Programski paket za sintezu optimalnih trajektorija gibanja mobilnog robota

Vardić, Petar Krešimir

Undergraduate thesis / Završni rad

2020

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Mechanical Engineering and Naval Architecture / Sveučilište u Zagrebu, Fakultet strojarstva i brodogradnje**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:235:234579

Rights / Prava: Attribution 4.0 International/Imenovanje 4.0 međunarodna

Download date / Datum preuzimanja: **2025-03-13**

Repository / Repozitorij:

Repository of Faculty of Mechanical Engineering and Naval Architecture University of Zagreb

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Petar Krešimir Vardić

Zagreb, 2020.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

PROGRAMSKI PAKET ZA SINTEZU OPTIMALNIH TRAJEKTORIJA GIBANJA MOBILNOG ROBOTA

Mentor:

Izv. prof. dr. sc. Andrej Jokić, dipl. ing.

Student:

Petar Krešimir Vardić

Zagreb, 2020.

Izjavljujem da sam ovaj rad izradio samostalno koristeći znanja stečena tijekom studija i navedenu literaturu.

Zahvaljujem se obitelji i prijateljima na podršci tijekom studija i mentoru izv. prof. dr. sc. Andreju Jokiću na strpljenju, savjetima i pomoći tijekom izrade završnoga rada.

Petar Krešimir Vardić

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za završne ispite studija strojarstva za smjerove:
proizvodno inženjerstvo, računalno inženjerstvo, industrijsko inženjerstvo i menadžment, inženjerstvo materijala i mehatronika i robotika

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa:	
Ur.broj:	

ZAVRŠNI ZADATAK

Student: **Petar Krešimir Vardić** Mat. Br.: 0035209271

Naslov rada na hrvatskom jeziku: **Programski paket za sintezu optimalnih trajektorija gibanja mobilnog robota**

Naslov rada na engleskom jeziku: **Software package for optimal trajectory synthesis in mobile robotics**

Opis zadatka:

U današnje vrijeme mobilni roboti su sve češće primijenjeni u praksi, primjerice u industrijskim postrojenjima, u lancima opskrbe (npr. posluživanje u skladištima), vojnoj industriji, spasilačkim misijama i svemirskim istraživanima. Jedna od ključnih zadaća u mobilnoj robotici je određivanje trajektorije gibanja robota od neke početne do ciljne pozicije, uz poštivanje eventualnih fizikalnih ograničenja na gibanje, npr. postojanje prepreka. Osnovni cilj ovog rada je razviti programski paket za generiranje optimalnih trajektorija gibanja mobilnog robota između dvije zadane pozicije u poznatoj okolini (poznate su prepreke i ostala eventualna ograničenja na gibanje). Ciljna funkcija u optimiranju je duljina trajektorije, a algoritam za traženje optimalne trajektorije treba se temeljiti na Dijkstrinom algoritmu za traženje najkraćeg puta na grafu.

U radu je potrebno ostvariti sljedeće:

1. Iz pregleda literature prikazati osnovne pristupe problemu sinteze trajektorija gibanja u mobilnoj robotici.
2. Prikazati Dijkstrin algoritam za traženje najkraćeg puta na grafu.
3. Formulirati problem optimalnog gibanja u mobilnoj robotici na takav način da se za rješavanje problema može koristiti Dijkstrin algoritam.
4. Osmisliti i razviti programski paket za generiranje optimalnih trajektorija u mobilnoj robotici.
5. Na nekoliko odabranih primjera ilustrirati rad razvijenog programskog paketa.
6. Razvijeni programski paket treba predati u elektronskom obliku uz izrađenu dokumentaciju za njegovo korištenje.

Zadatak zadan:
28. studenog 2019.

Datum predaje rada:
1. rok: 21. veljače 2020.
2. rok (izvanredni): 1. srpnja 2020.
3. rok: 17. rujna 2020.

Predviđeni datumi obrane:
1. rok: 24.2. – 28.2.2020.
2. rok (izvanredni): 3.7.2020.
3. rok: 21.9. - 25.9.2020.

Zadatak zadao:

Izv. prof. dr.sc. Andrej Jokić

Predsjednik Povjerenstva:

Prof. dr. sc. Branko Bauer

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

POPIS OZNAKA

Oznaka	Jedinica	Opis
G		Graf
V		Skup vrhova (Vertex)
E		Skup bridova (Edge)
u		vrh
v		vrh
$f(n)$		funkcija
$g(n)$		funkcija
$h(n)$		funkcija
O		O-notacija
Ω		Ω-notacija
Θ		Θ-notacija
C		konstanta
n_0		konstanta
C_1		konstanta
C_2		konstanta
b		faktor račvanja A* algoritma
d		dubina najkraćeg puta
P		početni vrh
Q		konačni vrh
$d[v]$		procjena najkraćeg puta vrha v
$\pi[v]$		atribut prethodnika vrha v
$\boldsymbol{Q}$		skup vrhova u Dijkstrinom algoritmu
$\boldsymbol{S}$		skup opuštenih vrhova
T_{rel}		Vrijeme izvršavanja *relax()* funkcije
T_{em}		Vrijeme izvršavanja *extract-min()* funkcije
F_k		k-ti Fibonaccijev broj
φ		zlatni rez
$\boldsymbol{W}$		skup prostora
$\boldsymbol{OB}$		skup prepreka
$\boldsymbol{A}$		skup robota
$\boldsymbol{\gamma}$		skup konfiguracije robota
$\boldsymbol{W_{free}}$		slobodni prostor
$\boldsymbol{W_d}$		diskretni prostor
w_i		i-ti član diskretnog prostora

SAŽETAK

U ovom završnom radu dan je pregled nekih osnovnih pristupa problemu sinteze trajektorije gibanja u mobilnoj robotici. Navedene su klasifikacije načina rješavanja i pojedini, često korišteni, algoritmi koji se koriste u slučaju poznate, statične okoline.

Nadalje je opisan algoritam odabran za rješavanje problema, Dijkstrin algoritam. Uz to su navedena i dodatna poboljšanja.

Rad se nastavlja formulacijom problema genriranja trajektorija mobilnog robota i opisivanjem procesa rješavanja te se konačno prikazuje programski paket osmišljen za generiranje optimalne trajektorije s karakterističnim primjerima.

Ključne riječi: planiranje puta, sinteza optimalne trajektorije, mobilna robotika, Dijkstrin algoritam

SUMMARY

In this bachelor thesis an overview of approaches to the problem of optimal trajectory synthesis is given. Solution classifications are presented, as well as several algorithms used in the case of a known, static enviroment.

Thereafter the algorithm which is chosen to solve the problem in this thesis, Dijkstra's algorithm, is presented. In addition, further improvements of the algorithm are presented in the thesis.

The thesis continues by formulating the problem of optimal trajectory generation for mobile robots and explaining the solution process. Finally the software package for optimal trajectory synthesis is shown together with several illustrative examples.

Key words: path planning, optimal trajectory synthesis, mobile robotics, Dijkstra's algorithm

1. UVOD

Za ljude je doći od jedne točke do druge najčešće trivijalan zadatak, no za robota takav problem predstavlja ozbiljan izazov. Završni zadatak se sastoji od pronalaženja trajektorije od početne do krajnje točke i pristup rješavanju se razlikuje ovisno o okolini, vrsti robota, primjeni i sl.

Kako bi sinteza trajektorije bila sigurna i djelotvorna potrebno je imati učinkovit algoritam za određivanje trajektorije. U pravilu je cilj minimizirati duljinu, što utječe na ostale vrijednosti kao što je vrijeme procesuiranja i utrošak energije potreban za ostvarivanje trajektorije.

U drugom poglavlju će se opisati općenita definicija problema sinteze optimalne trajektorije, uz navođenje različitih kategorija problema, kao i jedna kategorija algoritama za probleme, s fokusom na planiranje globalne trajektorije koristeći egzaktne i heurističke algoritme.

U narednom poglavlju će se pobliže definirati Dijkstrin algoritam na kojem će se temeljiti sinteza trajektorije i također spomenuti moguća poboljšanja koja mogu smanjiti vrijeme računanja najkraćeg puta.

U četvrtom poglavlju će se prikazati formulacija problema što će se svesti na formulaciju prostora i geometrije, dok će se u petome poglavlju objasniti pristup rješenju. Konačno u šestome će se poglavlju prikazati programski paket s karakterističnim primjerima.

2. SINTEZA OPTIMALNIH TRAJEKTORIJA GIBANJA U MOBILNOJ ROBOTICI

Kako bi mogli riješiti problem navigacije robota potrebno je odrediti gdje se robot nalazi, kamo ide, i kako će tamo doći.

To se preslikava na tri osnovne funkcije navigacije [1]:

- Lokalizacija: Određivanje položaja robota u okolini, koristeći različite senzore. Pozicija može biti određena u odnosu na okolinu, kao topološka koordinata ili u apsolutnim koordinatama.
- Mapiranje: Robotu je potrebna karta okoline u kojoj se giba, a ta karta može biti unaprijed ubačena u memoriju robotu ili se može stvarati dok robot otkriva svoju okolinu
- Planiranje trajektorije: Ako se želi pronaći trajektoriju, ciljna pozicija mora biti unaprijed poznata, nakon čega je moguće odrediti put kojim će robot ići. Samim time, potrebno je imati shemu određivanja adrese po kojoj robot može odrediti gdje ide od početne pozicije. Adrese, primjerice, mogu biti apsolutne ili relativne koordinate.

