zaštićenog kućišta kako bi ostalo otporne na negativne vanjske utjecaje. Na robot su spojeni motori te kotači kako bi se ono moglo kretati, a u svrhu realizacije autonomnog kretanja na robota su spojeni i senzori od kojih najvažniju ulogu imaju infracrveni te ultrazvučni senzore, a od velike pomoći je i kamera smještena na vrhu robota. Ultrazvučni senzor se koristi za određivanje udaljenosti od predmeta, npr. zida u labirintu, ali ga mi nećemo koristiti jer će naš labirint biti predstavljen crnim linijama na bijeloj podlozi. Koristit ćemo infracrveni senzor kako bismo pratili liniju na podlozi.



Slika 2.1 Mobilna robotska platforma

2.1 Infracrveni senzor

Princip rada moda za praćenje linija sličan je modu za izbjegavanje prepreka, samo što se koriste drugačiji senzori. Mod za praćenje linije koristi ITR2001/T infracrveni refleksivni senzor, čiji odašiljač prenosi infracrveno svjetlo tijekom cijelog vremena kretanja. Senzor ima sposobnost očitavanja brojnih boja za što ga je potrebno dodatno modificirati. Kada se infracrveno svjetlo odbije natrag zbog učitane crne linije na bijeloj podlozi, prima ga infracrveni prijemnik te će se robot kretati dokle god prima ta očitanja. Robot posjeduje infracrveni senzor sa pet kanala. Provjerom izlaza tih kanala, robot procjenjuje položaj crne linije kako bi kontrolirao svoje djelovanje.

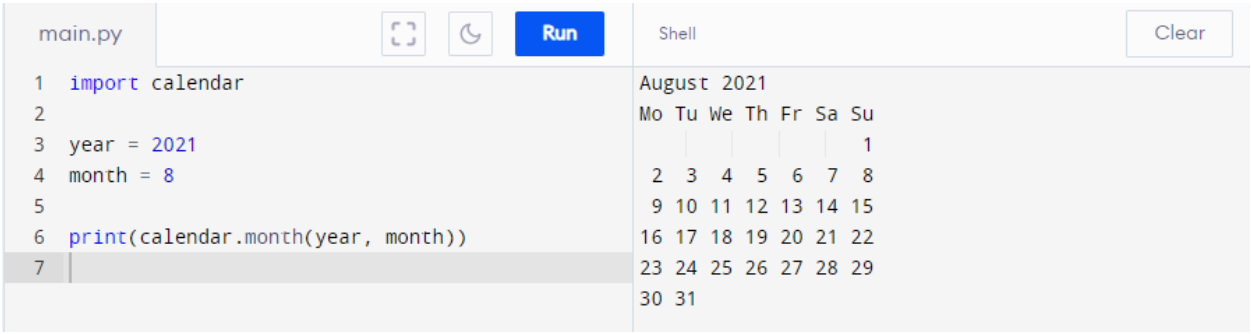
Slika 2.2 5-kanalni senzor za praćenje linije (izvor: https://es.aliexpress.com/item/32772715819.html)

3. PYTHON PROGRAMSKI JEZIK

Python je najbrže rastući i najpopularniji programski jezik na svijetu, ne samo među programerima, nego i među matematičarima, statističarima, znanstvenicima te računovođama. Guido van Rossum je čovjek koji stoji iza ovog jezika i osmislio ga je 1991. godine, a ime je dobio prema čuvenoj britanskoj komediji "Monty Python".

Odlikuje ga interaktivnost te objektno orijentirani pristup visoke razine. Upravo ta visoka razina nam olakšava zadatak jer se ne moramo brinuti o kompleksnim zadacima kao što je zauzeće memorije. Također, Python aplikacije možemo pokretati na Windows-u, Mac-u i Linux-u. Ima ogromnu podršku na internetu gdje gotovo uvijek možemo naći rješenje za naš potencijalni problem. Posjeduje veliki broj biblioteka i alata što nam uvelike štedi vrijeme.

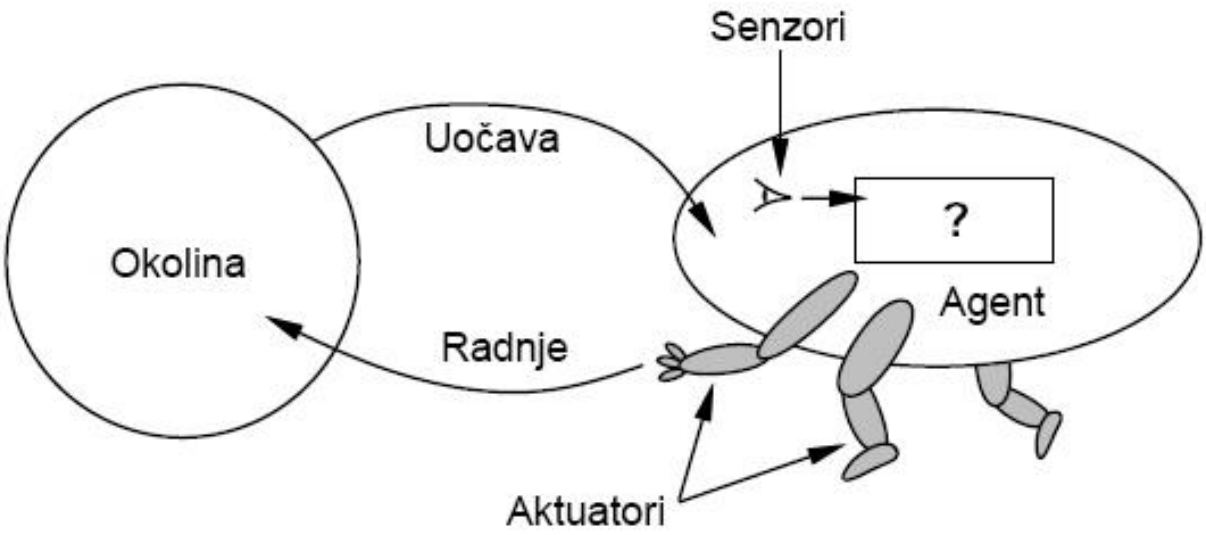
Njegova jednostavna sintaksa omogućava lakšu čitljivost, a velika fleksibilnost primjenu u mnogim granama računarstva, od video igara, matematičke analize podataka i njihove vizualizacije, umjetne inteligencije, automatizacije repetitivnih zadataka, web i mobilnih aplikacija, testiranja pa sve do hakiranja.



Slika 3.1 Primjer Python koda

4. INTELIGENTNI AGENT

U umjetnoj inteligenciji, inteligentni agent predstavlja autonomni entitet koji promatra i zapaža okolinu koristeći senzore te se ponaša kako korisnik zahtijeva koristeći aktuatore, što na našoj mobilnoj robotskoj platformi predstavlja kotačiće pomoću kojih se robot kreće. Također, inteligentni agenti mogu naučiti nešto novo ili koristiti postojeće znanje kako bi izvršili zadatak. Mogu biti opisani kao računalni softver koji bez tuđe pomoći na fleksibilan način izvršava postavljeni zadatak, a korisnika obaviještava o ishodu.

