Razvoj programske podrške za svladavanje labirinta na Raspbery Pi mobilnoj robotskoj platformi

Budimir, Marko

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

Sveučilišni preddiplomski studij Računarstva

PROGRAMSKE PODRŠKE ZA SVLADANJE LABIRINTA NA RASPBERY PI MOBILNOJ ROBOTSKOJ PLATFORMI

Završni rad

Marko Budimir

Osijek, 2021.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

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

Osijek, 17.09.2021.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada na preddiplomskom sveučilišnom studiju

Ime i prezime studenta:	Marko Budimir
Studij, smjer:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R4181, 23.07.2018.
OIB studenta:	16682771474
Mentor:	Izv. prof. dr. sc. Damir Blažević
Sumentor:	
Sumentor iz tvrtke:	
Naslov završnog rada:	Razvoj programske podrške za svladavanje labirinta na Raspbery Pi mobilnoj robotskoj platformi
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: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	17.09.2021.
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, 26.09.2021.

Ime i prezime studenta:	Marko Budimir
Studij:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R4181, 23.07.2018.
Turnitin podudaranje [%]:	10

Ovom izjavom izjavljujem da je rad pod nazivom**: Razvoj programske podrške za svladavanje labirinta na Raspbery Pi mobilnoj robotskoj platformi**

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

U ovom radu prikazan je jedan od načina rješavanja labirinta pomoću robota. Korišten je Alphabot2 robot koji sadrži Raspberry Pi 3 jednopločno računalo i različite senzore. Labirint je napravljen od trake nalijepljene na podu koju robot prati. Robot prati te linije koristeći reflektirajući infracrveni fotoelektrični senzor. Robot prvo prolazi kroz cijeli labirint, mapirajući ga. Nakon toga računa najkraći put te prolazi tim putem od početka do cilja. Za postizanje tog cilja korišteni su već poznati algoritmi kao što su Trémauxov i Dijkstrin algoritam. Pomoću ovih algoritama i gotovih funkcija robota napisan je program u programskom jeziku Python koji izvršava navedenu funkcionalnost. U prvom poglavlju opisan je tehnologija i algoritmi korišteni za rješavanje labirinta. Opisan je Alphabot2 robot, te korišteni senzori. Zatim su objašnjeni algoritmi za obilazak i pronalazak najkraćeg puta u labirintu. U drugom poglavlju objašnjena je izrada kretnje robota po labirintu, te implementacija navedenih algoritama. U trećem poglavlju su prikazani rezultati, te jedan primjer rješavanja labirinta.

1.1. Zadatak završnog rada

Odabrati i implementirati odgovarajući algoritam pogodan za mapiranje prostora i svladavanje proizvoljnog labirinta za primjenu na Raspberry Pi mobilnoj robotskoj platformi. Zadatak realizirati koristiti postojeće senzore na mobilnoj platformi. Labirint prikazati u obliku linija na podlozi. Rješenje temeljiti na open source tehnologijama.

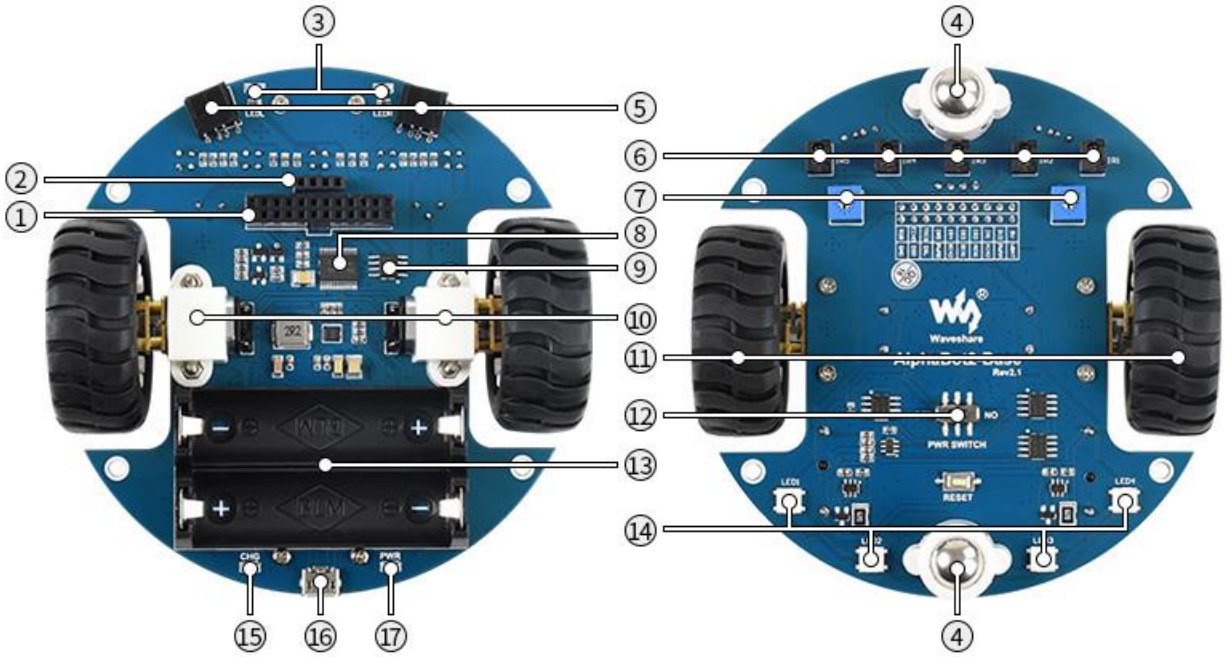
2. ALGORITMI I TEHNOLOGIJE ZA RJEŠAVANJE LABIRINTA

Labirint je struktura sastavljena od staze koja ima jedan početak i jedan kraj koji predstavlja cilj. Rješavanje labirinta je jedan od najpoznatijih problema, zbog čega su tijekom godina razvijeni mnogi algoritmi da bi se to postiglo. Dosta tih algoritama je usko povezano s teorijom grafova. Labirinti koji ne sadrže petlje poznati su kao "jednostavno povezani" ili "savršeni" labirinti i ekvivalentni su stablu u teoriji grafova [1]. Postoje dvije vrste algoritama za rješavanje labirinta. Prva je u slučaju da putnik nema predznanja o labirintu, a ti algoritmi su: metoda slučajnog odabira, algoritam „slijedi zid“, Pledgeov algoritam i Trémauxov algoritam. Druga je kada osoba ili računalni program vidi cijeli labirint odjednom, a za to se koriste: algoritam „slijepih ulica“ i algoritmi najkraćeg puta [2].

2.1. Robot

Alphabot2 je mobilni robot kojeg upravlja Raspberry Pi 3 jednopločno računalo. Raspberry Pi 3 je jedan od modela Raspberry Pi računala napravljen za edukacijske i osobne svrhe. Preporučeni operacijski sustav je Raspberry Pi OS razvijen na distribuciji Linux-a, te Python je jedan od preporučenih jezika za programiranje zbog njegove jednostavnosti [3][4]. Robot ima razne senzore, motore, LE diode i kameru kojima se može upravljati i dobiti povratne informacije. Za kretanje koristi N20 mikro prijenosne motore na koje su spojeni gumeni kotači, a za praćenje linije po nekoj površini koristi se 5 reflektirajućih infracrvenih fotoelektričnih senzora. Na slici 2.1. prikazan je Alphabot2, a na slici 2.2. pod brojem 6 prikazani su 5 senzora za praćenje linija [5].

Slika 2.1. Alphabot2 robot [5]

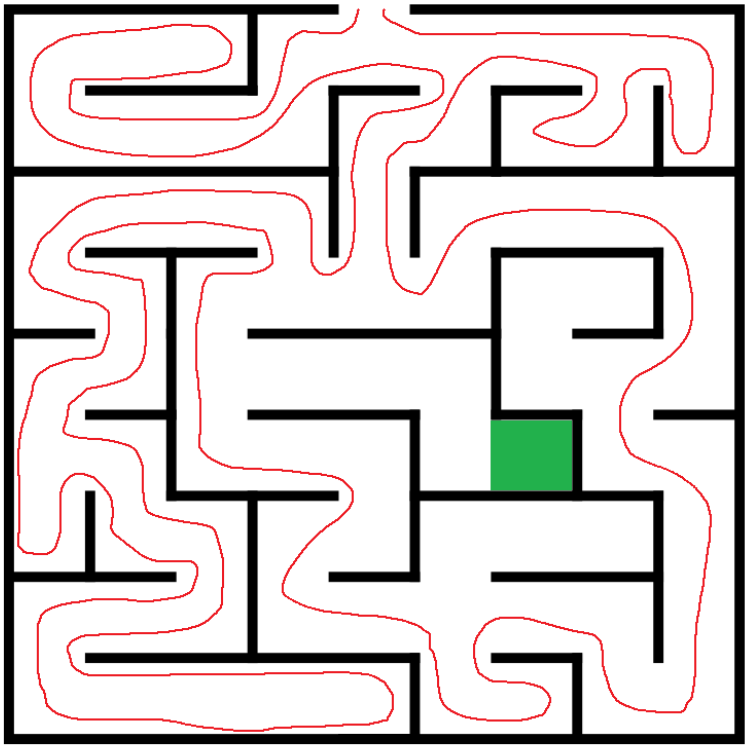


Slika 2.2. Donji dio Alphabot2 robota [5]

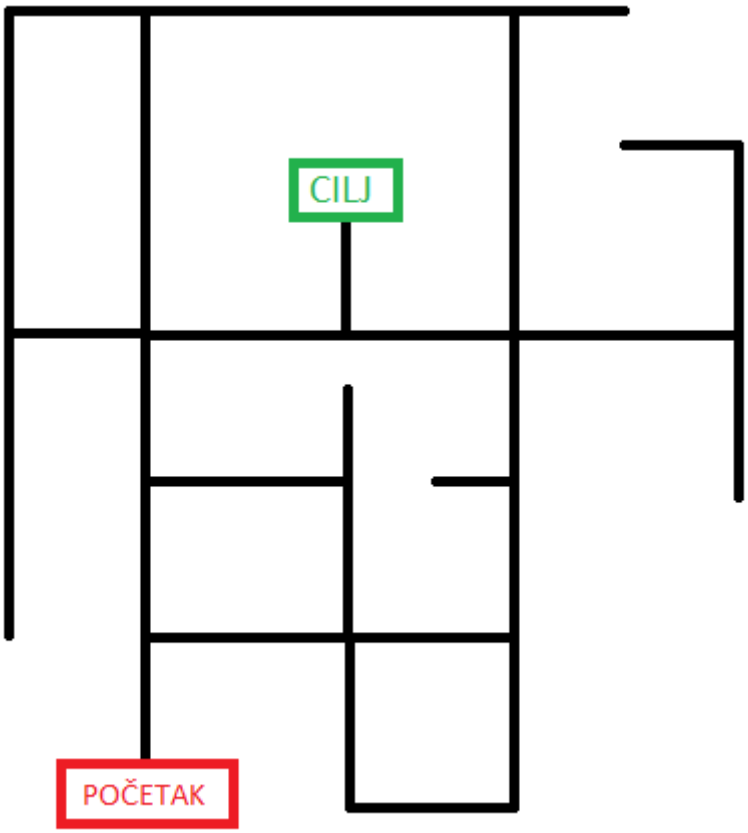
2.2. Prolazak robota kroz labirint

2.2.1. Algoritam „slijedi zid“

Jedan od najpoznatijih načina za rješavanje labirinta je algoritam „slijedi zid“, još zvan pravilo desne ili lijeve ruke. Ukratko, na ulazu u labirint odaberemo lijevi ili desni zid koji ćemo cijelim putem pratiti ili na svakom raskrižju odaberemo istu stranu gdje ćemo skrenuti i na taj način ćemo riješiti većinu tradicionalnih labirinata. Ako se cilj nalazi u sredini labirinta, a njegovi su zidovi izolirani tj. nisu povezani s početkom, tada se takav labirint ne može riješiti ovim algoritmom. Primjer labirinata koji se ne mogu riješiti ovim algoritmom prikazani su na slikama 2.3. i 2.4.



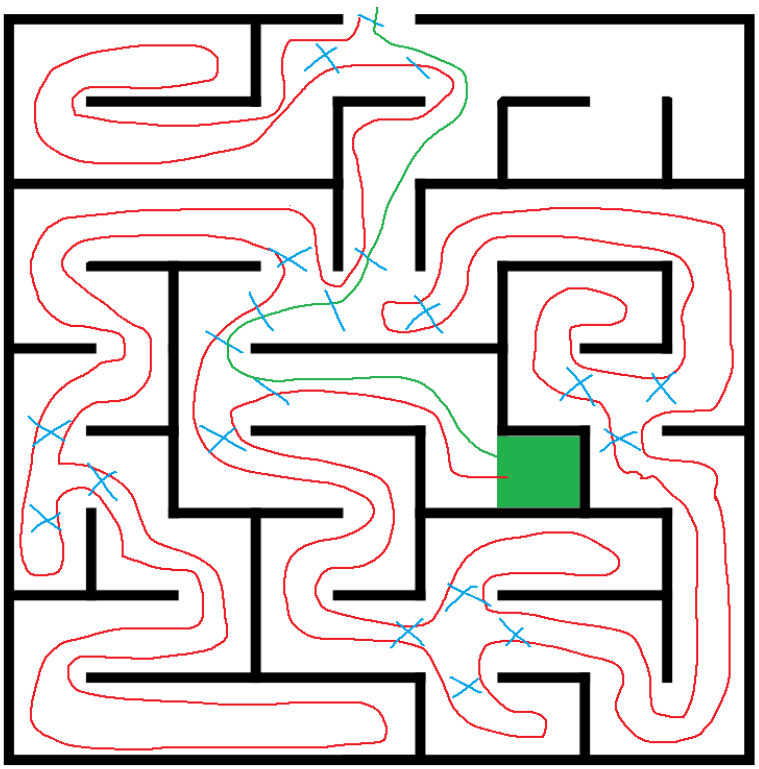
Slika 2.3. Nerješiv labirint za algoritam „slijedi zid“



Slika 2.4 Nerješiv labirint napravljen od linija za algoritam slijedi „zid“

2.2.2. Trémauxov algoritam

Trémauxov algoritam je učinkovit algoritam za pronalaženje izlaza iz labirinta. Ime dobiva po njegovom izumitelju Charles Pierre Trémauxu. Ovim algoritmom garantiran je pronalazak puta do cilja ako on postoji, ali nije garantiran pronalazak najkraćeg puta do tog cilja. Ako put do cilja ne postoji, algoritam će se vratiti na početak. Algoritam funkcionira tako da se označuje svaki put kroz koji prođemo. Put može biti označen niti jednom, jednom ili dva puta. Uvijek se bira put gdje je manje oznaka, tj. nema oznaka. U slučaju da su svi putevi označeni jednom tada se vraća putem kojim se došlo. Ako je put kojim se došlo označen dva puta, a ostali putevi su jednaki broj puta označeni tada se bira bilo koji od tih puteva. Ako je put označen dva puta, tim putem se više ne prolazi. Kada je cilj pronađen, vraća se na početak slijedeći puteve označene samo jednom [6]. Na slici 2.5. prikazan je prijašnji labirint riješen pomoću Trémauxovog algoritama.



Slika 2.5. Prikaz rješenja labirinta pomoću Trémauxovog algoritma

2.3. Najkraći put

Dijsktrin algoritam, koji je izumio Edsger Wybe Dijkstra, je algoritam za pronalazak najkraćeg puta između bilo koja dva čvora u grafu. Ovaj algoritam ima razne varijante, jedna od najpoznatijih je odabrati jedan čvor kao „izvor“ i pronaći najkraći put od „izvora“ do ostalih čvorova. Originalna metoda je služila da pronađe najkraći put između dva odabrana čvora[7].

Algoritam radi tako da čvorove svrsta u dvije skupine: obiđeni i neobiđeni. Prvi obiđeni čvor je početni, te od njega se gledaju najkraće udaljenosti do ostalih čvorova. Slijedeći čvor je onaj koji ima najkraću udaljenost od početnog, te on postaje sljedeći obiđeni čvor. Ostale udaljenosti čvorova se spremaju, te zatim se gleda najkraća udaljenost do čvorova povezanih obiđenim čvorom. Zatim se ponovo gleda najkraća udaljenost do neobiđenog čvor, te on postaje obiđeni. Taj postupak se ponavlja sve dok se ne prođu svi čvorovi ili ne dođe do željenog čvora[8].

2.4. Python

Krajem 1980-ih Gudio van Rossum je stvorio interpretirani programski jezik Python. Python je visoko razinski jezik opće namjene koji stavlja naglasak na čitljivost koda. Njegov objektno orijentirani pristup pomaže pri čistoći i jasnoći koda za male i velike projekte[9]. Prva verzija je izašla 1991. godine pod nazivom Python 0.9.0. Verzija 2.0 koja je izašla 2000. godine donijela je važne promjene kao što su poboljšanje funkcija lista te automatsko oslobađanje memorije. Zadnja glavna verzija 3.0 izašla je 2008 te je u uporabi i danas[10]. Programska podrška u Pythonu dinamički je pisana, tj. provjera o ispravnosti tipa podatka događa se tek kada program krene sa izvođenjem. Varijable i njihov tip ne moraju se deklarirati unaprijed.

Filozofija Pythona sumirana je u dokumentu The Zen of Python [11]:

- Lijepo je bolje nego ružno
- Eksplicitno je bolje nego implicitno
- Jednostavno je bolje nego složeno
- Složeno je bolje nego zamršeno
- Čitljivost je bitna

3. IZRADA

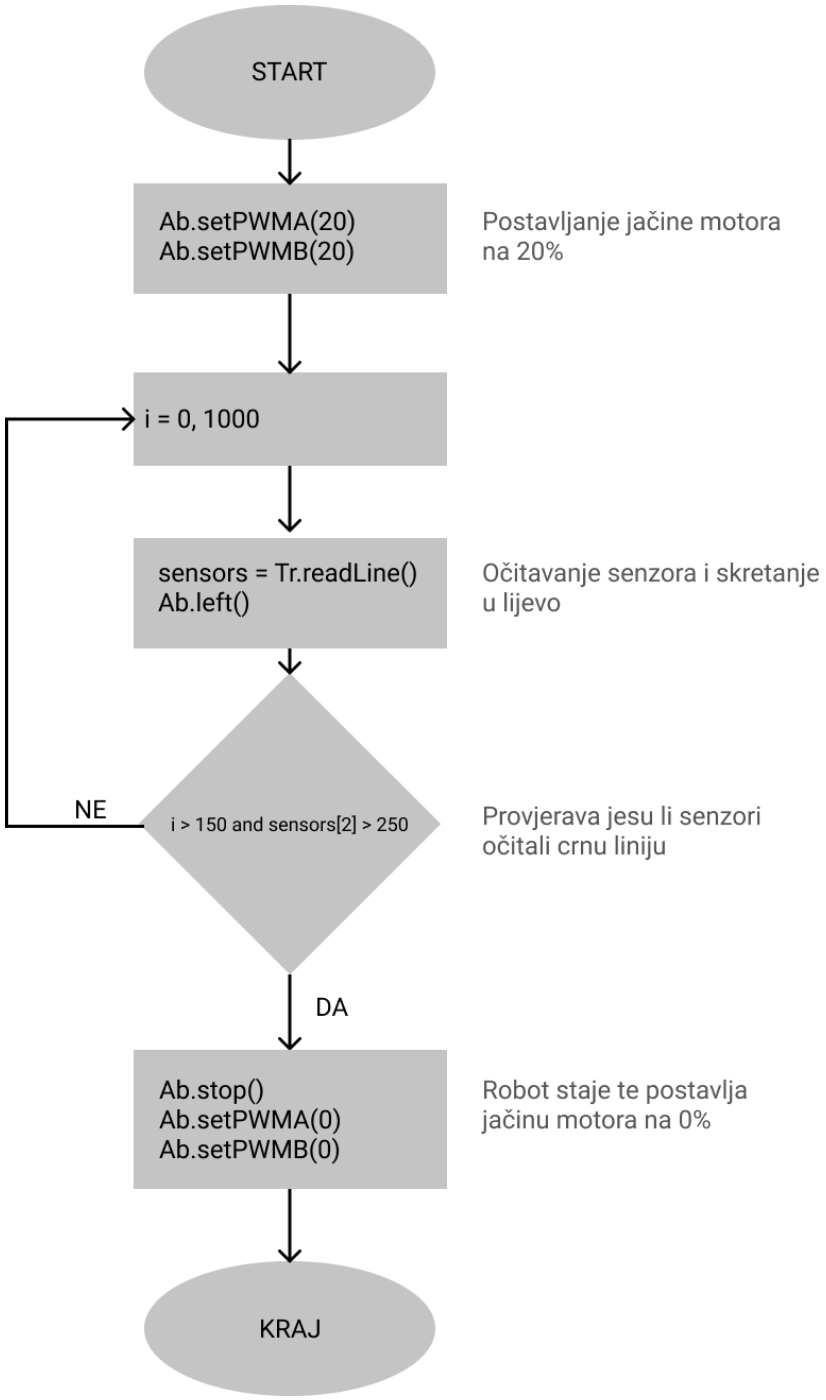
Za povezivanje s Alphabot2 robotom koristio se SSH protokol preko kojeg se vršila konfiguracija i realizacija samog zadatka. Programska podrška pomoću koje je ostvaren zadatak pisana je u programskom jeziku Python (v3.9).

