



UNIVERZITET U SARAJEVU
ELEKTROTEHNIČKI FAKULTET
ODSJEK ZA AUTOMATIKU I ELEKTRONIKU

ZAVRŠNI RAD
- PRVI CIKLUS STUDIJA -

Implementacija T-RRT algoritma za planiranje
putanje u konfiguracijskom prostoru baziranim
na težinskim mapama

Autor:
Dinno Koluh

Mentor:
Vanr. prof. dr. Bakir Lačević, dipl.ing.el.

Sarajevo
Decembar 2020.

Sažetak

Ovaj rad se bavi planiranjem putanje robotskih manipulatora zasnovanom na korištenju algoritama baziranih na uzorkovanju u konfiguracijskom prostoru nad kojim je definisana odgovarajuća težinska funkcija. Ključni algoritam korišten za planiranje putanje je T-RRT algoritam, koji je modifikacija baznog RRT algoritma. T-RRT algoritam traži putanju koja se proteže kroz regije težinske mape konfiguracijskog prostora koje predstavljaju "prevoje", "kanjone" i "doline" odnosno regije sa niskom težinom dok regije sa visokom težinom teži zaobići. Uz odgovarajuću težinsku funkciju koja modelira kriterij problema, T-RRT algoritmom se dobivaju putanje koje ispunjavaju zadati kriterij dok RRT algoritam taj zadatok, u opštem slučaju, ne može obaviti. Primjer kriterija kojeg ćemo koristiti je zahtijev da vrh manipulatora bude što udaljeniji od prepreka u radnom prostoru što pospješuje njegovu sigurnost.

Parametri koji su integrirani unutar T-RRT algoritma utiču na njegovu performansu i krajnji oblik putanje tako da je urađen veliki broj simulacija sa raznim vrstama robotskih manipulatora radi analize uticaja pojedinih parametara na putanju. Ovi primjeri su uspoređeni sa oblicima putanja koje je su dobivene RRT algoritmom. Svi algoritmi su implementirani u MATLAB programskom paketu.

Abstract

This paper deals with path planning for robot manipulators using sampling-based algorithms in a configuration space equipped by a specific cost function. The key algorithm used for path planning is the T-RRT algorithm which is a modification of the basic RRT algorithm. The T-RRT algorithm searches for a path which spans over the regions of the configuration space costmap which represent "passes", "canyons" and "valleys" i.e. regions which have a low cost, while it tends to bypass high cost regions. Using a convenient cost function, which models the problem's criterion, with the T-RRT algorithm we get paths which fulfill the stated criterion while the RRT algorithm, in the general case, is not able to carry out such a task. An example of a criterion which we are going to use is the requirement for the tip of the manipulator to be at the furthest possible distance from the obstacles in the working space which ensures its safety.

The parameters which are integrated inside the T-RRT algorithm influence its performance and the final shape of the path so a great number of simulations was conducted, with various types of robot manipulators, to analyze the influence of individual parameters on the path. These examples were compared with the shapes of paths obtained with the RRT algorithm. All the algorithms have been implemented within the MATLAB programming package.

Univerzitet u Sarajevu
Elektrotehnički fakultet Sarajevo
Odsjek: Automatika i elektronika

Izjava o autentičnosti rada

Završni rad

Prvi ciklus studija

Ime i prezime: Dinno Koluh

Naslov rada: Implementacija T-RRT algoritma za planiranje putanje u konfiguracijskom prostoru baziranim na težinskim mapama

Broj stranica: 66

Potvrđujem:

- da sam pročitao dokumente koji se odnose na plagijarizam, kako je to definirano Statutom Univerziteta u Sarajevu, Etičkim kodeksom Univerziteta u Sarajevu i pravilima studiranja koja se odnose na I i II ciklus studija, integrirani studijski program I i II ciklusa i III ciklus studija na Univerzitetu u Sarajevu, kao i uputama o plagijarizmu navedenim na web stranici Univerziteta u Sarajevu;
- da sam svjestan univerzitetskih disciplinskih pravila koja se tiču plagijarizma;
- da je rad koji predajem potpuno moj, samostalni rad, osim u dijelovima gdje je to naznačeno;
- da rad nije predat, u cjelini ili djelimično, za stjecanje zvanja na Univerzitetu u Sarajevu ili nekoj drugoj visokoškolskoj ustanovi;
- da sam jasno naznačio prisustvo citiranog ili parafraziranog materijala i da sam se referirao na sve izvore;
- da sam dosljedno naveo korištene i citirane izvore ili bibliografiju po nekom od preporučenih stilova citiranja, sa navođenjem potpune reference koja obuhvata potpuni bibliografski opis korištenog i citiranog izvora;
- da sam odgovarajuće naznačio svaku pomoć koju sam dobio pored pomoći mentora.

