Planiranje puta autonomne prskalice

Paro, Borna

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
Preddiplomski studij računarstva

Završni rad

Planiranje puta autonomne prskalice

Rijeka, srpanj 2022.

Borna Paro
0069088363

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
Preddiplomski studij računarstva

Završni rad

Planiranje puta autonomne prskalice

Mentor: prof. dr. sc. Kristijan Lenac

Rijeka, srpanj 2022.

Borna Paro
0069088363

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
POVJERENSTVO ZA ZAVRŠNE ISPITE

Rijeka, 15. ožujka 2022.

Zavod: **Zavod za računarstvo**
Predmet: **Algoritmi i strukture podataka**
Grana: **2.09.04 umjetna inteligencija**

ZADATAK ZA ZAVRŠNI RAD

Pristupnik: **Borna Paro (0069088363)**
Studij: Preddiplomski sveučilišni studij računarstva

Zadatak: **Planiranje puta autonomne prskalice / Autonomous sprayer path planning**

Opis zadatka:

Razviti programsku podršku za planiranje puta autonomne prskalice u ROS (Robot Operating System) sustavu. Za zadanu početnu poziciju prskalice, cilj je odrediti put kojime se pokriva zadana površina uz poštivanje ograničenja kretanja.

Rad mora biti napisan prema Uputama za pisanje diplomskih / završnih radova koje su objavljene na mrežnim stranicama studija.

Zadatak uručen pristupniku: 21. ožujka 2022.

Mentor:

Prof. dr. sc. Kristijan Lenac

Predsjednik povjerenstva za završni ispit:

Prof. dr. sc. Kristijan Lenac

Izjava o samostalnoj izradi rada

Izjavljujem da sam samostalno izradio završni rad na temu „Planiranje puta autonomne prskalice“ pod mentorstvom prof. dr. sc. Kristijana Lenca.

Rijeka, srpanj 2022.

Ime Prezime

Zahvala

Zahvaljujem se obitelji, svim prijateljima, kolegama, profesorima, asistentima koji su mi na bilo koji način pomogli, olakšali i uljepšali studiranje.

Sadržaj

Poglavlje 1

Uvod

Robotika je interdisciplinarna grana računarske znanosti i inženjerstva koja uključuje dizajniranje, kreiranje i programiranje robota. Robotika stvara strojeve koji mogu zamijeniti i replicirati ljudske akcije. Roboti se mogu koristiti u raznim situacijama, na primjer u opasnim okruženjima za čovjeka kao što je pronalaženje i deaktiviranje eksplozivnih naprava, odlazak u svemir, rad na visokim temperaturama, rad na zagađenim i radioaktivnih prostorima itd. [1]

Iako se roboti danas najviše koriste u ekstremnim uvjetima kao što su prethodno navedeni, velik dio robota također obavlja zadatke koji su ljudima naporni, repetitivni ili dosadni. Roboti mogu zahtijevati ljudsko upravljanje ili pak mogu biti potpuno autonomni što je od posebnog interesa zato što ne zahtijeva gotovo nikakvo ljudsko predznanje, nego robot radi sam sve što se od njega zahtjeva.

U ovom radu će biti prikazan i opisan jedan od repetitivnih poslova, a to je oprašivanje sadnica. Problem je zamišljen na način da je robotu dana mapa polja i koordinate sadnica te da robot za bilo kakvu orijentaciju sadnica nađe najkraći put, uz nikakvo ili što manje preklapanje, koji će proći između svih redova sadnica i oprašiti ih.

Za programiranje logike kretanja i ponašanja robota korišten je ROS2 (Robot Operating System), za kreiranje polja i simuliranje robota korišten je Gazebo, a za vizualizaciju poruka i markera korišten je RViz2.

Poglavlje 2

Razrada problema

Tema ovog rada, a i samog problema, je planiranje puta autonomne prskalice. Planiranje puta je jedan od čestih problema koji se rješava u mobilnoj robotici te ovisno o sustavu u kojemu se robot nalazi i specifikaciji samog problema, potrebno je uzeti u obzir razna ograničenja i zahtjeve koji moraju biti ispunjeni. Traženi zahtjev ovog problema je bio da robot mora obići svaki dio svake sadnice uz nikakvo ili što manje preklapanje pređenog puta. Bez navedenog zahtjeva, rješavanje problema bi bilo puno jednostavnije jer nakon što bi se generirale točke kroz koje robot mora proći, mogla bi krenuti navigacija gdje bi jedini zahtjev bio da robot mora obići svaku točku. No, takav način navigacije je neefikasan jer bi robot kroz neke dijelove polja prošao veliki broj puta, a kroz neke ne bi uopće te bi mogle ostati ne oprašene sadnice.

2.1 Mapa

Pretpostavka ovog rada je bila da je sustavu dana mapa u kojoj je robot lokaliziran. Mapa je dana kao par *YAML* i *PGM* datoteka koje ROS i RViz koriste za lokalizaciju i vizualizaciju. *YAML* datoteka opisuje metapodatke mape kao što su ime mape, rezolucija itd., a *PGM* datoteka, tj. slika, opisuje popunjenost svake ćelije, gdje bijela boja predstavlja prazan prostor, crna boja zauzet (neka prepreka ili bilo kakav drugi objekt), a nijanse sive boje predstavljaju neznanje. Osim slike *PGM* formata može se koristiti i *PNG* format, no *PGM* je bolje rješenje zato što zauzima manje

memorije. Ograničenje *PGM*-a je to što može prikazati samo nijanse sive boje [2], tj. sve nijanse između crne i bijele, ali to je ionako sve što je ROS-u potrebno.

Primjer koda unutar *YAML* datoteke prikazan je u sljedećem isječku koda (Isječak koda 2.1).

```
image: horizontalna_mapa.pgm
mode: trinary
resolution: 0.05
origin: [-1.07, -0.017, 0]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.25
```

Isječak koda 2.1 *Primjer YAML datoteke*

2.2 Polje i sadnice

Uz mapu, zadane su i koordinate okvira polja i koordinate sadnica. Točke su zadane u lokalnom koordinatnom sustavu mape gdje svaka koordinata predstavlja udaljenost točke po pojedinoj osi od ishodišta. Pošto se robot može kretati samo po xy ravnini, z os se zanemaruje. Pretpostavka ovog rada je da je svako polje pravokutnik, stoga su zadane četiri točke gdje je svaka točka jedan kut pravokutnika. Koordinate sadnica su zadane na način da su za svaki red sadnica zadane po dvije točke, početna i krajnja točka tog reda sadnice.

Uz koordinate okvira polja i sadnica još su zadane početna i krajnja ciljna pozicija do koje robot mora doći i zaustaviti se.

2.3 Model robota

Pretpostavka je da će se koristiti model robota koji može proći između svakog para reda sadnica te ima dovoljno manevarskog prostora za rotiranje i prolazak između ruba sadnica i okvira polja.