3.1. Kretanje

Alphabot2 dolazi sa gotovim setovima instrukcija za kretanje i korištenje infracrvenih senzora. Pomoću te dvije klase napravljena je jedna klasa za kretanje po labirintu nazvana „MazeMovement“. U njoj se nalaze 6 metoda od kojih je 5 za kretanje, a jedna za kalibriranje senzora. Primjer jedne metode prikazan je na slici 3.1, a dijagram toka za tu metodu prikazan je na slici 3.2.. Kada je metoda pozvana ona prvo postavlja jačinu motora na 20% maksimalne snage, kako bismo dobili optimalnu brzinu kretanja. Zatim ulazi u petlju koja sprema očitavanje senzora i naređuje robotu da skreće ulijevo sve dok ne vidi sljedeću crnu liniju koja predstavlja put.

Linija	***Kod***
1:	`def Left(self):`
2:	`    self.Ab.setPWMA(20)`
3:	`    self.Ab.setPWMB(20)`
4:	`    for i in range(0, 1000):`
5:	`        position,Sensors = self.TR.readLine()`
6:	`        self.Ab.left()`
7:	`        if(i > 150 and Sensors[2] > 250):`
8:	`            self.Ab.stop()`
9:	`            self.Ab.setPWMA(0)`
10:	`            self.Ab.setPWMB(0)`
11:	`            break`

Slika 3.1. Metoda za skretanje prema lijevo u labirintu



Slika 3.2. Dijagram toka za skretanje prema lijevo u labirintu

3.2. Mapiranje

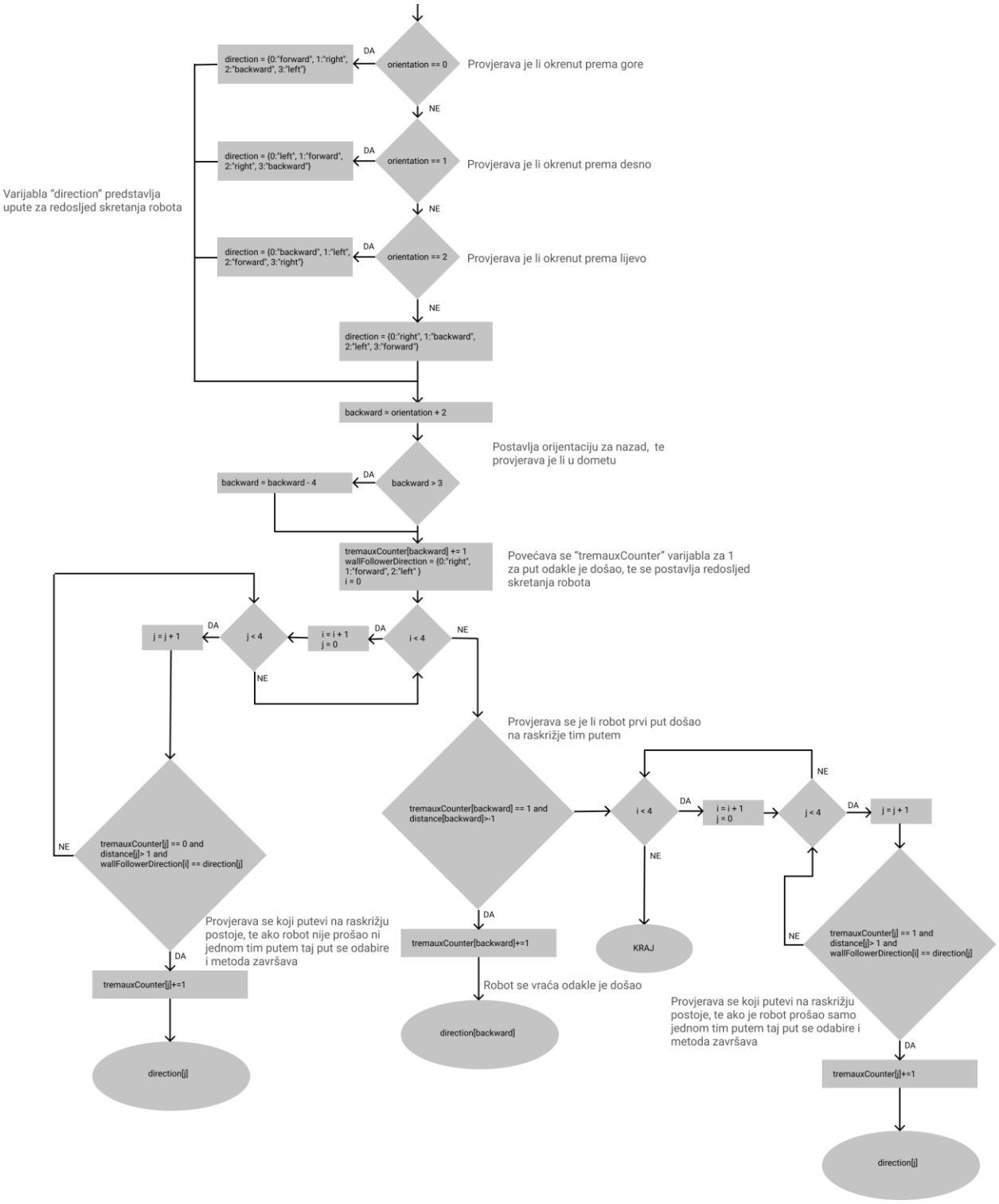
Za mapiranje labirinta koristio se Trémauxov algoritam. U slučaju kada je put kojim se došlo označen dva puta, a ostali putevi jednak broj puta, koristio se algoritam „slijedi zid“, tj. na svakom jednako označenom raskrižju robot će birati desno skretanje ako je moguće. Robot će mjeriti udaljenost od raskrižja do raskrižja, te će ih spremiti i označiti kao čvorove. Kada robot pronađe cilj spremit će ga kao čvor i nastavit će dalje sa Trémauxovim algoritmom dok se ne vrati na

početak. Na slici 3.3. prikazana je metoda za odabir skretanja na raskrižju, a na slici 3.4. prikazan je dijagram toka te metode. Metoda prima varijablu koja predstavlja trenutnu orijentaciju robota. Robot ima četiri orijentacije koje su prikazane pomoću brojeva: 0 – gore, 1 – desno, 2 – dolje i 3 – lijevo. Početna orijentacija robota je „gore“. Metoda u ovisnosti na orijentaciju postavlja varijablu smjer. Nakon toga povećava atribut „tremauxCounter“ za put kojim je došao robot do tog raskrižja. Taj atribut broji koliko puta se prošlo određenim putem. Na kraju metoda provjerava dostupne puteve, te prema Tremauxovom i „slijedi zid“ algoritmu provjerava koji put će odabrati. Tom putu povećava brojač i vraća odgovarajući smjer.

Linija ***Kod***

```
1:  def Choose(self, orientation):
2:      if(orientation == 0):
3:          direction = {0:"forward", 1:"right", 2:"backward", 3:"left"}
4:      elif(orientation == 1):
5:          direction = {0:"left", 1:"forward", 2:"right", 3:"backward"}
6:      elif(orientation == 2):
7:          direction = {0:"backward", 1:"left", 2:"forward", 3:"right"}
8:      else:
9:          direction = {0:"right", 1:"backward", 2:"left", 3:"forward"}
10:     backward = orientation + 2
11:     if(backward > 3):
12:         backward -= 4
13:     self.tremauxCounter[backward] += 1
14:     wallFollowerDirection = {0:"right", 1:"forward", 2:"left"}
15:     for i in range (3):
16:         for j in range(4):
17:             if(self.tremauxCounter[j] == 0 and self.distance[j]> 1 and
                    wallFollowerDirection[i] == direction[j]):
18:                 self.tremauxCounter[j]+=1
19:                 return direction[j]
20:     if(self.tremauxCounter[backward] == 1 and self.distance[backward]>-1):
21:         self.tremauxCounter[backward]+=1
22:         return direction[backward]
23:     for i in range (3):
24:         for j in range(4):
25:             if(self.tremauxCounter[j] == 1 and self.distance[j]>-1 and
                    wallFollowerDirection[i] == direction[j]):
26:                 self.tremauxCounter[j]+=1
27:                 return direction[j]
```

Slika 3.3. Metoda za odabir skretanja na raskrižju



Slika 3.4. Dijagram toka za odabir skretanja na raskrižju

3.3. Rješavanje labirinta

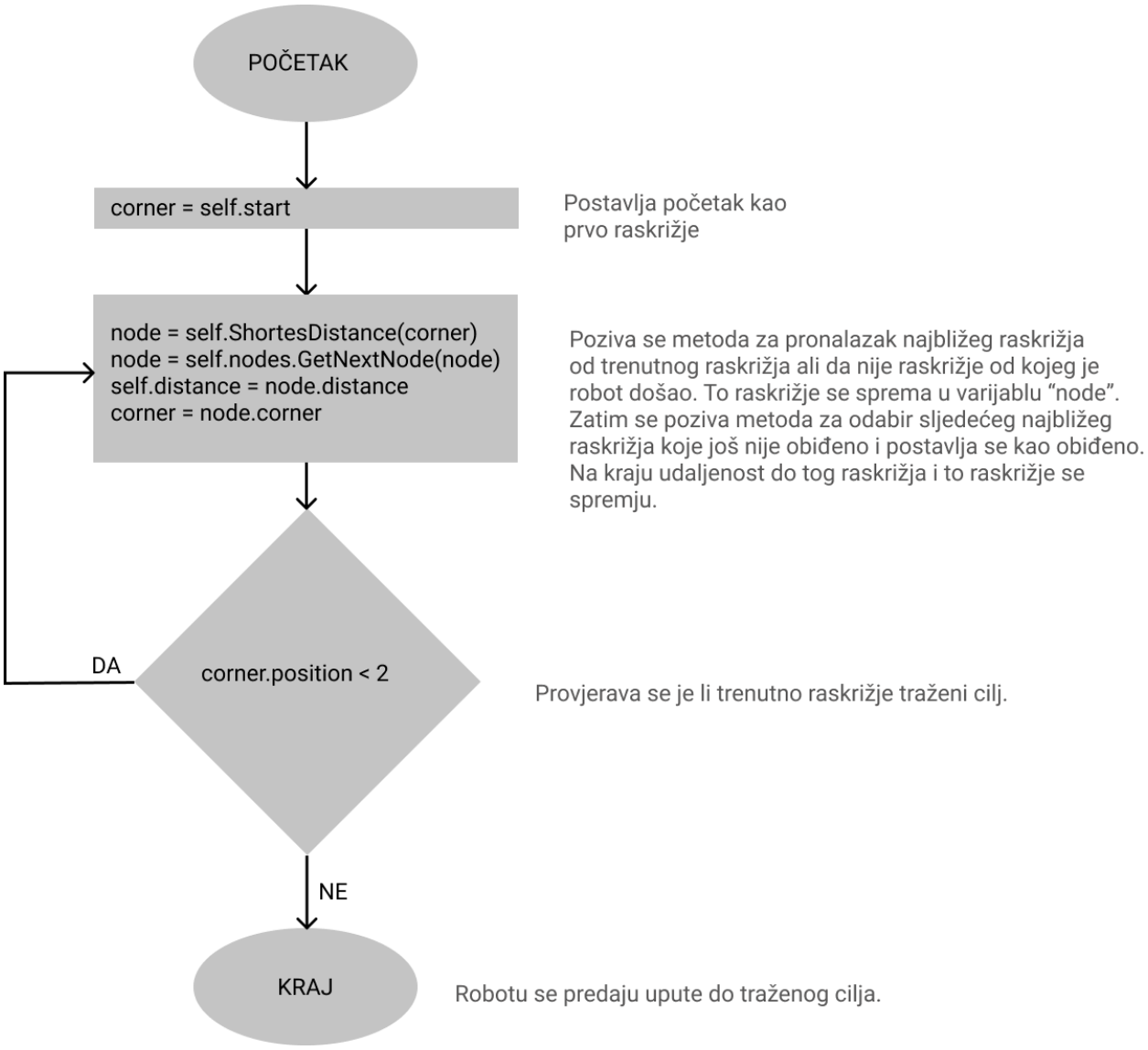
Kada se robot vrati na početak, pomoću Dijsktrinog algoritma računa najkraći put. Nakon izračuna robot prati taj put do cilja. Primjena Dijsktrinog algoritma prikazana je na slici 3.5, te blok dijagram

te metode prikazan je na slici 3.6. Metoda varijablu „corner“ postavlja na početnu vrijednost koja predstavlja početnu poziciju robota. Zatim ulazi u petlji u kojoj prvo postavlja varijablu „node“ na najkraću udaljenost od varijable „corner“ tj. od raskrižja. Varijabla „node“ u sebi sadržava varijablu „corner“, udaljenost tog raskrižja od početne pozicije, te varijablu „prev“ koja predstavlja raskrižje preko kojeg se došlo do „cornera“. Nakon toga metoda traži pomoću varijable „node“ najkraću udaljenost do neobiđenog raskrižja i sprema ga u „node“. Zatim tu udaljenost sprema i to raskrižje sprema u varijablu „corner“. Taj proces se ponavlja sve dokle varijabla „corner“ ne postane traženi cilj.

Linija ***Kod***

```
1:      def DijkstraAlgoritham(self):
2:          corner = self.start
3:          while(corner.position < 2):
4:              node = self.ShortesDistance(corner)
5:              node = self.nodes.GetNextNode(node)
6:              self.distance = node.distance
7:              corner = node.corner
```

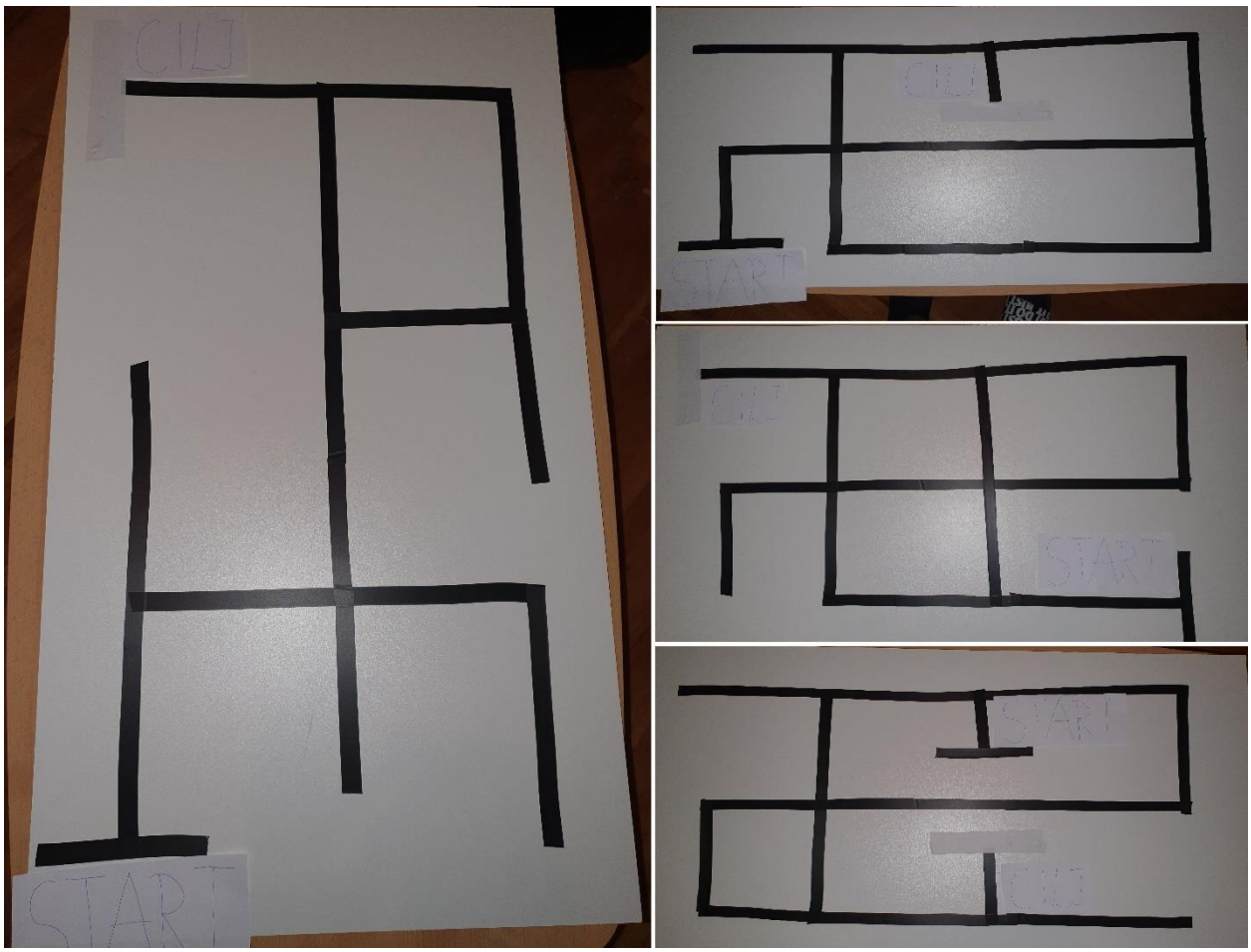
Slika 3.5. Metoda za računanje najkraćeg puta



Slika 3.6. Blok dijagram za računanje najkraćeg puta

4. REZULTATI

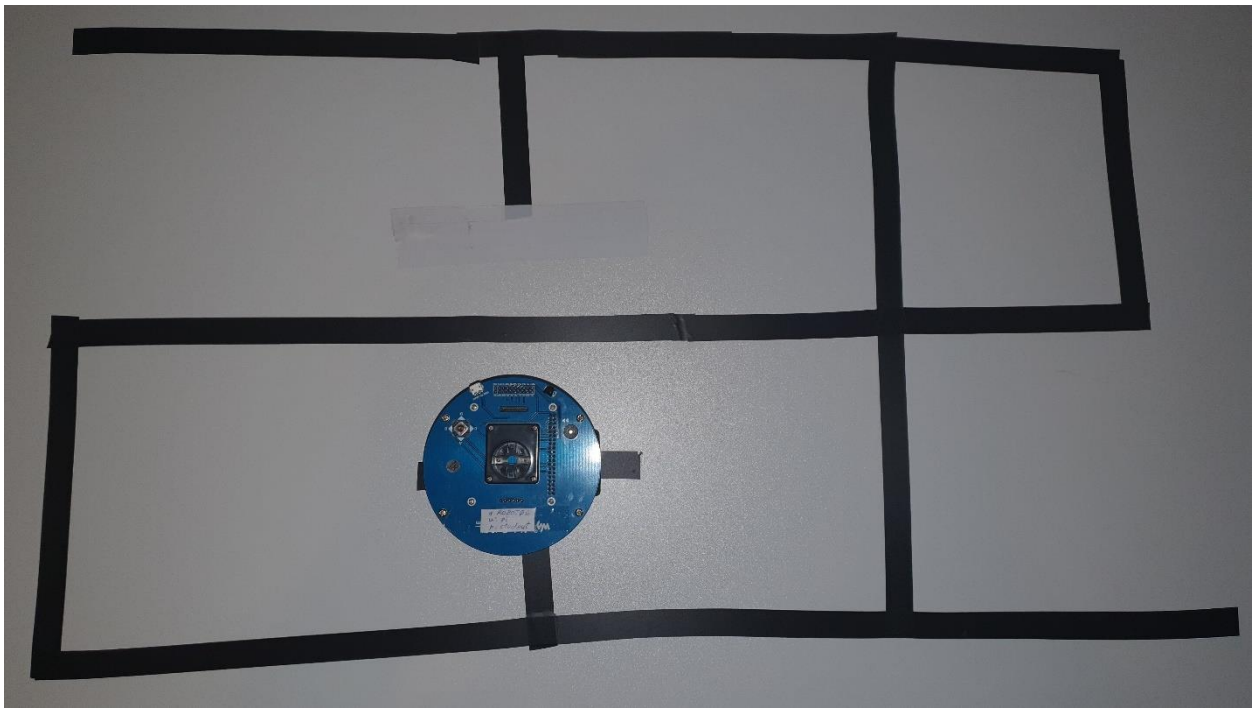
Za prikaz labirinta koristila se bijela ploča na kojoj su nalijepljene crne linije koje predstavljaju put. Početak je označen okomito zalijepljenom crnom linijom, a kraj okomito zalijepljenom bijelom linijom. Iako ljudsko oko teško raspoznaje razliku između bijele površine ploče i bijele linije, zbog različite refleksije ta dva materijala robotu je jasna razlika. Ovim načinom je postignuta jednostavna izmjena početka i cilja, te kompletnog labirinta. Zbog toga robot je testiran sa više različitih labirinata, te ih je sve uspješno riješio. Na slici 4.1. prikazano je nekoliko verzija labirinta.