Potpis studenta:

U Sarajevu, ————— 2020. godine

Sadržaj

1 Uvod i motivacija	1
1.1 Generalizacija problema traženja putanje	1
1.2 Neki od algoritama planiranja putanje	2
2 RRT algoritam i njegove modifikacije	5
2.1 Klasični RRT algoritam	5
2.1.1 Definisanje parametara RRT-a	5
2.1.2 Opis algoritma	7
2.2 BiRRT algoritam	9
2.2.1 Upravljanje širenjem stabala	12
2.3 Biased RRT/BiRRT algoritam	13
3 Težinske mape i težinske funkcije	15
4 T-RRT algoritam	17
4.1 Implementacija funkcije <code>testTransition</code>	17
4.2 Implementacija funkcije <code>minExpandControl</code>	21
5 Robotski manipulatori i njihov matematski model	24
5.1 DH parametri	24
5.2 Tipične strukture robotskih manipulatora	25
5.3 Radni i konfiguracijski prostor robotskih manipulatora	27
6 Modeliranje prepreka u radnom i konfiguracijskom prostoru	30
6.1 Prepreka kao konveksni omotač	30
6.2 Testiranje sudara i dobivanje minimalne udaljenosti od prepreka	31
7 Integracija RRT baziranih algoritama sa modelom robotskih manipulatora	34
7.1 Implementacija modela robotskih manipulatora u MATLAB-u	34
7.2 Implementacija bitnih funkcija	35
7.3 Integrisani algoritam za planiranje putanje robotskog manipulatora	39
7.4 Generisanje težinske funkcije manipulatora	41
8 Testiranje, simulacije, rezultati i diskusija	43
8.1 Testiranje robotskih manipulatora	44
8.1.1 Dvosegmentna planarna ruka	45
8.1.2 Trosegmentna planarna ruka	45
8.1.3 Antropomorfna ruka	45
8.1.4 Sferna šaka	46
8.1.5 Kombinacija antropomorfne ruke i sferne šake	46
8.2 Uticaj ključnih parametara T-RRT algoritma na njegovu performansu	46
9 Zaključak	50

Popis slika

1	2D konfiguracijski prostor \mathcal{C} sačinjen od $\mathcal{C}_{ob_1}, \mathcal{C}_{ob_2}$ i \mathcal{C}_{free} i ograničen pravougaonikom stranica a i b	2
2	Primjena algoritma zasnovanog na pretragi mreže u 2D konfiguracijskom prostoru.	2
3	Primjena algoritma zasnovanog na uzorkovanju u 2D konfiguracijskom prostoru.	3
4	Primjena PRM algoritma u 2D konfiguracijskom prostoru (za dati čvor i , $m = 3$, $k = 2$).	3
5	Potencijalno polje u 2D konfiguracijskom prostoru.	4
6	boundary struktura za 3D hiperkocku.	6
7	Pregled gradnje RRT stabla za razne slučajeve.	8
8	Primjena RRT algoritma u 2D konfiguracijskom prostoru sa preprekama sa sljedećim parametrima: <code>maxIter = 1000, deltaQ = 1, minDistance = 1, L = 27.3346</code>	8
9	Primjena RRT algoritma u 3D konfiguracijskom prostoru sa preprekama sa sljedećim parametrima: <code>maxIter = 1000, deltaQ = 1, minDistance = 3, L = 42.6999</code>	8
10	Primjena BiRRT algoritma u 2D konfiguracijskom prostoru sa preprekama sa sljedećim parametrima: <code>maxIter = 1000, deltaQ = 1, eta = 0.50, L = 26.5909</code>	11
11	Primjena BiRRT algoritma u 3D konfiguracijskom prostoru sa preprekama sa sljedećim parametrima: <code>maxIter = 1000, deltaQ = 1, eta = 0.50, L = 34.6349</code>	12
12	Primjena BiRRT algoritma u 2D konfiguracijskom prostoru sa preprekama sa sljedećim parametrima: <code>maxIter = 1000, deltaQ = 1, eta = 0.20, L = 42.1545</code>	12
13	Primjena Biased BiRRT algoritma u 2D konfiguracijskom prostoru sa preprekama sa sljedećim parametrima: <code>maxIter = 1000, deltaQ = 1, eta = 0.50, mi = [0.80, 0.80], L = 25.1631</code>	14
14	Primjena Biased BiRRT algoritma u 2D konfiguracijskom prostoru sa preprekama sa sljedećim parametrima: <code>maxIter = 1000, deltaQ = 1, eta = 0.25, mi = [0.60, 0.30], L = 28.8219</code>	14
15	Primjena RRT algoritma u 3D konfiguracijskom prostoru sa preprekama sa sljedećim parametrima: <code>maxIter = 1000, deltaQ = 1, minDistance = 3, eta = 0.30, mi = [0.25, 0.30], L = 34.4868</code>	14
16	Težinska mapa generisana težinskom funkcijom: $f_c(\mathbf{x}) = 0.25 \cdot \ \mathbf{x}_0 - \mathbf{x}\ ^2$. Kriterij težinske funkcije je minimalna udaljenost do ciljane konfiguracije \mathbf{x}_0	15
17	Težinska mapa generisana težinskom funkcijom: $f_c(\mathbf{x}) = \exp(-0.05 \cdot \ \mathbf{x}_0 - \mathbf{x}\ ^2) + \exp(-0.2 \cdot \ \mathbf{x}_1 - \mathbf{x}\ ^2)$. Kriterij težinske funkcije je što veća udaljenost od konfiguracija \mathbf{x}_0 i \mathbf{x}_1	16
18	Primjena T-RRT algoritma (bez <code>minExpandControl</code>) na težinskoj mapi u 2D konfiguracijskom prostoru. Parametri T-RRT algoritma: <code>maxIter = 10000, deltaQ = 1, minDistance = 2, eta = 0.50, mi = [0.10, 0.10], T = 10^{-6}, K = 0.26, alpha = 1.25, c_max = 0.38, maxFails = 15, L = 118.3633</code>	20
19	Uticaj <code>minExpandControl</code> funkcije unutar T-RRT algoritma nakon 10000 iteracija.	22