Poglavlje 3

Korištene tehnologije i alati

3.1 ROS

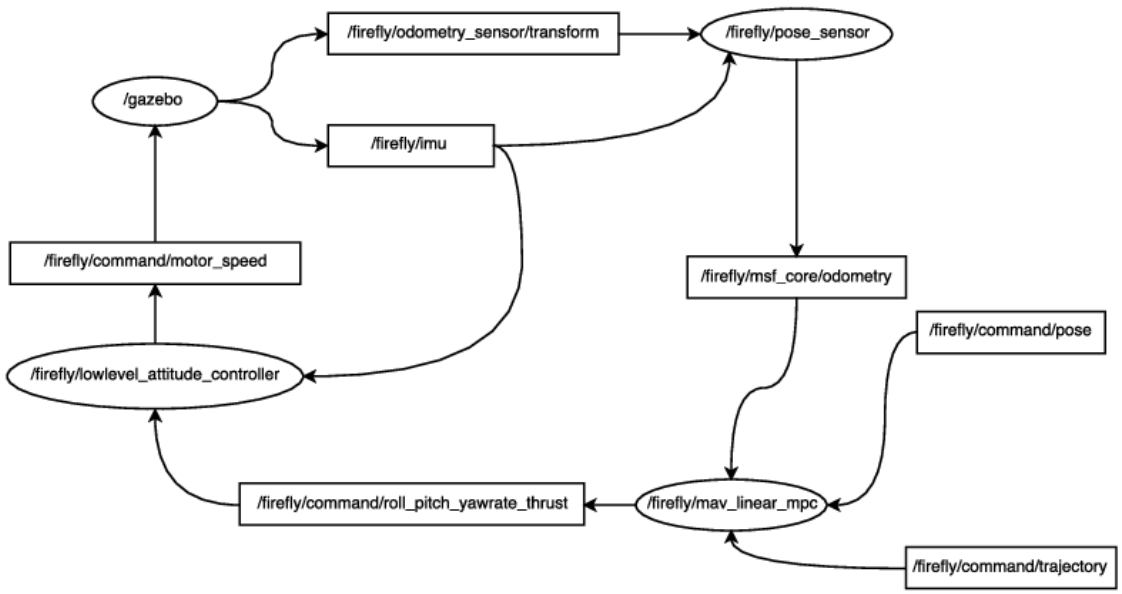
Robotski operacijski sustav (eng. Robot Operating System, skraćeno ROS) je kolekcija međusoftvera (eng. middleware) otvorenog koda (eng. open-source) koji služe za razvijanje softvera za upravljanje robotima. Iako u svom nazivu ima *operacijski sustav* zbog funkcionalnosti koje pruža kao što su: apstrakcija hardvera, kontrola uređaja na niskoj razini, implementacija uobičajenih funkcionalnosti, prijenos poruka između procesa i upravljanje paketima itd., on zapravo nije samostalni operacijski sustav, nego mora biti instaliran na operacijskom sustavu kao što je GNU/Linux.

3.2 Arhitektura ROS sustava

3.2.1 Model računalnog grafa

ROS procesi su prikazani kao čvorovi u grafu koji su povezani bridovima koji se zovu teme (eng. topic). ROS čvorovi mogu prosljeđivati poruke drugim čvorovima kroz teme, stvarati zahtjeve prema serverima koji su drugi čvorovi ili pružati usluge servera drugim čvorovima [3].

Na sljedećoj slici (Slika 3.1) je prikazan jednostavni ROS graf. Elipse predstavljaju čvorove, a pravokutnici teme.



Slika 3.1 *Prikaz ROS grafa [4]*

3.2.2 Čvorovi (eng. Nodes)

Čvor predstavlja proces koji se izvršava unutar ROS grafa [5]. Svaki čvor ima svoje ime koje je poželjno da bude jedinstveno da ne bi došlo do neočekivanih problema. U praksi, robotski sustav se sastoji od više čvorova od kojih svaki obavlja svoju zadaću. Na primjer, jedan čvor upravlja kotačima i kontrolira brzinu, drugi služi za lokalizaciju i mapiranje, treći šalje poruke o svojoj lokaciji da bi se mogao vizualizirat itd.

3.2.3 Poruke (eng. Messages)

Čvorovi međusobno komuniciraju prosljeđivanjem poruka. Poruka je struktura podataka koja može biti jednostavna npr. jedan cijeli broj, a može biti i kompleksna da unutar sebe sadrži polja, instance drugih poruka itd. [6]

3.2.4 Teme (eng. Topics)

Teme su transportne linije preko kojih čvorovi primaju i šalju poruke [7]. Svaka tema ima svoje ime koje treba biti jedinstveno. Da bi se poslala poruka, čvor mora objaviti (eng. publish) poruku na odgovarajuću temu, a da bi se primila čvor se mora pretplatiti (eng. subscribe) na odgovarajuću temu. *Publish/subscribe* model je anoniman, ni jedan čvor ne zna tko objavljuje, a tko je pretplaćen na temu, a čvor koji je pretplaćen na neku temu prima sve poruke s te teme. Primjer poruka koje se šalju na temu su: mjerenja senzora, naredbe za upravljanje kotačima i ubrzanjem, informacije o stanju robota itd. Za razliku od servisa, objavljivanje i pretplaćivanje na temu se može prekinuti.

3.2.5 Servisi (eng. Services)

Servis se sastoji od poslužitelja koji čeka na zahtjev i od klijenta koji šalje zahtjev poslužitelju [8]. Servis je uglavnom neka kratka jednostavna radnja koja se brzo izvrši i nije repetitivna ni periodična i rijetko se dešava. Zahtjev za servis se ne može prekinuti, za razliku od tema i akcija.

3.2.6 Akcije (eng. Actions)

Akcija je kombinacija tema i servisa. Koristi se kod radnji kojima treba duže vremena da se izvrše. Sastoji se od tri djela, *goal*, *feedback* i *result*. Akcija započinje tako da klijent napravi zahtjev prema akcija serveru te ga on prihvati i sve dok se radnja ne izvrši, server klijentu šalje informacije o izvršavanju preko *feedback* teme [9]. Za razliku od servisa, akcija se može prekinuti.

3.2.7 Usporedba ROS1 i ROS2

ROS1 je kreirala tvrtka Willow Garage 2007. godine te su kroz iskustvo i godine koje su prošle uvidjeli koje bitne karakteristike nedostaju i što bi se moglo poboljšati. Nažalost, ako bi išli mijenjati postojeću implementaciju sustav bi postao nestabilan i sklon pucanju, stoga se odlučilo da će se "iz nule" napraviti novi sustav.

Glavna razlika između navedenih sustava je ta što više ne postoji i nije potreban *ROS Master* koji se je uvijek morao prvi pokretati i služio je kao DNS server za čvorove, bez njega čvorovi ne bi mogli komunicirati jer se ne bi mogli međusobno pronaći [10]. ROS2 sustav nije centralizirani sustav, svaki čvor može pronaći bilo koji čvor. Ova karakteristika je vrlo korisna zato što čvorovi više nisu ovisni o globalnom *master* čvoru te to omogućuje kreiranje u potpunosti distribuiranih sustava.