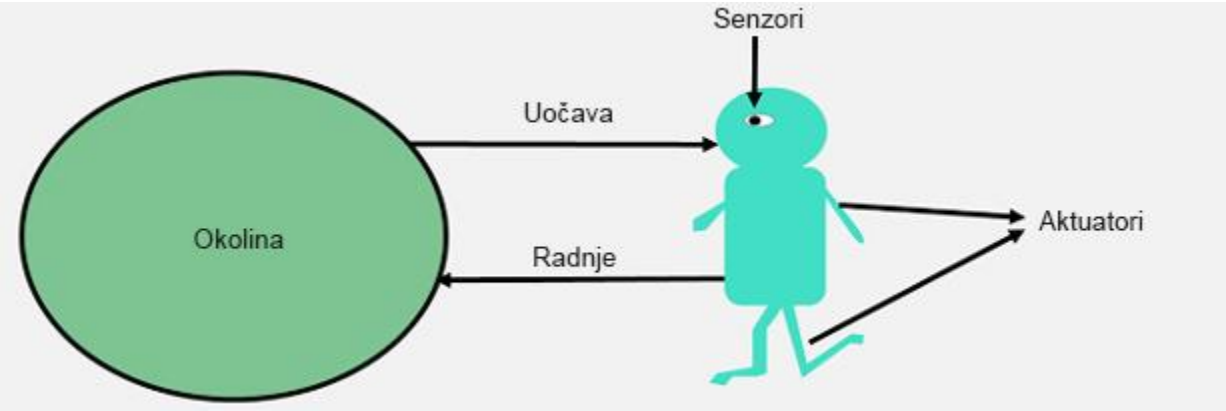


Slika 4.1 Princip rada inteligentnog agenta

U osnovi, inteligentni agent je spoj arhitekture i programa. Ima senzore preko kojih prima ulazna očitanja iz okoline u koju je postavljen. Glavni dio agenta je program koji određuje njegovo ponašanje, taj dio korisnik piše i uvjetuje. On upravlja senzorima i

obrađuje njihove informacije. Naposljetku se pomoću aktuatora izvršava postavljeni zadatak u okolini.

Učenje agenta može biti ugrađeno znanje (postojeće), za vrijeme izvođenja akcija ili iz iskustva koje je stekao. Agent koristeći senzore ima uvid u stanje okoliša u bilo kojem trenutku te isti ti senzori lociraju sve bitno za izvršenje akcije. Bitno da je senzori rade ispravno kako se ne bi narušio zadatak. Radna okolina agenta obuhvaća četiri kriterija racionalnosti: mjera uspješnosti, okolina, efektori te senzori, koji zajedno tvore naziv PEAS (eng. Performance measure, Environment, Actuator, Sensor). Definiranje PEAS-a je prvi korak u dizajnu inteligentnog agenta.



Slika 4.2 Jednostavan prikaz PEAS-a

Četiri su vrste agenata: agent vođen ciljem, agent vođen kvalitetom, refleksivni agent te model-refleksivni agent. Agent vođen ciljem često ne posjeduje dovoljno informacija za donođenje ispravne odluke stoga mu je potreban cilj kojega će pomno planirati. Odlikuje ga manja efikasnost. Agent vođen kvalitetom se temelji na razini zadovoljstva koja nam govore kako pojedina stanja agenta priželjkujemo više od drugih. Može se opisati funkcijom zadovoljstva koja željenim stanjima pridružuje broj koji opisuje razinu zadovoljstva. Refleksivni agent je najjednostavniji agent čija se akcija temelji na

trenutnom zapažanju i on je smanjene inteligencije. Često mogu upasti u beskonačne petlje prilikom izvršenja zadatka te mu okolina mora biti vidiljiva. Model-refleksivni agent je ažurirana verzija refleksivnog agenta koji se bavi slučajevima kada okolina nije potpuno vidljiva te agent treba pamtiti stanja kako bi mogao nastaviti sa akcijom.

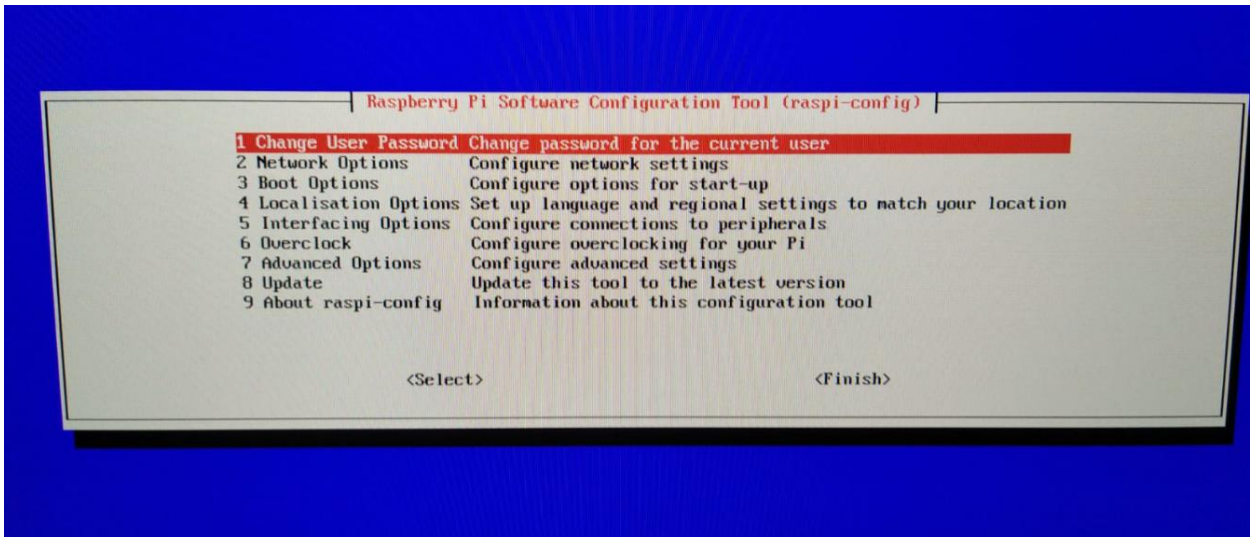
5. PROGRAMSKO RJEŠENJE

Za programsku implentaciju koristiti ću Python. Prvo ću razviti kod za pokretanje motora, zatim za praćenje linije te naposljetku implementirati algoritme za rješavanja labirinta te sve to povezati sa inteligentnim agentom. Tijekom dizajniranja agenta prvo je ptorebno definirati i opisati radnu okolinu (PEAS). Mjera uspješnosti (eng. **P**erformance Measure) predstavlja jedinicu uspješnosti rješavanja zadatka, odnosno je li agent uspio ili nije. Okoliš (eng. **E**nvironment) predstavlja okolinu oko agenta, mjesto gdje on obavlja zadatak. U našem slučaju okoliš je predstavljen sa crnim linijama na bijeloj podlozi. Aktuatori (eng. **A**ctuator) je sredstvo kretanja našega agenta, odnosno kotači. Senzor (eng. Sensor) kojega ćemo korsititi je infracrveni senzor ugrađen na mobilnu robotsku platformu pomoću kojega ćemo pratiti linije u labirinut. Naknadno, potrebno se povezati na Raspbian.