20	Primjena T-RRT algoritma na težinskoj mapi u 2D konfiguracijskom prostoru. Parametri T-RRT algoritma: <code>maxIter</code> = 10000, <code>deltaQ</code> = 1, <code>minDistance</code> = 2, <code>eta</code> = 0.50, <code>mi</code> = [0.10, 0.10], $T = 10^{-6}$, $K = 0.26$, <code>alpha</code> = 1.25, <code>c_max</code> = 0.38, <code>maxFails</code> = 15, <code>rho</code> = 0.05, $L = 110.1681$	22
21	Prikaz generalnog slučaja DH parametara za dva segmenta.	24
22	Razni robotski manipulatori.	25
23	Početni i krajnji položaj dvosegmentne planarne ruke u radnom i konfiguracijskom prostoru. Parametri manipulatora: $\mathbf{a} = [1 \ 1]^T$, $\mathbf{q}_s = [-30^\circ \ -120^\circ]^T$ i $\mathbf{q}_f = [0^\circ \ 150^\circ]^T$	28
24	Korištenje BiRRT algoritma u konfiguracijskom prostoru se manifestira kao kretanje manipulatora u relanom svijetu tj. radnom prostoru.	29
25	Konveksni omotač skupa tačaka u 2D prostoru.	30
26	Prikaz konveksnih omotača u prvobitnom i rotiranom položaju. Uglovi rotacije odgovarajućih omotača: $\alpha_1 = 30^\circ$, $\alpha_2 = -15^\circ$ i $\alpha_3 = -90^\circ$ oko prve tačke omotača.	31
27	Prikaz prozora u 3D prostoru, sačinjenog od četiri konveksna omotača, u prvobitnom položaju i nakon rotacije oko z ose za ugao $\alpha = 90^\circ$	31
28	Tačke p_1 i p_2 su u slobodnom prostoru ali cijeli segment nije (a). Cijeli segment u slobodnom prostoru (b). Nedovoljna podjela dovodi do netačne interpretacije da je cijeli segment u slobodnom prostoru (c). Tačna interpretacija ali velik broj podjela segmenta što usporava algoritam (d).	32
29	Težinska mapa u konfiguracijskom prostoru manipulatora. Generisano težinskom funkcijom $f_c(\mathbf{q}) = \exp(-3 \cdot \text{getRobotTipToObstaclesDistance}(\text{robot}(\mathbf{q})))$ koju ćemo koristiti za buduće testiranje.	42
30	Reprezentacija konfiguracijskog prostora kao torus.	42
31	Planiranje kretanja dvosegmentnom planarnom rukom sa BiRRT algoritmom.	45
32	Planiranje kretanja dvosegmentnom planarnom rukom sa T-RRT algoritmom.	45
33	Planiranje kretanja trosegmentnom planarnom rukom sa BiRRT/T-RRT algoritmima u radnom prostoru.	45
34	Planiranje kretanja antropomorfnom rukom sa BiRRT/T-RRT algoritmima u radnom prostoru.	45
35	Planiranje kretanja sfernom šakom sa BiRRT/T-RRT algoritmima u radnom prostoru.	46
36	Planiranje kretanja kombinacijom antropomorfne ruke i sferne šake sa BiRRT/T-RRT algoritmima u radnom prostoru.	46