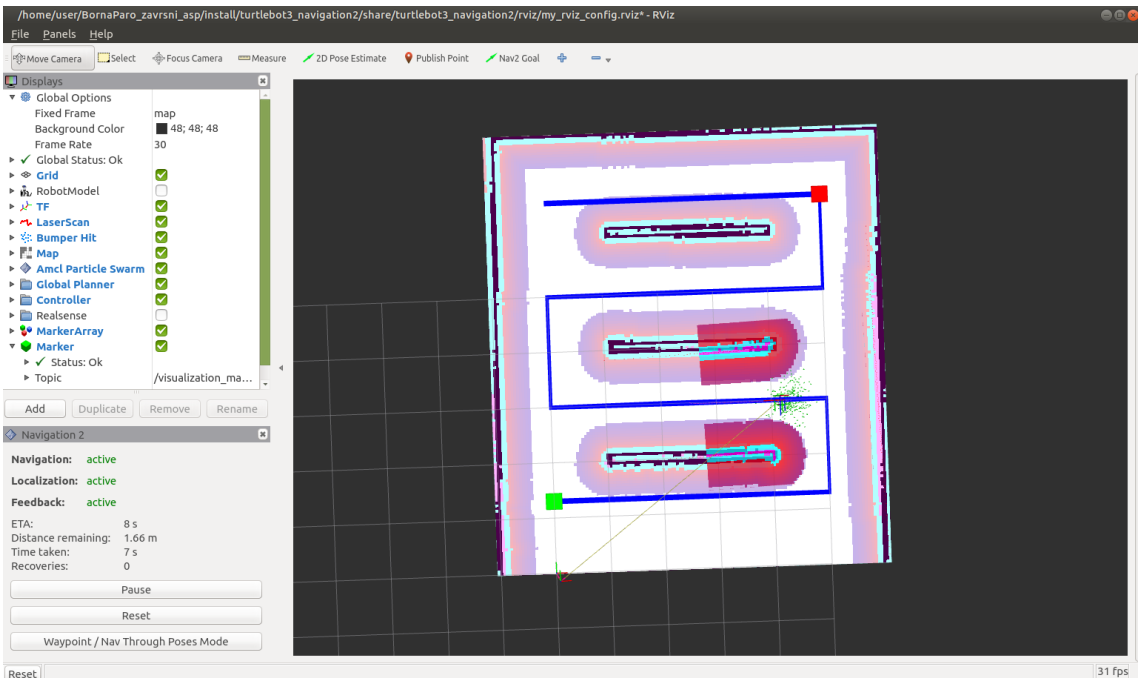
Za izradu ovog rada koristio sam ROS2 sustav pošto je noviji, pruža dugoročniju podršku, bolje je konstruiran te se savjetuje da se koristi ROS2 sustav jer kad istekne podrška za ROS1 sav kod će se morati prebaciti u ROS2. No, unatoč prednostima koje ROS2 pruža, danas se još uvijek poprilično koristi ROS1 zbog stabilnih alata koje podržava, a nisu još implementirani unutar ROS2 te zbog opsežnije dokumentacije

3.3 Gazebo simulator

Gazebo je 3D simulator otvorenog koda koji služi za simuliranje robota i njihovog okruženja. Unutar sebe ima integriran *ODE* sustav za simulaciju fizike nad objektima, *OpenGL* za prikazivanje te podršku za simulaciju raznih senzora i aktuatora [11]. Vrlo je popularan simulator zbog visokih performansi te zbog grafičkog sučelja preko kojeg se može jednostavno modelirati svijet s raznim preprekama te se jednostavno mogu modelirati roboti s raznim konfiguracijama i senzorima koje će koristiti.

3.4 RViz2

ROS2 visualization ili skraćeno RViz2 je 3D grafičko sučelje koje služi za vizualizaciju raznih podataka koji se dešavaju unutar ROS2 sustava [12]. Pomoću različitih umetaka (eng. plugin) mogu se prikazivati razne informacije kao npr. mapa, transformacije, informacije o modelu robota, informacije senzora, markeri itd.



Slika 3.2 *Prikaz RViz2 grafičkog sučelja*

3.5 TurtleBot3 Burger

Za izradu ovog rada koristio sam robota *TurtleBot3 Burger* prvenstveno zbog toga što sam već radio s tim robotom te zbog toga što za njega postoji najviše dokumentacije i tutoriala na Internetu koji objašnjavaju njegovo korištenje što mi je olakšalo izradu rada i zaobilaženje problema na koje sam naišao koristeći drugog robota.

Slika 3.3 *Prikaz TurtleBot3 Burger robota [13]*

3.6 Nav2

Nav2 je projekt koji je nasljednik ROS Navigation Stack-a. Nav2 unutar sebe ima razne pakete kojima je glavni cilj pronaći siguran način da mobilni robot dođe od točke A do točke B. Neke mogućnosti koje nav2 paketi nude su: dinamičko planiranje puta, izračunavanje brzine za motore, izbjegavanje prepreka itd. [14]

3.7 slam_toolbox

Slam Toolbox je paket koji ukomponira informacije dobivene iz laserskih skenera te radi istovremenu lokalizaciju i mapiranje (na eng. skraćeno SLAM) [15]. Paket mi je bio potreban kako bi mogao lokalizirati robota u novom okruženju te kreirati mapu tog okruženja sa svim preprekama koje ga okružuju.

3.8 VirtualBox

Oracle VM VirtualBox je virtualizacijski softver koji omogućuje korisniku da na jednom uređaju koristi više operacijskih sustava [16].

Cijeli Ubuntu Mate sa svim potrebnim ROS2, Gazebo i ostalim paketima i potrebnim aplikacijama je instaliran na VirtualBox virtualnoj mašini. Primarni razlog korištenja virtualne mašine je manjak dodatnog diska na koji bih mogao instalirati Linux distribuciju, a dodatna prednost je ako se slučajno obrišu neke bitne datoteke za pokretanje operacijskog sustava ili ako se nešto poremeti, puno je lakše ponovo instalirati virtualnu mašinu nego očistiti cijeli disk ili particiju i sve ponovo instaliravati.

Mane korištenja virtualne mašine bi bile što ne može koristiti sve jezgre procesora, svu radnu memoriju i memoriju grafičke kartice, no za potrebe ovog rada to nije stvaralo probleme.

Poglavlje 4

Implementacija i analiza rješenja

Sve funkcionalnosti i sva logika je pisana u programskom jeziku Python koristeći klijentsku knjižnicu *rclpy*. Odabrao sam Python umjesto C++-a zato što je brže i lakše programiranje te se na manje stvari mora paziti zato što je interpreterski jezik i samim time je "labaviji", tj. programer na manje stvari mora misliti i uzimati u obzir. Neki od nedostataka Pythona u usporedbi s C++-om su ti što je sporiji i zauzima više memorije, no za potrebe ove simulacije to nije predstavljalo problem.