U ovom radu razmatrat će se funkcija planiranja trajektorije, koja se može riješiti na različite načine ovisno o poznatim parametrima i pristupu modeliranja, optimiziranja i pronalaska puta.

2.1. Klasifikacija načina rješavanja

Sinteza optimalne trajektorije se ponajprije dijeli na globalnu i lokalnu. Globalna sinteza trajektorije podrazumijeva da robot unaprijed ima informacije o okolini u kojoj djeluje dok lokalna sinteza podrazumijeva da je većina informacija nepoznata robotu prije početka djelovanja [2].

Zatim se način rješavanja može klasificirati po načinu modeliranja okoline [2][3], što omogućuje jednostavnije shvaćanje prostora u kojem se djeluje te smanjuje nepotrebno planiranje, samim time smanjujući i vrijeme potrebno da bi se dobila trajektorija.

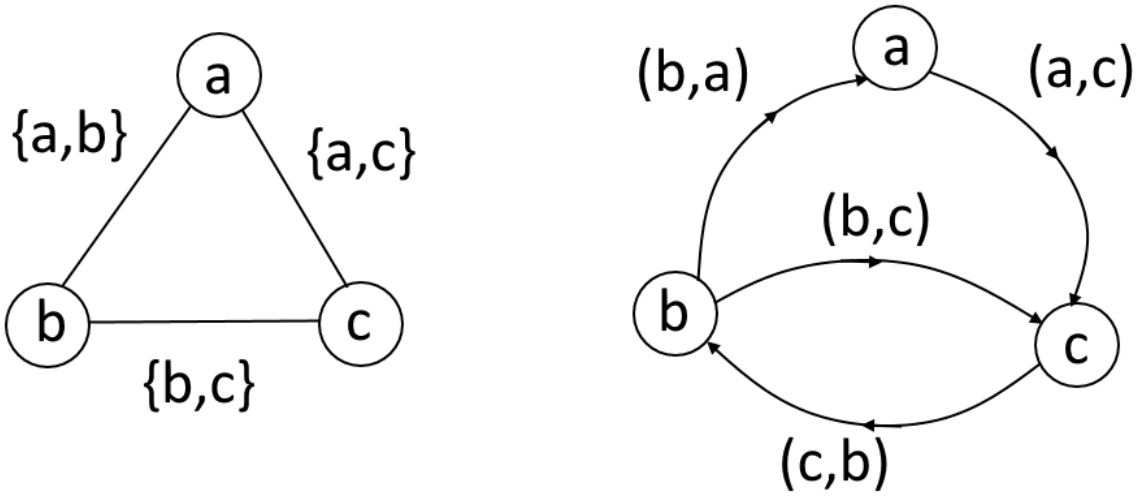
Sljedeći faktor je kriterij optimizacije, u kojima se ističu efikasnost, preciznost i sigurnost [4]. Po tim kriterijima, robot bi trebao doći do cilja uz minimalni utrošak energije [1], što se često može svesti na minimizaciju puta [5].

Posljednji faktor je odabir algoritma, a nudi se niz različitih opcija, uključujući korištenje umjetne inteligencije kao što su genetski algoritmi, ali i algoritmi za pronalaženje puta kao što su Dijkstrin algoritam, D* algoritam i A* algoritam [2].

2.1.1. Algoritmi za pronalaženje puta

Algoritmi za pronalaženje puta su vrsta algoritama koji su formulirani za grafove te određuju put između dva vrha u grafu, i to na temelju odabira bridova koji čine put između dva vrha.

U općenitom slučaju graf G definira kroz dva skupa $G=(V,E)$, gdje je V skup vrhova, a E je skup bridova, tj. parova vrhova (u,v). Grafovi se dijele na neusmjerene i usmjerene. Neusmjereni graf sadrži neuređene parove $\{u,v\}=\{v,u\}$ dok usmjereni graf ima bridove s usmjerenim parovima (u,v) [6].



Slika 1. Primjeri neusmjerenog (lijevo) i usmjerenog (desno) grafa

Nadalje, put se definira kao uređen (sortiran) skup bridova po kojima se prolazi, a duljina puta je zbroj težina (vrijednosti) bridova koji se nalaze na putu. Težine udaljenosti između dva vrha se mogu spremati na različite načine, ponajprije kao liste susjedstva i matrice susjedstva.

Među najčešće korištene algoritme za pronalaženje najkraćeg puta između dva vrha u grafu spada Dijkstrin algoritam, te se koristi čak i u području umjetne inteligencije [7]. Dijkstrin algoritam pronalazi optimalan put, ali zato mora provjeriti svaki vrh dok ne dođe do cilja. Kako bi se ubrzao proces pronalaženja puta, postoji mogućnost korištenja heurističkih algoritama, vrste algoritama koji su brži, ali zato nisu egzaktni.

Heuristički algoritmi su vrsta algoritama koji koristeći različite metode ubrzavaju rješavanje problema, ali zauzvrat gube na kvaliteti rješenja. Metode se temelje na teorijskim pretpostavkama, rezultatima eksperimenata ili jednostavno na iskustvu. Algoritmi tom metodom dobivaju na brzini, ali smanjuju točnost, preciznost i cjelovitost rješenja. Upravo zbog tih razloga heuristički algoritmi ne daju nužno najkraći put.

Glavni heuristički algoritmi koji se koriste u rješavanju problema sinteze optimalne trajektorije su A* i D*. Primjerice, A* algoritam u svom izračunu minimizira funkciju *f(n)=g(n)+h(n)*, gdje je *g(n)* funkcija težine brida dok je *h(n)* heuristička funkcija koja se koristi da bi se algoritam ubrzao.

2.1.1.1. Kompleksnost

Kompleksnost algoritma se definira preko O-notacije, koja pokazuje red veličine funkcije algoritma [8].

Dok O-notacija predstavlja gornju granicu, **Ω**-notacija predstavlja donju granicu, stoga se u računarstvu češće koristi O-notacija [9]

$$O\big(g(n)\big) = \{f(n): postoje\ konstante\ C, n_0 > 0\ takve\ da\ vrijedi\ 0 \leq f(n) \leq Cg(n)\ za\ sve\ n > n_0\} \quad \textbf{(1)}$$

$$\Omega\big(g(n)\big) = \{f(n): postoje\ konstante\ C, n_0 > 0\ takve\ da\ vrijedi\ 0 \leq Cg(n) \leq f(n)\ za\ sve\ n > n_0\} \quad \textbf{(2)}$$

Također se može koristiti Θ-notacija koja sadrži i donju i gornju granicu.

$$\Theta\big(g(n)\big) = \{f(n): postoje\ konstante\ C_1, C_2, n_0 > 0\ takve\ da\ vrijedi\ \ 0 \leq C_1 g(n) \leq f(n) \leq C_2 g(n)\ za\ sve\ n > n_0\} \quad \textbf{(3)}$$

Takva se notacija može definirati i tako da ako je *f(n)=O(g(n))* i *f(n)=Ω(g(n))* onda vrijedi i *f(n)=Θ(g(n))* [10].

Kod algoritma se kompleksnost može odnositi na vrijeme potrebno da se izvrši, ali i količinu memorije potrebnu tijekom procesa. U slučaju algoritma za pronalaženje puta najčešće se razmatra vremenska kompleksnost.

2.1.1.2. Usporedba algoritama

U tablici [Tablica 1] su prikazane glavne razlike između navedenih algoritama. Primjerice, kompleksnost Dijkstrinog algoritma ovisi o skupu bridova *E* i skupu vrhova *V* dok kompleksnost A* algoritma ovisi samo o skupu bridova, što je za taj algoritam proporcionalno potenciji s bazom *b*, što je faktor račvanja u A* algoritmu i eksponentom *d* koji označava dubinu traženog puta (broj vrhova kroz koji se prolazi tim putem). Po tim glavnim stavkama se može vidjeti da je prednost Dijkstrinog algoritma optimalnost, ali mu je zato mana veća vremenska kompleksnost. A* algoritam ju također posjeduje pod uvjetom prihvatljivosti heurističke funkcije *h(n)*, što znači da funkcija nikada ne precjenjuje stvarnu vrijednost brida. Također valja naglasiti da je u slučaju *h(n)=0* A* algoritam ekvivalentan Dijkstrinom algoritmu. Posljednji algoritam D*, je već samim imenom povezan s A* algoritmom, jer naziv proizlazi od Dynamic A*, a specifično se radi o tome da se *h(n)* u D* algoritmu može mijenjati po potrebi. Od tud se može izvesti zaključak da je Dijkstrin algoritam posebni slučaj A* algoritma, koji je sam posebni slučaj D* algoritma.