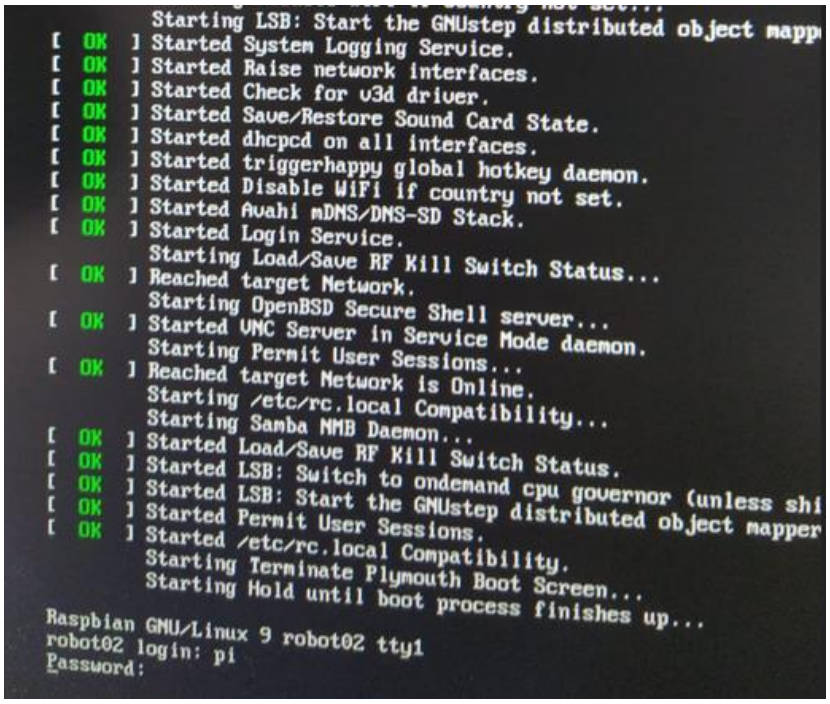
5.1 Povezivanje

Za povezivanje su nam potrebni zaslon (TV sa HDMI ulazom također može poslužiti), tipkovnica, mobilna robotska platforma te HDMI kabel. Spomenutim kablom povežemo robotsku platformu na zaslon, a tipkovnicu spojimo na mobilnu robotsku platformu koristeći USB ulaz. Prvi korak je unos korisničkog imena i lozinke, što je navedeno na samom robotu.

Također je potrebno konfigurirati Raspberry Pi kako bismo ga mogli pravilno koristiti. Nužno je uključiti mu sve senzore, isključiti pristup preko UART-a i spojiti ga na Internet koristeći SSH protocol kako bismo saznali IP adresu mobilne platforme preko koje ćemo se spojiti na Putty. To radimo preko naredbe *hostname -I*. Putty je vrsta terminala koji služi sa serijsku komunikaciju sa uređajem i za prijenos datoteka. Naredbom *startx* ulazimo u grafičko sučelje operacijskog sustava.



Slika 5.1 Konfiguracija operacijskog sustava

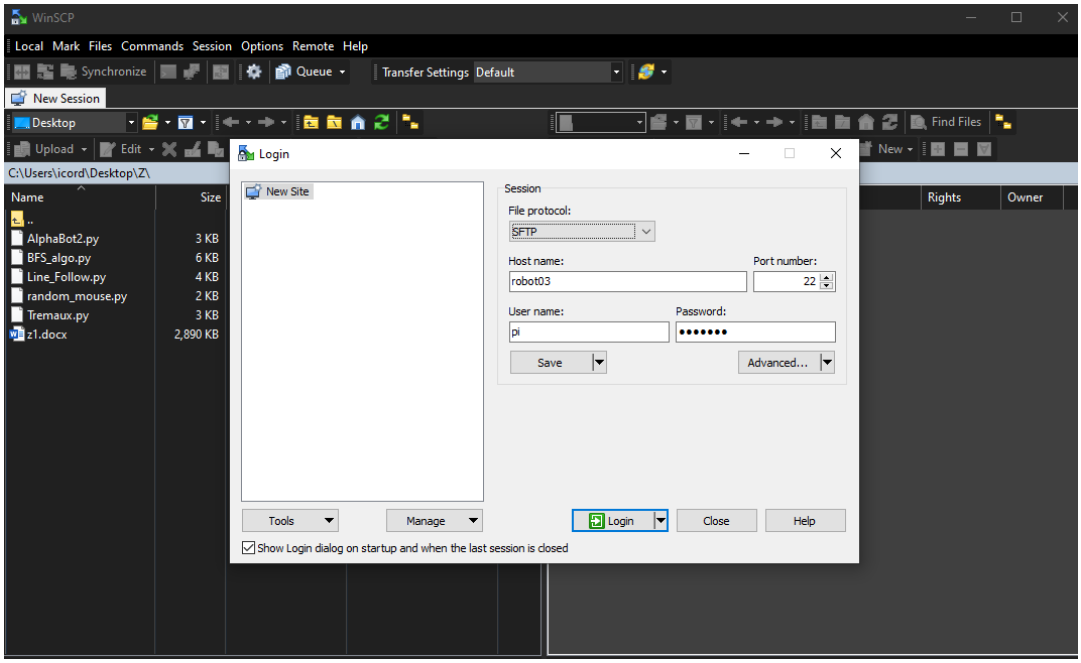


Slika 5.2 Povezivanje na Raspbian operacijski sustav

Prije same implementacije programskog koda, potrebno je instalirati Python bibliotoke za komunikaciju sa senzorima te sveobuhvatnu povezanost mobilne robotske platforme i programskog koda. Najvažnija biblioteka je ona WiringPI. Napisana u C++ programskom jeziku, omogućuje komunikacija sa pinovima, njihovo postavljanje vrijednosti te upotrebu senzora.

5.2 Pokretanje motora

Za pokretanje motora moramo definirati pinove preko kojih su kotači povezani na Raspberry Pi računalo te smjer vrtnje. Koristiti ćemo WinSCP, softver koji služi sa sigurno slanje datoteka na udaljeni uređaj koristeći SFTP protokol.