Popis tabela

1	DH parametri N-segmentne planarne ruke.	26
2	DH parametri antropomorfne ruke.	26
3	DH parametri sferne šake.	27
4	DH parametri kombinacije antropomorfne ruke i sferne šake.	27
5	Informacije o parametrima manipulatora.	44
6	Informacije o parametrima BiRRT/TRRT algoritma.	44
7	Informacije o rezultatima simulacija.	44
8	Uticaj parametara <code>alpha</code> i <code>maxFails</code> na predeni put vrha trosegmentne planarne ruke.	47
9	Uticaj parametara <code>alpha</code> i <code>maxFails</code> na duzinu puta u konfiguracijskom prostoru trosegmentne planarne ruke.	47
10	Uticaj parametara <code>alpha</code> i <code>maxFails</code> na prosječnu težinu puta na težinskoj mapi trosegmentne planarne ruke.	47
11	Uticaj parametara <code>alpha</code> i <code>maxFails</code> na broj potrebnih iteracija T-RRT algoritma za trosegmentnu planarnu ruku.	47
12	Uticaj parametara <code>alpha</code> i <code>maxFails</code> na vrijeme izvršavanja T-RRT algoritma za trosegmentnu planarnu ruku.	48
13	Uticaj parametra <code>rho</code> na performanse T-RRT algoritma za trosegmentnu planarnu ruku.	48
14	Uticaj parametra <code>rho</code> na performanse T-RRT algoritma za brdovitu težinsku mapu.	48

1 Uvod i motivacija

Problem planiranja putanje (eng. *motion planning*) je široko područje mnogih inženjerskih disciplina, od robotike pa sve do problema proteinskog savijanja [5] (eng. *protein folding*) u biologiji. U ovom radu ćemo se bazirati na problemu planiranja putanje robotiskih manipulatora u 2D i 3D prostorima koji sadrže prepreke. Uz ovu činjenicu želimo da robotski manipulator također bude *autonoman* tj. da može sam, bez našeg djelovanja, pronaći traženu putanju. Obično planiranje putanje od početne do krajnje tačke bez prepreka za robotske manipulatore je manje-više trivijalno ali uvođenje prepreka u radni prostor robota dosta komplikuje problem. Daljnja komplikacija planiranja putanje robotskih manipulatora jeste što se sastoje iz više pokretnih segmenata što dovodi do toga za svaki pojedinačni segment moramo definisati putanju. Dodatni problem koji se uvodi jeste optimizacija nađene putanje po nekom kriteriju. Taj kriterij, kada govorimo o robotskim manipulatorima (od sada samo manipulatori) je najčešće maksimiziranje udaljenosti manipulatora od prepreka tako da ne dođe do sudara ili da bude na sigurnoj udaljenosti od prepreke - iako se mogu zadati i drugi kriteriji, u zavisnosti od problema.

Svi navedeni problemi i zahtijevi su doveli do razvoja raznih algoritama u oblasti planiranja putanje o kojima ćemo pričati u sljedećem potpoglavlju.

1.1 Generalizacija problema traženja putanje

Da bi se manipulator kretao po nekoj definisanoj putanji, ona se najčešće diskretizira i šalje se kao skup diskretnih komandi manipulatoru i on ih izvršava sekvencialno. Razdvojimo, za sad, kinematiku manipulatora i skup komandi koje mu moramo poslati. Taj skup komandi su razna stanja manipulatora u prostoru u kojem se nalazi i njih dobivamo kao izlaz algoritama planiranja putanje.

Iz ovog razloga pretpostavimo za sada da ne govorimo o manipulatorima nego o dvije tačke u n -dimenzionalnom prostoru koji sadrži prepreke i naš cilj je doći od početne do krajnje tačke. n -dimenzionalni prostor u kojem tražimo putanju ćemo zvati *konfiguracijski prostor* (eng. *configuration space*) ali ćemo dati opširniju definiciju ovog termina kasnije kada se vratimo na manipulatore. Označimo spomenuti konfiguracijski prostor sa \mathcal{C} i tačku, odnosno *konfiguraciju* u tom prostoru sa \mathbf{q} . Označimo prostor gdje se nalaze prepreke (eng. *obstacles*) odnosno prostor kojem ne možemo pristupiti sa \mathcal{C}_{ob} . Pretpostavimo da imamo p prepreka koje nisu fizičke spojene i svaka od njih zauzima prostor \mathcal{C}_{ob_i} onda je prostor kojem ne možemo pristupiti dat kao unija svih tih prepreka odnosno:

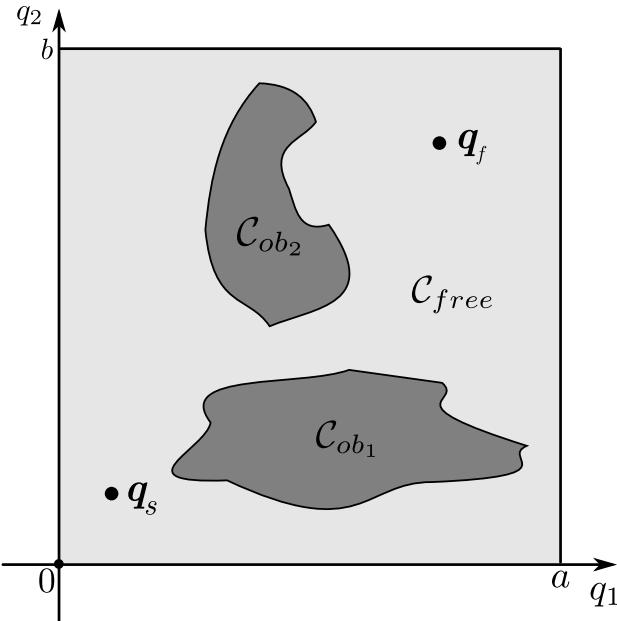
$$\mathcal{C}_{ob} = \bigcup_{i=1}^p \mathcal{C}_{ob_i} \quad (1.1)$$

Zadnja komponenta konfiguracijskog prostora jeste prostor kojem možemo pristupiti (eng. *free space*) i označimo ga sa \mathcal{C}_{free} . Sada slijedi da unija svih prepreka i slobodnog prostora daje cijeli konfiguracijski prostor odnosno:

$$\mathcal{C} = \mathcal{C}_{free} \cup \mathcal{C}_{ob} \quad (1.2)$$

Prostor \mathcal{C}_{free} ne mora biti spojen što je posljedica izgleda \mathcal{C}_{ob} što bi značilo da ne možemo napraviti putanju između bilo koje dvije tačke koje pripadaju \mathcal{C}_{free} prostoru. Na slici 1 je prikazan primjer 2D

konfiguracijskog prostora uz startnu \mathbf{q}_s i finalnu \mathbf{q}_f konfiguraciju. Konfiguracijski prostor je definisan kao pravougaonik $\mathcal{C} = \{(q_1, q_2) \in \mathbb{R}^2 : 0 < q_1 < a, 0 < q_2 < b\}$. Primjetimo da možemo napraviti putanju između bilo koje dvije tačke unutar \mathcal{C}_{free} mada u generalnom slučaju to ne mora vrijediti.



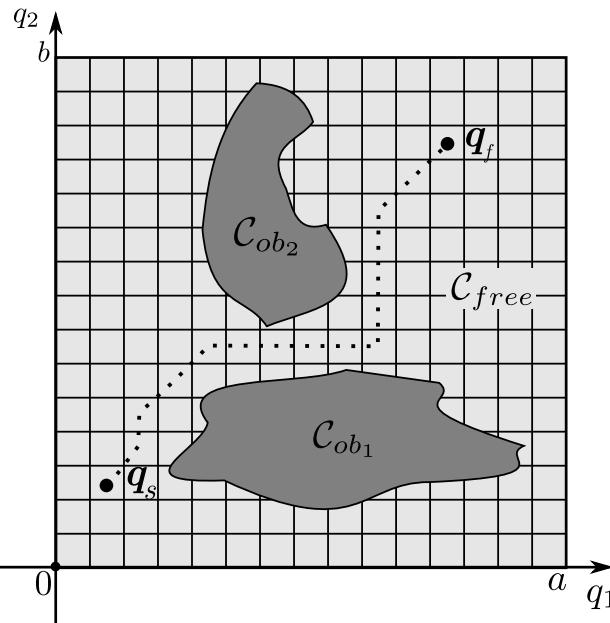
Slika 1: 2D konfiguracijski prostor \mathcal{C} sačinjen od $\mathcal{C}_{ob1}, \mathcal{C}_{ob2}$ i \mathcal{C}_{free} i ograničen pravougaonikom stranica a i b .