Tablica 1. Usporedba heurističkih algoritama

	Dijkstra	A*	D*
Kompleksnost	$\Theta(\|E\|+\|V\|\log\|V\|)$	$O(\|E\|)=O(b^d)$	Varijabilno
Optimalni put	Da	Ako je heuristička funkcija prihvatljiva (admissible)	Ne, ovisi o pretpostavkama
Okolina	Statična	Statična	Dinamična

Kako se u ovome radu pretpostavlja statična okolina te se traži optimalna trajektorija, za sintezu će se koristiti Dijkstrin algoritam.

3. DIJKSTRIN ALGORITAM

Dijkstrin algoritam je algoritam za traženje najkraćeg puta, osmislio ga je E. W. Dijkstra 1956. godine i objavio 1959. godine. Dijkstra opisuje problem traženja najkraćeg puta po nenegativnim bridovima grafa (u izvornom tekstu stabla) od izvora *P* do cilja *Q* [11].

3.1. Definicija problema

Sa zadanim grafom *G*, na kojem se nalaze vrhovi *P* i *Q*, tražimo najkraći put od *P* do *Q* (što također može biti i najkraći put od *Q* do *P*, ako je graf neusmjeren). Put se, kao što je već spomenuto u potpoglavlju 2.1.1, sastoji od vrhova i bridova između tih vrhova. Duljina puta je zbroj težina bridova, što znači da za pronalažanje najkraćeg puta želimo minimizirati funkciju duljine puta.

3.2. Dijkstrin algoritam

Dijkstrin algoritam pronalazi najkraće puteve od izvornog vrha *P* (u pseudokodu *s*) do svakog drugog vrha unutar skupa vrhova V grafa *G*, $V[G]$. Proces se započinje incijaliziranjem s funkcijom *Initialize-Single-Source()*, čiji je pseudokod prikazan na slici.

```
INITIALIZE-SINGLE-SOURCE(G, s)
1  for each vertex v ∈ V[G]
2        do d[v] ← ∞
3           π[v] ← NIL
4  d[s] ← 0
```

Slika 2. Pseudokod za funkciju *Initalize-Single-Source()* [6]

Inicijalizacija uzima svaki vrh *v* skupa vrhova *V* koji pripadaju grafu *G* i uzima da je u početku atribut *d[v]*, koji predstavlja gornju granicu najkraćeg puta od *s* do *v*, koji se u literaturi također naziva procjena najkraćeg puta [6]. Drugi atribut *π[v]* je prethodnik, koji označava vrh koji prethodi traženome vrhu *v* u najkraćem putu. Ispočetka će svaki prethodnik biti NIL, odnosno prazna vrijednost. Konačno će se atributu *d[s]* za početni vrh *s* dodijeliti vrijednost 0, jer je duljina puta od vrha do samog sebe jednaka nuli.

Nakon inicijalizacije izvora stvara se ispočetka prazan skup ***S***, u koji se prebacuju vrhovi do kojih su najkraći putevi određeni. Potom se stvara skup ***Q***, koji je ispočetka jednak skupu vrhova *V* grafa *G,* ***Q***=*V[G],* a sadrži elemente sortirane po *d[v]* atributu (po vrijednosti tog atributa, od manjeg prema većem). Dijkstrin algoritam će se izvršavati dok god skup ***Q*** ne isprazni, odnosno dok se skup ***S*** ne popuni. Ovo se tipično izvodi koristeći *while* petlju.

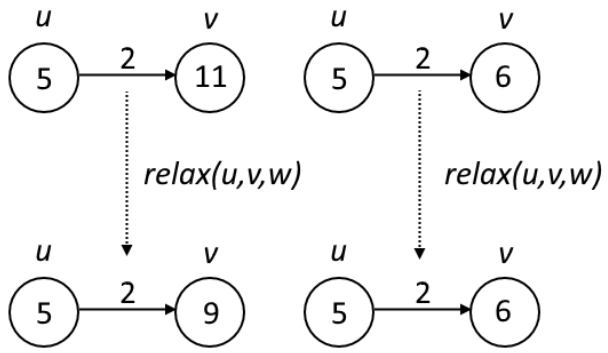
Unutar petlje algoritam će izvući vrh iz skupa ***Q*** s minimalnom vrijednosti atributa d[] i pridodati tu vrijednost vrhu *u*. Pseudokod za funkciju *Extract-min()* neće biti prikazan jer ovisi o odabranoj strukturi podataka. Ova funkcija iz zadanog skupa izuzima element s najmanjom vrijednosti. Kako je vrh *u* izvučen iz skupa ***Q*** potrebno ga je dodati skupu ***S***. U danom pseudokodu se koristi unija $\boldsymbol{S} \leftarrow \boldsymbol{S} \cup \{u\}$.

Konačno se za svaki vrh *v* koji je susjed vrha *u* provodi tzv. *opuštanje* bridova (eng. *relax*)između vrhova *v* i vrha *u* koristeći vrijednosti bridova u skupu *w*. Funkcija *relax(u,v,w)* provjerava je li trenutna procjena najkraćeg puta do vrha *v* veća od zbroja iznosa duljine brida između *u* i *v*, *w(u,v)* i procjene najkraćeg puta vrha *u*. U slučaju da jest, onda se procjeni najkraćeg puta vrha *v*, *d[v]* dodjeljuje prethodno spomenuti zbroj. Uz to se postavlja vrh *u* kao atribut prethodnika vrha *v*, *π[v]*.

```
RELAX(u, v, w)
1   if d[v] > d[u] + w(u, v)
2      then d[v] ← d[u] + w(u, v)
3           π[v] ← u
```

Slika 3. Pseudokod opuštanja brida [6]

Na narednoj slici je prikazan primjer opuštanja brida u slučaju da je gore prikazana nejednakost unutar *if* petlje zadovoljena i u slučaju da ta nejednakost nije zadovoljena.

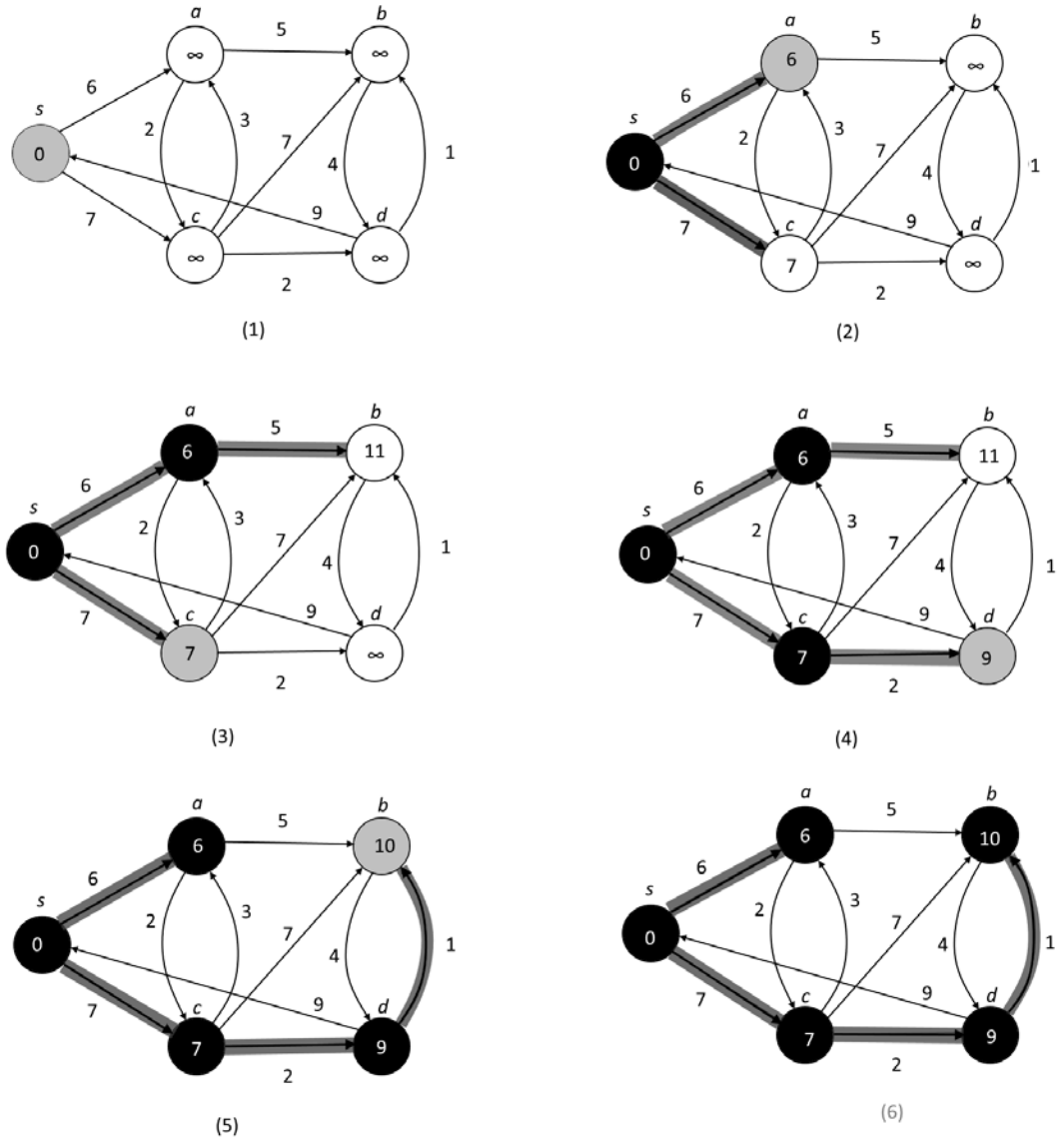


Slika 4. Primjer opuštanja brida. Lijevo je u slučaju zadovoljene nejednakosti, a desno u slučaju kad jednakost nije zadovoljena.