Slika 5.3 Povezivanje mobilne robotske platforme sa WinSCP

Napisanu skriptu sa kodom spremamo kao Python datoteku te ju potom prebacujemo na operacijski sustav robota koji pokreće skriptu. GPIO (eng. General-

purpose input/output) je standardno sučelje koje se koristi za povezivanje mikrokontrolera s drugim elektroničkim uređajima. Koristiti ćemo *Rpi.GPIO* biblioteku koja sadrži veliki broj klasa i modula koja služi za upravljanje GPIO sučeljem na Raspberry Pi-u, odnosno definiramo koji su pinovi ulaz, a koji izlaz. Također, potrebno je definirati i pinove koji djeluju na motor.

Metodom __init__ postavljamo i definiramo pinove koristeći brojčane vrijednosti, a nakon toga svim pinovima GPIO vrijednosti postavimo na OUT. Brojevi pinova se iščitavaju standardno, kako je određeno za Raspberry Pi uređaje. Koristimo PWM (eng. Pulse Width Modulation) metodu kojom upravljamo razinama izlazeće energije na kotače, odnosno, kontroliramo brzinu motora.

AlphaBot2.py | Line_Follow.py

```
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()
```

Slika 5.4 Osposobljavanja rada motora definiranjem pinova

Sljedeće na redu su metode za smjerova kretanja motora, pokretanje i zaustavljanje. Temelje se na pozivanju Bool varijable na izlaz motora, ovisno o željenom smjeru. Koristeći PWM postavljamo željenu brzinu kretanja, kojom kasnije možemo manipulirati, ovisno kreće li se robot brže ili sporije od očekivanog.

```
30      def forward(self):
31          self.PWMA.ChangeDutyCycle(self.PA)
32          self.PWMB.ChangeDutyCycle(self.PB)
33          GPIO.output(self.AIN1,GPIO.LOW)
34          GPIO.output(self.AIN2,GPIO.HIGH)
35          GPIO.output(self.BIN1,GPIO.LOW)
36          GPIO.output(self.BIN2,GPIO.HIGH)
37
38
39      def stop(self):
40          self.PWMA.ChangeDutyCycle(0)
41          self.PWMB.ChangeDutyCycle(0)
42          GPIO.output(self.AIN1,GPIO.LOW)
43          GPIO.output(self.AIN2,GPIO.LOW)
44          GPIO.output(self.BIN1,GPIO.LOW)
45          GPIO.output(self.BIN2,GPIO.LOW)
46
47      def backward(self):
48          self.PWMA.ChangeDutyCycle(self.PA)
49          self.PWMB.ChangeDutyCycle(self.PB)
50          GPIO.output(self.AIN1,GPIO.HIGH)
51          GPIO.output(self.AIN2,GPIO.LOW)
52          GPIO.output(self.BIN1,GPIO.HIGH)
53          GPIO.output(self.BIN2,GPIO.LOW)
54
55
56      def left(self):
57          self.PWMA.ChangeDutyCycle(30)
58          self.PWMB.ChangeDutyCycle(30)
59          GPIO.output(self.AIN1,GPIO.HIGH)
60          GPIO.output(self.AIN2,GPIO.LOW)
61          GPIO.output(self.BIN1,GPIO.LOW)
62          GPIO.output(self.BIN2,GPIO.HIGH)
63
64
65      def right(self):
66          self.PWMA.ChangeDutyCycle(30)
67          self.PWMB.ChangeDutyCycle(30)
68          GPIO.output(self.AIN1,GPIO.LOW)
69          GPIO.output(self.AIN2,GPIO.HIGH)
70          GPIO.output(self.BIN1,GPIO.HIGH)
71          GPIO.output(self.BIN2,GPIO.LOW)
```

Slika 5.5 Funkcije koje upravljaju radom motora

5.3 Praćenje linije

Ovaj programski kod zapravo služi za implementaciju infracrvenog senzora ITR2001/T sa platforme koji nam može poslužiti da prati crnu liniju na bijeloj podlozi. Senzor za praćenje linije ima pet izlaza na čije vrijednosti utječe udaljenost i boja otkrivenog objekta. Objekt sa većom refleksijom ima veću izlaznu vrijednost, a onaj sa nižom refleksijom ima manju izlaznu vrijednost. Kada se infracrveni senzor približi crnoj liniji, izlazna vrijednost biti će manja te je zato veoma lako dobiti udaljenost od crne linije provjerom izlaza. Prije početka rada, neophodno je uključiti TRSensors biblioteku koja ima predefiniran set kontroliranja dobivenih podataka koji su učitani sa senzora upravo u svrhu praćenja linije.

Prvi dio koda biblioteke predstavlja proces normalizacije dobivenih vrijednost sa izlaza, koji je neophodan za smanjenje utjeca iz okoliša i raznoraznih čimbenika. Različiti senzori mogu davati različite razultate za boju i udaljenost, a bitno nam je ustanoviti minimalnu i maksimalnu vrijednost sa senzora te je stoga potrebno normalizirati vrijednosti. Nakon transformacije vrijednosti, izlazne vrijednosti su u rasponu od 0 do 1000. To su krajnje vrijednosti i ne predstavljaju povoljan položaj, što znači da ako učitamo te vrijednosti, naš senzor nije točno na sredini crne crte, čemu težimo. Program će u nekoliko ponavljanja tražiti vrijednosti senzora kako bi dobio odgovarajuću minimalnu i maksimalnu vrijednost.

Zatim dobivamo pet skupova vrijednosti o udaljenosti senzora i crne linije te bismo trebali koristiti prosjek za pretvaranje tih vrijednosti u određivanje središnje linije. Za otkrivanje gdje se točno nalazi sredina crne linije, moramo poštovati uvjet da debljina linije ne smije prelaziti 16 mm, što odgovara razmaku dva senzora.

Naposljetku koristimo PID (eng. Proportional, Integral, Derivative) algoritam kako bi robot nesmetano radio. U najidealnijem slučaju crna linija ostaje u sredini. Varijabla *proportional* rezultat je razlike trenutne i objektivne pozicije, što zapravo predstavlja pogrešku položaja. Ako je broj pozitivan, robot se nalazi desno od crne linije te ako je broj