1.2 Neki od algoritama planiranja putanje

Najpoznatiji i najviše korišteni algoritmi za rješavanje spomenute generalizacije su:

1. Algoritmi zasnovni na pretragi mreže (eng. *grid-based*)

Ovi algoritmi su zasnovani na diskretiziranju cijelog konfiguracijskog prostora mrežom tako da svaka konfiguracija odgovara jednom polju mreže. Iz trenutne u susjednu konfiguraciju možemo preći ako centre dva susjedna polja možemo spojiti tako da ne sjeku prepreku. Vrijeme izvršavanja zavisi od rezolucije mreže. Nakon diskretiziranja konfiguracijskog prostora pretražni algoritmi (eng. *search algorithms*) (kao A* algoritam) se mogu koristiti za traženje putanje između startne i finalne konfiguracije. Za nisko-dimenzionalne prostore ovaj algoritam je brz i efikasan ali kako se dimenzija u kojoj operiramo povećava vrijeme izvršavanja eksponencijalno raste tako da nije namijenjen za visoko-dimenzionalne prostore. Na slici 2 je prikazana ideja ovih algoritama.



Slika 2: Primjena algoritma zasnovanog na pretragi mreže u 2D konfiguracijskom prostoru.

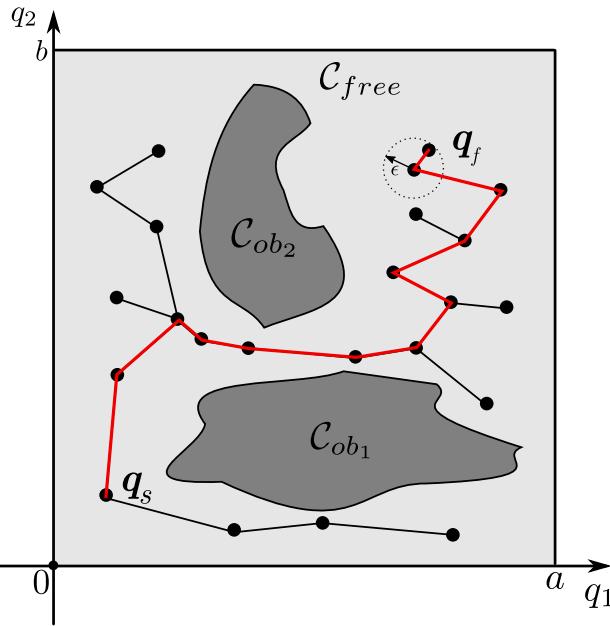
2. Algoritmi zasnovni na uzorkovanju (eng. *sampling-based*)

Algoritmi zasnovani na uzorkovanju rade na principu uzorkovanja konfiguracijskog prostora i konstrukcije puta tj. putokaza (eng. *roadmap*) od uzorkovanih konfiguracija. Uzorci se biraju u \mathcal{C} ali samo oni uzorci koji pripadaju \mathcal{C}_{free} se uzimaju u obzir. Uzorci $\mathbf{q} \in \mathcal{C}_{free}$ se zatim pokušavaju spojiti u matematsku strukturu koja je graf (eng. *graph*) gdje su čvorovi (eng. *nodes*) grafa nađeni uzorci \mathbf{q} , a grane (eng. *edges*) grafa su spojevi između pojedinih čvorova. Bitna napomena je da mora biti i prisutan neki oblik algoritma za detekciju sudara sa preprekama jer uslijed prepreka neće moći doći do veze između proizvoljnih čvorova. Kao početni čvor biramo početnu konfiguraciju \mathbf{q}_s , i kada dobijemo uzorak koji se nalazi u predefinisanoj ϵ okolini finalne konfiguracije \mathbf{q}_f i ako postoji skup čvorova i grana između dva navedena čvora takvih da možemo doći od početnog do finalnog čvora onda kažemo da smo pronašli rješenje. Dvije najpoznatije vrste algoritama zasnovanih na uzorkovanju su:

- RRT (eng. *Rapidly Random Exploring Trees*) algoritam [2]
- PRM (eng. *Probabilistic Roadmap*) algoritam [3]

Ovi algoritmi imaju probabilističku prirodu tj. uzorci u \mathcal{C} se biraju po nekoj probabilističkoj raspodjeli. Kako raste broj broj uzoraka tako i raste dio prostora kojeg smo pretražili. Pokazano je da su ovi algoritmi *probabilistički kompletni* (eng. *probabilistically complete*) što znači kako broj uzoraka teži ka beskonačnosti, vjerovatnoća da ćemo pronaći rješenje, ako ono postoji, teži ka 1. Ova činjenica ima i slabost, a to je da nam ovi algoritmi ne mogu reći da li postoji rješenje ili ne. Zbog ovog se u općem slučaju uzima maksimalan broj uzoraka i ako nakon tog