Rclpy je Python klijentska knjižnica koja se izgrađuje povrh *rcl* knjižnice pisane u programskom jeziku C koja služi kao aplikacijsko programsko sučelje za ROS funkcionalnosti. *Rclpy* pruža apstrakciju visokog levela za implementiranje čvorova, tema, akcija, servisa itd. [17]

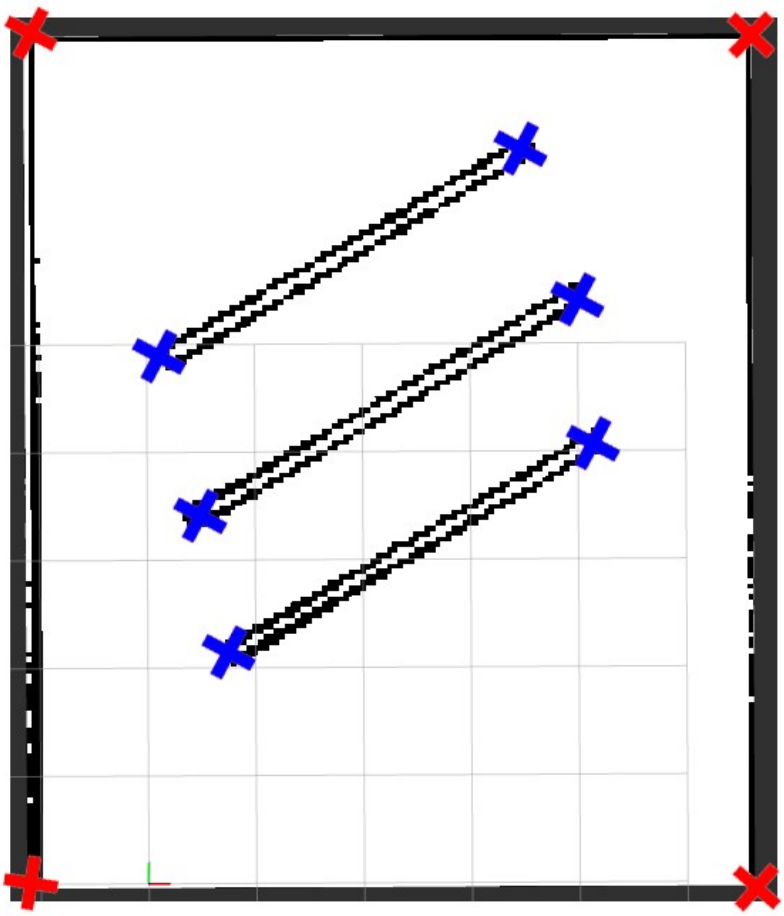
4.1 Dohvaćanje koordinata sustava

Nakon što se pokrene skripta za pokretanje simulacije, ona pokreće izvršnu (eng. executable) datoteku *findPath.py* unutar koje se nalaze najbitnije funkcionalnosti.

Pri samom pokretanju datoteke, ona stvara objekt *navigator* koji je instanca klase *BasicNavigator* koji se nalazi unutar *nav2* paketa te samim time nasljeđuje sve njegove metode. Objekt *navigator* objavljuje svoju inicijalnu poziciju uz pomoć *setInitialPose* metode koja se nalazi unutar *nav2* paketa. Objavljivanje inicijalne pozicije je bitno kako bi se robot mogao lokalizirati u okruženju u kojem se nalazi te

kako bi se poslije mogao precizno kretati.

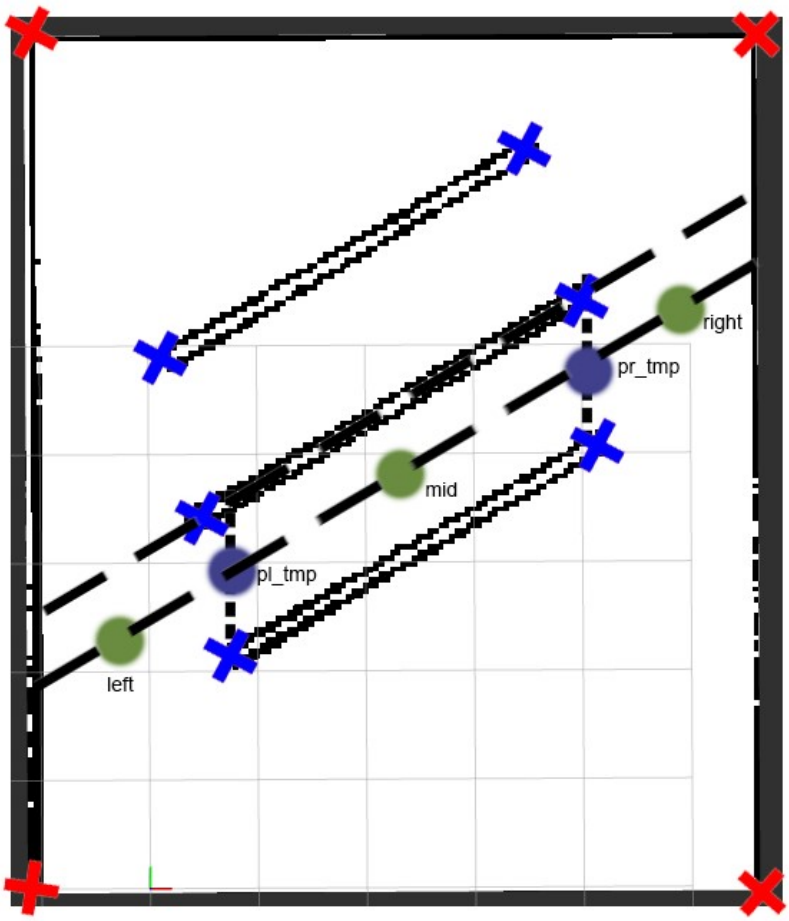
Ovisno o mapi koja se koristi, dohvaćaju se predefinirane lokalne koordinate sadnica i okvira polja za tu mapu. Prikaz koordinata koje se dohvaćaju za polje gdje su sadnice orijentirane u koso možemo vidjeti na slici 4.1 gdje crveni križići označavaju koordinate okvira polja, a plavi križići označavaju koordinate početka i kraja sadnica pojedinog retka.