negativan, robot se nalazi lijevo od crne linije. *Integral* predstavlja sumu svih pogrešaka. Što je broj veći, robot se udaljava od linije. *Derivacija* predstavlja brzinu odziva robota.

```
TR = TRSensor()
Ab = AlphaBot2()
Ab.stop()
print("Line follow Example")
time.sleep(0.5)
for i in range(0,100):
    if(i<25 or i>= 75):
        Ab.right()
        Ab.setPWMA(30)
        Ab.setPWMB(30)
    else:
        Ab.left()
        Ab.setPWMA(30)
        Ab.setPWMB(30)
    TR.calibrate()
Ab.stop()
print(TR.calibratedMin)
print(TR.calibratedMax)
while (GPIO.input(Button) != 0):
    position,Sensors = TR.readLine()
    print(position,Sensors)
    time.sleep(0.05)
Ab.forward()

while True:
    position,Sensors = TR.readLine()
    if(Sensors[0] >900 and Sensors[1] >900 and Sensors[2] >900 and Sensors[3] >900 and Sensors[4] >900):
        Ab.setPWMA(0)
        Ab.setPWMB(0);
    else:
        #"proportional" treba biti 0 kada smo na liniji
        proportional = position - 2000

        #računanje brzine odziva i sume pogrešaka
        derivative = proportional - last_proportional
        integral += proportional

        #zapamti posljednju poziciju
        last_proportional = proportional

        #razlika između dva dijela motora. Ako je broj pozitivan, robot skreće u desno, i obrnuto
        #magnituda broja određuje oštrinu skretanja

        power_difference = proportional/30  + integral/10000 + derivative*2;

        if (power_difference > maximum):
            power_difference = maximum
        if (power_difference < - maximum):
            power_difference = - maximum
        print(position,power_difference)
        if (power_difference < 0):
            Ab.setPWMA(maximum + power_difference)
            Ab.setPWMB(maximum);
        else:
            Ab.setPWMA(maximum);
            Ab.setPWMB(maximum - power_difference)
```

Slika 5.6 Programsko rješenje za praćenje pozicije

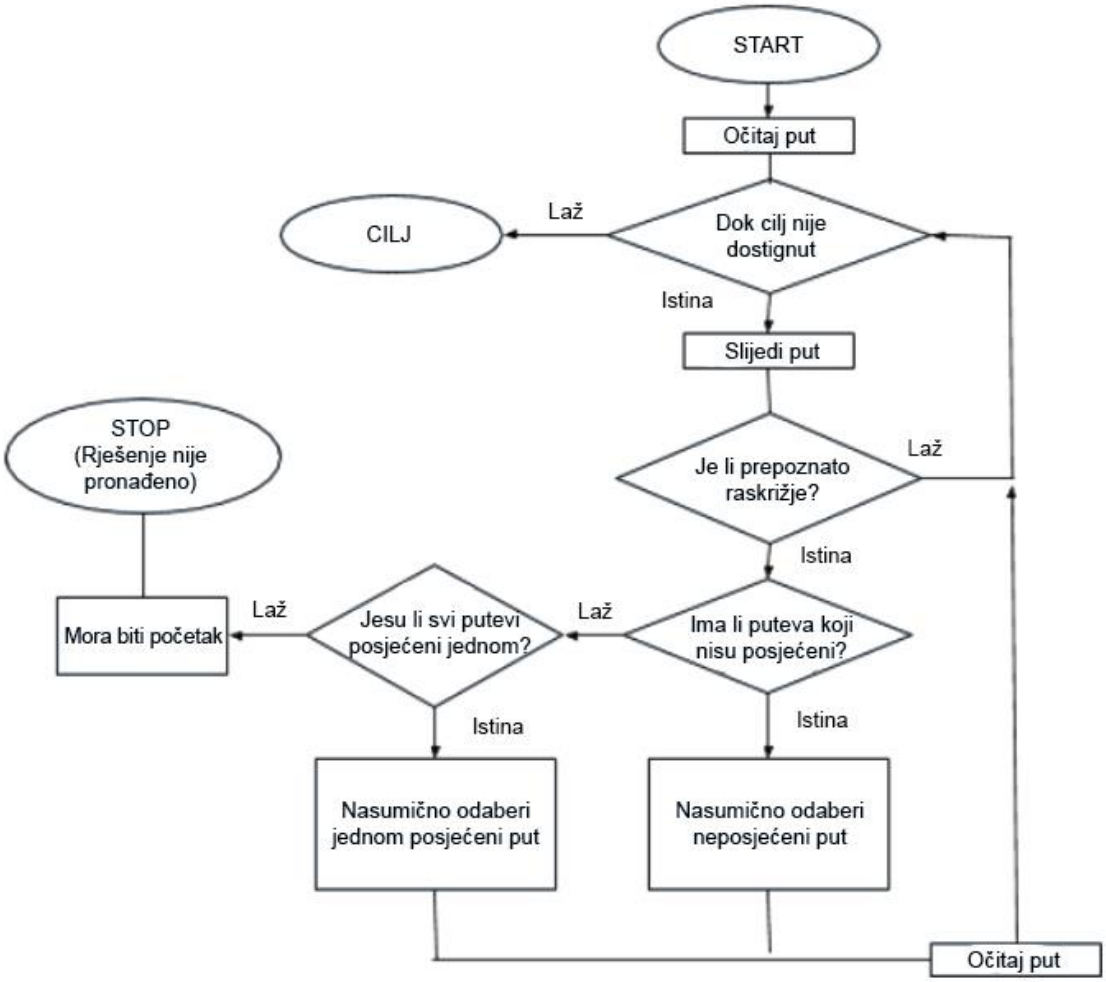
5.4 Trémaux algoritam

Ovaj algoritam kojega je implementirao Charles Pierre Trémaux je veoma učinkovita metoda za rješavanje labirinta te će sigurno raditi za sve labirinte koji su dobro definirani, iako sigurno neće pronaći najkraći put, odnosno neće biti najbrži i najefikasniji.

Najvažnija stavka ovog algoritma je da se put koji se već prošao više ne smije prijeći. Kada agent pronađe slijepi dio labirinta koji je već označen kao posjećen, mora se okrenuti i vratiti natrag. Također, kada se pronađe novo "raskrižje", smjer agenta u kojem nastavlja tražiti izlaz iz labirinta odabran je nasumično. Trémaux algoritam može razlikovati tri različita slučaja: prazan, što znači da taj dio labirinta još nije istražen, posjećen jednom, što znači da je taj dio labirinta već posjećen, te posjećen dva puta, što ukazuje da je agent išao tim putem i bio je prisiljen vratiti se, odnosno išao je pogrešnim putem.

Koristeći ovaj algoritam, agent može imati dva ishoda. Prvi je kada se pronađe rješenje, odnosno izlaz iz labirinta, dok drugi ishod opisuje slučaj u kojem je nemoguće pronaći izlaz. To se događa kad se agent vrati na mjesto odakle je krenuo, odnosno početnu točku.

Spomenuti algoritam ima svoje prednosti i nedostatke. Logika algoritma jamči izlaz za bilo koji labirint, pa čak i one kompleksnije. Agent pamti izlaz iz labirinta, odnosno evidentira koje je dijelove kojim redoslijedom prošao i ukoliko ne uspije naći izlaz u prvome pokušaju, u jednom od sljedećih će sigurno uspjeti izbjegavajući slijepe dijelove labirinta.