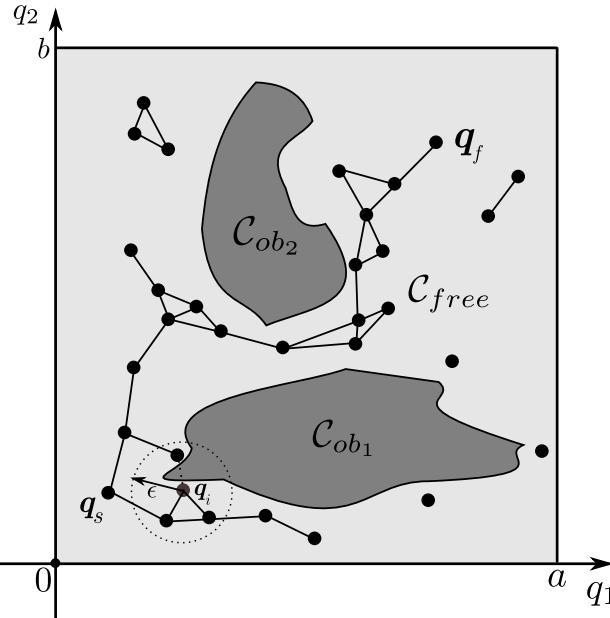
broja uzoraka ne možemo naći put između startne i finalne konfiguracije onda smatramo da nema rješenja. Na slici 3 je prikazana ideja ovih algoritama.



Slika 3: Primjena algoritma zasnovanog na uzorkovanju u 2D konfiguracijskom prostoru.

RRT i PRM algoritmi pripadaju istoj porodici algoritama ali se suštinski razlikuju po tome kako se konstruiše graf. Pošto je RRT algoritam glavna tema ovog rada posvetili smo cijelo poglavlje o tome kako se on implementira tako da nećemo ovdje o tome pričati već ćemo samo dati kratko objašnjenje kako radi PRM algoritam.

Generišemo n uzoraka u \mathcal{C} i odbacimo one uzorke koji su u \mathcal{C}_{ob} . Uzmimo i -ti uzorak i prođimo kroz sve uzorke koje imamo. Odaberimo m uzoraka koji su unutar ϵ okoline i -tog uzorka i spojimo ih sa i -tim uzorkom uz korištenje algoritma za detekciju prepreka. Prepostavimo da možemo izvršiti spajanje k od m nađenih uzoraka sa uzorkom i . U graf dodajemo novonastale grane gdje se svakoj nastaloj vezi tj. grani dodaje težina koja odgovara udaljenosti dva čvora i nastavljamo tako dalje do n -tog uzorka. Na kraju ćemo imati spojen graf koji može sadržavati konture što može dovesti do redundancije ali korištenjem nekih od poznatih algoritama u teoriji grafova za pronađak minimalnog puta (npr. Dijkstrin algoritam) tj. puta sa minimalnom težinom, što je u našem slučaju dužina između dva čvora, možemo dobiti put bez redundancija. Na slici 4 je prikazan primjer PRM algoritma uz konstrukciju grana i -tog čvora.



Slika 4: Primjena PRM algoritma u 2D konfiguracijskom prostoru (za dati čvor i , $m = 3$, $k = 2$).

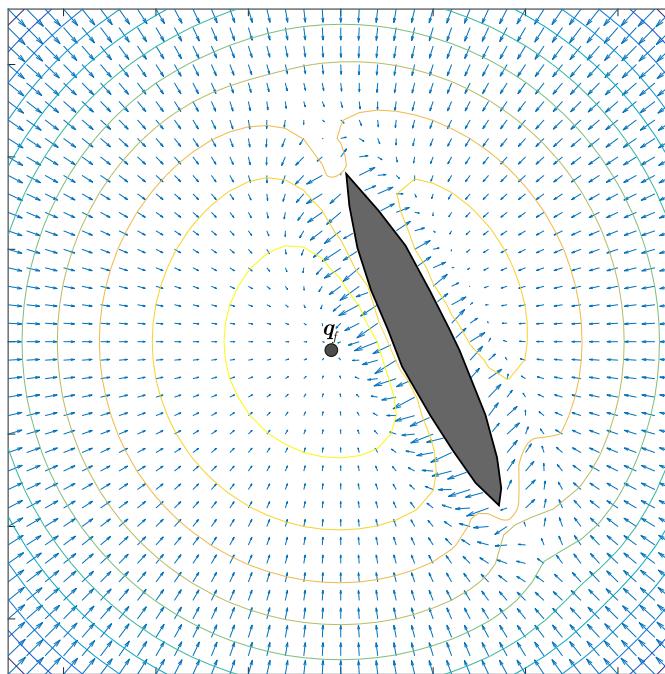
RRT algoritam se razlikuje u načinu konstrukcije grafa i finalnom izgledu grafa koji je ustvari stablo tj. nema kontura što omogućava brži pronalazak puta i graf je u cijelosti spojen dok kod PRM algoritma možemo imati dijelove grafa koji nisu povezani (mada se ovo može prevazići do određene mjere sa varijabilnom ϵ okolinom).