Kako su sada poznati svi elementi Dijkstrinog algoritma možemo predstaviti njegov pseudokod, kako slijedi.

```
DIJKSTRA(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  S ← ∅
3  Q ← V[G]
4  while Q ≠ ∅
5       do u ← EXTRACT-MIN(Q)
6          S ← S ∪ {u}
7          for each vertex v ∈ Adj[u]
8               do RELAX(u, v, w)
```

Slika 5. Pseudokod Dijkstrinog algoritma [6]



Slika 6. Primjer Dijkstrinog algoritma s 5 vrhova

Radi ilustracije Dijkstrinog algoritma, na Slici 6. prikazan je primjer s relativno malim grafom. U prikazanome primjeru se Dijkstrin algoritama izvršava s izvorom *s* koji se nalazi krajnje lijevo. Trenutna procjena najkraćeg puta u svakom koraku algoritma je broj koji se na slici nalazi unutar vrhova, a markirani bridovi predstavljaju bridove najkraćeg puta, te se gledajući od konačnog vrha prema početnome tim putem mogu odrediti prethodnici. Bijeli vrhovi su još u skupu ***Q***, a trenutni minimum iz tog skupa (vrh *u*) je obojan sivo. Zacrnjeni vrhovi su oni koji su prebačeni u skup ***S***. Prikaz 1 pokazuje stanje nakon izvršavanja funkcije *initialize-single-source()* i prve iteracije *while* petlje. Svaki sljedeći prikaz pokazuje narednu iteraciju *while* petlje. Posljednji prikaz 6 predstavlja konačno stanje nakon izvršavanja cijelog algoritma.

Općenito je kompleksnost Dijkstrinog algoritma $\Theta(|E|T_{rel}+|V|T_{em})$, gdje je T_{rel} vrijeme izvršavanja *relax()* funkcije dok je T_{em} vrijeme izvršavanja *extract-min()* funkcije. U svom najjednostavnijem obliku se koristi lista, a *extract-min()* se svodi na linearno pretraživanje te liste. U tom slučaju je kompleksnost $\Theta(|E|+V^2)= \Theta(V^2)$ [12].

U slučaju pojave negativne težine brida, može se dogoditi da je potrebno naknadno promijeniti najkraći put, ali kako je Dijkstrin algoritam pohlepan (greedy) i ne provjerava vrhove ponovno nakon što su prebačeni u otvoreni skup, rezultat će biti da najkraći put do nekog vrha nije točan, stoga se kod Dijkstrinog algoritma koriste isključivo grafovi s nenegativnim bridovima, dakle bridovi s težinom ≥ 0 [12]. Ako želimo uzimati u obzir i negativne bridove preporuča se korištenje Bellman-Ford algoritma [6].

3.3. Dodatna poboljšanja

Moguće je dobiti i manju kompleksnost algoritma od one prikazane u prethodnom poglavlju, ovisno o strukturama podataka koje su korištene i o uvjetima koji se pretpostavljaju.

Prvo se uzima u obzir način spremanja informacija o susjedima. Predstavlja se mogućnost korištenja ili liste ili matrice susjeda. Lista susjeda je adekvatnija za rijetke grafove, dakle one u kojima broj bridova E uvelike manji od V^2, kvadrata broja vrhova. Naspram tome, matrica susjeda daje bolji rezultat ukoliko se radi o gustom grafu gdje vrijedi da je E veći od V^2 [6][12].

U specifičnim slučajevima, primjerice kada vrijednosti bridova mali, može se uz različite strukture podataka smanjiti kompleksnost [13][14][15][16]. U specifičnome slučaju cjelobrojnih težina i neusmjerenog grafa može se dodatno smanjiti kompleknost na $\Theta(|E|)$ [17].

Smanjenje kompleksnosti na temelju strukture podataka se oslanja na apstraktnu strukturu podataka, redove s prioritetom minimuma da bi se dobila kompleksnost $\Theta((|E|+|V|)log|V|)$ [6]. Poboljšanje se tada temelji na smanjenju kompleknosti funkcija *extract-min()* i *relax()*. Realizacija prethodno navedenih redova se dobiva u obliku hrpi, a za njih vrijedi da je funkcija *extract-min()* vremenske kompleksnosti $\Theta(1)$, dakle u konstantnome vremenu. Kako je funkcija *extract-min()* ista, odabir tipa hrpe će ovisiti o kompleksnosti izvršavanja funkcije *relax()*, što u slučaju hrpi ovisi o funkciji *decrease-key()* koja mijenja vrijednosti (argumente) hrpe, zbog toga što nakon izvlačenja minimuma iz hrpe potrebno je istu restrukturirati, jer tada hrpa ima novi minimum. Tako će za općeniti slučaj, najbolju

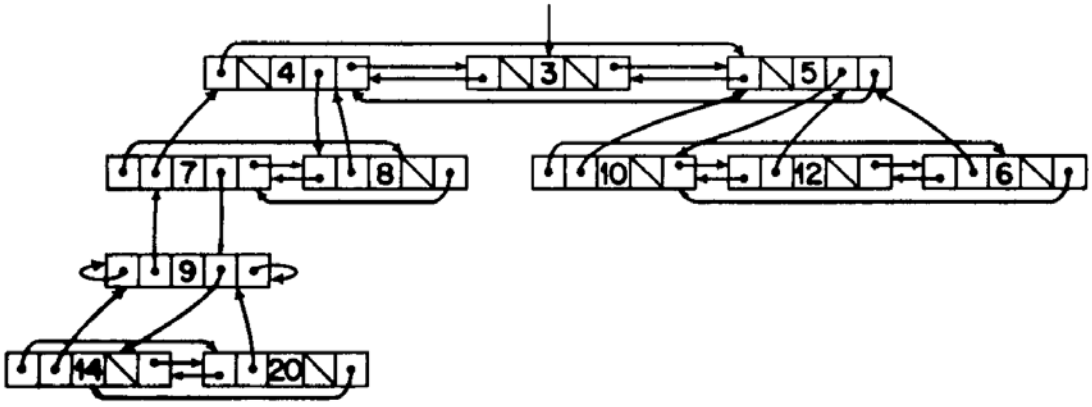
kompleksnost postizati Fibonaccijeve hrpe, čija *decrease-key()* funkcija ima kompleksnost $\Theta(1)$[6].

3.3.1. Fibonaccijeve hrpe

Fibonaccijeva hrpa je struktura podataka koja se sastoji od stabala sortiranih po redu hrpa koje zadovoljavaju svojstvo minimalnog korijena, što znači da je korijen uvijek minimum stabla, što i omogućuje već spomenuto brzo određivanje minimuma.

Dodatno pravilo je da čvor ranga k ima najmanje $F_{k+2}>\varphi^k$ djece, uključujući sebe, gdje je F_k k-ti Fibonaccijev broj ($F_0=0$, $F_1=1$, $F_k=F_{k-1}+F_{k-2}$). Odatle potječe i naziv Fibonaccijeve hrpe [18].

Kompleksnost funkcija proizlazi iz te posebne strukture hrpe, koja je prikazana na Slici 3. gdje se može vidjeti da je primjerice pokazivač (pointer) uvijek inicijalno u vrhu, koji je zbog definicije hrpe uvijek minimum.



Slika 7. Fibonaccijeva hrpa s naznačenim pokazivačima (pointerima)[Fib]

Stoga Fibonaccijeve hrpe imaju kompleksnost *O(1)* za *extract-min()* (koji je zapravo ekvivalentan funkciji *find-min()*) i kompleknost *O(1)* za *decrease-key()* [18] pa se ukupna kompleksnost Dijkstrinog algoritma s implementiranim Fibonaccijevim hrpama svodi na $\Theta(|E|+|V|log|V|)$[6][12].

Unatoč najboljoj kompleksnosti, implementacija Fibonaccijevih hrpi u praksi ima više problema. Ponajprije, implementacija Fibonaccijevih hrpa u programskim jezicima je često zahtjevna [19][20]. Također, zbog velikih vremenskih konstanti, koje se u O i Θ notaciji zanemaruju, i korištenja velike količine memorije po čvoru rezultira u tome da u stvarnim

slučajevima ne dolazi do velikog smanjenja vremena računanja, makar novija istraživanja pokazuju da rezultiraju u boljoj kompleksnosti od različith vrsta hrpa [21].