Slika 4.1. Labirinti

Na slikama od 4.2 do 4.30. prikazan je proces mapiranja labirinta. U tablicama od 4.1 do 4.29. prikazane su varijable koje se robot sprema pri mapiranju labirinta. Varijabla „corner“ predstavlja raskrižje labirinta. Za lakši prikaz svako raskrižje je prikazano jednim slovom, dok su u kodu prikazani memorijskom adresom. Gore, desno, dolje i lijevo u tablici predstavlja atribute varijable „corner“ koji pokazuju na raskrižja gore, desno, dolje ili lijevo od tog trenutnog raskrižja. Navedeni smjerovi ovise o početku labirinta. Npr. gore je ravno sve dok robot ne skrene, ako

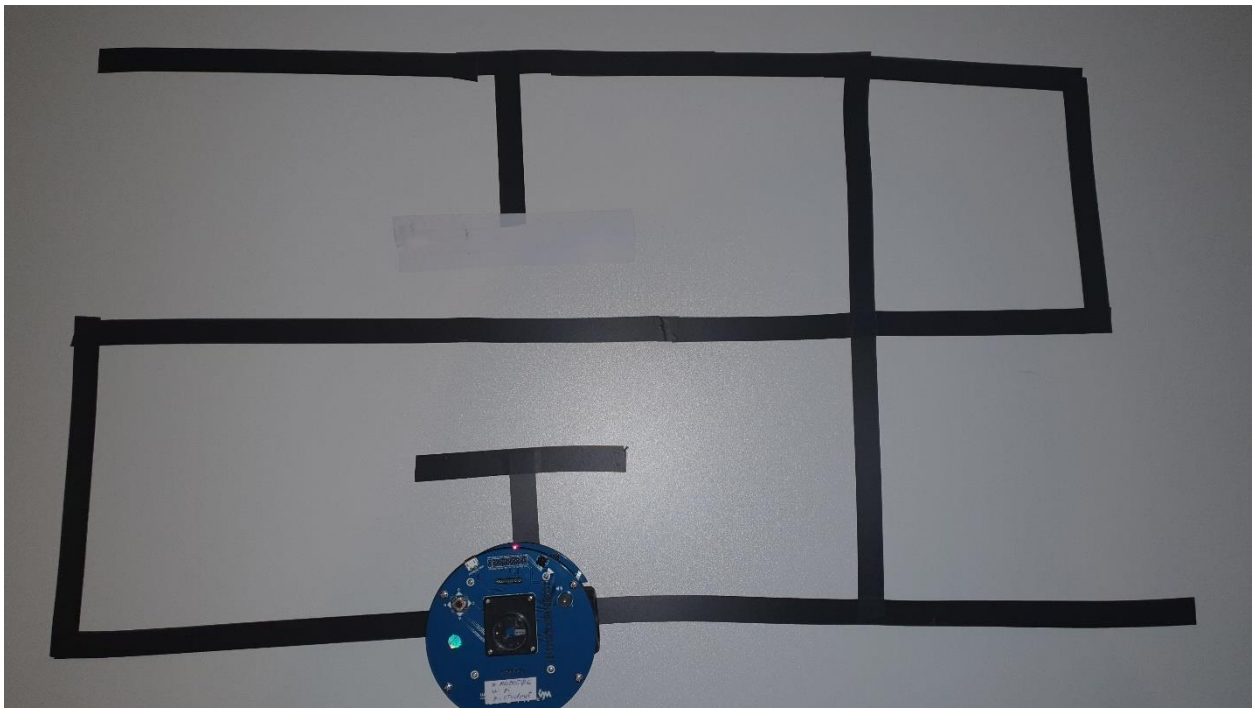
skrene desno onda će gore biti lijevo, a desno biti ravno naspram trenutne pozicije robota. Robot broji svaki put s koje strane je došao na određeno raskrižje te to sprema u varijablu „tremauxCounter“. Za lakši prikaz u tablici je „tremauxCounter“ zamijenjen sa TC, te je u zagradi naveden smjer koji je ranije objašnjen. U tablici je podebljano slovo raskrižja na kojem se robot trenutno nalazi. Također su podebljanje promjene koje se vrše tijekom dolaska na određeni dio labirinta. Na kraju cilj je dodatno označen podvučenom crtom.



Slika 4.2.

Tablica 4.1.

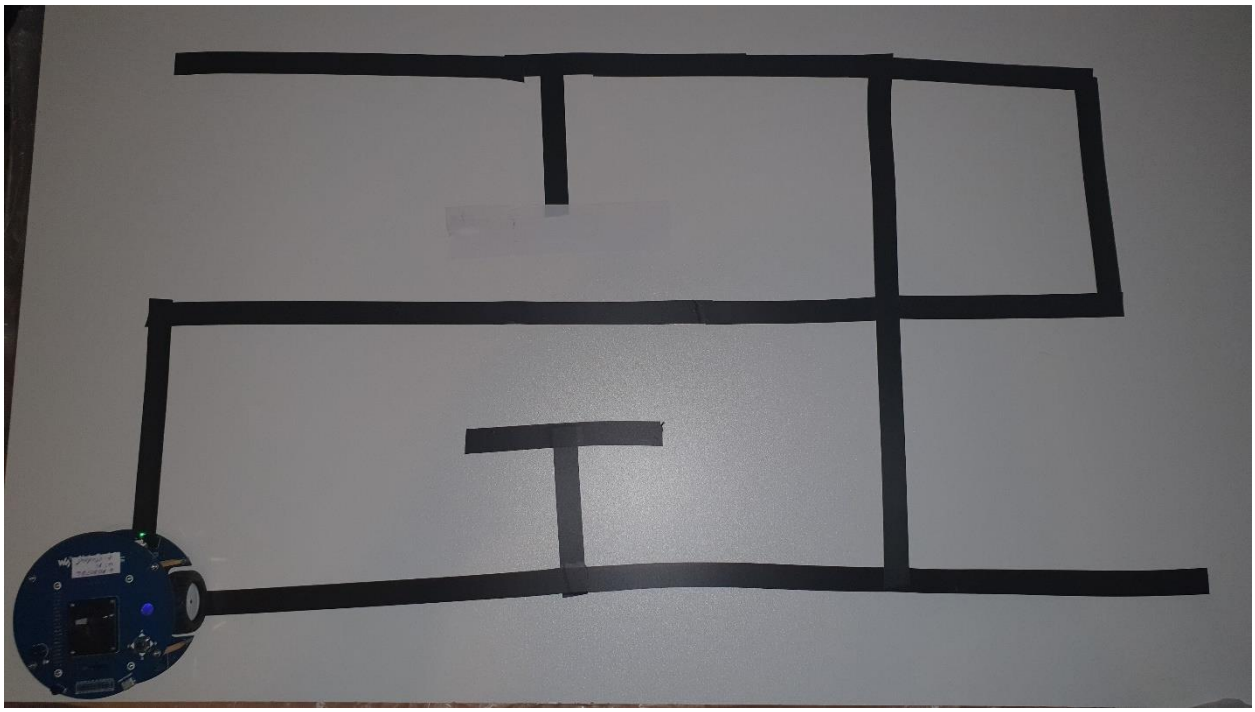
„corner“	**Gore**	**Desno**	**Dolje**	**Lijevo**	**TC[0]**	**TC[1]**	**TC[2]**	**TC[3]**
A	/	/	/	/	0	0	0	0



Slika 4.3.

Tablica 4.2.

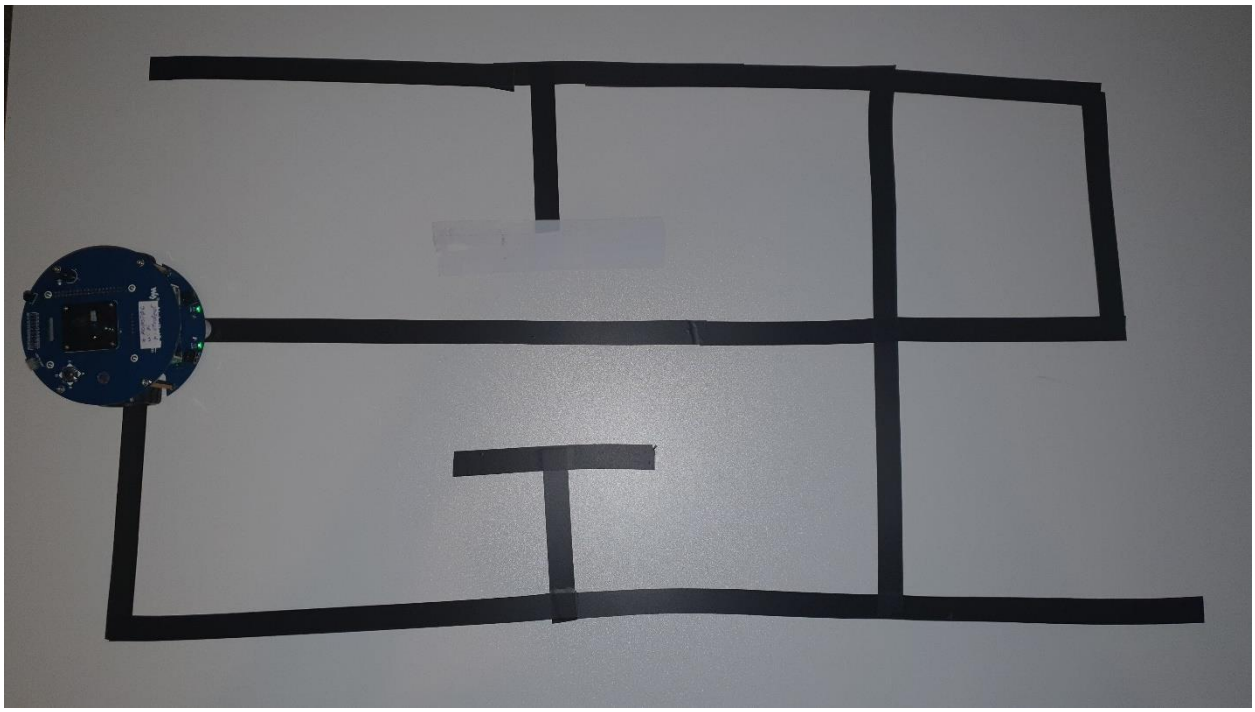
„corner“	**Gore**	**Desno**	**Dolje**	**Lijevo**	**TC[0]**	**TC[1]**	**TC[2]**	**TC[3]**
A	**B**	/	/	/	**1**	0	0	0
B	/	/	**A**	/	0	0	**1**	0



Slika 4.4.

Tablica 4.3.

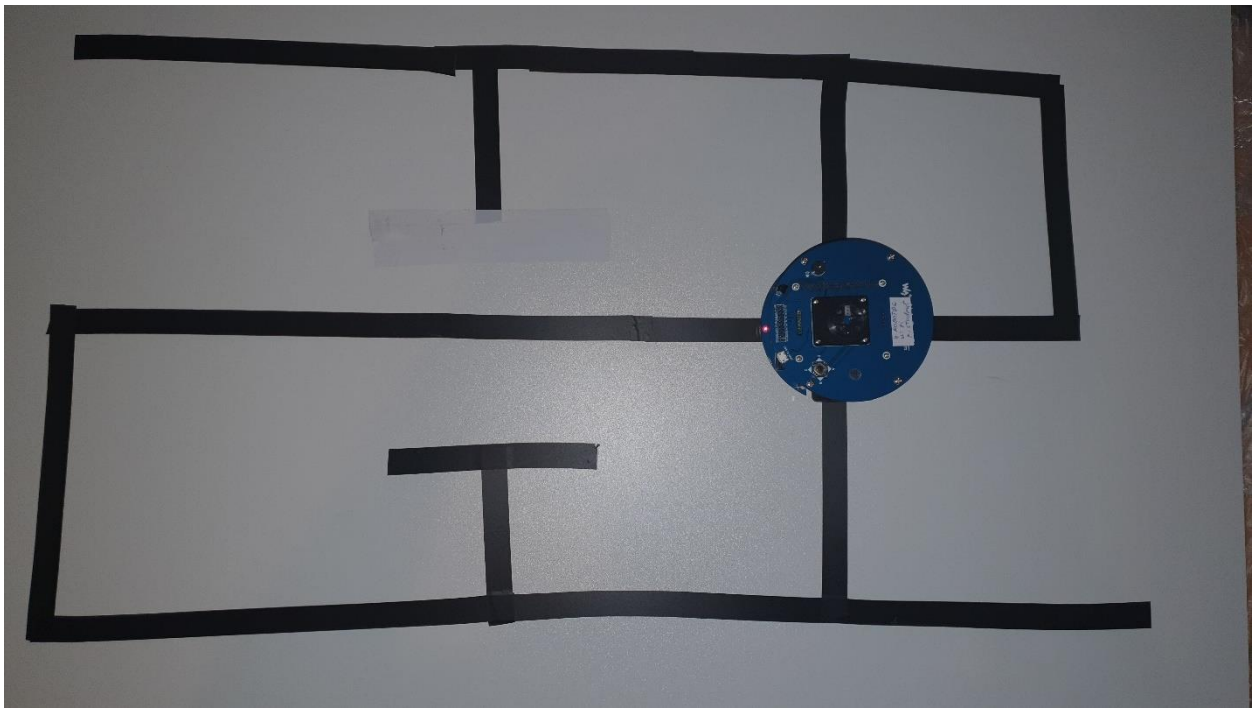
„corner“	**Gore**	**Desno**	**Dolje**	**Lijevo**	**TC[0]**	**TC[1]**	**TC[2]**	**TC[3]**
A	B	/	/	/	1	0	0	0
B	/	**C**	A	/	0	**1**	1	0
C	**/**	/	/	**B**	0	0	0	**1**



Slika 4.5.

Tablica 4.4.

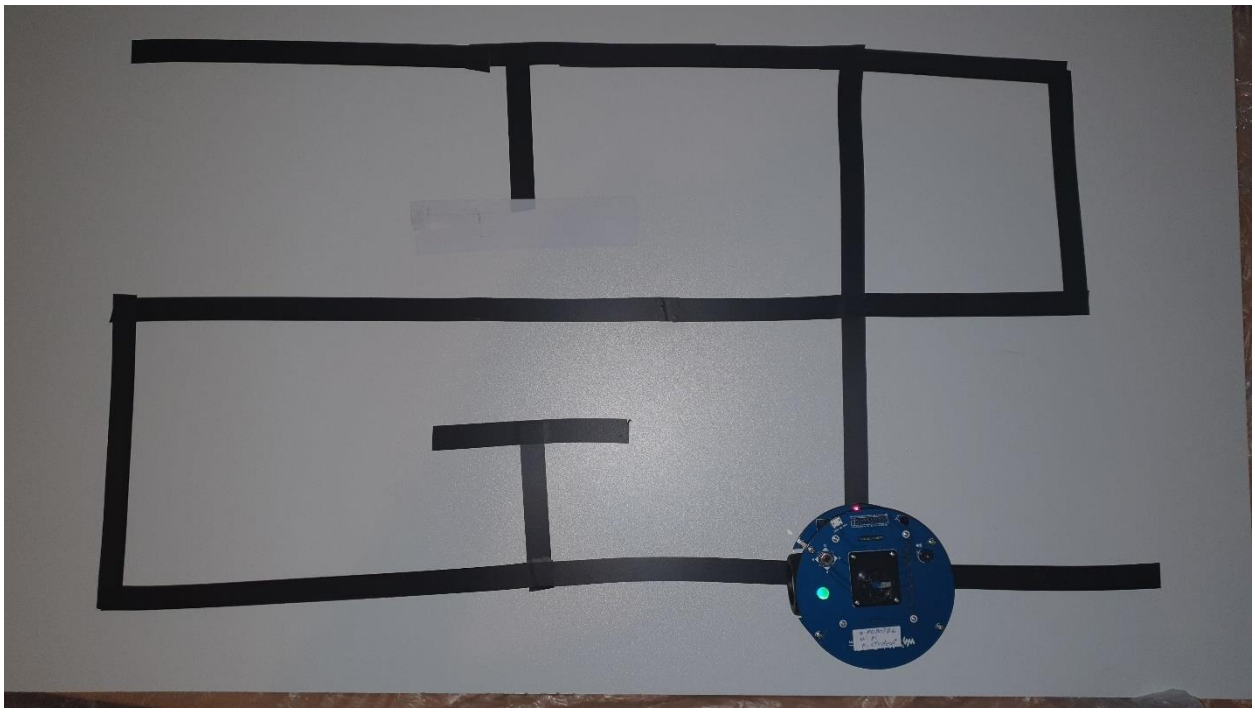
„corner“	**Gore**	**Desno**	**Dolje**	**Lijevo**	**TC[0]**	**TC[1]**	**TC[2]**	**TC[3]**
A	B	/	/	/	1	0	0	0
B	/	C	A	/	0	1	1	0
C	**/**	/	**D**	B	0	0	**1**	1
D	**C**	/	/	/	**1**	0	0	0



Slika 4.6.

Tablica 4.5.

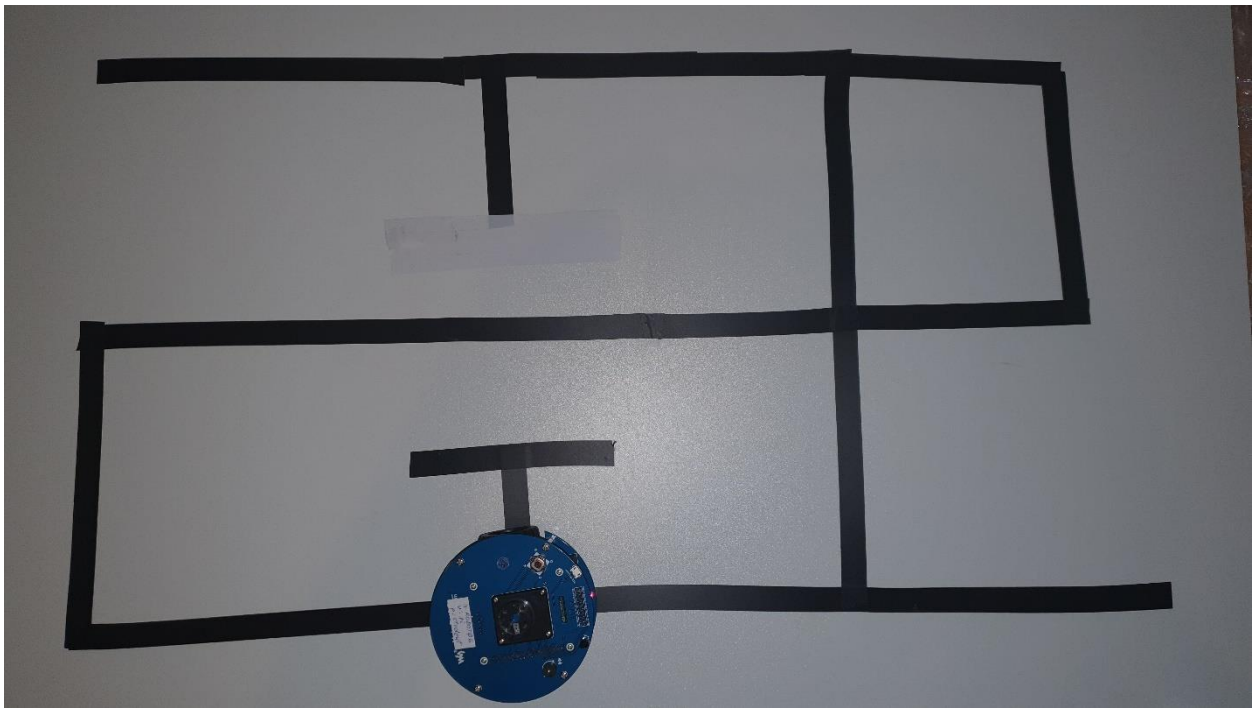
„corner“	**Gore**	**Desno**	**Dolje**	**Lijevo**	**TC[0]**	**TC[1]**	**TC[2]**	**TC[3]**
A	B	/	/	/	1	0	0	0
B	/	C	A	/	0	1	1	0
C	**/**	/	D	B	0	0	1	1
D	C	/	/	**E**	1	0	0	**1**
E	/	**D**	/	/	0	**1**	0	0



Slika 4.7.