Slika 4.1 *Prikaz zadanih koordinata okvira polja i sadnica*

4.2 Izračun točaka obilaženja

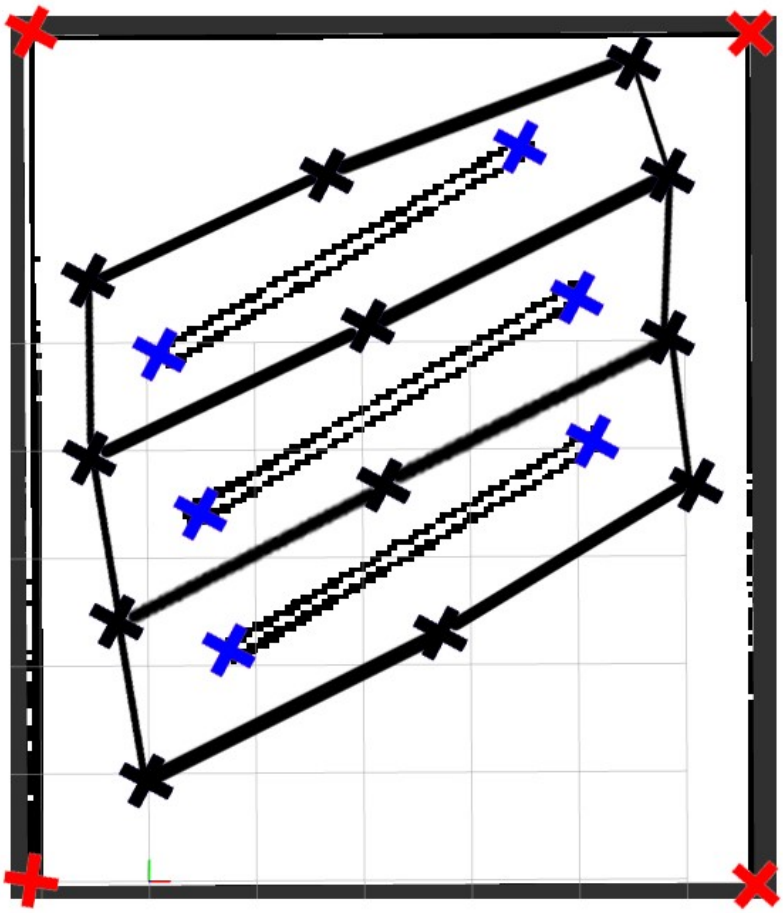
Nakon dobivenih koordinata okvira polja i sadnica, izračunavaju se koordinate koje se nalaze između parova sadnica kroz koje robot mora proći kako bi oprašio sve sadnice. Izračun koordinata je sljedeći, prvo se moraju izračunati privremene točke koje se nalaze na polovici udaljenosti po y osi između jednog para sadnica. Nakon što se izračunaju privremene točke, kroz njih se provuče pravac te se odrede trajne točke koje su na pola udaljenosti između ruba sadnica i okvira polja po x osi, a leže na tom pravcu. Srednja točka se nalazi na pola puta između lijeve i desne trajne točke po x osi, a leži na prethodno definiranom pravcu. Na slici 4.2 je grafički prikazan izračun točaka gdje su *pl_tmp* i *pr_tmp* privremene, a *left*, *mid* i *right* trajne točke. U slučaju da je prevelika udaljenost između prve sadnice i donjeg okvira polja i zadnje sadnice i gornjeg okvira polja, napravit će se izračun da robot bude što bliže sadnicama.



Slika 4.2 *Prikaz određivanja točaka kroz koje robot mora proći*

4.3 Kreiranje grafa

Novo dobivene točke se povezuju u matricu susjedstva koja služi kao graf struktura podataka nad kojom će se vršiti istraživanje puta kojim robot treba ići kako bi oprašio sve sadnice uz minimalno preklapanje prođenog puta. Na slici 4.3 crnim križićima su označene točke kroz koje robot mora proći, one predstavljaju čvorove grafa, dok crne linije koje ih povezuju predstavljaju bridove grafa, težina svakog brida je 1.



Slika 4.3 *Prikaz točaka, kroz koje robot mora proć, povezane u graf*

4.4 Traženje optimalnog puta

Nakon generirane matrice susjedstva potrebno je odrediti kojim redoslijedom će robot obići svaku točku tako da ne bude preklapanja ili da bude što manje preklapanja. Problem određivanja puta može se svesti na pojednostavljenu verziju problema trgovačkog putnika (eng. Travelling salesman problem). Problem trgovačkog putnika postavlja pitanje: "Za danu listu gradova i udaljenosti između svakog para gradova, koji je najkraći put koji će posjetiti svaki grad točno jednom i vratiti se u početni grad?" [18]. Za razliku od navedenog problema, problem ovog rada je pojednostavljen na način da se nije potrebno vratiti u početni čvor (grad) te da je težina svakog brida (udaljenost gradova) jednaka, tj. iznosi 1.

Pronalaženje puta započinje na način da se poziva statička metoda *solve* iz klase *PathFinder* koja kao argumente prima indeks početne i krajnje ciljne točke te matricu susjedstva. Metoda *solve* poziva statičku metodu *search*, također iz klase *PathFinder*, koja pokušava pronaći sve puteve koji posjećuju svaki čvor (točku) točno jednom. Metoda *search* implementira algoritam *unatražnog pretraživanja* (eng. backtracking). *Unatražno pretraživanje* spada pod algoritme pretraživanja grafa i stabla koji inkrementalno stvara kandidate koji bi mogli biti rješenje, no ako utvrdi da taj kandidat nije validan, odbacuje ga i traži nove kandidate [19].

Prikaz metoda može se vidjeti u sljedećem isječku koda (Isječak koda 4.1).

```
1  import queue
2
3  class PathFinder:
4
5      @staticmethod
6      def is_valid_state(state, adjMat):
7      # check if it is a valid solution
8          for val in range(len(adjMat[0])):
9              if val not in state:
10                 return False
11
12         return True #svi nodeovi unutar state-a
13
```

```
14      @staticmethod
15      def get_neighbours(current, adjMat):
16          tmp = []
17          for index, val in enumerate(adjMat[current]):
18              if val != 0:
19                  tmp.append(index)
20          return tmp
21
22      @staticmethod
23      def search(current, adjMat, state, solutions):
24          if PathFinder.is_valid_state(state, adjMat):
25              solutions.append(state.copy())
26              return
27
28          for neighbour in PathFinder.get_neighbours(current,
    adjMat):
29              if neighbour not in state:
30                  state.append(neighbour)
31                  PathFinder.search(neighbour, adjMat, state,
    solutions)
32                  state.pop()
```

Isječak koda 4.1 *Dio metoda klase PathFinder*

search metoda započinje provjerom ako su kandidati, u ovom slučaju varijable unutar liste *state*, validni. Kandidati su validni ako se svi indeksi nalaze unutar liste što znači da je posjećen svaki čvor, ako je to slučaj, onda se lista trenutnih kandidata stavlja na kraj liste *solutions*. Ako kandidati nisu validni, pretraživanje se nastavlja. Za svaki čvor iterira se kroz njegove susjede, listu susjeda vraća metoda *get_ neighbours*. U slučaju da se susjed već ne nalazi unutar liste *state*, stavlja ga se u listu *state* te se onda nad njim vrši pretraga, kao kod pretraživanja grafa u dubinu. Ako se utvrdi da trenutni čvor nema više neposjećenih susjeda, a kandidati nisu validni, čvor se uklanja iz liste kandidata, pomoću rekurzije se vraća na prethodni čvor te se nastavlja s pretraživanjem. Pretraživanje staje kada se isprobaju sve kombinacije puteva.

Nakon povratka u *solve* metodu provjerava se ako postoji rješenje. Rješenje će uvijek postojati osim u nekim slučajevima kada bi se za početnu ili krajnju ciljnu točku uzeo čvor koji se nalazi u središtu grafa.