4. FORMULACIJA PROBLEMA

Da bi se riješio problem sinteze optimalne trajektorije potrebno je modelirati problem na takav način da ga algoritam može riješiti.

Zadan je prostor (okolina), sve prepreke i ostala eventualna ograničenja na gibanje. Zadane su dvije pozicije, početna točka i krajnja točka, odnosno cilj. Problem se definira kao problem generiranja optimalne trajektorije koja te dvije točke ima kao krajnje. Trajektorija mora obilaziti prepreke i držati se eventualnih ograničenja na gibanje.

4.1. Formulacija prostora

U ovome radu će se pretpostavljati $\mathbb{R}^2$ prostor, dakle dvodimenzonalna površina. Taj radni prostor, $\boldsymbol{W} \subset \mathbb{R}^2$, će moći sadržavati prepreke (obstacles) $\boldsymbol{OB} \subset \boldsymbol{W}$, a unutar prostora će se kretati robot $\boldsymbol{A}$ s konfiguracijom $\boldsymbol{\gamma}$, koja sadrži eventualna ograničenja na gibanje, po koordinatnom sustavu vezanom uz $\boldsymbol{W}$ [22].

Takav robot $\boldsymbol{A}$ će se moći kretati u slobodnome prostoru $\boldsymbol{W_{free}}$, što je prostor koji ne uključuje prepreke $\boldsymbol{OB}$ [23]:

$$\boldsymbol{W_{free}} = \{\boldsymbol{W} \setminus \boldsymbol{OB}\} \tag{4}$$

5. OPIS RJEŠENJA

Nakon što je formuliran problem, potrebno je prilagoditi zadane argumente problema (prostor, početna točka i cilj) Dijkstrinom algoritmu.

5.1. Diskretizacija prostora

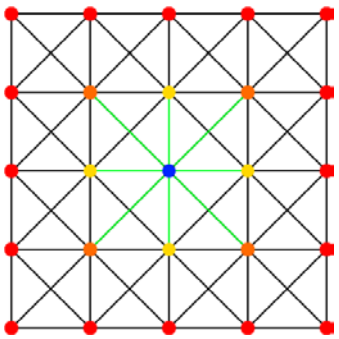
Dijkstrin algoritam kao ulaz uzima graf, što je diskretni oblik prostora, stoga je za rješavanje problema potrebno od kontinuiranog skupa $\boldsymbol{W}$ ili $\boldsymbol{W_{free}}$ dobiti diskretizirani skup $\boldsymbol{W_d}$. Diskretni skup će se dobiti tako da se odabere skup vrhova kod kojeg vrijedi [24]:

$$\boldsymbol{W_d} = \{w_i \in \boldsymbol{W} \mid 1 < i \leq N\} \tag{5}$$

U jednadžbi je evidentno da će gustoća diskretnog grafa ovisiti o *N*, broju vrhova uzetih iz kontinuiranog skupa (uz pretpostavku ravnomjerne raspodjele vrhova u kontinuiranom prostoru). Detalji realizacije će biti detaljnije opisani u sljedećem poglavlju.

5.2. Formulacija geometrije

Da bi se navedene udaljenosti mogle definirati potrebno je odrediti geometriju prostora. U ovom slučaju će se uzeti kvadratna (pravokutna) mreža vrhova, a robot će se moći gibati prema horizontalnim i vertikalnim susjedima, ali i prema dijagonalnima. Na slici 8. se može vidjeti primjer mreže vrhova, s plavim izvornim vrhom, njegovim žutim horizontalnim ili vertikalnim susjedima i narančastim dijagonalnim susjedima. Zeleno su označeni bridovi između plavoga vrha i susjeda. Crveni su označeni ostali vrhovi u mreži.



Slika 8. Primjer mreže vrhova

U jednadžbi (6) se vidi da će dijagonalna udaljenost biti za $\sqrt{2}$ veća od horizontalne, što odgovara euklidskoj normi [25].

$$d = 1$$
$$d_2 = \sqrt{2} \quad \textbf{(6)}$$

Ta udaljenost će se naknadno u sklopu samoga grafa uvećati za faktor koji ovisi o udaljenosti neposrednih susjeda u smjeru x ili y osi, odnosno Manhattan normi.

5.3. Odabir strukture podataka

Kako će se po geomtriji u ovome slučaju raditi o relativno gustome grafu, što znači da je broj bridova E veći od kvadrata broja vrhova V^2, graf će se spremiti u obliku matrice susjeda, jer je takav oblik prikladniji za guste grafove [6][12].

5.4. Sinteza optimalne trajektorije

S dobivenim grafom su pripremljeni svi ulazni podatci potrebni za Dijkstrin algoritam. Kako se s Dijkstrinim algoritmom dobiva najkraći put od odabranog izvora s do svakog drugoga vrha, za dobivanje tražene optimalne trajektorije do ciljnog vrha q potrebno je pozvati procjenu najkraćeg puta $d[q]$, nakon što algoritam izvrši jer je upravo to vrijednost najkraćeg puta od s do q. Prateći niz prethodnika, počevši od q, dobiva se optimalna trajektorija.

6. PROGRAMSKI PAKET S PRIMJERIMA

Programski paket je napisan u programskome jeziku MATLAB zbog jednostavne manipulacije matrica i grafičkog prikaza. Uz opis funkcija koje pripadaju programskome paketu, također će se prikazati odabrani ilustrativni primjeri na kojima će se vidjeti kako bi se mobilni robot kretao koristeći ovdje razvijeni programski paket.

6.1. Opis funkcija

Programski paket se sastoji od tri funkcije, glavne su *dgraf* funkcija koja generira mrežu točaka (vrhova) koristeći koordinate prostorije i prepreka kao ulaze. Druga funkcija *dijkstra* je implementacija Dijkstrinog algoritma u MATLAB-u, koja pak koristi podatke dobivene *dgraf* funkcijom kao ulaz. Pozivom funkcije *dijkstra()* dobiva se vrijednost najkraćeg puta kao i skup vrhova koje se na tom putu nalaze. Također je u paket dodana treća funkcija, pomoćna funkcija *koord*, preko koje određujemo indekse koordinata najbližemu proizvoljnome vrhu. Navedena funkcija stoga omogućuje proizvoljno biranje početnog odnosno krajnjeg vrha puta.

6.1.1. dgraf()

Funkcija *dgraf()* služi za diskretizaciju kontinuiranog prostora i dobivanja grafa u obliku matrice susjedstva, ali i dodatnih varijabli za grafički prikaz. Sintaksa poziva funkcije je sljedeća:

```
[dadj, xd, yd, xpk, ypk, varargout]=dgraf(xp, yp, rez, varargin)
```

Ulazni argumenti u funkciju su vektori stupci `xp` i `yp`, koji predstavljaju x odnosno y koordinate granica prostora ***W*** u kojem će se robot gibati. Argument `rez` (rezolucija) specificira horizontalnu, odnosno vertikalnu udaljenost između dva susjeda u mreži vrhova. Ta tri navedena argumenta su obavezna za funkcioniranje funkcije *dgraf()*. Sljedeće je `varargin` što u MATLAB-u označava varijabilni broj ulaza. U ovome slučaju varijabilan je broj prepreka ***OB***. Prepreke se trebaju unositi kao i `xp` i `yp`, dakle u obliku vektora stupaca koji sadrže koordinate granica prepreka ***OB***.

Primjerice, moguće je u funkciju *dgraf()* unijeti `dgraf(xp, yp, rez, xob1, yob1, xob2, yob2)` što znači da će se raditi o prostoru s dvije različite prepreke. Valja naglasiti da je obavezno unijeti paran broj varijabilnih ulaza, pošto je svaka prepreka u dvodimenzionalnome prostoru definirana s x i y koordinatama. Nadalje, zbog toga što se unutar funkcije *dgraf()* koristi MATLAB-ova funkcija *boundary()* [26] i *inpolygon()* [27], potrebno je uvijek uvrštavati koordinate u obliku vektora stupca.

Funkcija će uzeti ulazne argumente i s njima generirati mrežu točaka u obliku pravokutnika koja prekriva prostor. Uz to će se stvoriti inicijalna matrica susjeda za tu mrežu.

Zatim će se provjeriti koji se vrhovi nalaze unutar granica prostora, a izvan granica prepreka koristeći funkciju *inpolygon()*. Vrhovi koji se ne nalaze u tom području će se odbaciti, a uz to će se također i odbaciti svi vrhovi koje se nalaze u neposrednoj blizini nekog ruba, što će uvelike smanjiti slučajeve gdje bi hipotetski robot bio bliže rubu nego što mu vlastite dimenzije dopuštaju. Koristeći te informacije o odbačenim vrhovima, izbrisat će se redovi i stupci matrice susjeda u kojima su definirane udaljenosti (vrijednosti bridova) tih vrhova do njihovih susjeda.