Tablica 4.6.

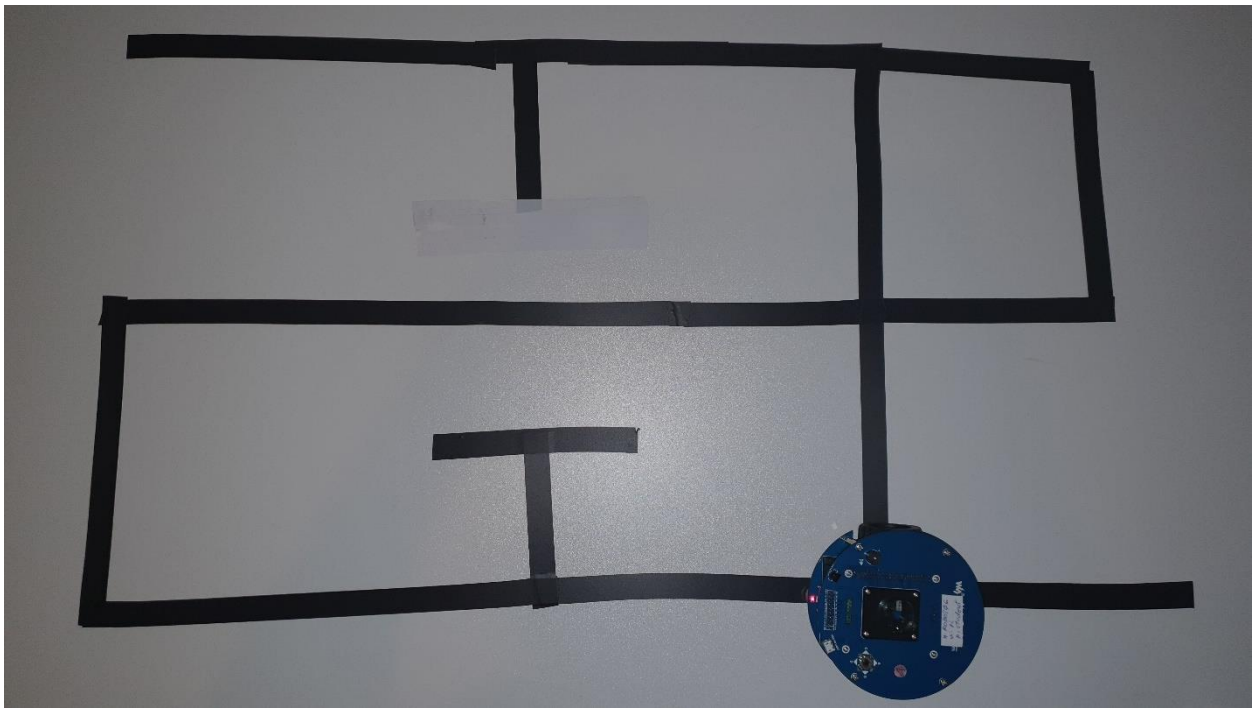
„corner“	**Gore**	**Desno**	**Dolje**	**Lijevo**	**TC[0]**	**TC[1]**	**TC[2]**	**TC[3]**
A	B	/	/	/	1	0	0	0
B	/	C	A	/	0	1	1	0
C	**/**	/	D	B	0	0	1	1
D	C	/	/	E	1	0	0	1
E	**F**	D	/	/	**1**	1	0	0
F	/	/	**E**	/	0	0	**1**	0



Slika 4.8.

Tablica 4.7.

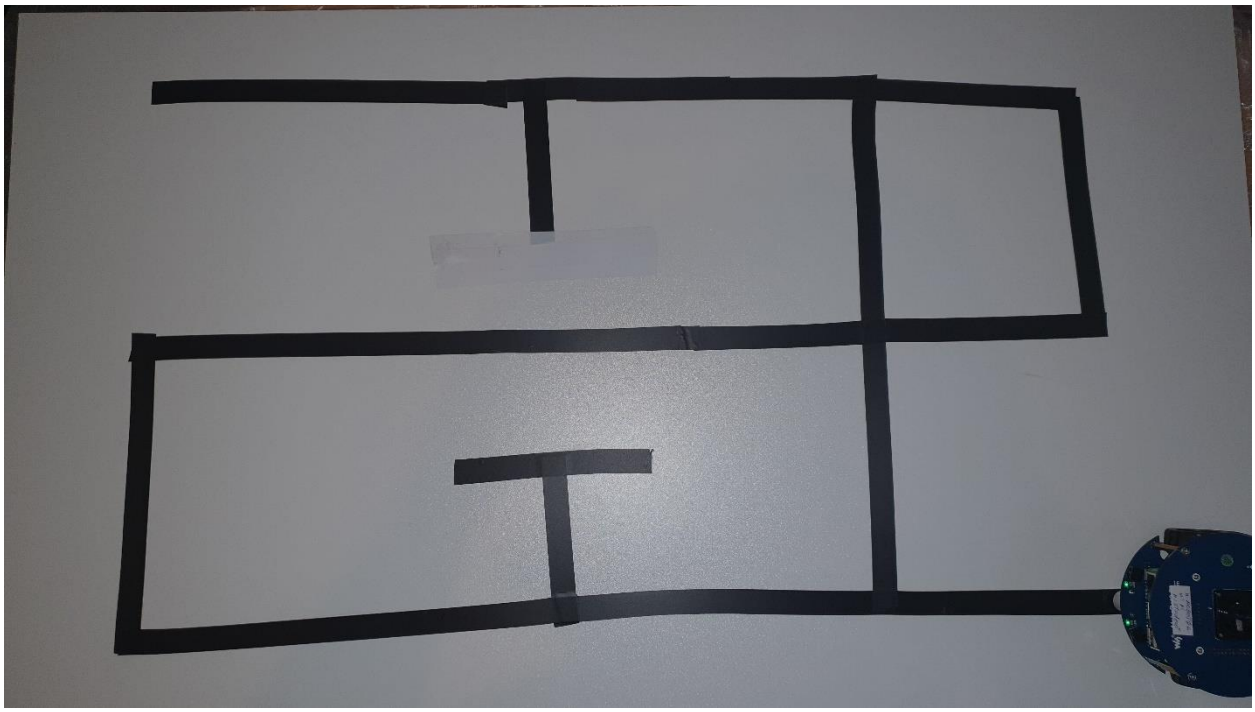
„corner“	**Gore**	**Desno**	**Dolje**	**Lijevo**	**TC[0]**	**TC[1]**	**TC[2]**	**TC[3]**
A	B	/	/	/	1	0	0	0
B	/	C	A	**F**	0	1	1	**1**
C	**/**	/	D	B	0	0	1	1
D	C	/	/	E	1	0	0	1
E	F	D	/	/	1	1	0	0
F	/	**B**	E	/	0	**1**	1	0



Slika 4.9.

Tablica 4.8.

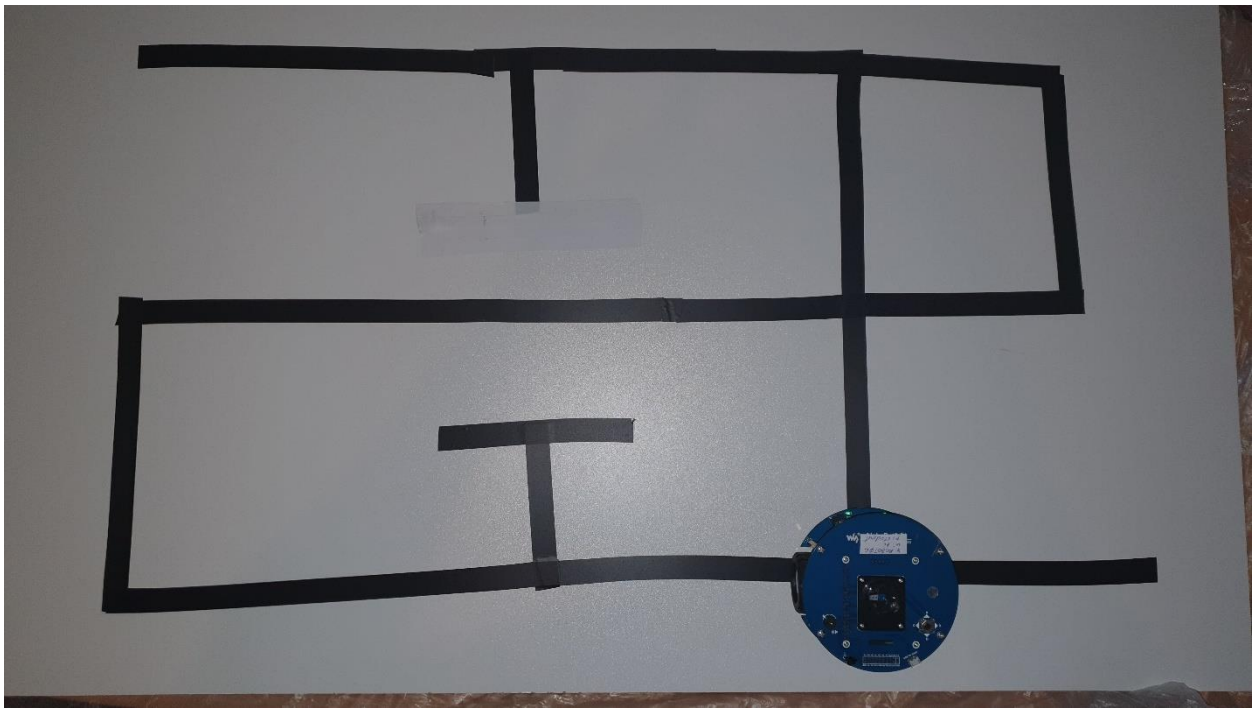
„corner“	**Gore**	**Desno**	**Dolje**	**Lijevo**	**TC[0]**	**TC[1]**	**TC[2]**	**TC[3]**
A	B	/	/	/	1	0	0	0
B	/	C	A	/	0	1	1	**2**
C	**/**	/	D	B	0	0	1	1
D	C	/	/	E	1	0	0	1
E	F	D	/	/	1	1	0	0
F	/	/	E	/	0	**2**	1	0



Slika 4.10.

Tablica 4.9.

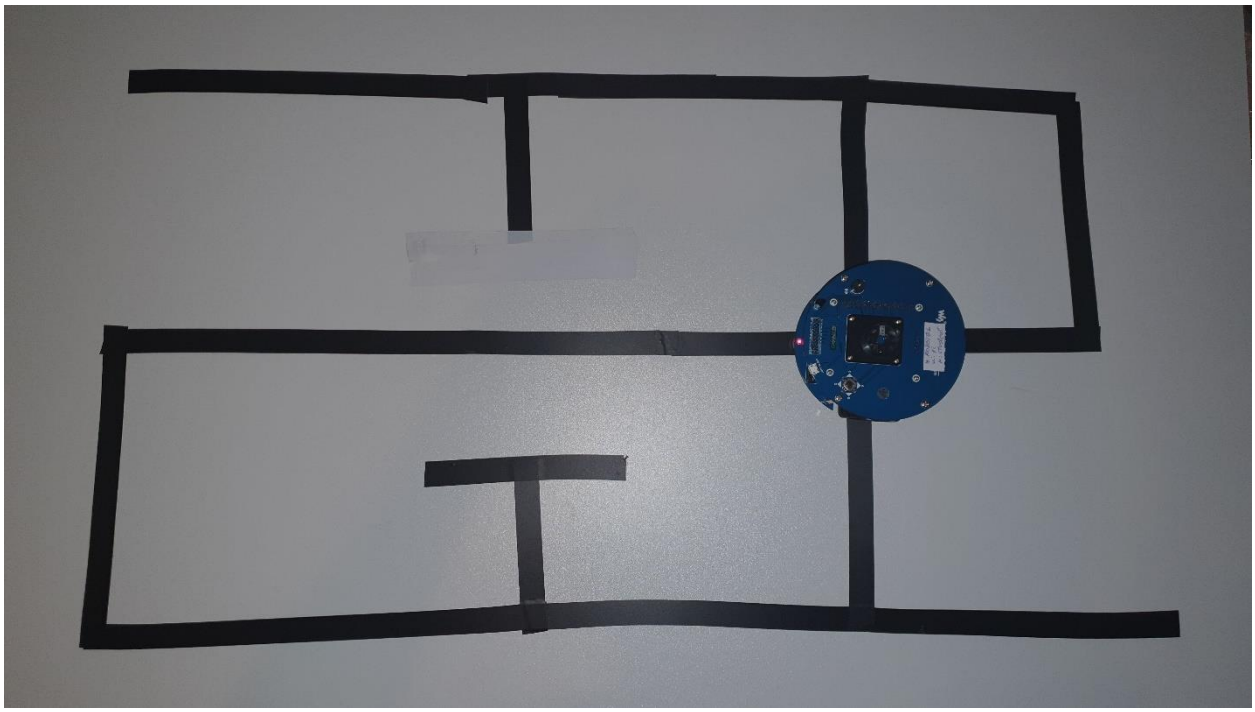
„corner“	**Gore**	**Desno**	**Dolje**	**Lijevo**	**TC[0]**	**TC[1]**	**TC[2]**	**TC[3]**
A	B	/	/	/	1	0	0	0
B	/	C	A	F	0	1	1	2
C	/	/	D	B	0	0	1	1
D	C	/	/	E	1	0	0	1
E	F	D	/	/	1	1	0	0
F	/	B	E	/	0	0	1	**1**



Slika 4.11.

Tablica 4.10.

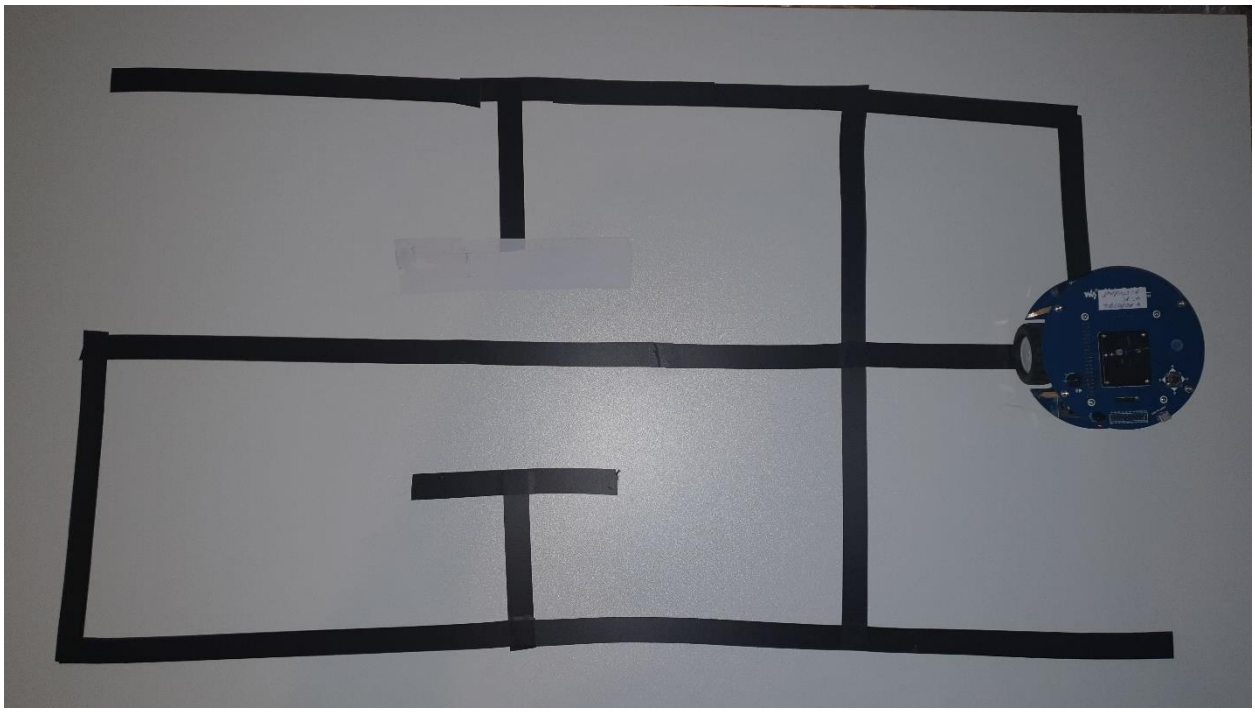
„corner“	**Gore**	**Desno**	**Dolje**	**Lijevo**	**TC[0]**	**TC[1]**	**TC[2]**	**TC[3]**
A	B	/	/	/	1	0	0	0
B	/	C	A	F	0	1	1	2
C	/	/	D	B	0	0	1	1
D	C	/	/	E	1	0	0	1
E	F	D	/	/	1	1	0	0
F	/	B	E	/	0	2	1	**2**



Slika 4.12.

Tablica 4.11.

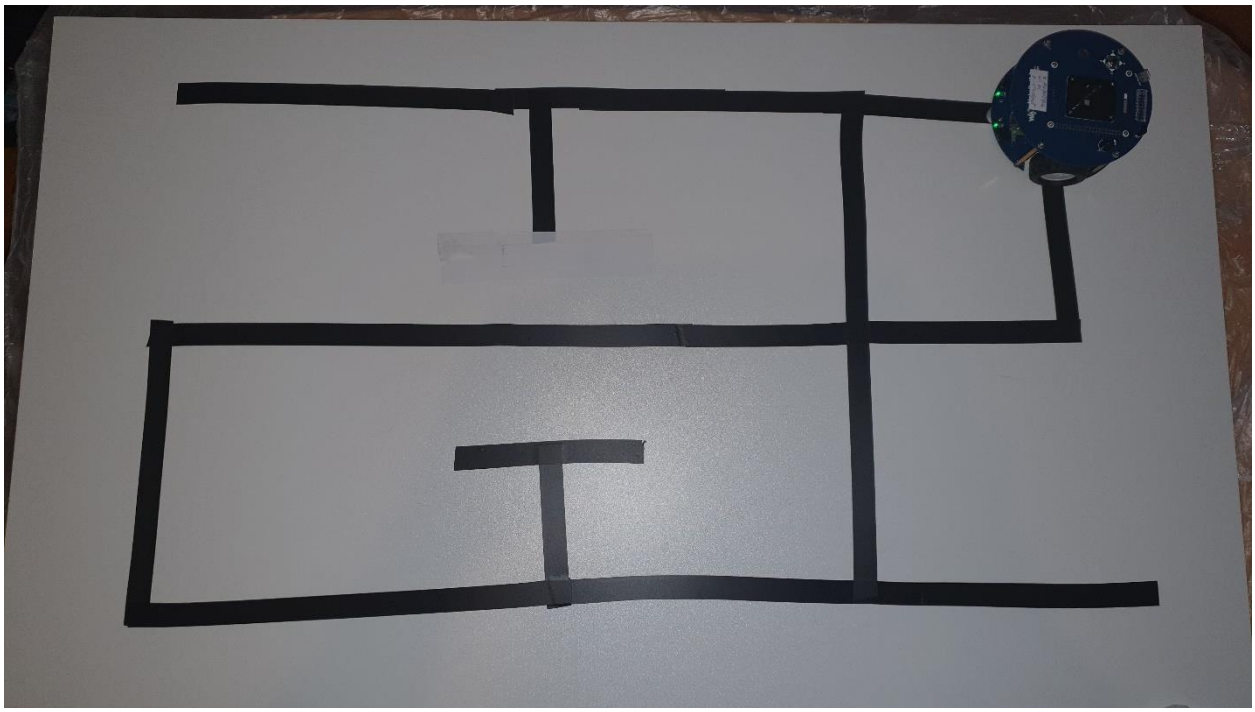
„corner“	**Gore**	**Desno**	**Dolje**	**Lijevo**	**TC[0]**	**TC[1]**	**TC[2]**	**TC[3]**
A	B	/	/	/	1	0	0	0
B	/	C	A	F	0	1	1	2
C	/	/	D	B	0	0	1	1
D	C	/	/	E	1	0	0	1
E	F	D	/	/	**2**	1	0	0
F	/	B	E	/	0	2	**2**	2