Slika 5.7 Dijagram toka Trémaux algoritma

```
class Tremaux(MazeSolveAlgo):
    def __init__(self):
        self.visited_cells = {}

    def _solve(self):
        self.visited_cells = {}
        solution = []

        current = self.start
        solution.append(current)
        self._visit(current)

        if self._on_edge(self.start):
            current = self._push_edge(self.start)
            solution.append(current)
            self._visit(current)

        while not self._within_one(solution[-1], self.end):
            ns = self._find_unblocked_neighbors(solution[-1])
            nxt = self._what_next(ns, solution)
            solution.append(self._midpoint(solution[-1], nxt))
            solution.append(nxt)
            self._visit(nxt)
        return [solution]

    def _visit(self, cell):
        if cell not in self.visited_cells:
            self.visited_cells[cell] = 0

        self.visited_cells[cell] += 1

    def _get_visit_count(self, cell):
        if cell not in self.visited_cells:
            return 0
        else:
            return self.visited_cells[cell] if self.visited_cells[cell] < 3 else 2

    def _what_next(self, ns, solution):
        if len(ns) <= 1:
            return ns[0]
        visit_counts = {}
        for neighbor in ns:
            visit_count = self._get_visit_count(neighbor)
            if visit_count not in visit_counts:
                visit_counts[visit_count] = []
            visit_counts[visit_count].append(neighbor)
        if 0 in visit_counts:
            return choice(visit_counts[0])
        elif 1 in visit_counts:
            if len(visit_counts[1]) > 1 and len(solution) > 2 and solution[-3] in visit_counts[1]:
                visit_counts[1].remove(solution[-3])
            return choice(visit_counts[1])
        else:
            if len(visit_counts[2]) > 1 and len(solution) > 2 and solution[-3] in visit_counts[2]:
                visit_counts[2].remove(solution[-3])
            return choice(visit_counts[2])
```

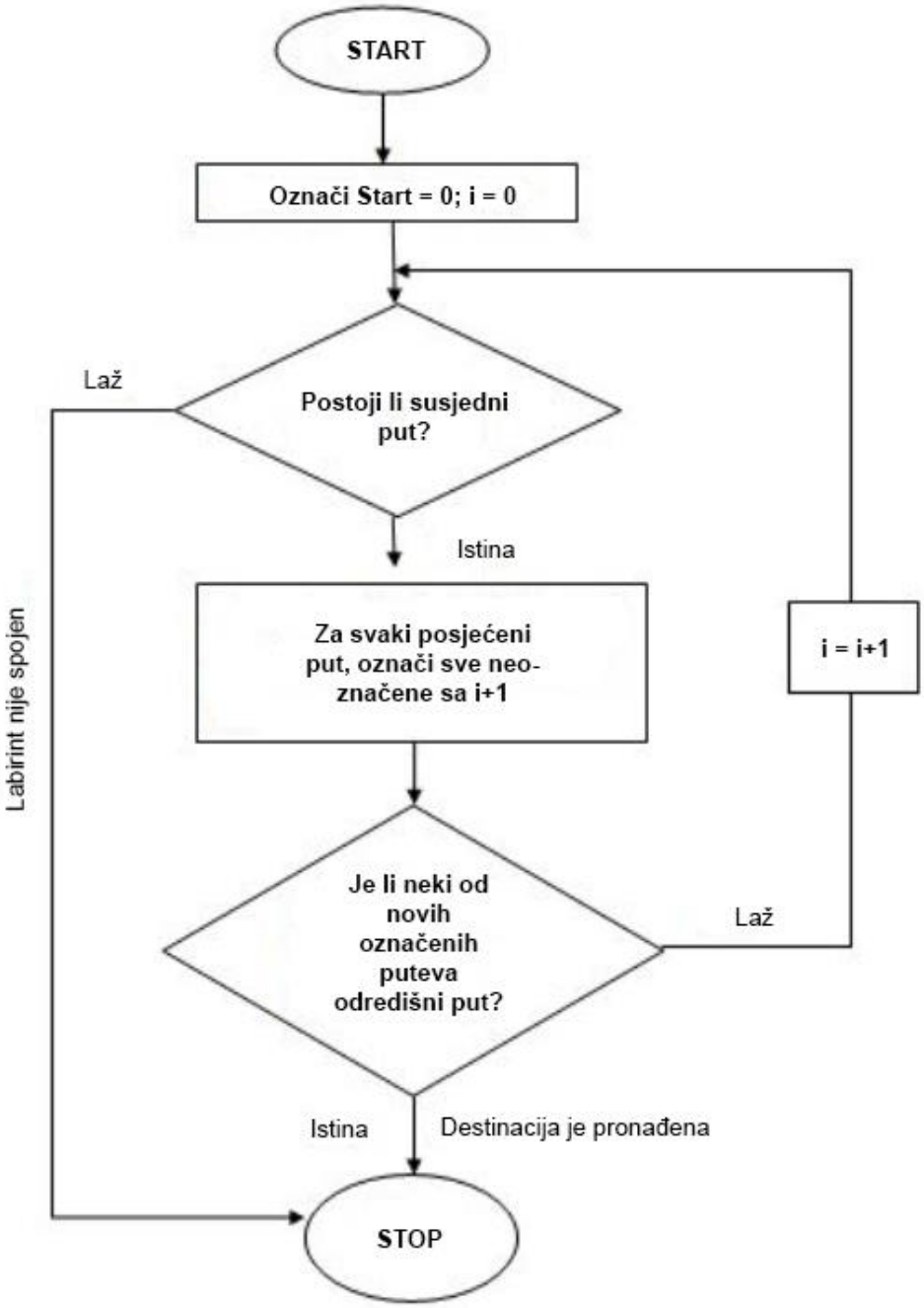
Slika 5.8 Trémaux algoritam

5.5 Algoritam najkraćeg puta

U slučaju kada labirint ima više mogućih rješenje, klijent možda želi uštedjeti na vremenu i pronaći najbrži put između početne i završne točke labirinta. Nekoliko je algoritama za pronalaženje najkraćeg puta te svi potječu iz teorije grafova, a neki od njih su Dijkstrin algoritam, Bellman-Ford algoritam, A* algoritam za pretraživanje i brojni drugi.

Mi ćemo koristiti BFS (eng. Breadth-First Search) algoritam. Ukoliko zamislimo da je labirint jedan veliki graf gdje svako skretanje predstavlja jedan čvor u grafu, ovaj algoritam moguće je primijeniti i na našeg agenta.

Algoritam radi tako da krene od početnog čvora, u ovom slučaju starta labirinta i istražuje najbliže čvorove, odnosno najbliža skretanja, tj. mjesto gdje se labirint grana. Zatim odabere to najbliže grananje te prolazi kroz sve neposjećene čvorove. Algoritam iterativno slijedi postupak za svaki od najbližih čvorova dok ne dođe do kraja labirinta te istražuje sve susjede svih čvorova i osigurava da se svaki čvor posjeti točno jedan put.