U slučaju da postoje rješenja, uzima se ono kojem je zadnji element u listi jednak krajnjoj ciljnoj točki do koje robot treba doći, tj. varijabla *end*. Ako to pak nije slučaj, onda se iterira kroz rješenja te se uzima ono kojem je zadnji element u listi najbliži krajnjoj ciljnoj točki. Traženje najbližeg rješenja se nalazi pomoću pretraživanja grafa u širinu, tj. pomoću metode *bfs*.

Metode *solve* i *bfs* prikazane su u sljedećem isječku koda (Isječak koda 4.2).

```
34      @staticmethod
35      def solve(start, end, adjMat):
36          solutions = []
37          state = [start]
38          PathFinder.search(start, adjMat, state, solutions)
39
40          print(f'solutions for starting node = {start}, end node
    = {end}', solutions)
41          if not solutions:
42              print("couldn't find path that visits each node
    only once")
43              return solutions
44
45          for sol in solutions:
46              if sol[-1] == end:
47                  print(f'shortest path for starting node = {
    start}, end node = {end}', sol)
48                  return sol
49
50          min_len = [9999, 9999] #[<index_solution_path-a>, <
    duzina_najkraceg_solution_path-a>]
51          for index, sol in enumerate(solutions): #od zadnjeg
    elem u toj soluciji do tocke do koje zelimo doc
52              shortest_len = PathFinder.bfs(sol[-1], end, adjMat)
```

```
    #od zadnjeg elem u toj soluciji do tocke do koje zelimo doc
    , moze bfs jer je svima tezina 1
53              if shortest_len < min_len[1]:
54                  min_len[0] = index
55                  min_len[1] = shortest_len
56              print(f'shortest path for starting node = {start},
    end node = {end} after bfs', shortest_len)
57
58          print("min_len", min_len)
59          solutions[min_len[0]].append(end)
60          return solutions[min_len[0]]
61
62      @staticmethod
63      def bfs(start, end, adjMat):
64          visited = []
65          q = queue.Queue()
66          q.put([start, 0]) #node, weight (tj duzina puta)
67
68          while not q.empty():
69              current = q.get()
70
71              for neighbour in PathFinder.get_neighbours(current
    [0], adjMat):
72                  if neighbour in visited: #posjecen
73                      continue
74
75                  if neighbour == end: #dosao do kraja
76                      return current[1] + 1
77
78                  q.put([neighbour, current[1] + 1])
79
80              visited.append(current[0])
```

Isječak koda 4.2 *Prikaz solve i bfs metode*

4.5 Vizualizacija puta

Prikaz putanje obilaženja se realizira tako što na temu *visualization_marker* novo kreirani čvor objavljuje koordinate točaka, a RViz2 koji je pretplaćen na navedenu temu vizualizira put.

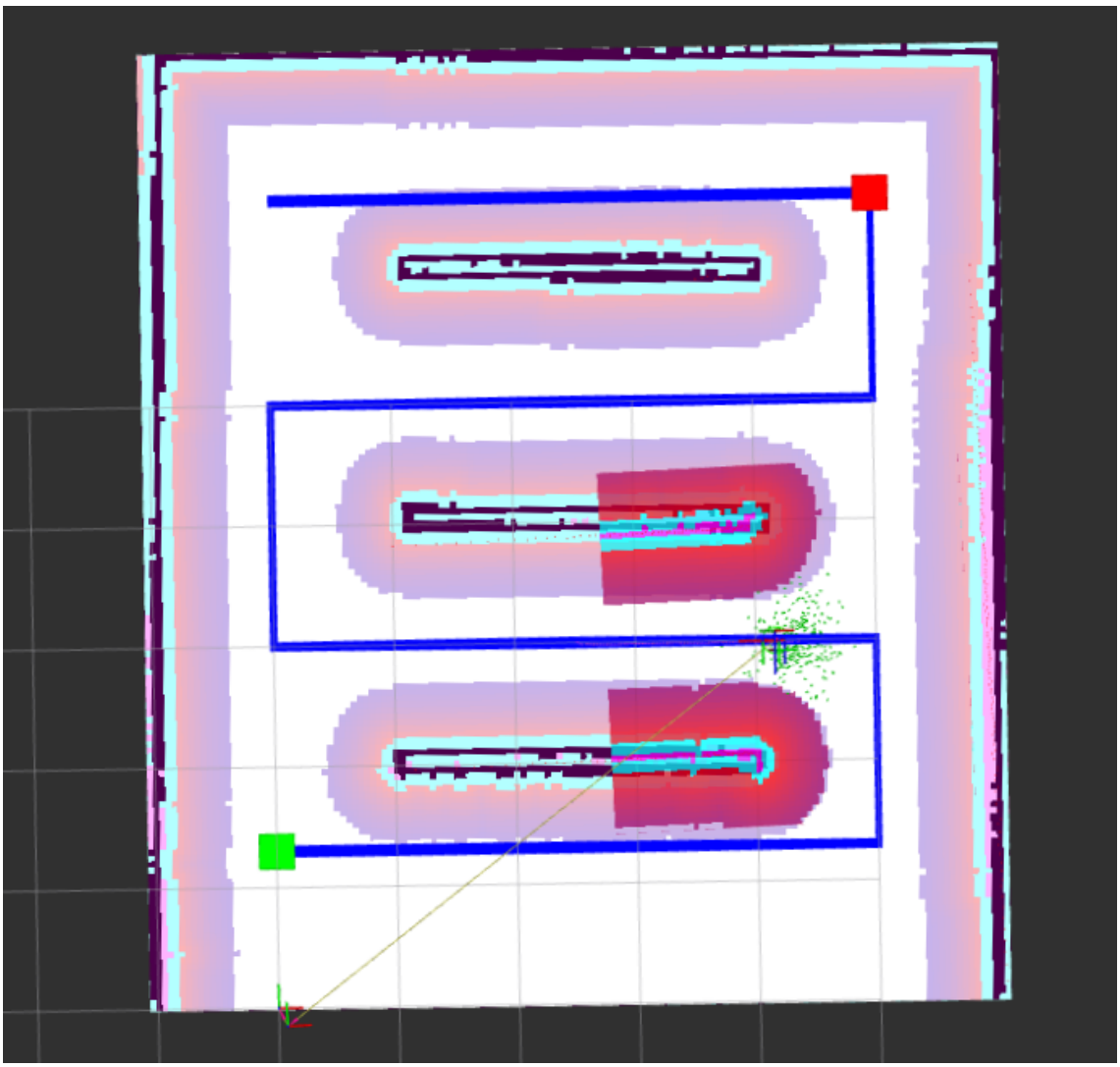
4.6 Navigacija

Navigaciju robota omogućava metoda *followWaypoints* koja se nalazi unutar nav2 paketa. Metoda kao argument prima listu *PoseStamped* točaka kroz koje robot navigira redoslijedom kako su poredane u listi sve dok ne dođe do zadnje točke nakon koje navigacija staje.