Tako će se dobiti izlazni argumenti. Prvi izlazni argument je `dadj`, koji predstavlja matricu susjeda potrebnu za Dijkstrin algoritam. Sljedeći argumenti nisu potrebni za algoritam, ali se koriste u grafičkom prikazu rješenja. Vektori `xd` i `yd` skupa čine mrežu točaka odnosno vrhova potrebnih za grafički prikaz. Vektori `xpk` i `ypk` predstavljaju ulazne vektore `xp` i `yp`, dakle granicu prostora, ali u obliku koji je adekvatan za grafički prikaz koristeći MATLAB-ovu funkciju *plot()*. Na sličan način funkcioniraju varijabilni izlazi `varargout`. Ti izlazi predstavljaju granice prepreka, isto u adekvatnom obliku.

Tako će se za prethodno navedeni primjer moći dobiti izlazi

```
[dadj, xd, yd, xpk, ypk, xobk1, yobk1, xobk2, yobk2]
    =dgraf(xp, yp, rez, xob1, yob1, xob2, yob2)
```

gdje `xobk1`, `yobk1`, `xobk2`, `yobk2` predstavljaju izlazne oblike varijabilnih ulaza.

ronačno će se dobiti matrica susjedstva, mreža vrhova koji zadovoljavaju već prethodno navedene uvjete i koordinate granica u adekvatnom obliku za grafički prikaz.

6.1.2. dijkstra()

Sljedeća funkcija je *dijkstra()*, uz pomoć koje se dobiva najkraći put. Ulazi u funkciju su matrica susjeda, početni vrh i konačni vrh.

```
[min_put, dn] = dijkstra(D_ADJ, ds, dt)
```

Ulazi u funkciju su `D_ADJ` što je matrica susjeda dobivena funkcijom *dgraf()* i indeksi u matrici susjeda početnoga vrha `ds` i konačnoga vrha `dt`. Algoritam se ponaša slično originalnoj formulaciji, jedino se prebacivanje iz skupa $\boldsymbol{Q}$ u otvoreni skup $\boldsymbol{S}$ zamjenjuje vektorom `S`, koji je inicijalno popunjen nulama, a svaki put kada se neki vrh prebaci, nula se na odgovarajućem indeksu u vektoru zamjenjuje s jedinicom. Uz to se dodaje još jedna petlja preko koje se dobiva vektor indeksa točaka na najkraćem putu. Vektor se dobiva tako da se počevši od indeksa konačnoga vrha dodaju indeksi na početak vektora, tako da svakome vrhu prethodi njegov najbliži susjed. Indeksi točaka odgovaraju atributu $\pi[v]$ spomenutome u poglavlju 3.

Izlazi algoritma su iznos najkraćeg puta `min_put` (zbroj vrijednosti bridova koji se nalaze na putu) i vektor indeksa vrhova od početnog vrha do cilja `dn`.

Kako se radi o vrlo jednostavnoj realizaciji algoritma potrebno je napomenuti da ako gustoća mreže nije dovoljno velika, može se dogoditi da jedan ili više vrhova budu kompletno odvojeni od ostalih, dakle pojava svojevrsnog otoka vrhova koji nemaju nikakvu vezu s ostalim vrhovima, što će rezultirati u beskonačnoj petlji jer ih algoritam neće moći označiti kao opuštene. Uzimajući taj problem u obzir, treba birati gustoću mreže u funkciji *dgraf()* na takav način da se svaka točka u rezultirajućoj mreži bude indirektno povezana s ostalim vrhovima.

6.1.3. koord()

Funkcija *koord()* je pomoćna funkcija koja omogućuje jednostavno biranje početnoga odnosno konačnoga vrha.

```
[k_ind] = koord(xa,ya,xd,yd)
```

Ulazni argumenti `xa` i `ya` su koordinate točke za koju se traži najbliži vrh. Ulazni argumenti `xd` i `yd` su koordinate mreže. Izlazni argument je `k_ind` indeks traženoga vrha.

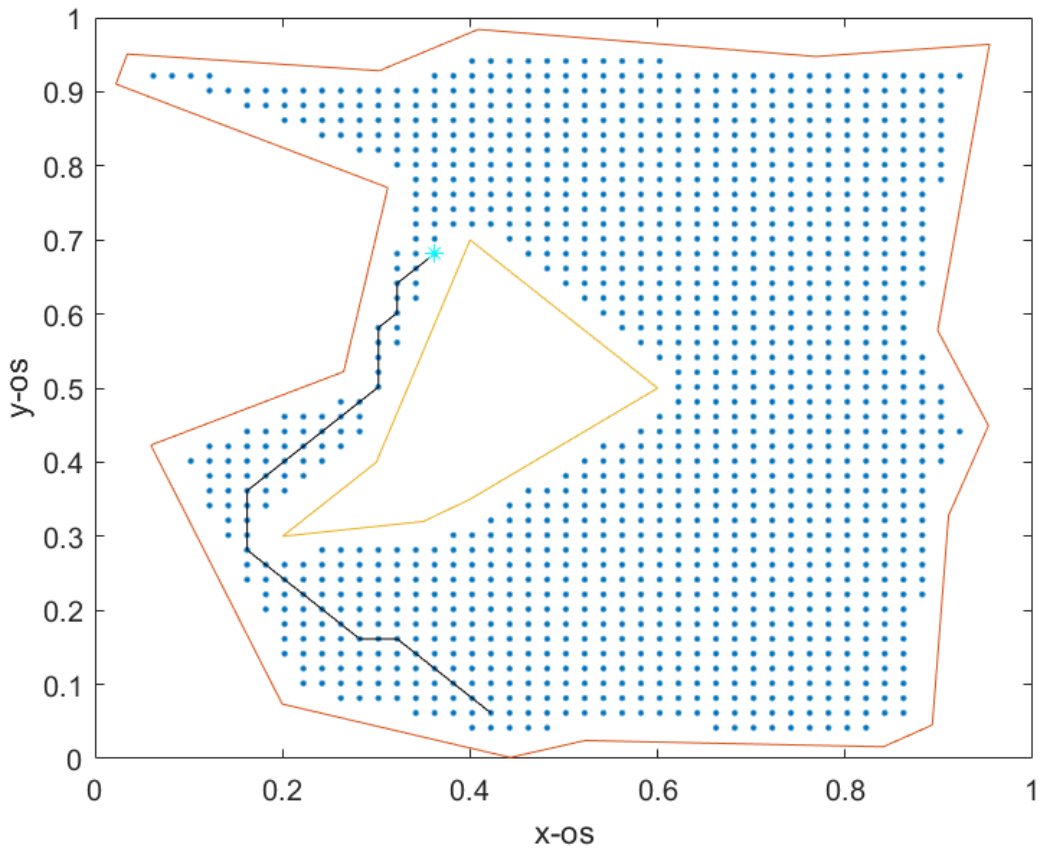
Druga opcija je da se naknadno u mrežu doda željeni vrh ili vrhovi, što podrazumijeva i dodavanje vrijednosti bridova u matricu susjeda.

6.2. Primjeri

Uz funkcije su također dani primjeri. Primjeri uključuju nasumično generirane prostore i prepreke, koji su ilustrativni jer jasno pokazuju kako bi se mobilni robot mogao kretati.

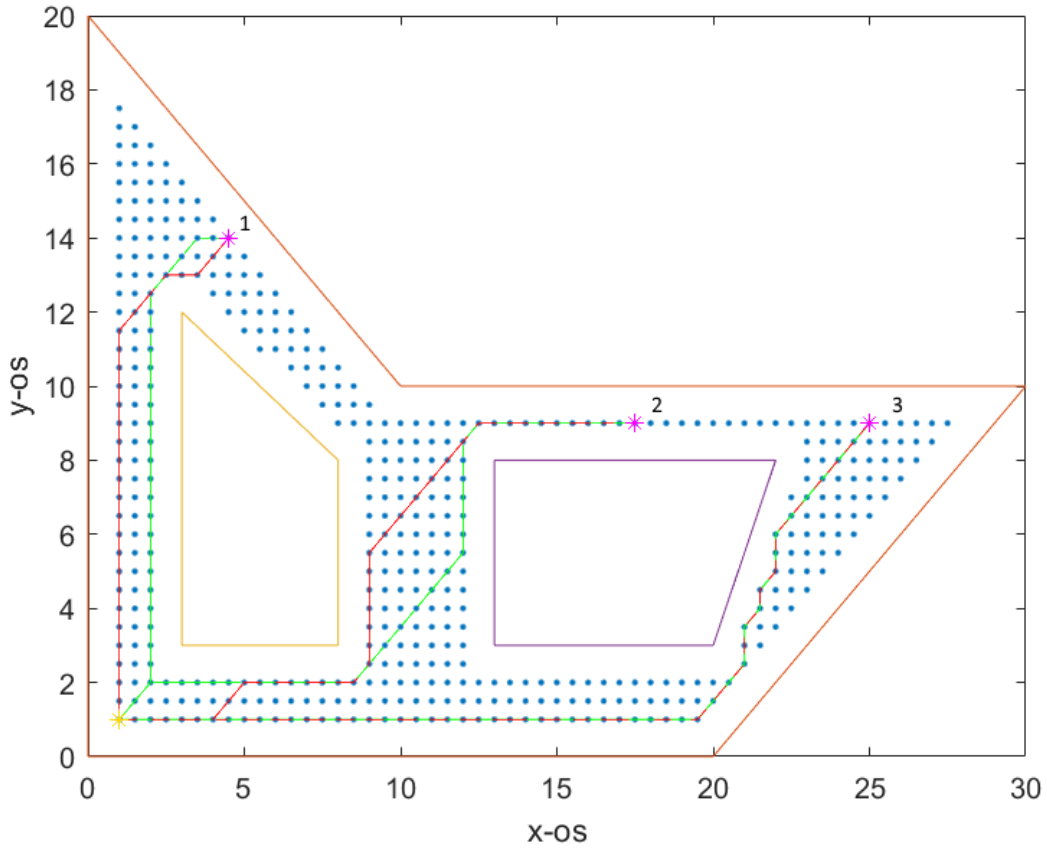
Na svakom grafičkome prikazu nalaze se rubovi prostorije i prepreka, mreža vrhova po kojoj se robot može gibati i put po kojem se robot giba od početnoga do konačnoga vrha. Konačni vrh je posebno istaknut da bi se znao smjer u kojem se robot kreće.