Slika 4.13

Tablica 4.12

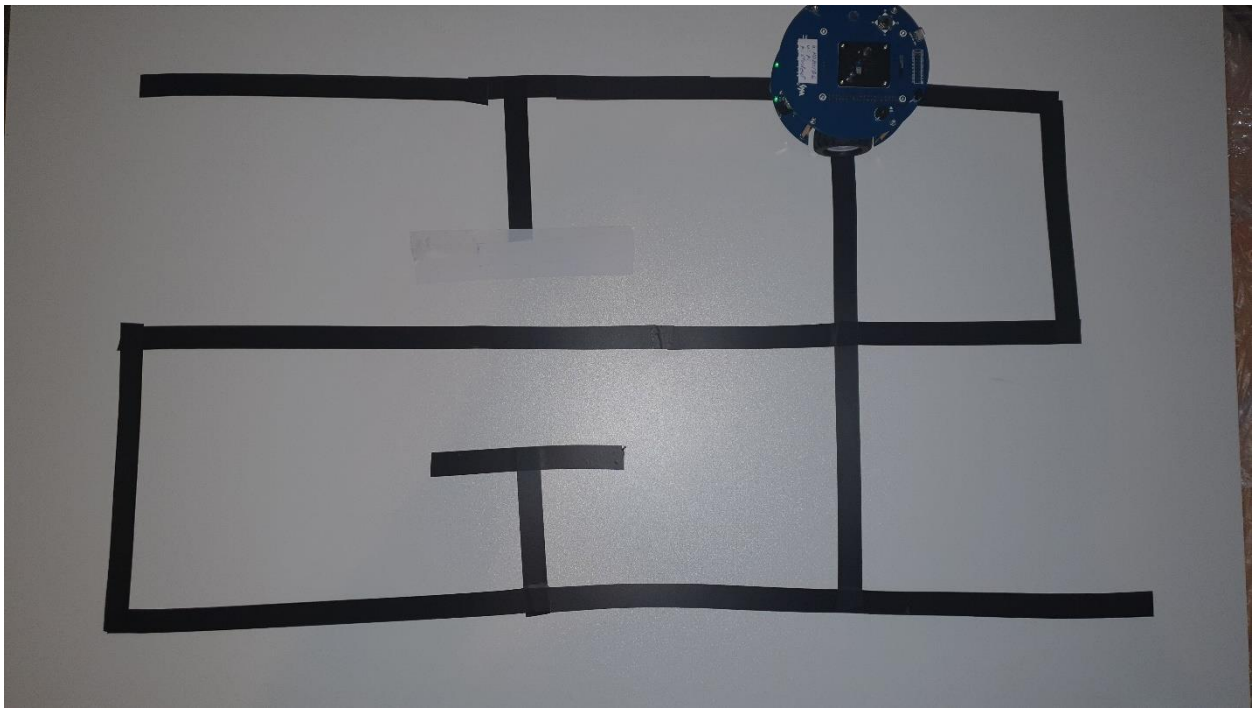
„corner“	**Gore**	**Desno**	**Dolje**	**Lijevo**	**TC[0]**	**TC[1]**	**TC[2]**	**TC[3]**
A	B	/	/	/	1	0	0	0
B	/	C	A	F	0	1	1	2
C	/	/	D	B	0	0	1	1
D	C	/	/	E	1	0	0	1
E	F	D	/	**G**	2	1	0	**1**
F	/	B	E	/	0	2	2	2
G	/	**E**	/	/	0	**1**	0	0



Slika 4.14.

Tablica 4.13.

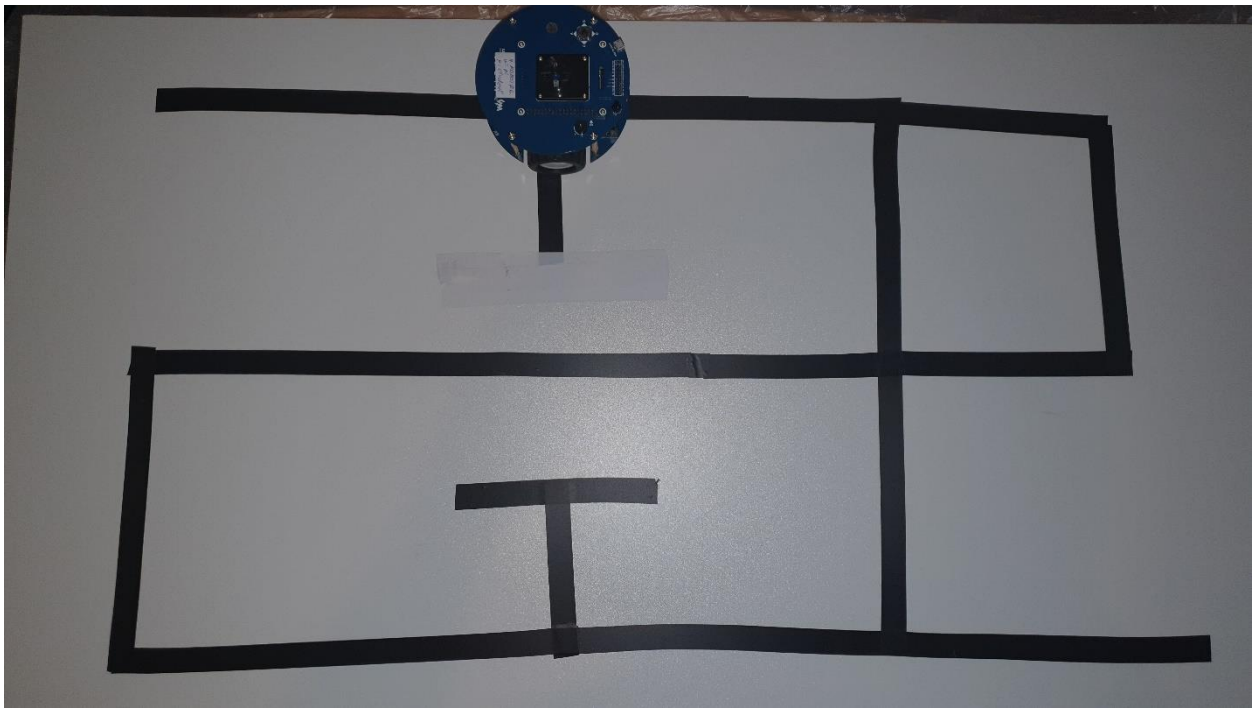
„corner“	**Gore**	**Desno**	**Dolje**	**Lijevo**	**TC[0]**	**TC[1]**	**TC[2]**	**TC[3]**
A	B	/	/	/	1	0	0	0
B	/	C	A	F	0	1	1	2
C	**/**	/	D	B	0	0	1	1
D	C	/	/	E	1	0	0	1
E	F	D	/	G	2	1	0	1
F	/	B	E	/	0	2	2	2
G	/	E	**H**	/	0	1	**1**	0
H	**G**	/	/	/	**1**	0	0	0



Slika 4.15.

Tablica 4.14.

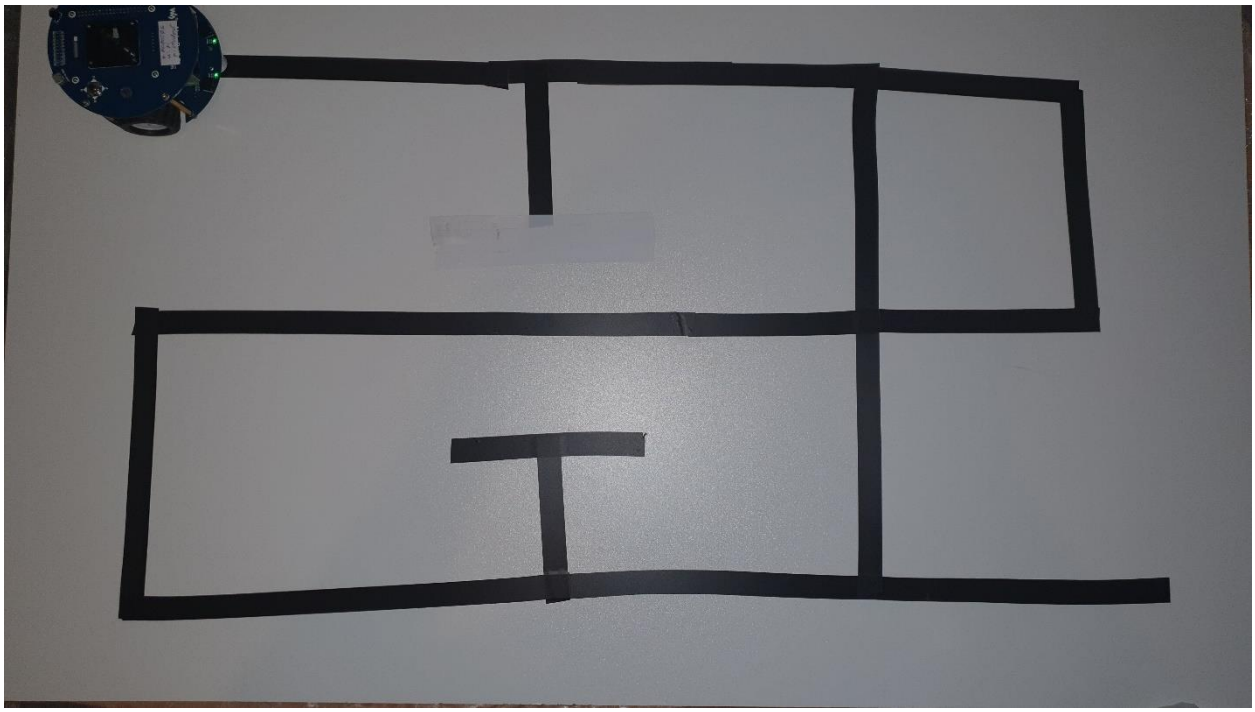
„corner“	**Gore**	**Desno**	**Dolje**	**Lijevo**	**TC[0]**	**TC[1]**	**TC[2]**	**TC[3]**
A	B	/	/	/	1	0	0	0
B	/	C	A	F	0	1	1	2
C	/	/	D	B	0	0	1	1
D	C	/	/	E	1	0	0	1
E	F	D	/	G	2	1	0	1
F	/	B	E	/	0	2	2	2
G	/	E	H	/	0	1	1	0
H	G	**I**	/	/	1	**1**	0	0
I	/	/	/	**H**	0	0	0	**1**



Slika 4.16.

Tablica 4.15.

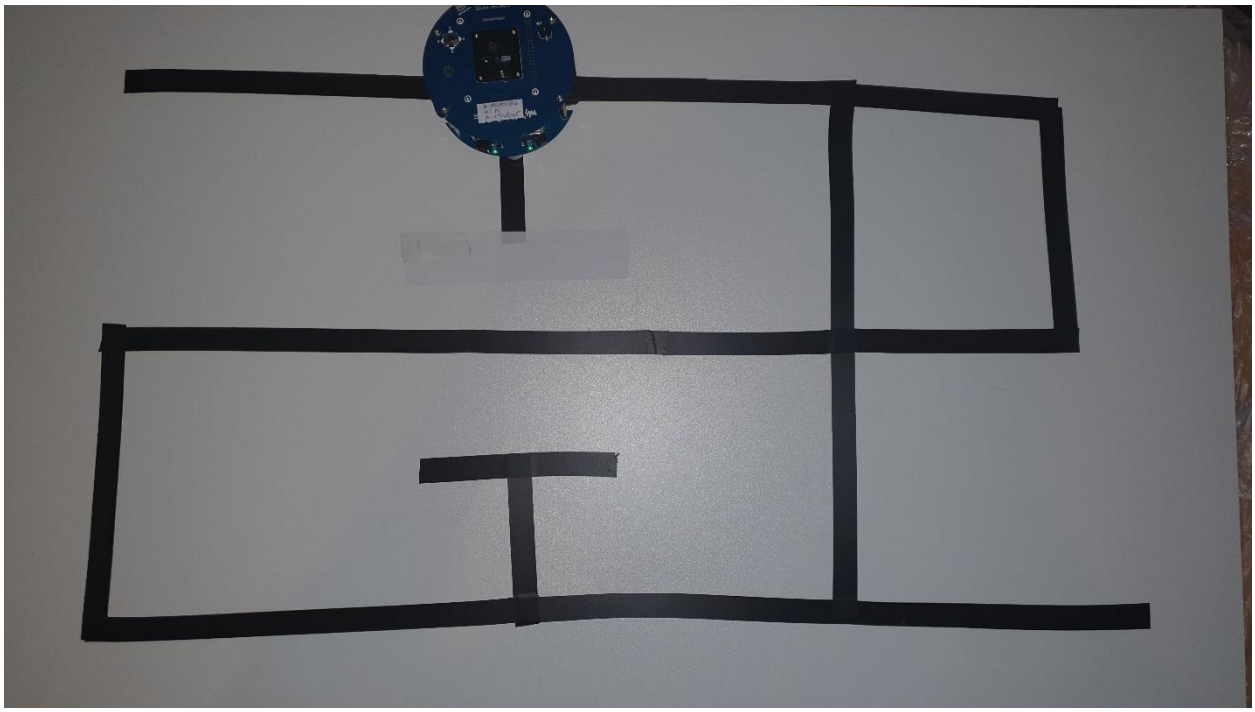
„corner“	**Gore**	**Desno**	**Dolje**	**Lijevo**	**TC[0]**	**TC[1]**	**TC[2]**	**TC[3]**
A	B	/	/	/	1	0	0	0
B	/	C	A	F	0	1	1	2
C	/	/	D	B	0	0	1	1
D	C	/	/	E	1	0	0	1
E	F	D	/	G	2	1	0	1
F	/	B	E	/	0	2	2	2
G	/	E	H	/	0	1	1	0
H	G	I	/	/	1	1	0	0
I	/	**J**	/	H	0	**1**	0	1
J	/	/	/	**I**	0	0	0	**1**



Slika 4.17.

Tablica 4.16.

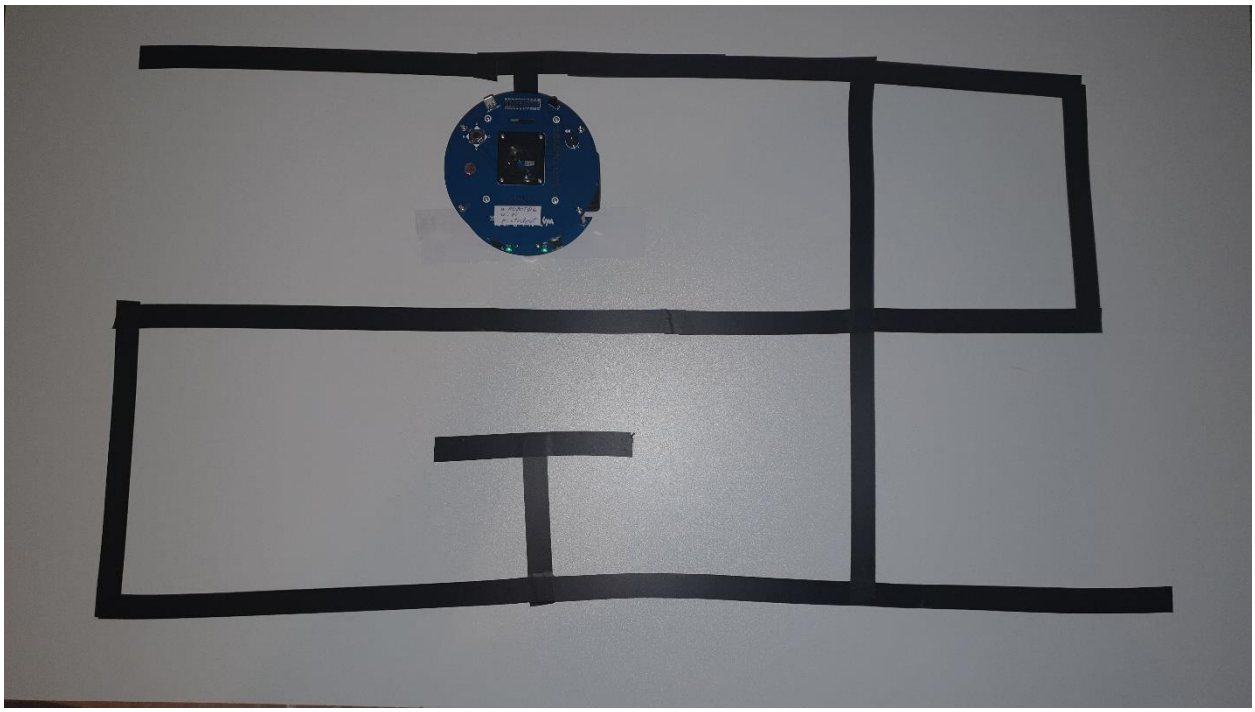
„corner“	**Gore**	**Desno**	**Dolje**	**Lijevo**	**TC[0]**	**TC[1]**	**TC[2]**	**TC[3]**
A	B	/	/	/	1	0	0	0
B	/	C	A	F	0	1	1	2
C	/	/	D	B	0	0	1	1
D	C	/	/	E	1	0	0	1
E	F	D	/	G	2	1	0	1
F	/	B	E	/	0	2	2	2
G	/	E	H	/	0	1	1	0
H	G	I	/	/	1	1	0	0
I	/	J	/	H	0	1	0	1
J	/	/	/	I	0	**1**	0	1



Slika 4.18.

Tablica 4.17.

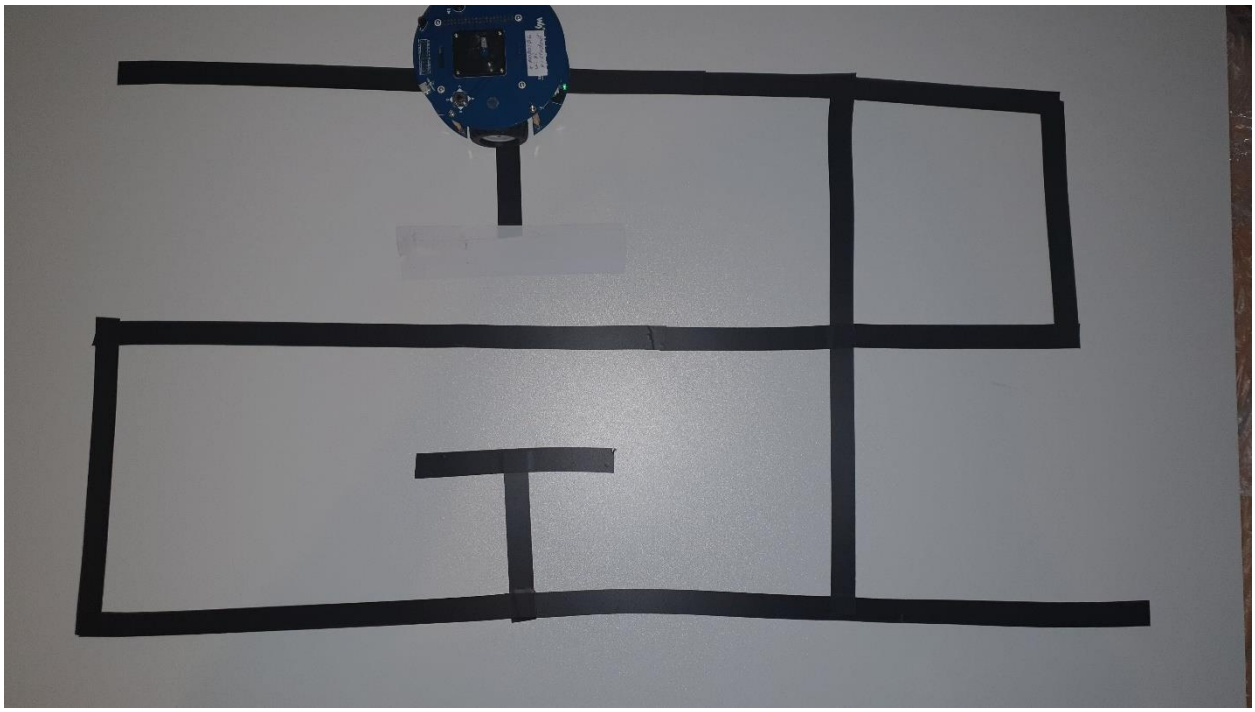
„corner“	**Gore**	**Desno**	**Dolje**	**Lijevo**	**TC[0]**	**TC[1]**	**TC[2]**	**TC[3]**
A	B	/	/	/	1	0	0	0
B	/	C	A	F	0	1	1	2
C	/	/	D	B	0	0	1	1
D	C	/	/	E	1	0	0	1
E	F	D	/	G	2	1	0	1
F	/	B	E	/	0	2	2	2
G	/	E	H	/	0	1	1	0
H	G	I	/	/	1	1	0	0
I	/	J	/	H	0	1	0	1
J	/	/	/	I	0	**2**	0	1



Slika 4.19.