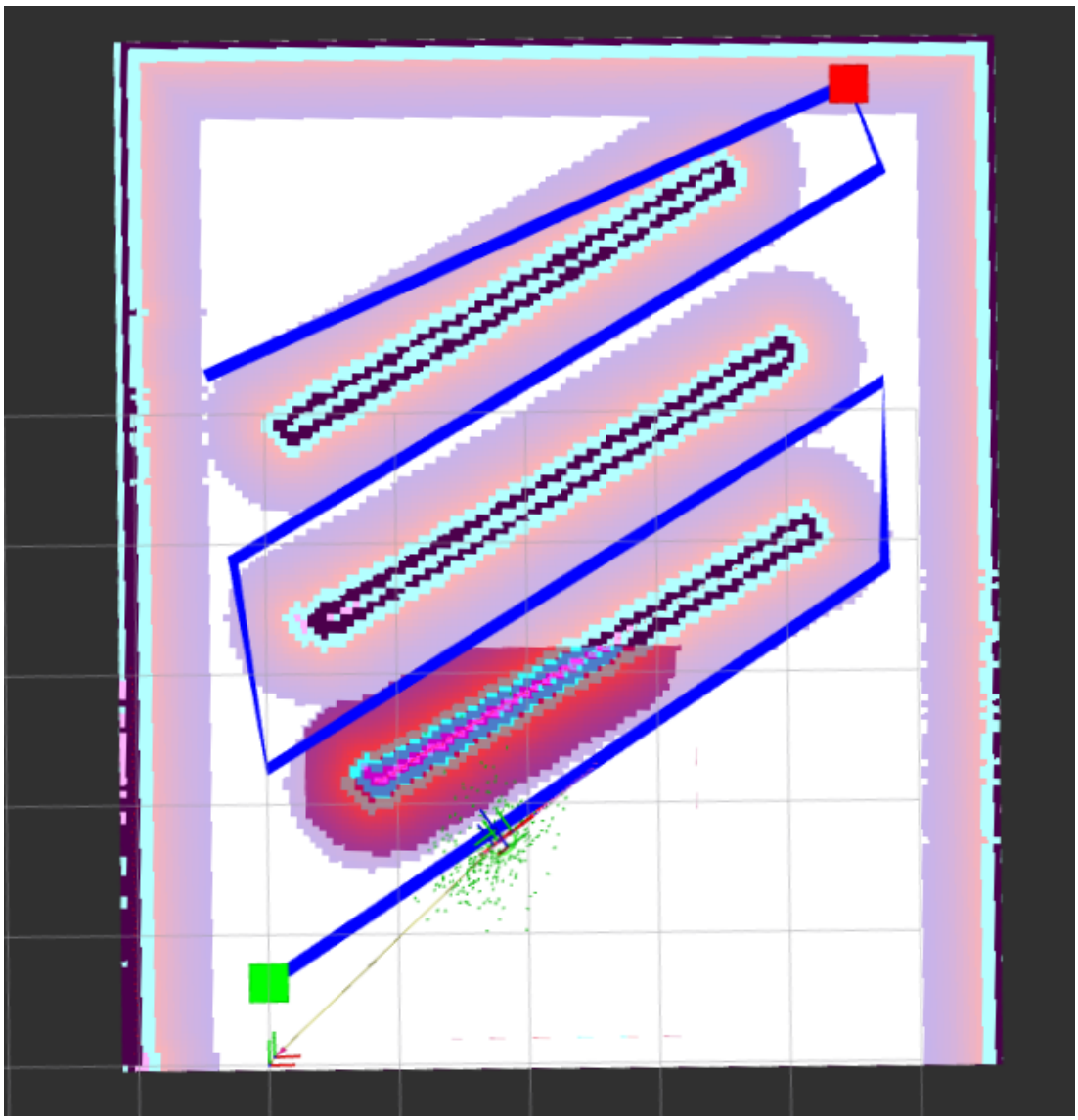
4.7 Prikaz simulacije

Na sljedećim slikama su prikazane simulacije robota za tri različite orijentacije polja unutar RViz2 vizualizacije. Zeleni kvadratić predstavlja početnu točku robota, a crveni kvadratić predstavlja krajnju ciljnu točku do koje robot mora doći i tamo se zaustaviti.

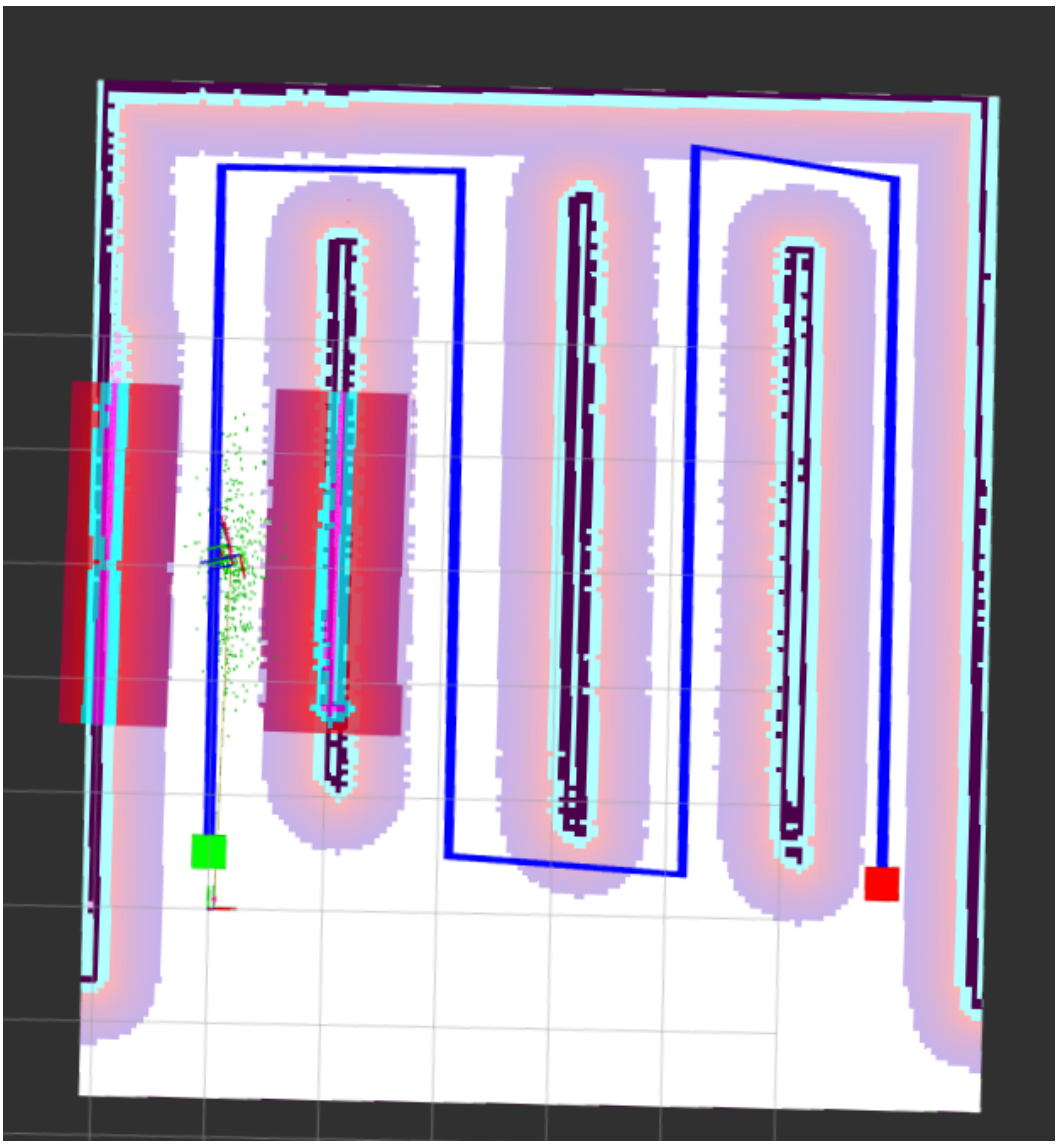
Na slikama 4.4 i 4.5 se može uočiti da iako je robot došao do krajnje ciljne točke (crveni kvadratić), nije gotov s oprašivanjem zato što mu je ostao još cijeli lijevi dio sadnice kojeg nije obišao, tek kad sve dijelove obiđe ide ka krajnjoj ciljnoj točki gdje se zaustavlja.