Na prvome primjeru se vidi nasumično generirani skup vrhova s jednom preprekom. Na tom se primjeru također može vidjeti kako robot prolazi kroz područje koje bi moglo biti prethodno spomenuti otok vrhova odvojen od ostalih da gustoća točaka nije dovoljno velika. Tako je u ovome primjeru odabrana minimalna udaljenost do susjednoga vrha (rezolucija) u iznosu 0,02.



Slika 9. Prvi primjer, nasumično generirana prostorija s preprekom. Konačni vrh označen je zvijezdicom

U drugome primjeru se može vidjeti prostorija s dvije prepreke, te su istovremeno prikazana tri para puteva. Svaki par puteva se sastoji od crvenoga puta, čiji je početni vrh (izvor) u donjem lijevom kutu prostorije, a konačni vrh (cilj) je označen magenta zvijezdom. Svaki crveni put ima svog zelenoga para, čija trajektorija ima iste krajeve i iznos, ali su zamijenjeni izvor i cilj pa je tako svim zelenim putevima cilj u donjem lijevom kutu prostorije, označen zlatnom zvijezdom.



Slika 10. Drugi primjer s tri para puteva. Crvenim putevima je cilj magenta zvijezda, dok je zelenima ciljni vrh žuta zvijezda.

Na tim primjerima se može vidjeti kako algoritam bira put i kako u slučaju više mogućih najkraćih puteva rješenje može biti drukčije (glede odabranih vrhova, iznos puta je uvijek isti) ovisno o tome koji je vrh početni a koji ciljni.

Ta razlika proizlazi iz toga što se u funkciji *dijkstra()* put određuje tako da se počevši od konačnog vrha dodaju prethodnici koji su najbliži početnom vrhu, na način biranja puta se može gledati na takav način da, gledajući od cilja, put se stvara tako da se svakim korakom maksimalno približava početnome vrhu, odnosno podput je u pravilu najkraći mogući.

Iznimke se mogu pojaviti kao u prvome primjeru i trećem paru drugoga primjera, što je rezultat načina na koji funkcija *min()* bira izlaz u slučaju da više opcija odgovara, i to da uzima prvu ponuđenu opciju [28]. Stoga na odabir trajektorije u slučaju više mogućih rješenja također utječe smjer u kojem Dijkstrin algoritam uzima vrhove u obzir za opuštanje, slijeva nadesno i odozdo prema gore u ovom programskome paketu.

7. ZAKLJUČAK

Problem sinteze optimalne trajektorije predstavlja izazov u skupini problema navigacije mobilnih robota zbog različitih faktora kao što su uvjeti okoline i mogućnost dobivanja informacija o njoj.

Korištenje Dijkstrinog algoritma je odabrano s pretpostavkom uvjeta globalne sinteze trajektorije i statične okoline. Prednost korištenja ovog algoritma je što on daje optimalnu trajektoriju (unutar preciznosti definirane diskretizacije prostora).

Glavni nedostatak Dijkstrinog algoritma je značajnija kompleksnost algoritma, a proizlazi iz egzaktnosti, jer algoritam mora provjeriti sve dostupne vrhove.

Zbog tog nedostatka navedena su različita poboljšanja, koja se, za razliku od Dijkstrinog algoritma, u većini slučajeva temelje na posebnim slučajevima. Iako su ti algoritmi osmišljeni koristeći, primjerice heuristike, te pružaju suboptimalna rješenja, zbog navedene posebnosti suboptimalna rješenja mogu biti veoma blizu optimalnome.

LITERATURA

[1] Koubaa, A. et al. Introduction to Mobile Robot Path Planning. In: Robot Path Planning and Cooperation: Foundations, Algorithms and Experimentations. Springer International Publishing; 2018. 10.1007/978-3-319-77042-0_1.

[2] Zhang H, Lin W, Chen A. Path Planning for the Mobile Robot: A Review. Symmetry. 2018;10(10):450. 10.3390/sym10100450.

[3] Wang Y, Yu X, Liang X. Design and Implementation of Global Path Planning System for Unmanned Surface Vehicle among Multiple Task Points. International Journal of Vehicle Autonomous Systems. 2018;14:82. 10.1504/IJVAS.2018.093119.

[4] Nayl T. et al. Obstacles Avoidance for an Articulated Robot Using Modified Smooth Path Planning. In: 2017 International Conference on Computer and Applications (ICCA), 2017.

[5] Atef A. A. Optimal Trajectory Planning of Manipulators: A review. In: Journal of Engineering, Science and Technology Vol 2, No. 1, 2007.

[6] Cormen T.H, Leiserson C.E, Rivest R.L, Stein C. Introduction to Algorithms, The MIT Press, 2002.

[7] Felner A. Position Paper: Dijkstra's Algorithm Vs. Uniform Cost Search or A Case Against Dijkstra's Algorithm, 2011.

[8] Vitány P.M.B, Meertens L. Big Omega Versus the Wild Functions. 1985.

[9] Balcázar J L, Gabarró J. Nonuniform complexity classes specified by lower and upper bounds. In: Informatique théorique et applications, p. 177-194. 1989.

[10] Knuth D. Big Omicron and big Omega and big Theta. 1976.

[11] Dijkstra E.W. A Note on Two Problems in Connexion with Graphs. In: Numerische Mathematik. 1959.

[12] Mehlhorn K, Sanders P. Algorithms and Data Structures: The Basic Toolbox. Springer, 2007.

[13] Dial R. B. Algorithm 360: Shortest-path forest with topological ordering [H]. Communications of the ACM. 1969;12(11):632–633.

[14] Ahuja R.K. et. al. Faster Algorithms for the Shortest Path Problem. In: Journal of the ACM. 1990;37(2): 213–223.

[15] Thorup M. On RAM Priority Queues. In: SIAM Journal on Computing. 2000;30 (1): 86–109.

[16] Raman R. Recent results on the single-source shortest paths problem. In: SIGACT News. 1997;28 (2): 81–87.

[17] Thorup M. Undirected single-source shortest paths with positive integer weights in linear time. In: Journal of the ACM. 1999;46 (3): 362–394.

[18] Freedman M.L, Tarjan R.E. Fibonacci heaps and their uses in improved network optimization algorithms. In: Journal of the Association for Computing Machinery. 1987;34 (3): 596–615.

[19] https://maryrosecook.com/blog/post/the-fibonacci-heap-ruins-my-life Citirano 17. rujna 2020.

[20] https://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/FibonacciHeaps.pdf Citirano 17. rujna 2020.

[21] Larkin D, Sen S, Tarjan R. A Back-to-Basics Empirical Study of Priority Queues. In: Proceedings of the Sixteenth Workshop on Algorithm Engineering and Experiments. 2014;61–72.

[22] Latombe J-C. Robot Motion Planning. Springer, 1991.

[23] Barraquand J, Latombe J-C. Robot Motion Planning: A Distributed Representation Approach. In: The International Journal of Robotics Research, 1991;10(6):628-649.

[24] Yershov D, LaValle S. Simplicial Dijkstra and A∗ Algorithms: From Graphs to Continuous Spaces. In: Advanced Robotics, 2012;1-21.

[25] http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html Citirano 17. rujna 2020.

[26] https://uk.mathworks.com/help/matlab/ref/boundary.html Citirano 17. rujna 2020.

[27] https://uk.mathworks.com/help/matlab/ref/inpolygon.html Citirano 17. rujna 2020.

[28] https://uk.mathworks.com/help/matlab/ref/min.html Citirano 17. rujna 2020.