Tablica 4.18.

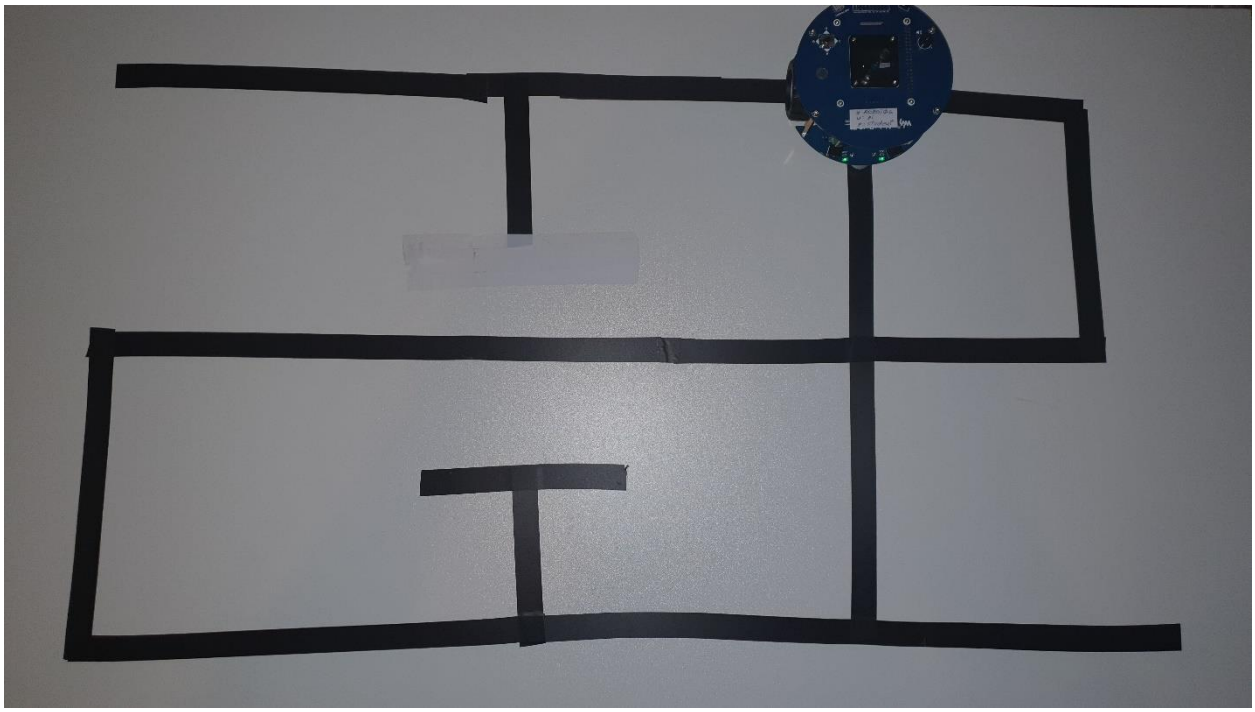
„corner“	**Gore**	**Desno**	**Dolje**	**Lijevo**	**TC[0]**	**TC[1]**	**TC[2]**	**TC[3]**
A	B	/	/	/	1	0	0	0
B	/	C	A	F	0	1	1	2
C	**/**	/	D	B	0	0	1	1
D	C	/	/	E	1	0	0	1
E	F	D	/	G	2	1	0	1
F	/	B	E	/	0	2	2	2
G	/	E	H	/	0	1	1	0
H	G	I	/	/	1	1	0	0
I	/	J	/	H	0	1	0	1
J	**K**	/	/	I	**1**	2	0	1
K	/	/	**I**	/	0	0	**1**	0



Slika 4.20.

Tablica 4.19.

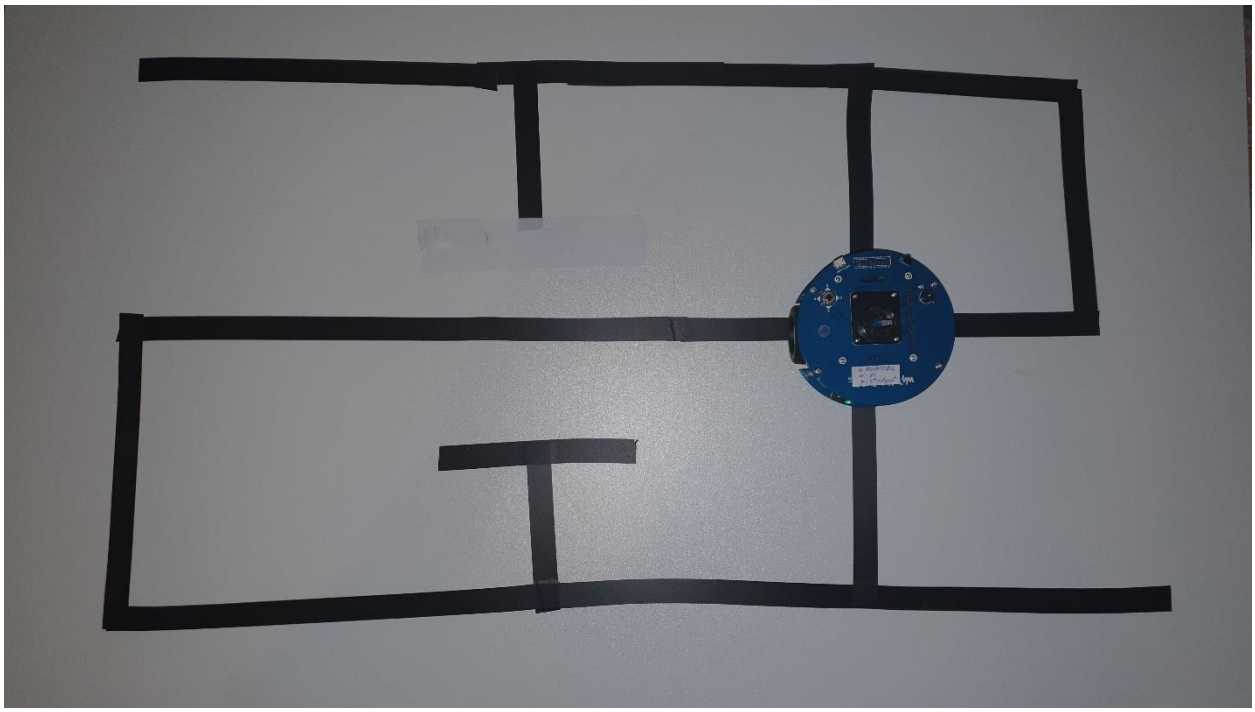
„corner“	**Gore**	**Desno**	**Dolje**	**Lijevo**	**TC[0]**	**TC[1]**	**TC[2]**	**TC[3]**
A	B	/	/	/	1	0	0	0
B	/	C	A	F	0	1	1	2
C	/	/	D	B	0	0	1	1
D	C	/	/	E	1	0	0	1
E	F	D	/	G	2	1	0	1
F	/	B	E	/	0	2	2	2
G	/	E	H	/	0	1	1	0
H	G	I	/	/	1	1	0	0
I	/	J	/	H	0	1	0	1
J	K	/	/	I	**2**	2	0	1
K	/	/	I	/	0	0	**2**	0



Slika 4.21.

Tablica 4.20.

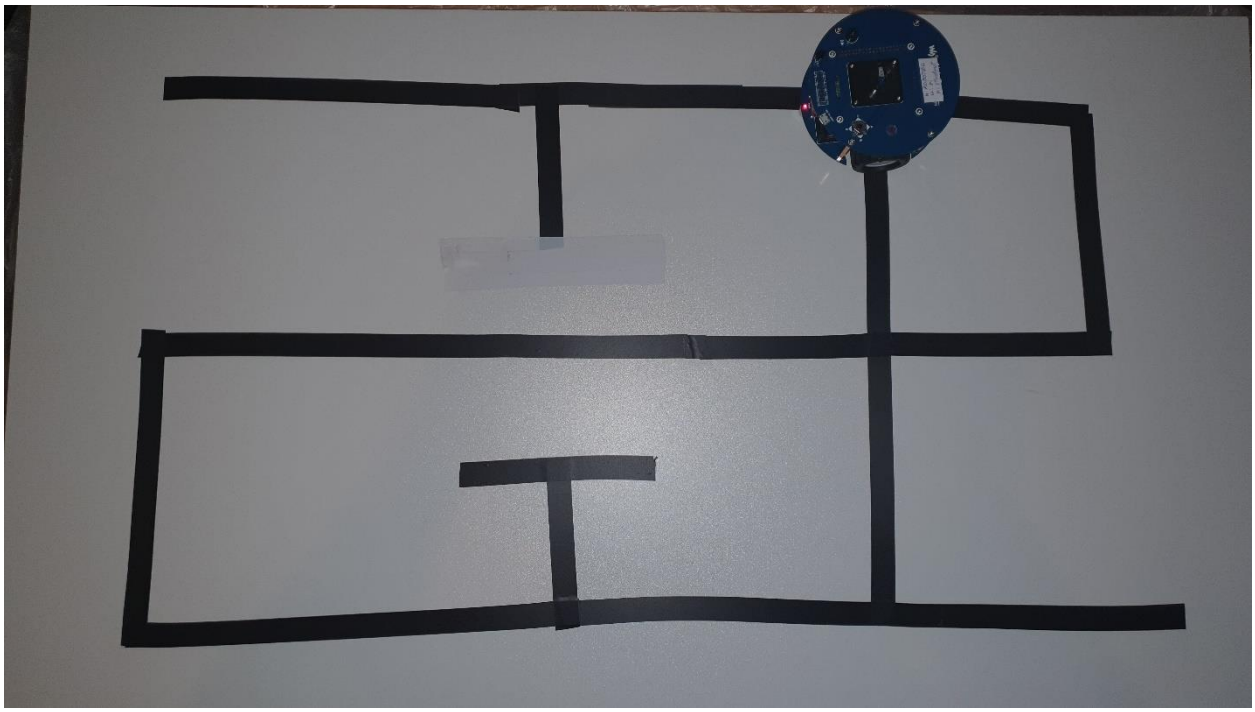
„corner“	**Gore**	**Desno**	**Dolje**	**Lijevo**	**TC[0]**	**TC[1]**	**TC[2]**	**TC[3]**
A	B	/	/	/	1	0	0	0
B	/	C	A	F	0	1	1	2
C	/	/	D	B	0	0	1	1
D	C	/	/	E	1	0	0	1
E	F	D	/	G	2	1	0	1
F	/	B	E	/	0	2	2	2
G	/	E	H	/	0	1	1	0
H	G	I	/	/	1	1	0	0
I	/	J	/	H	0	**2**	0	1
J	K	/	/	I	2	2	0	**2**
K	/	/	I	/	0	0	2	0



Slika 4.22.

Tablica 4.21.

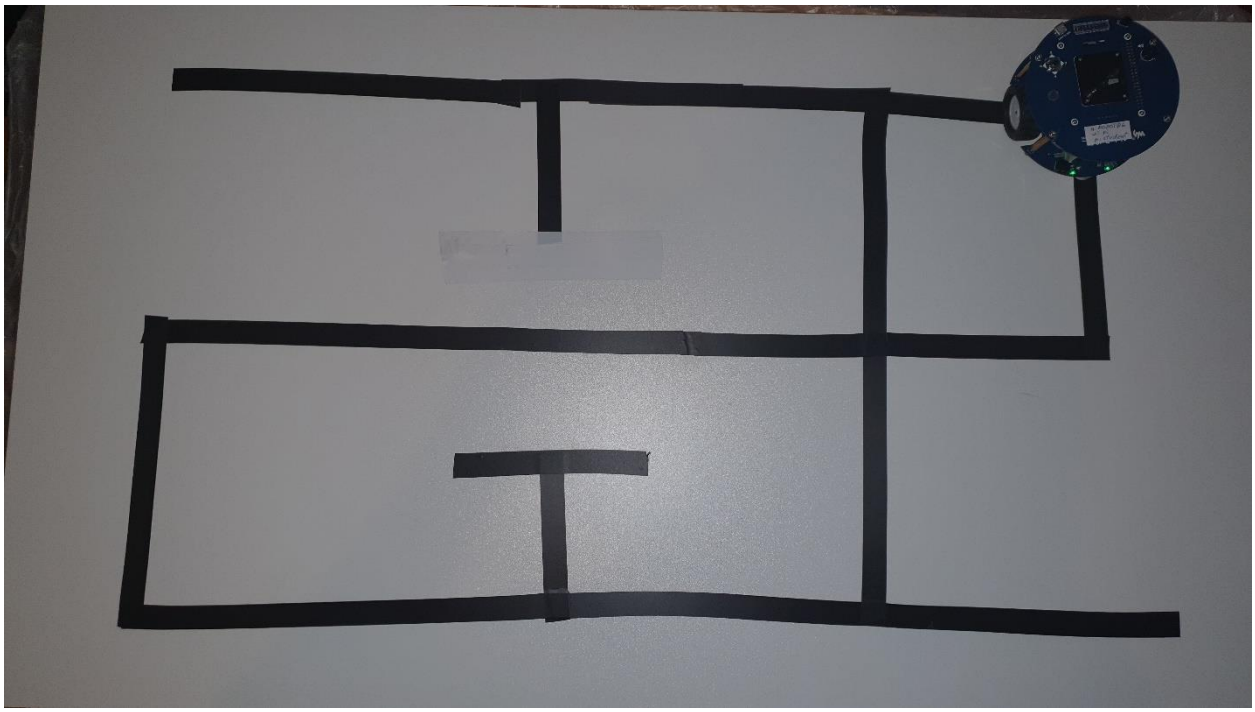
„corner“	**Gore**	**Desno**	**Dolje**	**Lijevo**	**TC[0]**	**TC[1]**	**TC[2]**	**TC[3]**
A	B	/	/	/	1	0	0	0
B	/	C	A	F	0	1	1	2
C	**/**	/	D	B	0	0	1	1
D	C	/	/	E	1	0	0	1
E	F	D	**I**	G	2	1	**1**	1
F	/	B	E	/	0	2	2	2
G	/	E	H	/	0	1	1	0
H	G	I	/	/	1	1	0	0
I	**E**	J	/	H	**1**	2	0	1
J	K	/	/	I	2	2	0	2
K	/	/	I	/	0	0	2	0



Slika 4.23.

Tablica 4.22.

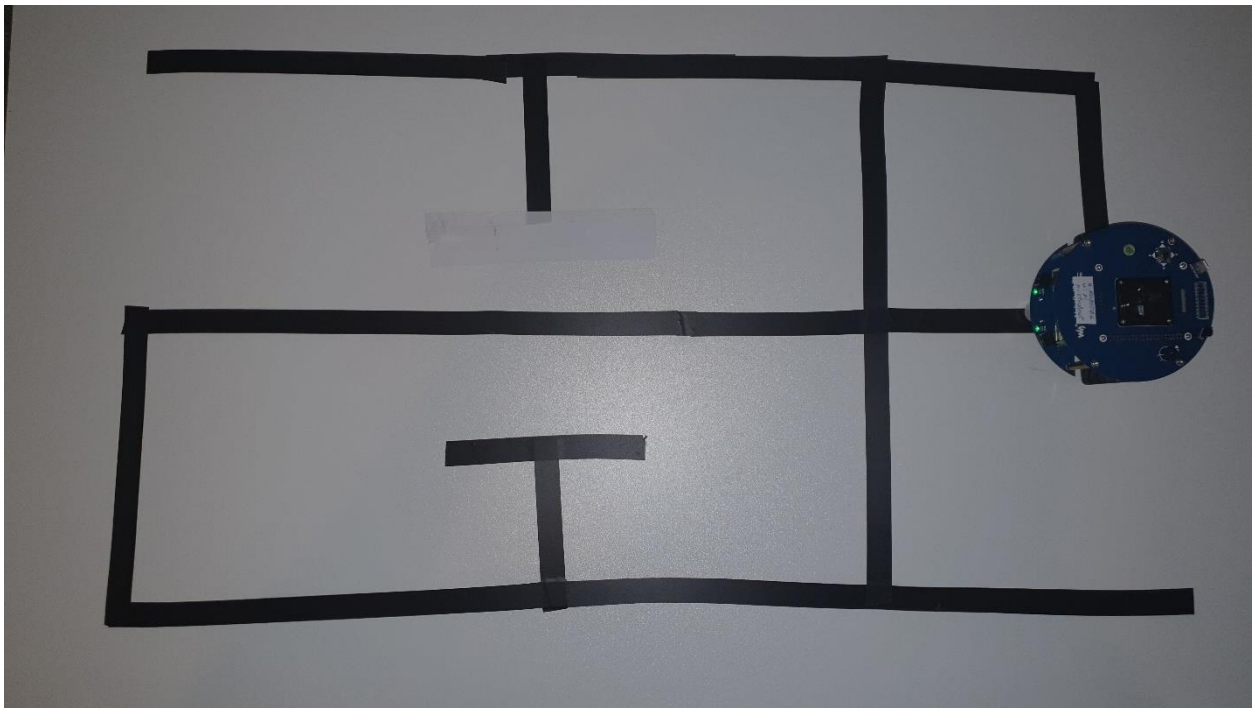
„corner“	**Gore**	**Desno**	**Dolje**	**Lijevo**	**TC[0]**	**TC[1]**	**TC[2]**	**TC[3]**
A	B	/	/	/	1	0	0	0
B	/	C	A	F	0	1	1	2
C	/	/	D	B	0	0	1	1
D	C	/	/	E	1	0	0	1
E	F	D	I	G	2	1	**2**	1
F	/	B	E	/	0	2	2	2
G	/	E	H	/	0	1	1	0
H	G	I	/	/	1	1	0	0
I	E	J	/	H	**2**	2	0	1
J	K	/	/	I	2	2	0	2
K	/	/	I	/	0	0	2	0



Slika 4.24.

Tablica 4.23.

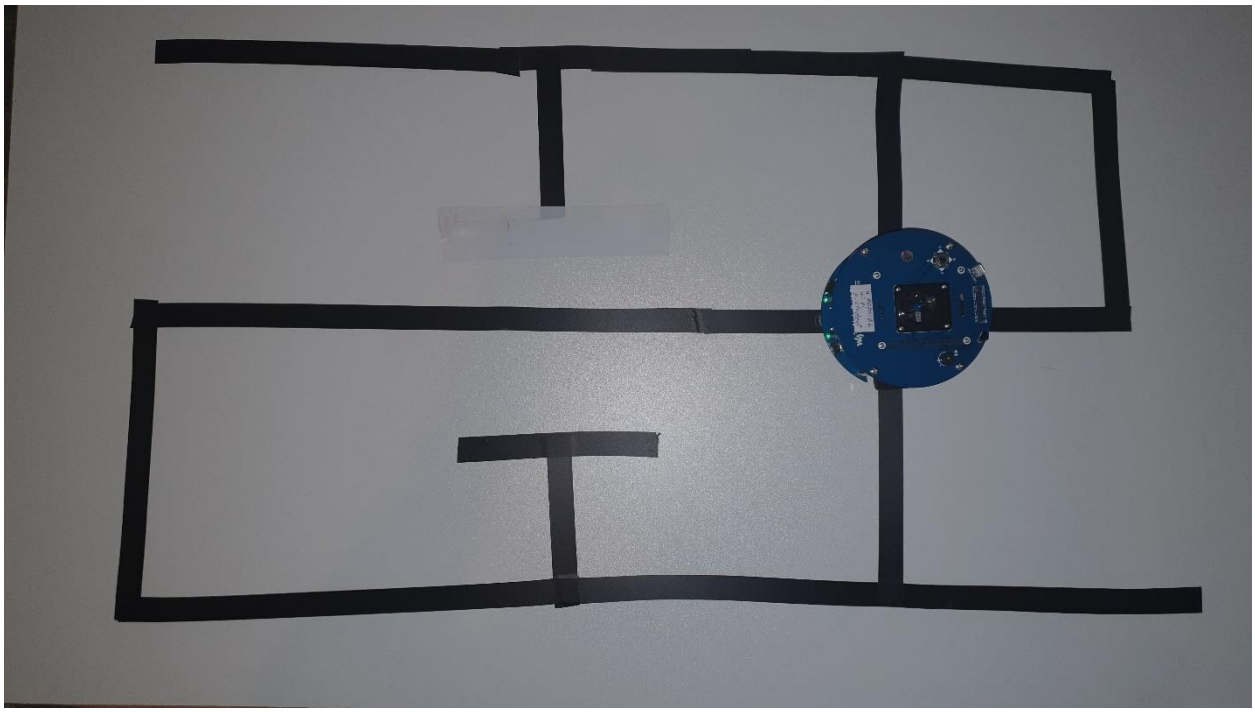
„corner“	**Gore**	**Desno**	**Dolje**	**Lijevo**	**TC[0]**	**TC[1]**	**TC[2]**	**TC[3]**
A	B	/	/	/	1	0	0	0
B	/	C	A	F	0	1	1	2
C	/	/	D	B	0	0	1	1
D	C	/	/	E	1	0	0	1
E	F	D	I	G	2	1	2	1
F	/	B	E	/	0	2	2	2
G	/	E	H	/	0	1	1	0
H	G	I	/	/	1	**2**	0	0
I	E	J	/	H	2	2	0	**2**
J	K	/	/	I	2	2	0	2
K	/	/	I	/	0	0	2	0



Slika 4.25.