Slika 4.4 *RViz2, primjer horizontalne orijentacije sadnica*



Slika 4.5 *RViz2, primjer kose orijentacije sadnica*



Slika 4.6 *RViz2, primjer vertikalne orijentacije sadnica*

Poglavlje 5

Zaključak

Cilj ovog završnog rada je bio napraviti sustav i napisati algoritam koji će pronaći optimalan put kretanja robota tako da obiđe sve sadnice u polju, a potom navigirati robota kroz polje i sve to vizualizirati. Trenutno je sustav isproban na tri različite orijentacije sadnica te bi samim time trebao raditi i za sve ostale orijentacije, jer jedina stavka koja se mijenja je vektor smjera pojedinog retka sadnica.

Tijekom izrade ovog rada naučio sam mnogo o ROS2 sustavu, navigaciji, te alatima s kojima je ROS usko povezan, kao što su RViz i Gazebo. Također sam se pobliže upoznao s backtracking algoritmima. Implementiranje algoritma mi je puno pomoglo u samom shvaćaju logike iza te skupine algoritama te mi je pomoglo u jasnijem shvaćaju i korištenju rekurzije. Susreo sam se i s raznim problemima, ponajviše zbog manjka dokumentacije te zbog raspršenosti različitih implementacija na ROS1 i ROS2 sustav.

Moje mišljenje je da je najveći problem ROS-a manjak dokumentacije i primjera korištenja paketa i funkcija. Čak i ako se nađe primjer za neki paket iz ROS1, nema primjera za taj isti paket u ROS2 ili pak paket uopće ne postoji. Iako je implementacija funkcionalnosti u ROS1 i ROS2 poprilično slična, početniku otežava učenje. U mojem slučaju, učenje ROS-a nakon prođenih osnovnih tutoriala se u većini svelo na analiziranje kodova s javno dostupnih repozitorija, pretpostavljanje načina na koji funkcioniraju te njihovo manipuliranje da rade ono što želim.

Nadogradnja na ovaj rad bi bila da okvir polja može biti bilo kakav poligon, a ne

samo pravokutnik. Također, da se umjesto lokalno zadanih točaka i laserskog skenera koriste GPS pozicije, na taj način bi se mogla robotu dat mapa s GPS koordinatama, zadat početne GPS koordinate robota i robot bi se mogao kretati kroz to polje bez prethodne lokalizacije u njemu.

Bibliografija

[1] "Robotics", članak s Interneta, srpanj 2022.

[2] ".PGM File Extension", FileInfo.com, srpanj 2022.

[3] "Robot Operating System", članak s Interneta, srpanj 2022.

[4] ROS graph, slika preuzeta s Interneta, srpanj 2022.

[5] "ROS/Tutorials/UnderstandingNodes – ROS Wiki". ROS.org. Open Robotics, srpanj 2022.

[6] "msg – ROS Wiki". ROS.org. Open Robotics, srpanj 2022.

[7] "ROS/Tutorials/UnderstandingTopics – ROS Wiki". ROS.org. Open Robotics, srpanj 2022.

[8] "ROS/Tutorials/UnderstandingServicesParams – ROS Wiki". ROS.org. Open Robotics, srpanj 2022.

[9] "ROS Actions Overview", MathWorks, srpanj 2022.

[10] "ROS1 vs ROS2, Practical Overview For ROS Developers", članak s Interneta, srpanj 2022.

[11] Ackerman, Evan (2016-02-04). "Latest Version of Gazebo Simulator Makes It Easier Than Ever to Not Build a Robot". IEEE Spectrum. IEEE

[12] "rviz – ROS Wiki". ROS.org. Open Robotics, srpanj 2022.

[13] TurtleBot3 Burger, slika preuzeta s Interneta, srpanj 2022.

[14] S. Macenski, F. Martín, R. White, J. Clavero. The Marathon 2: A Navigation System. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2020.

[15] Macenski, S., Jambrecic I., "SLAM Toolbox: SLAM for the dynamic world", Journal of Open Source Software, 6(61), 2783, 2021.

[16] "Oracle VM VirtualBox Overview", lipanj, 2021.

[17] "ROS 2 Client Interfaces (Client Libraries)", ROS2 službena dokumentacija, srpanj 2022.

[18] "Travelling salesman problem", s Interneta, srpanj 2022.

[19] Gurari, Eitan (1999). "CIS 680: DATA STRUCTURES: Chapter 19: Backtracking Algorithms"

Popis slika

Isječci koda

Pojmovnik

DNS Domain Name System. 7

GPS Global Positioning System. 25

ODE Open Dynamics Engine. 7

PGM Portable Gray Map. 2

PNG Portable Network Graphics. 2

ROS Robot Operating System. 4

SLAM Simultaneous localization and mapping. 9

YAML Yet Another Markup Language ili YAML Ain't Markup Language. 2

Sažetak

U ovom radu je opisan algoritam za traženje najkraćeg puta koji će obići sve sadnice uz nula ili što manje preklapanja koristeći Python programski jezik. Uz algoritam, u radu je opisana i prikazana simulacija koja je kreirana koristeći ROS2, Gazebo simulator, RViz2. Opisan je rad ROS-a, korišteni paketi i alati te model robota. Prikazani su najbitniji dijelovi koda.

Ključne riječi — **robotika, ROS2, planiranje puta, mobilni roboti, autonomni roboti, backtracking**

Abstract

This paper describes an algorithm for finding the shortest path that will visit all seedlings with zero or as little overlap using the Python programming language. In addition to the algorithm, the paper describes and shows the simulation created using ROS2, Gazebo simulator and RViz2. The operation of ROS, robot model, the packages and tools used are described. The most important parts of the code are presented.

Keywords — **robotics, ROS2, path planning, mobile robots, autonomous robots, backtracking**