PRILOZI

I. MATLAB kôd

a. funkcija *dgraf()*

```
function [dadj, xd, yd, xpk, ypk, varargout]=dgraf(xp, yp, rez, varargin)
    %xp i yp su koordinate granica prostorije unutar kojeg se robot krece
    %varargin je celija u koja se sastoji od proizvoljnog broja prepreka
    %vrijednosti prepreka je potrebno unositi u istom formatu kao i
    %prostorije, dakle prvo koordinate osi x pa osi y jedne prepreke
    %nakon cega se moze na isti nacin dodati jos prepreka
    %rez govori koliki je razmak izmedu dvije susjedne tocke
    %u horizontalnom odnosno vertikalnom smjeru, svojevrsna rezolucija
    %dadj je matrica susjeda, potrebna za dijkstrin algoritam

    pr = boundary(xp,yp); %generiranje granice prostora
    %xo = [min(xp) max(xp) min(xp) max(xp)]; %stvaranje            pravokutnika
    %yo = [min(yp) min(yp) max(yp) max(yp)]; %           opisanog
    nop = (nargin -
 3)/2; %broj prepreka je manji za 3 zbog koordinata prostora i gustoce mreze, a pola zbog t
oga što za prepreku trebaju i x i y koordinate
    pre = cell(nop,2); %inicijaliziranje arraya, ubrzava se proces
    pre(:,1) = varargin(1:2:end).'; %x koordinata bi trebala biti svaki neparni varargin
    pre(:,2) = varargin(2:2:end).'; %y koordinata bi trebala biti svaki parni varargin
    kop = cell(nop,1); %incijaliziranje arraya za granicu
    for ik = 1:nop
        kop{ik} = boundary(pre{ik,1},pre{ik,2}); %dodavanje elemenata u array
    end
    %gen mreze
    xmg = min(xp):rez:max(xp); %odredivanje gustoce mreze
    ymg = min(yp):rez:max(yp);
    xymg = length(xmg)*length(ymg); %odredivanje ukupnog broja tocaka, ponajprije zbog jedn
ostavnosti
    eg = norm([rez rez]);      % razmak izmedu dva dijagonalna susjeda
    [xm, ym] = meshgrid(xmg,ymg); %meshgrid da bi se stvarno radilo o mrezi
    dadj = ones(xymg)*inf; %incijaliziranje matrice te postavljanje svih vrijednosti na inf
    for i1 = 1:xymg
        dadj(i1,i1) = 0; %tocka je sama sebi najbliza
        for j1 = 1:xymg
            if (i1-
j1 == 1)&&(mod(i1,length(xmg)) ~= 1) %susjedi, koji su u istom retku, mod provjerava radi l
i se o tocki na rubu
                dadj(i1,j1) = rez;
                dadj(j1,i1) = rez;
            end
            if (i1-j1) == length(xmg) %stupci
                dadj(i1,j1)=rez;
                dadj(j1,i1)=rez;
            end
            if  ((i1-j1) == length(xmg)+1)&&(mod(i1,length(xmg))~=1) %dijagonalno dolje-
lijevo, ako je modul 1, radi se o tocki na lijevom rubu,a ona nema poveznice dolje-lijevo
                dadj(i1,j1)=eg;
                dadj(j1,i1)=eg;     %je
            end
            if  ((i1-j1) == length(xmg)-1)&&(mod(i1,length(xmg))~=0) %dijagonalno dolje-
desno, ako je modul 0, radi se o tocki na desnom rubu, a ona nema poveznice dolje-desno
                dadj(i1,j1)=eg;
                dadj(j1,i1)=eg;
            end
        end
    end
```

```
    [inp, onp] = inpolygon(xm,ym,xp(pr),yp(pr)); %provjerava se je li tocka unutar prostora
    io_p = cell(nop,2); %inicijalizacija celije za pre
    for i_p = 1:nop %dodavanje vrijednosti u celiju
        [io_p{i_p,1},io_p{i_p,2}] = inpolygon(xm,ym,pre{i_p,1}(kop{i_p}),pre{i_p,2}(kop{i_p
}));
    end
    dap = inp & ~onp; %da bi tocka u ovom slucaju bila u prostoru mora takoder ne biti na g
ranici jer po defaultu se na granici smatra unutar
    for ida = 1:nop %petlja preko koje odredujemo da zelimo samo tocke koje jesu u prostoru
, ali nisu u prepreci
        dap = dap & ~io_p{ida,1};
    end
    dap_temp = double(dap); %privremeni dap jer je originalni logicki array
    dt1=size(dap_temp,1);
    dt2=size(dap_temp,2);
    for it1 = 1:dt1
        for it2 = 1:dt2
            if dap_temp(it1,it2) == 1 %provjeravamo tocke koje su unutar prostora
                %ako je iden od narednih uvjeta zadovoljen, tocka ima
                %susjeda koji je nula, stoga ju zelimo zanemariti jer robot
                %ne bi trebao ici preblizu ruba
                if (it1<dt1)&&(it2<dt2)&&(dap_temp(it1+1,it2+1) == 0)
                    dap_temp(it1,it2) = 2;
                    continue
                end
                if (it1>1)&&(it2<dt2)&&(dap_temp(it1-1,it2+1) == 0)
                    dap_temp(it1,it2) = 2;
                    continue
                end
                if (it1<dt1)&&(it2>1)&&(dap_temp(it1+1,it2-1) == 0)
                    dap_temp(it1,it2) = 2;
                    continue
                end
                if (it1>1)&&(it2>1)&&(dap_temp(it1-1,it2-1) == 0)
                    dap_temp(it1,it2) = 2;
                    continue
                end
                if (it1<dt1)&&(dap_temp(it1+1,it2) == 0)
                    dap_temp(it1,it2) = 2;
                    continue
                end
                if (it2<dt2)&&(dap_temp(it1,it2+1) == 0)
                    dap_temp(it1,it2) = 2;
                    continue
                end
                if (it1>1)&&(dap_temp(it1-1,it2) == 0)
                    dap_temp(it1,it2) = 2;
                    continue
                end
                if (it2>1)&&(dap_temp(it1,it2-1) == 0)
                    dap_temp(it1,it2) = 2;
                    continue
                end
            end
        end
    end
    %moramo takoder provjeriti sve krajnje tocke jer bi inace moglo biti
    %iznimki
    dap_temp(1,:) = 2;
    dap_temp(:,1) = 2;
    dap_temp(dt1,:) = 2;
    dap_temp(:,dt2) = 2;
    dap_temp(dap_temp == 2) = 0; %sve vrijednosti 2 pretvaramo u 0
    dap = boolean(dap_temp); %dap se vraca u logicki array da bi se kao takav mogao dalje k
oristit
```

```
    dap2 = reshape(dap.',[xymg,1]); %pretvaramo matricu u listu da bi mogli preko nje napra
viti petlju za eliminaciju
    for i2 = length(dap2):-
1:1 %pocinjemo od kraja da ne bi morali uzimati u obzir promjenu indeksa nakon brisanja
        if dap2(i2) == 0 %ako je vrijednost 0 tocka je izvan prostora
            dadj(:,i2) = []; %brisanje stupca
            dadj(i2,:) = []; %brisanje retka
        end
    end
    xdt = xm.';%transponirano mrezu i dap jer matlab logical indexing ide stupac po stupac
    ydt = ym.';%zbog matrice susjedstva potrebno je da ide red po red
    xd = xdt(dap.');
    yd = ydt(dap.');
    xpk = xp(pr);
    ypk = yp(pr);
    varargout = cell(nop*2,1);%inicijaliziranje varijablnih izlaznih arg
    for i3 = 1:nop
        varargout{2*i3-1} = pre{i3,1}(kop{i3});
        varargout{2*i3} = pre{i3,2}(kop{i3});
        %varijabilni izlazi, obrubi prepreka
    end
end
```

b. funkcija *dijkstra()*

```
function [min_put, dn] = dijkstra(D_ADJ, ds, dt)
%D_ADJ je (simetricna) matrica susjedstva
%ds je pocetna tocka
%dt je konacna tocka
%min_put je najkraca udaljenost izmedu ds i dt
%dn su tocke po kojima se dolazi od ds do dt (ukljucno)
n=size(D_ADJ,1);
S(1:n) = 0;     %vektor posjecenih tocki
dist(1:n) = inf;   % najkraca udaljenost izvora i svakog drugog tocke
prev(1:n) = n+1;    % prethodni tocka, najbolji tocka za doci do bilo kojeg drugog tocke
dist(ds) = 0;
while sum(S)~=n
    tren=[];
    for i=1:n
        if S(i)==0
            tren=[tren dist(i)];
        else
            tren=[tren inf];
        end
    end
    [~, u]=min(tren);
    S(u)=1;
    for i=1:n
        if(dist(u)+D_ADJ(u,i))<dist(i)
            dist(i)=dist(u)+D_ADJ(u,i);
            prev(i)=u;
        end
    end
end
dn = dt;
while dn(1) ~= ds
    if prev(dn(1))<=n
        dn=[prev(dn(1)) dn];
    else
        error('Error')
    end
end
min_put = dist(dt);
end
```

c. funkcija *koord()*

```
function [k_ind] = koord(xa,ya,xd,yd)
%A su koordinate koje trazimo
%B su koordinate mreze
A = [xa ya];
B = [xb yb];
dist2 = sum((B - A) .^ 2, 2); %euklidska udaljenost
[~,k_ind]=min(dist2); %trazimo indeks najblize tocke
end
```