Tablica 4.24.

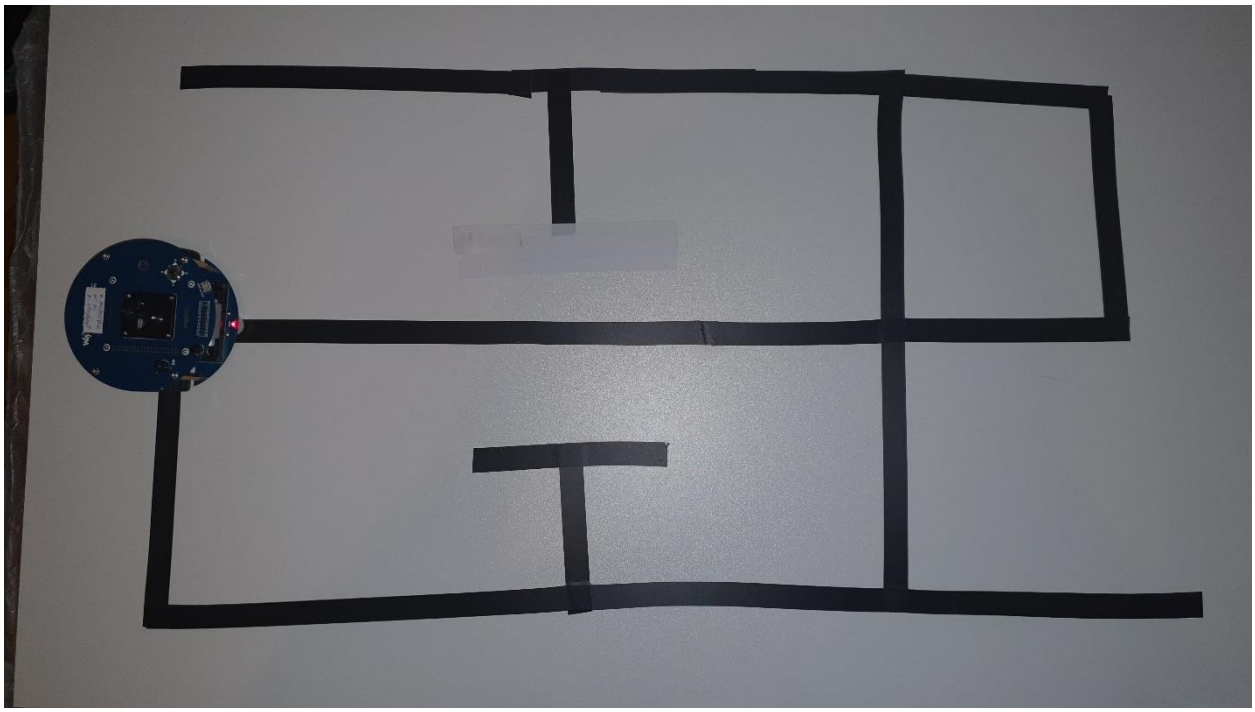
„corner“	**Gore**	**Desno**	**Dolje**	**Lijevo**	**TC[0]**	**TC[1]**	**TC[2]**	**TC[3]**
A	B	/	/	/	1	0	0	0
B	/	C	A	F	0	1	1	2
C	/	/	D	B	0	0	1	1
D	C	/	/	E	1	0	0	1
E	F	D	I	G	2	1	2	1
F	/	B	E	/	0	2	2	2
G	/	E	H	/	0	1	**2**	0
H	G	I	/	/	**2**	2	0	0
I	E	J	/	H	2	2	0	2
J	K	/	/	I	2	2	0	2
K	/	/	I	/	0	0	2	0



Slika 4.26.

Tablica 4.25.

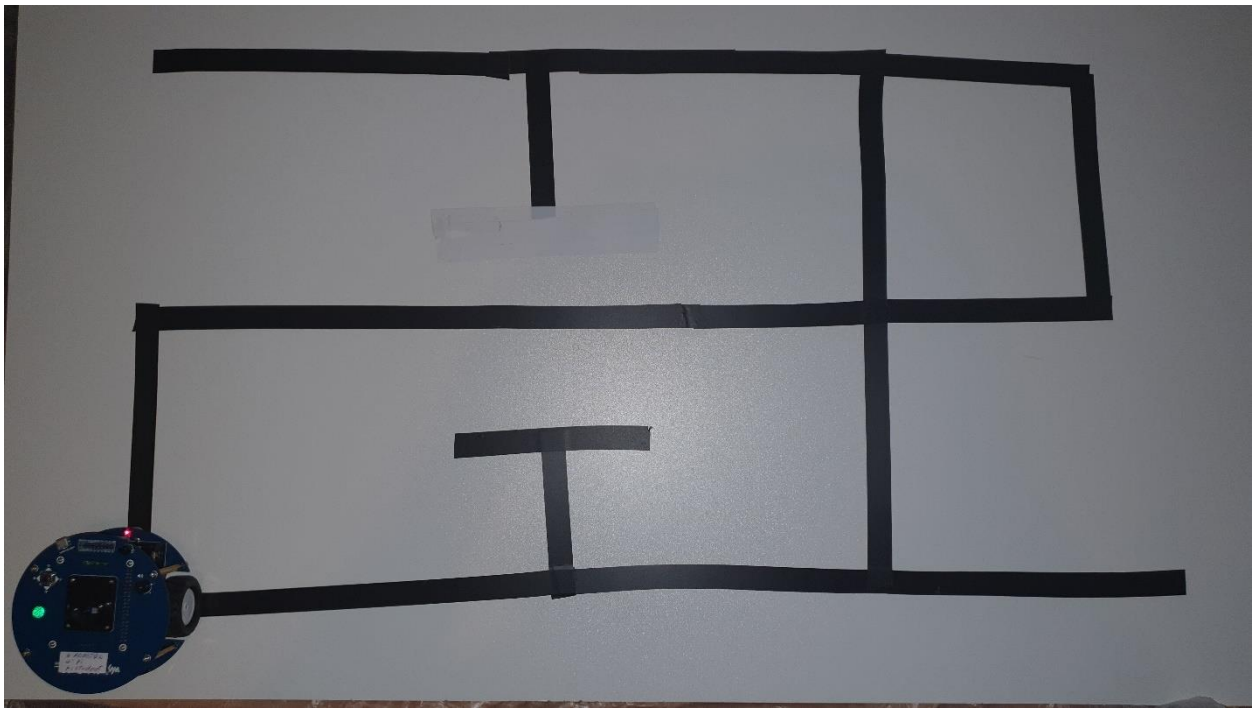
„corner“	**Gore**	**Desno**	**Dolje**	**Lijevo**	**TC[0]**	**TC[1]**	**TC[2]**	**TC[3]**
A	B	/	/	/	1	0	0	0
B	/	C	A	F	0	1	1	2
C	/	/	D	B	0	0	1	1
D	C	/	/	E	1	0	0	1
E	F	D	I	G	2	1	2	**2**
F	/	B	E	/	0	2	2	2
G	/	E	H	/	0	**2**	2	0
H	G	I	/	/	2	2	0	0
I	E	J	/	H	2	2	0	2
J	K	/	/	I	2	2	0	2
K	/	/	I	/	0	0	2	0



Slika 4.27.

Tablica 4.26.

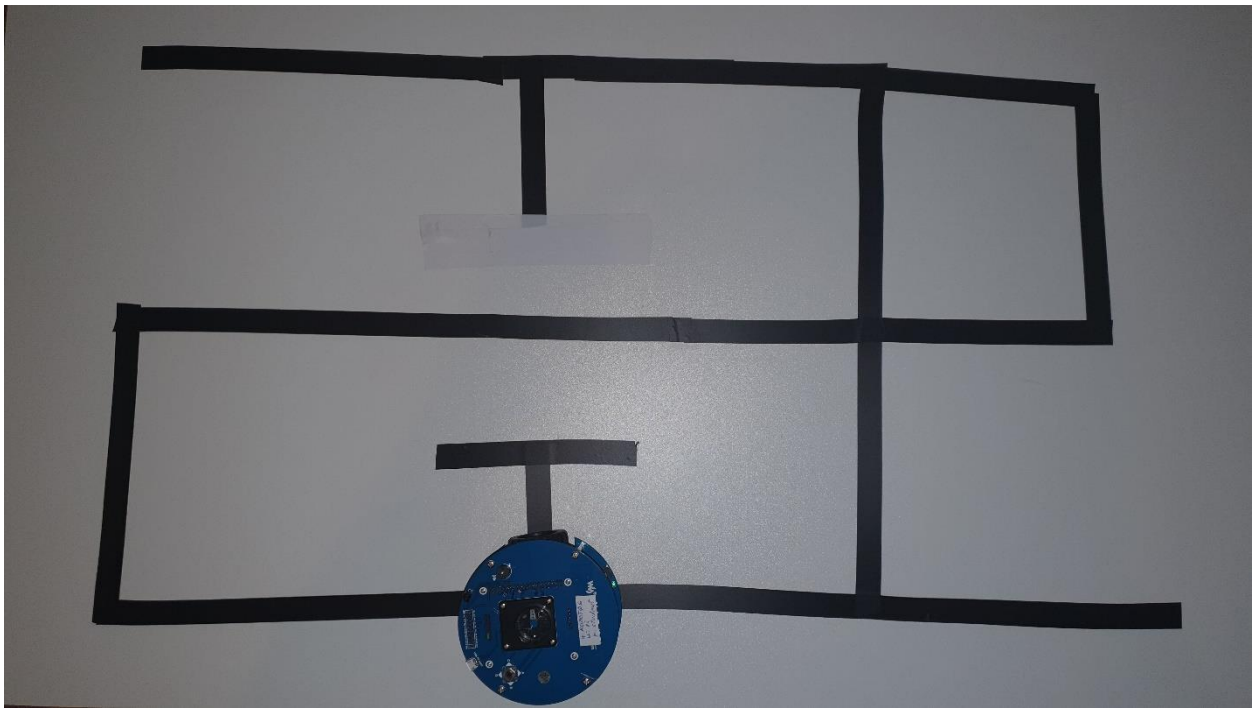
„corner“	**Gore**	**Desno**	**Dolje**	**Lijevo**	**TC[0]**	**TC[1]**	**TC[2]**	**TC[3]**
A	B	/	/	/	1	0	0	0
B	/	C	A	F	0	1	1	2
C	/	/	D	B	0	0	1	1
D	C	/	/	E	1	0	0	**2**
E	F	D	I	G	2	**2**	2	2
F	/	B	E	/	0	2	2	2
G	/	E	H	/	0	2	2	0
H	G	I	/	/	2	2	0	0
I	E	J	/	H	2	2	0	2
J	K	/	/	I	2	2	0	2
K	/	/	I	/	0	0	2	0



Slika 4.28.

Tablica 4.27.

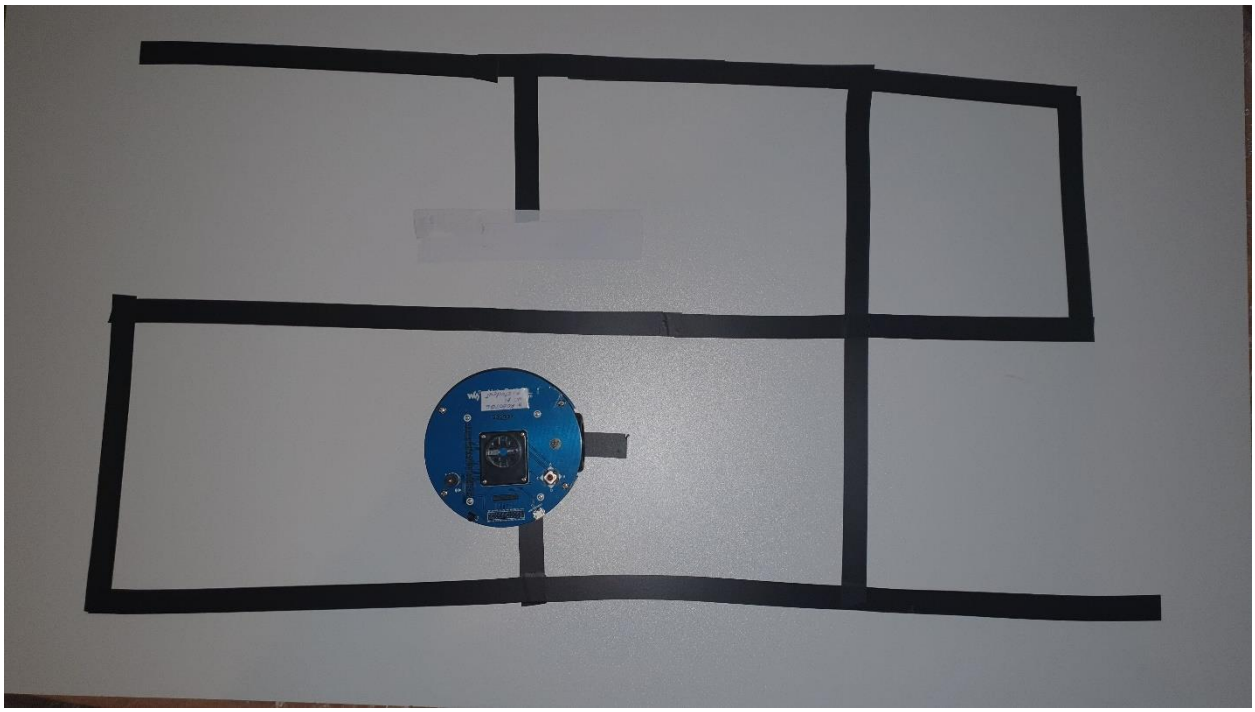
„corner“	**Gore**	**Desno**	**Dolje**	**Lijevo**	**TC[0]**	**TC[1]**	**TC[2]**	**TC[3]**
A	B	/	/	/	1	0	0	0
B	/	C	A	F	0	1	1	2
C	**/**	/	D	B	0	0	**2**	1
D	C	/	/	E	**2**	0	0	2
E	F	D	I	G	2	2	2	2
F	/	B	E	/	0	2	2	2
G	/	E	H	/	0	2	2	0
H	G	I	/	/	2	2	0	0
I	E	J	/	H	2	2	0	2
J	K	/	/	I	2	2	0	2
K	/	/	I	/	0	0	2	0



Slika 4.29.

Tablica 4.28

„corner“	**Gore**	**Desno**	**Dolje**	**Lijevo**	**TC[0]**	**TC[1]**	**TC[2]**	**TC[3]**
A	B	/	/	/	1	0	0	0
B	/	C	A	F	0	**2**	1	2
C	/	/	D	B	0	0	2	**2**
D	C	/	/	E	2	0	0	2
E	F	D	I	G	2	2	2	2
F	/	B	E	/	0	2	2	2
G	/	E	H	/	0	2	2	0
H	G	I	/	/	2	2	0	0
I	E	J	/	H	2	2	0	2
J	K	/	/	I	2	2	0	2
K	/	/	I	/	0	0	2	0



Slika 4.30.

Tablica 4.29.

„corner“	**Gore**	**Desno**	**Dolje**	**Lijevo**	**TC[0]**	**TC[1]**	**TC[2]**	**TC[3]**
A	B	/	/	/	**2**	0	0	0
B	/	C	A	F	0	2	**2**	2
C	/	/	D	B	0	0	2	2
D	C	/	/	E	2	0	0	2
E	F	D	I	G	2	2	2	2
F	/	B	E	/	0	2	2	2
G	/	E	H	/	0	2	2	0
H	G	I	/	/	2	2	0	0
I	E	J	/	H	2	2	0	2
J	K	/	/	I	2	2	0	2
K	/	/	I	/	0	0	2	0

Sljedeće tablice od 4.30. do 4.38. su prikaz Dijsktrinog algoritma za pronalazak najkraćeg puta. Kao što je prije spomenuto varijabla „node“ sadrži atribute „corner“, „prevCorner“ i „distance“. Atribut „corner“ je trenutno raskrižje varijable „node“, „prevCorner“ je raskrižje kojim se došlo do tog trenutnog raskrižja. Atribut „distance“ je udaljenost tog raskrižja od početka. Nakon što

algoritam dođe do raskrižja označenog kao cilj tada staje. Zatim najkraći put se dobije tako da se dođe od cilja do početka prateći prethodna raskrižaja i pohrane se. Iz tablice 4.38. primjetan je najkraći put: A, B, F, E, I, J, K.

Tablica 4.30.

„node“	**„corner“**	**„prevCorner“**	**„distance“**
1.	A	/	0

Tablica 4.31.

„node“	**„corner“**	**„prevCorner“**	**„distance“**
1.	A	/	0
2.	B	A	210

Tablica 4.32.

„node“	**„corner“**	**„prevCorner“**	**„distance**
1.	A	/	0
2.	B	A	210
3.	C	B	903
4.	F	B	714

Tablica 4.33.

„node“	**„corner“**	**„prevCorner“**	**„distance“**
1.	A	/	0
2.	B	A	210
3.	C	B	903
4.	F	B	714
5.	E	F	1155

Tablica 4.34.

„node“	**„corner“**	**„prevCorner“**	**„distance“**
1.	A	/	0
2.	B	A	210
3.	C	B	903
4.	F	B	714
5.	E	F	1155
6.	G	E	1512
7.	I	E	1554

Tablica 4.35.

„node“	**„corner“**	**„prevCorner“**	**„distance“**
1.	A	/	0
2.	B	A	210
3.	C	B	903
4.	F	B	714
5.	E	F	1155
6.	G	E	1512
7.	I	E	1554
8.	H	G	1890

Tablica 4.36.

„node“	**„corner“**	**„prevCorner“**	**„distance“**
1.	A	/	0
2.	B	A	210
3.	C	B	903
4.	F	B	714
5.	E	F	1155
6.	G	E	1512
7.	I	E	1554
8.	H	G	1890
9.	J	I	2100

Tablica 4.37.

„node“	**„corner“**	**„prevCorner“**	**„distance“**
1.	A	/	0
2.	B	A	210
3.	C	B	903
4.	F	B	714
5.	E	F	1155
6.	G	E	1512
7.	I	E	1554
8.	H	G	1890
9.	J	I	2100

Tablica 4.38.

„node“	**„corner“**	**„prevCorner“**	**„distance“**
1.	A	/	0
2.	B	A	210
3.	C	B	903
4.	F	B	714
5.	E	F	1155
6.	G	E	1512
7.	I	E	1554
8.	H	G	1890
9.	J	I	2100
10.	K	J	2646

Zbog ljudske pogreške pri izradi ravnih linija labirinta i razlike u snazi motora u ovisnosti o postotku baterije, robot nije sve labirinte riješio iz prvog pokušaja. Raznim pokušajima ustanovljeno je da pri jačini motora robota postavljena na 20%, te razina baterije je između 20% i 70% robot radi minimalne pogreške.

5. ZAKLJUČAK

Postoje razni algoritmi za prolazak kroz labirint, te za pronalaženje najkraćeg puta već istraženog labirinta. Zbog manjka radova i istraživanja o mapiranju labirinta u ovom završnom radu je primijenjen Trémauxov algoritam za prolazak kroz labirint, te je isti modificiran tako da istraži cijeli labirint. Za pronalazak najkraćeg puta istraženog labirinta koristio se Dijsktrin algoritam. U radu su objašnjeni i primijenjeni prije spomenuti algoritmi. Slikama i tablicama prikazan je cjelokupno rješavanje jednog labirinta. Pri testiranju tih algoritama uočeni su i opisani problemi koji nisu bili uračunati tijekom izrade rada. Za unapređenje efikasnosti rada i poboljšanje primjenjivosti istog preporuča se veća podrobnost pri implementaciji kretanja robota, te eventualna implementacija efikasnijih algoritama za pronalaženje najkraćeg puta.