Slika 5.9 Dijagram toka BFS algoritma

Tremaux.py | BFS_algo.py

```
class ShortestPaths(MazeSolveAlgo):
    def _solve(self):
        self.start_edge = self._on_edge(self.start)
        self.end_edge = self._on_edge(self.end)

        start = self.start
        if self.start_edge:
            start = self._push_edge(self.start)
        start_posis = self._find_unblocked_neighbors(start)
        solutions = []
        for sp in start_posis:
            if self.start_edge:
                solutions.append([start, self._midpoint(start, sp), sp])
            else:
                solutions.append([self._midpoint(start, sp), sp])
        num_unfinished = len(solutions)
        while num_unfinished > 0:
            for s in range(len(solutions)):
                if solutions[s][-1] in solutions[s][:-1]:
                    solutions[s].append(None)
                elif self._within_one(solutions[s][-1], self.end):
                    solutions[s].append(None)
                elif solutions[s][-1] is not None:
                    if len(solutions[s]) > 1:
                        if self._midpoint(solutions[s][-1], solutions[s][-2]) == self.end:
                            solutions[s].append(None)
                            continue
                    ns = self._find_unblocked_neighbors(solutions[s][-1])
                    ns = [n for n in ns if n not in solutions[s][-2:]]

                    if len(ns) == 0:
                        solutions[s].append(None)
                    elif len(ns) == 1:
                        solutions[s].append(self._midpoint(ns[0], solutions[s][-1]))
                        solutions[s].append(ns[0])
                    else:
                        for j in range(1, len(ns)):
                            nxt = [self._midpoint(ns[j], solutions[s][-1]), ns[j]]
                            solutions.append(list(solutions[s]) + nxt)
                        solutions[s].append(self._midpoint(ns[0], solutions[s][-1]))
                        solutions[s].append(ns[0])
            num_unfinished = sum(map(lambda sol: 0 if sol[-1] is None else 1, solutions))
        solutions = self._clean_up(solutions)
        return solutions
```

Slika 5.10 BFS algoritam

5.6 Random mouse algoritam

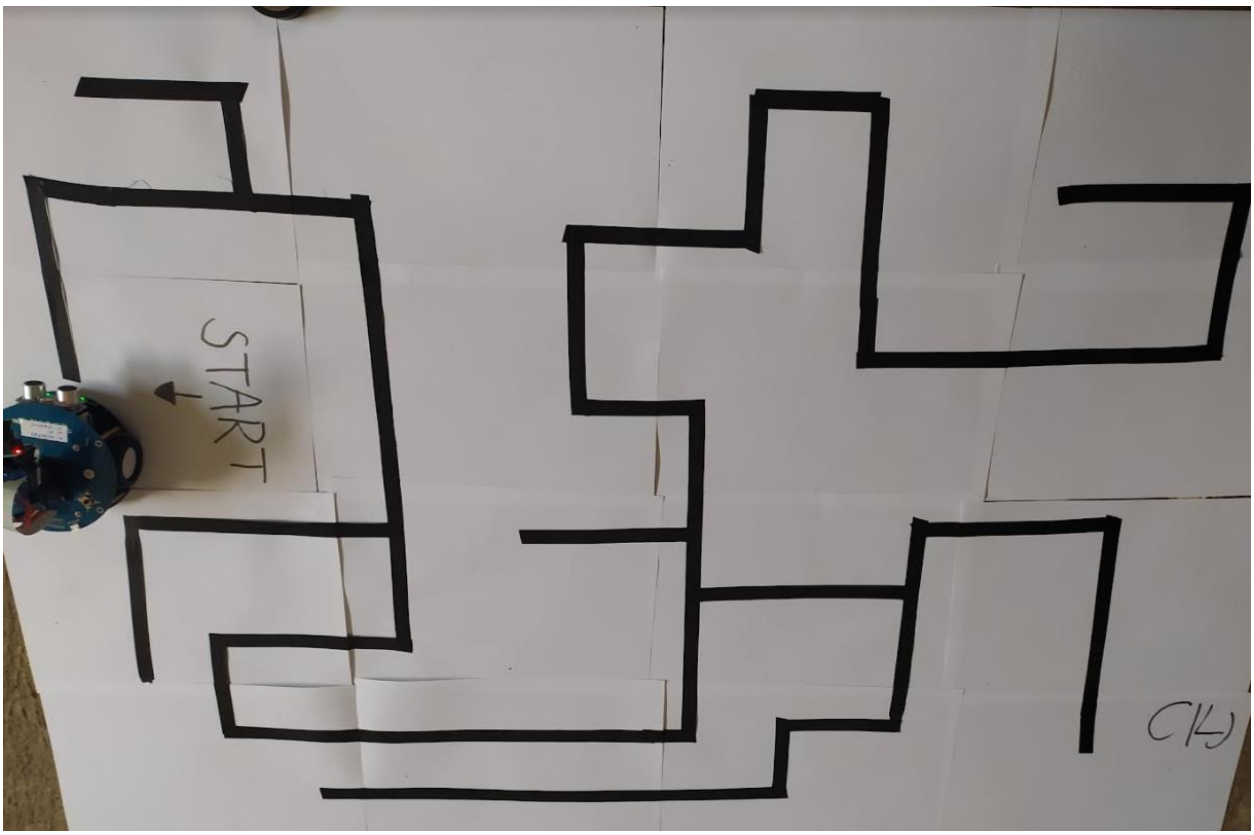
Ovaj algoritam je veoma trivijalan i jednostavan način za rješavanje labirinta. Agent slijedi put do grananja labirinta te potom slučajno odlučuje o sljedećem smjeru. Taj postupak se ponavlja dokle god ne dođe do kraja labirinta. Metoda se u pravilu ne smeta efikasnom jer za zahtjevnije i velike labirinte ne mora uvijek pronaći izlaz, a i ako pronađe, to može dugo trajati.

```
6   class RandomMouse(MazeSolveAlgo):
7
8       def _solve(self):
9           solution = []
10          current = self.start
11          if self._on_edge(self.start):
12              current = self._push_edge(self.start)
13          solution.append(current)
14
15          #odaberi random skretanje i idi do kraja te vrati rješenje
16          while not self._within_one(solution[-1], self.end):
17              ns = self._find_unblocked_neighbors(solution[-1])
18
19              nxt = choice(ns)
20              solution.append(self._midpoint(solution[-1], nxt))
21              solution.append(nxt)
22
23          return [solution]
```

Slika 5.11 Random Mouse algoritam

6. TESTIRANJE

Prije samog početka testiranja rada inteligentnog agenta, moramo napraviti labirint. Labirint će se sastojati od bijele podloge sa crnim linijama te će imati početak i kraj. Ukupne je dužine 201 cm, dok je širina linije 15 mm. Pokrenuvši robota, primjećujemo se agent kreće prebrzo i neće moći reagirati na vrijeme. Problem je riješen smanjivanjem brzine kretanja. Vremensko trajanje svakog algoritma radi usporedbe efikasnosti mjeriti ćemo štopericom.



Slika 6.1 Labirint

Svaki algoritam testirati ćemo pet puta na istome labirintu (Slika 6.1) i mjeriti vrijeme te naposljetku vidjeti koliko su efikasni u stvarnom vremenu.