LITERATURA

[1] Hollis Williams, The Mathematics of Mazes, 2020., dostupno na https://www.researchgate.net/publication/343615560_The_Mathematics_of_Mazes [pristupljeno: 5.9.2021.]

[2] Walter D. Pullen, Maze Classification, 2021., dostupno na https://www.astrolog.org/labyrnth/algrithm.htm [pristupljeno: 5.9.2021.]

[3] Raspberry Pi, Operating system images, 2021., dostupno na https://www.raspberrypi.org/software/operating-systems/ [pristupljeno: 4.9.2021.]

[4] Raspberry Pi, Raspberry Pi Documentation, 2021., dostupno na https://www.raspberrypi.org/documentation/computers/os.html#python [pristupljeno: 4.9.2021.]

[5] Waveshare, Alphabot2 robot building kit for Raspberry Pi 3 Model B, 2021., dostupno na https://www.waveshare.com/alphabot2-pi.htm [pristupljeno: 7.9.2021.]

[6] E. Lucas, Récréations Mathématiques Volume I, Gauthier-Villars, Francuska, 1882.

[7] P. Deuflhard, A. Hohmann, A note on two problem in connexion with graphs, Numerische Mathematik, 1, 1, 269–271, Prosinac 1959.

[8] T. Cormen, C. Leiserson, R. Rivest, C. Stein, Dijkstra's algorithm, MIT Press, Sjedinjene Američke Države, 1990.

[9] D. Kuhlman, A Python Book: Beginning Python, Advanced Python, and Python Exercises, Platypus Global Media, Sjedinjene Američke Države, 2011.

[10] G. van Rossum, The History of Python: A Brief Timeline of Python, 2009., dostupno na https://python-history.blogspot.com/2009/01/brief-timeline-of-python.html [pristupljeno: 10.9.2021.]

[11] T. Peters, PEP 20 – The Zen of Pyrhon, 2004., dostupno na https://www.python.org/dev/peps/pep-0020/ [pristupljeno: 10.9.2021.]

SAŽETAK

Zadatak završnog rada je implantacija algoritama za mapiranje i pronalaska najkraćeg puta u labirintu. U radu je korišten Alphabot2 robot i implementirana je programska podrška napisana u programskom jeziku Python. U teorijskom dijelu rada opisano je što je labirint, te uvod u algoritme obilaska i pronalaska najkraćeg puta labirinta. Zatim je opisan Alphabot2 robot, njegova oprema, specifikacije i raznovrsni senzori. Nakon toga su objašnjeni algoritmi korišteni za izradu rada i programski jezik Python. U drugom dijelu prikazana je primjena tih algoritama za postizanje mapiranja i pronalaska najkraćeg puta. Na kraju je objašnjena izrada labirinta, te je prokomentirana uspješnost i kvaliteta programa.

Ključne riječi: Alphabot2, Dijkstra, labirint, mapiranje, najkraći put, robot, senzori, Trémaux

ABSTRACT

The task of this final dissertation is the implantation of mapping algorithms and finding the shortest path in a maze. The Alphabot2 robot was used in the work and the software was implemented in the Python programming language. The theoretical part of the paper describes what a labyrinth is, as well as making an introduction to algorithms for visiting and finding the shortest path in a labyrinth. Next, the Alphabot2 robot, its equipment, specifications and various sensors are described. The algorithms used in the paper and the Python programming language are then explained. The second part shows the application of these algorithms to achieve mapping and finding the shortest path. And finally, the makings of the labyrinth are explained, and the success and quality of the program are observed.

Keywords: Alphabot2, Dijkstra, maze, mapping, shortest path, robot, sensors, Trémaux

TITLE

Software to master the maze on the Raspbery Pi mobile robotic platform

ŽIVOTOPIS

Marko Budimir rođen je 18.06.1999. godine u Vukovaru. Nakon završene osnovne škole Tordinci, 2014. godine upisuje „Tehničkoj školi Ruđera Boškovića“ u Vinkovcima. 2018. godine upisuje Fakultet elektrotehnike, računarstva i informacijskih tehnologija u Osijeku.

PRILOZI

Prilog 1. Izvorni kod aplikacije:

MazeMovement.py

```
import RPi.GPIO as GPIO
from AlphaBot2 import AlphaBot2
from TRSensors import TRSensor

class MazeMovement(object):
        def __init__(self, Ab, TR, speed = 25):
                self.TR = TR
                self.Ab = Ab
                self.maximum = speed
                self.integral = 0
                self.last_proportional = 0
                self.Ab.setPWMA(speed)
                self.Ab.setPWMB(speed)


        def Forward(self):
                self.Ab.setPWMA(self.maximum)
                self.Ab.setPWMB(self.maximum)
                distance = 0
                while(True):
                        position,Sensors = self.TR.readLine()
                        if(Sensors[0] > 650 or Sensors[4] > 650 or
                                (Sensors[0] < 300 and Sensors[1] < 300 and Sensors[2] < 300 and S
ensors[3] < 300 and Sensors[4] < 300)):
                                self.Ab.stop()
                                self.Ab.setPWMA(0)
                                self.Ab.setPWMB(0)
                                return distance
                        self.Ab.forward()
                        distance +=1
                        proportional = position - 2000

                        derivative = proportional - self.last_proportional
                        self.integral += proportional

                        self.last_proportional = proportional

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

                        if (power_difference > self.maximum):
                                power_difference = self.maximum
                        if (power_difference < - self.maximum):
                                power_difference = - self.maximum
```

```
                if (power_difference < 0):
                        self.Ab.setPWMA(self.maximum + power_difference)
                        self.Ab.setPWMB(self.maximum)
                else:
                        self.Ab.setPWMA(self.maximum)
                        self.Ab.setPWMB(self.maximum - power_difference)


def Right(self):
        self.Ab.setPWMA(20)
        self.Ab.setPWMB(20)
        for i in range(0, 1000):
                position,Sensors = self.TR.readLine()
                self.Ab.right()
                if(i > 150 and Sensors[2] > 250):
                        self.Ab.stop()
                        self.Ab.setPWMA(0)
                        self.Ab.setPWMB(0)
                        break

def Left(self):
        self.Ab.setPWMA(20)
        self.Ab.setPWMB(20)
        for i in range(0, 1000):
                position,Sensors = self.TR.readLine()
                self.Ab.left()
                if(i > 150 and Sensors[2] > 250):
                        self.Ab.stop()
                        self.Ab.setPWMA(0)
                        self.Ab.setPWMB(0)
                        break

def Turn180(self):
        self.Ab.setPWMA(20)
        self.Ab.setPWMB(20)
        for i in range(0, 1000):
                position,Sensors = self.TR.readLine()
                self.Ab.right()
                if(i > 400 and Sensors[2] > 250):
                        self.Ab.stop()
                        self.Ab.setPWMA(0)
                        self.Ab.setPWMB(0)
                        break

def DeadEndTurn(self):
        position,Sensors = self.TR.readLine()
        while(Sensors[2] < 250):
                position,Sensors = self.TR.readLine()
                self.Ab.setPWMA(20)
```

```
                        self.Ab.setPWMB(20)
                        self.Ab.right()
                self.Ab.stop()
                self.Ab.setPWMA(0)
                self.Ab.setPWMB(0)

        def Calibrate(self):
                for i in range(0,100):
                        if(i<25 or i>= 75):
                                self.Ab.right()
                                self.Ab.setPWMA(30)
                                self.Ab.setPWMB(30)
                        else:
                                self.Ab.left()
                                self.Ab.setPWMA(30)
                                self.Ab.setPWMB(30)
                        self.TR.calibrate()
                self.Ab.stop()
```

Tremaux.py

```
import RPi.GPIO as GPIO
from AlphaBot2 import AlphaBot2
from TRSensors import TRSensor
from MazeMovement import MazeMovement
from Corner import Corner
import time
from Points import Points
from Dijkstra import Dijkstra

Button = 7
right = -1
left = -1
forward = -1

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(Button,GPIO.IN,GPIO.PUD UP)

while (GPIO.input(Button) != 0):
        time.sleep(0.05)

TR = TRSensor()
Ab = AlphaBot2()
Movement = MazeMovement( Ab, TR, 25)
time.sleep(0.5)
Movement.Calibrate()
time.sleep(3)
```

```
while (GPIO.input(Button) != 0):
        time.sleep(0.05)

start = Corner()
start.SetPostion(1)
dijkstra = Dijkstra(start)
points = Points()
corner = start
distance = 0
direction = -1
orientation = 0
points.AddPoint(corner, distance, orientation, start)

while True:
        position,Sensors = TR.readLine()
        if(orientation > 3):
                orientation -= 4
        elif(orientation < 0):
                orientation +=4
        if(Sensors[0] <300 and Sensors[1] < 300 and Sensors[2] < 300 and Sensors[3] < 300 and Sen
sors[4] < 300 and
                Sensors[0] > 100 and Sensors[1] > 100 and Sensors[2] > 100 and Sensors[3] > 100 a
nd Sensors[4] > 100):
                prev = corner
                corner = Corner(right, left, forward, orientation)
                points.AddPoint(corner, distance, orientation, prev)
                corner.SetDistance(distance, orientation, prev)
                corner.SetPoint(prev, orientation)
                corner.SetPostion(2)
                orientation+=2
                if(orientation > 3):
                        orientation -= 4
                elif(orientation < 0):
                        orientation +=4
                Ab.stop()
                Ab.setPWMA(0)
                Ab.setPWMB(0)
                time.sleep(0.5)
                Movement.Turn180()
                time.sleep(1)
                direction = ""
        if(right == 0 or left == 0 or forward == 0):
                check = points.CheckPoint(corner, distance, orientation)
                prev = corner
                if(check != None ):
                        corner = check
                        check = None
                        if(corner.position == 1):
```

```
                        Movement.Turn180()
                        orientation = 0
                        break
        else:
                corner = Corner(right, left, forward, orientation)
                points.AddPoint(corner, distance, orientation, prev)
        direction = corner.Choose(orientation)
        corner.SetDistance(distance, orientation, prev)
        corner.SetPoint(prev, orientation)
        right = -1
        left = -1
        forward = -1
if(direction == "right"):
        Ab.stop()
        Ab.setPWMA(0)
        Ab.setPWMB(0)
        time.sleep(0.5)
        Movement.Right()
        direction = ""
        time.sleep(3)
        orientation +=1
elif(direction == "forward"):
        Ab.stop()
        Ab.setPWMA(0)
        Ab.setPWMB(0)
        time.sleep(0.5)
        direction = ""
        time.sleep(3)
elif(direction == "left"):
        Ab.stop()
        Ab.setPWMA(0)
        Ab.setPWMB(0)
        time.sleep(0.5)
        Movement.Left()
        direction = ""
        time.sleep(3)
        orientation -=1
elif(direction == "backward"):
        Ab.stop()
        Ab.setPWMA(0)
        Ab.setPWMB(0)
        time.sleep(0.5)
        Movement.Turn180()
        direction = ""
        time.sleep(3)
        orientation +=2
elif((Sensors[4] >650 or Sensors[0] >650)):
        Ab.stop()
        Ab.setPWMA(20)
```

```
                Ab.setPWMA(20)
                Ab.forward()
                time.sleep(0.04)
                Ab.setPWMA(0)
                Ab.setPWMB(0)
                Ab.stop()
                position,Sensors = TR.readLine()
                if((Sensors[4] + Sensors[3]) > 900):
                    right = 0
                if((Sensors[0] + Sensors[1]) > 900):
                    left = 0
                Ab.setPWMA(20)
                Ab.setPWMB(20)
                Ab.forward()
                time.sleep(0.18)
                Ab.stop()
                Ab.setPWMA(0)
                Ab.setPWMB(0)
                position,Sensors = TR.readLine()
                if((Sensors[1] + Sensors[2] + Sensors[3]) > 1000):
                    forward = 0
                time.sleep(3)
        elif(Sensors[0] < 100 and Sensors[1] < 100 and Sensors[2] < 100 and Sensors[3] < 100 and
Sensors[4] < 100):
                Ab.stop()
                Ab.setPWMA(0)
                Ab.setPWMB(0)
                time.sleep(3)
                Movement.DeadEndTurn()
                orientation+=2
                time.sleep(3)
                Movement.Forward()
                distance = 0
        else:
                newDistance = Movement.Forward()
                if(newDistance > 0):
                        distance = newDistance


corners =dijkstra.FindShortesPath()
corners.pop()
while True:
        Movement.Forward()
        Ab.stop()
        position,Sensors = TR.readLine()
        if(Sensors[0] <300 and Sensors[1] < 300 and Sensors[2] < 300 and Sensors[3] < 300 and Sen
sors[4] < 300 and
                Sensors[0] > 100 and Sensors[1] > 100 and Sensors[2] > 100 and Sensors[3] > 100 a
nd Sensors[4] > 100):
```

```
                break
        Ab.setPWMA(20)
        Ab.setPWMB(20)
        Ab.forward()
        time.sleep(0.18)
        Ab.stop()
        Ab.setPWMA(0)
        Ab.setPWMB(0)
        corner = corners.pop()
        nextCorner = corners.__getitem__(-1)
        direction = corner.GetDirection(orientation, nextCorner)
        if(direction == "Right"):
                time.sleep(0.5)
                Movement.Right()
                time.sleep(2)
                direction = ""
                orientation +=1
        elif(direction == "Forward"):
                direction = ""
                time.sleep(2)
        elif(direction == "Left"):
                time.sleep(0.5)
                Movement.Left()
                time.sleep(2)
                orientation -=1
        if(orientation > 3):
                orientation -= 4
        elif(orientation < 0):
                orientation +=4
```

Dijsktra.py

```
from Corner import Corner

class Node(object):
        def __init__(self, corner, distance, prev):
                self.corner = corner
                self.distance = distance
                self.prev = prev

class Nodes(object):
        def __init__(self):
                self.nodes = list()
                self.prevNode = None

        def AddNode(self, corner, distance = 0, prev = None):
                check = 1
                for node in self.nodes:
```

```
                        if(node.corner == corner):
                                if(distance < node.distance):
                                        node.distance = distance
                                        node.prev = prev
                                check = 0
                if(check):
                    self.nodes.append(Node(corner, distance, prev))

        def GetNextNode(self, node):
                if(node == None):
                        node = self.prevNode
                        node.distance = 99999
                distance = node.distance
                nextNode = node
                for n in self.nodes:
                        if(distance > n.distance and n.corner.position < 0):
                                distance = n.distance
                                nextNode = n
                self.prevNode = nextNode
                return nextNode

        def GetLastNode(self):
                return self.nodes.__getitem__(-1)

class Dijkstra(object):
        def __init__(self, start):
                self.start = start
                self.nodes = Nodes()
                self.nodes.AddNode(start)
                self.distance = 0

        def DijkstraAlgoritham(self):
                corner = self.start
                while(corner.position < 2):
                        node = self.ShortesDistance(corner)
                        node = self.nodes.GetNextNode(node)
                        self.distance = node.distance
                        corner = node.corner

        def ShortesDistance(self, corner):
                if(corner.position < 0):
                        corner.position = 0
                minDistance = 99999
                node = None
                for i in range(4):
                        nextCorner = corner.GetPoint(i)
                        if(nextCorner != None and (nextCorner.position < 0 or nextCorner.position
 == 2)):
                                distance = corner.distance[i]
```

```
                                if(distance > 0):
                                        self.nodes.AddNode(corner.GetPoint(i), distance + self.di
stance, corner)
                                        if(minDistance >= distance):
                                                minDistance = distance
                                                node = self.nodes.GetLastNode()

                return node

        def FindShortesPath(self):
                self.DijkstraAlgoritham()
                node = self.nodes.GetLastNode()
                corners = list()
                corner = node.corner
                while(node.prev != None):
                        for n in self.nodes.nodes:
                                if(n.corner == corner):
                                        corners.append(corner)
                                        corner = n.prev
                                        node = n
                                        break
                return corners
```

Corner.py

```
class Corner(object):
        def __init__(self, right = -1, left = -1, forward = -1, orientation = 0):
                self.x = 0
                self.y = 0
                self.position = -1
                self.Left = None
                self.Right = None
                self.Up = None
                self.Down = None
                self.tremauxCounter = {0:0, 1:0, 2:0, 3:0}
                self.orientation = orientation
                if(orientation == 0):
                        self.distance = {0:forward, 1:right, 2:0, 3:left}
                elif(orientation == 1):
                        self.distance = {0:left, 1:forward, 2:right, 3:0}
                elif(orientation == 2):
                        self.distance = {0:0, 1:left, 2:forward, 3:right}
                else:
                        self.distance = {0:right, 1:0, 2:left, 3:forward}

        def Choose(self, orientation):
                if(orientation == 0):
                        direction = {0:"forward", 1:"right", 2:"backward", 3:"left"}
                elif(orientation == 1):
```

```
                        direction = {0:"left", 1:"forward", 2:"right", 3:"backward"}
                elif(orientation == 2):
                        direction = {0:"backward", 1:"left", 2:"forward", 3:"right"}
                else:
                        direction = {0:"right", 1:"backward", 2:"left", 3:"forward"}

                backward = orientation + 2
                if(backward > 3):
                        backward -= 4
                self.tremauxCounter[backward] += 1

                wallFollowerDirection = {0:"right", 1:"forward", 2:"left"}
                for i in range (3):
                        for j in range(4):
                                if(self.tremauxCounter[j] == 0 and self.distance[j]>-
1 and wallFollowerDirection[i] == direction[j]):
                                        self.tremauxCounter[j]+=1
                                        return direction[j]

                if(self.tremauxCounter[backward] == 1 and self.distance[backward]>-1):
                        self.tremauxCounter[backward]+=1
                        return direction[backward]

                for i in range (3):
                        for j in range(4):
                                if(self.tremauxCounter[j] == 1 and self.distance[j]>-
1 and wallFollowerDirection[i] == direction[j]):
                                        self.tremauxCounter[j]+=1
                                        return direction[j]

        def SetDistance(self, distance, orientation, prev):
                backward = orientation + 2
                if(backward > 3):
                        backward -= 4
                self.distance[backward] = distance
                prev.distance[orientation] = distance

        def SetPoint(self, prev, orientation):
                if(orientation == 0):
                        prev.Up = self
                        self.Down = prev
                elif(orientation == 1):
                        prev.Right = self
                        self.Left = prev
                elif(orientation == 2):
                        prev.Down = self
                        self.Up = prev
                else:
                        prev.Left = self
```

```
                self.Right = prev

    def GetPoint(self, orientation):
            if(orientation == 0):
                    return self.Up
            elif(orientation == 1):
                    return self.Right
            elif(orientation == 2):
                    return self.Down
            elif(orientation == 3):
                    return self.Left
            else:
                    return None

    def GetDirection(self, orientation, corner):
            if(orientation == 0):
                    if(corner == self.Up):
                            return "Forward"
                    elif(corner == self.Right):
                            return "Right"
                    elif(corner == self.Left):
                            return "Left"
            elif(orientation == 1):
                    if(corner == self.Right):
                            return "Forward"
                    elif(corner == self.Down):
                            return "Right"
                    elif(corner == self.Up):
                            return "Left"
            elif(orientation == 2):
                    if(corner == self.Down):
                            return "Forward"
                    elif(corner == self.Left):
                            return "Right"
                    elif(corner == self.Right):
                            return "Left"
            else:
                    if(corner == self.Left):
                            return "Forward"
                    elif(corner == self.Up):
                            return "Right"
                    elif(corner == self.Down):
                            return "Left"

    def SetPostion(self, positon):
            self.position = positon
```

Points.py

```
from Corner import Corner
import math

class Point(object):
        def __init__(self, corner):
                self.x = corner.x
                self.y = corner.y
                self.corner = corner

class Points(object):
        def __init__(self):
                self.points = list()

        def CheckPoint(self, corner, distance, orientation):
                x = corner.x
                y = corner.y
                if(orientation == 0):
                        x = x+distance
                elif(orientation == 1):
                        y = y+distance
                elif(orientation == 2):
                        x = x-distance
                else:
                        y = y-distance
                for point in self.points:
                        if(math.sqrt(pow(point.x - x, 2) + pow(point.y - y, 2)) <250):
                                return point.corner

                return None

        def AddPoint(self, corner, distance, orientation, prev):
                if(orientation == 0):
                        corner.x = prev.x+distance
                        corner.y = prev.y
                elif(orientation == 1):
                        corner.x = prev.x
                        corner.y = prev.y+distance
                elif(orientation == 2):
                        corner.x = prev.x-distance
                        corner.y = prev.y
                else:
                        corner.x = prev.x
                        corner.y = prev.y-distance
                self.points.append(Point(corner))
```

Prilog 2. CD/DVD medij sa Završnim radom u .pdf obliku, programskim kodom i video zapisom rješavanja labirinta.