Prvi algoritam kojega ćemo testirati je Trémaux algoritam. Agent je bio uspješan u svih pet pokušaja. Agent je u prvome pokušaju ušao u samo jedan slijepi dio labirinta, odnosio dio labirinta u kojem se morao okrenuti i vratiti nazad do raskrižja gdje nasumično bira daljni smjer kretnje Nakon toga je nasumičnim odabirom smjera kretnje najkraćim putem uz još jedan ulazak u slijepi dio labirinta pronašao izlaz. Agent je identičan smjer ponovio i kod ostala četiri ponavljanja.

BFS algoritam je sljedeći na redu za testiranje. Algoritam je bio efikasan u svih pet ponavljanja, odnosno agent je uspješno pronašao izlaz iz labirinta. Vremena izvođenja algoritma su identična jer je samo jedan mogući izlaz iz labirinta, odnosno agent pronalazi najkraće rješenje ukoliko postoji više najkraćih rješenja.

Posljednji algoritam kojega ćemo provjeriti je Random Mouse algoritam. Iako je agent svaki put pronašao izlaz iz labirinta, algoritam se pokazao dosta neefikasnim jer mu je trebalo puno vremena. Neučinkovitost algoritma u smislu razlike u vremenu rješavanja možemo objasniti u samoj osnovi algoritma koja se sastoji od nasumičnog kretanja agenta. Agent svaki put nasumično bira gdje će skrenuti, ne pamti koje je dijelove labirinta prošao i nije zajamčeno da će kompleksnije labirinte ikada uspjeti riješiti.

Tablica 6.1 Rezultati testiranja

	prvo ponavljanje	drugo ponavljanje	treće ponavljanje	četvrto ponavljanje	peto ponavljanje
Trémaux	75 s	75 s	75 s	75 s	75 s
Breadth-First Search	65 s	65 s	65 s	65 s	65 s
Random Mouse	120 s	120 s	150 s	150 s	120 s

Iz tablice rezultata testiranja rada inteligentnog agenta, zaključujemo kako je efikasnost sva tri algoritma 100%. Agent je koristeći Trémaux algoritam u prvome pokušaju zalutao u slijepi dio labirinta, označio ga i zapamtio te je nastavio do kraja labirinta odabirući nasumičan smjer kretnje. BFS algoritam se pokazao najefikasnijim jer se ipak radi o algoritmu najkraćeg puta. Agent je istraživajući neposjećene čvorove najbližeg grananja vrlo brzo pronašao izlaz iz labirinta. Random Mouse algoritam se pokazao najdugotrajnijim jer je agent proizvoljno na slučajan način birao skretanja te za veće labirinte ovaj algoritam može trajati dugo.

7. ZAKLJUČAK

U ovome završnom radu upoznali smo se sa Raspberry Pi platformom koja je idealna za razvijanje programskih vještina za početnike. Pruža mogućnost izrade jednostavnih i složenih robotskih radnji. Korištena mobilna robotska platforma, AlphaBot2-Pi uz definiranje svojstava i odgovarajućeg programskog koda pretvorena je u inteligentnog agenta koji rješava dani zadatak.

Prije realizacije inteligentnog agenta, bilo se nužno upoznati sa samom mobilnom robotskom platformom i njezinim senzorima te definirati pinove i implementirati programska rješenje za pokretanje robota, te za praćenje linije, što zapravo predstavlja implementaciju infracrvenog senzora. Spomenuti senzor nam je poslužio za praćenje crne linije na bijeloj podlozi što je zapravo predstavljalo labirint. Naposljetku smo odabrali tri algoritma za rješavanje labirinta koja smo testirali u sigurnom okruženju. Uspoređujući vremensku efikasnost, najbrži algoritam je bio algoritam najkraćeg puta.

LITERATURA

[1] AlphaBot2-Pi, dostupno na: https://www.waveshare.com/wiki/AlphaBot2-Pi [5.7.2021]

[2] AlphaBot2, dostupno na: https://www.open-electronics.org/alphabot2-the-opensource-robot/ [6.7.2021]

[3] AlphaBot korisničke upute, dostupno na: https://www.mouser.com/pdfdocs/Alphabot2-user-manual-en.pdf [5.7.2021]

[4] M. Lutz, Learning Python, 5th Edition, 2013.

[5] S. J. Russell, P. Norvig, Artificial Intelligence, A Modern Approach, 4th US ed, 2020.

[6] Inteligentni agent, dostupno na: https://en.wikipedia.org/wiki/Intelligent_agent [5.7.2021]

[7] GPIO pinovi, dostupno na: https://www.ics.com/blog/control-raspberry-pi-gpio-pins-python [5.7.2021]

[8] WinSCP povezivanje, dostupno na: https://behind-the-scenes.net/using-winscp-to-connect-to-a-raspberry-pi/ [22.8.2021]

[9] Algoritmi za rješavanje labirinta, dostupno na: https://en.wikipedia.org/wiki/Maze-solving_algorithm [29.8.2021]

[10] Algoritmi za rješavanje labirinta, dostupno na: http://www.astrolog.org/labyrnth/algrithm.htm#solve [29.8.2021]

[11] J. Buck, Mazes for Programmers: Code Your Own Twisty Little Passages 1st Edition, 2011.

SAŽETAK

U završnom radu programski je ostvaren inteligentni agent na mobilnoj robotskoj platformi koji, koristeći nekoliko različitih algoritama, riješava labirint predstavljen crnim linijama na bijeloj podlozi. Pri tom je korišten programski jezik Python. Prije početka rada, nužno je bilo osposobiti agenta za pokretanje i programski implementirati kod za praćenje linije. Korišteni algoritmi su jedni od brojnih postojećih načina za rješavanje labirinta. Testiranjem inteligentnog agenta u nekoliko ponavljanja potvrđena je njegova ispravnost, efikasnost i pogodnost za korištenje.

Ključne riječi: agent, algoritam, labirint, Python, robot

ABSTRACT

In the final work, an intelligent agent was programmatically realized on a mobile robotic platform which, using several different algorithms, solves a labyrinth represented by black lines on a white background. The Python programming language was used. Before starting work, it was necessary to train the startup agent and programmatically implement the line following code. The algorithms used are one of many existing ways to solve the maze. Testing of the intelligent agent in several repetitions confirmed its correctness, efficiency and suitability for use.

Keywords: agent, algorithm maze, Python, robot

ŽIVOTOPIS

Ivan Čordašić rođen je 24. rujna 1999. u Vinkovcima. Završio je Osnovnu školu Ivana Gorana Kovačića u rodnome gradu nakon čega upisuje opći smjer u vinkovačkoj gimnaziji koji završava sa odličnim uspjehom. Trenutno pohađa treću godinu preddiplomskog studija računarstva na osječkom FERIT-u te mu je želja nastaviti obrazovanje na diplomskom studiju Automobilsko računarstvo i komunikacije.