3. Umjetna potencijalna polja (eng. *artificial potential fields*)

Dvije prethodne vrste algoritama o kojima smo pričali pripadaju vrsti *off-line* algoritama planiranja putanja tj. vrijedi pretpostavka da znamo gdje se nalaze prepreke u prostoru prije nego što se manipulator počne kretati. Ovo je snažna pretpostavka jer u mnogim situacijama u praksi nemamo informaciju o rasporedu prepreka u radnom prostoru manipulatora ili još komplikovaniji slučaj kada su prepreke pokretne. Ovo ograničava prethodne algoritme i zbog toga ćemo nešto reći o *on-line* planiranju putanje algoritma umjetnih potencijalnih polja.

Tačka u konfiguracijskom prostoru koja predstavlja stanje manipulatora se kreće pod uticajem nekog potencijalnog polja U koje je dobiveno kao superpozicija privlačnog (eng. *attractive*) potencijala finalne konfiguracije \mathbf{q}_f i odbojnog (eng. *repulsive*) potencijala prostora prepreka \mathcal{C}_{ob} [6]. Planiranje putanje se vrši inkrementalno tako da umjetna sila djeluje na i -tu konfiguraciju u smjeru negativnog gradijenta $-\nabla U(\mathbf{q})$ potencijala što indicira kretanje u smjeru cilja. Analogija ove metode je kretanje tačkastog naboja u električnom polju. Najveći nedostatak ove metode jeste mogućnost da upadnemo u lokalni minimum i tako ne uspijemo dobiti traženu putanju. Na slici 5 je prikazan izgled potencijalnog polja konfiguracijskog prostora sa jednom preprekom. Strelice pokazuju smjer kojim se moramo kretati da bismo stigli do finalne konfiguracije. Primjetimo da što smo dalji od cilja to su strelice duže što označava veći privlačni potencijal i veću umjetnu силу koja nas vodi ka finalnoj konfiguraciji. Kako se bližimo cilju

tako i privlačni potencijal opada. U blizini prepreke imamo duže strelice što označava odbojni potencijal.



Slika 5: Potencijalno polje u 2D konfiguracijskom prostoru.

2 RRT algoritam i njegove modifikacije

RRT (eng. *Rapidly Random Exploring Trees*) [2] algoritam je daleko najčešće korišten algoritam zasnovan na uzorkovanju zbog njegove pretražne moći, jednostvane implementacije i vremena izvršavanja u višedimenzionalnim prostorima sa preprekama. RRT i njegove nadogradnje se već koriste u velikoj mjeri u planiranju putanja robota i robotskih manipulatora što ćemo i mi koristiti u ovom radu. RRT je randomiziran algoritam koji se može smatrati vrstom *Monte-Carlo* algoritama koji na osnovu randomiziranog uzorkovanja dovode do numeričkih rješenja [7].

Kao što je već spomenuto, RRT algoritam je probabilistički kompletan tako da kako generišemo nove uzorke tako se povećava vjerovatnoća pronalska rješenja. Glavna struktura RRT-a je graf (eng. *graph*) ili preciznije stablo (eng. *tree*) jer konstruišemo graf bez kontura. U nastavku ćemo detaljno objasniti kako se konstruiše RRT algoritam.

2.1 Klasični RRT algoritam

Kako smo već kazali, sa RRT algoritmom generišemo strukturu koja se zove stablo i koje se sastoji od čvorova (eng. *node, vertex*) (zvat ćemo ih i konfiguracije ili uzorci) i grana (eng. *edge*). Označimo stablo kao uredjen par:

$$\mathcal{T} = (\mathcal{Q}, \mathcal{V}) \quad (2.1)$$

gdje je \mathcal{Q} skup svih čvorova (koristit ćemo oznaku \mathcal{Q} jer pojedine čvorove (konfiguracije) označavamo sa \mathbf{q}), a \mathcal{V} je skup svih grana koje su *neuređeni parovi* (jer imamo neusmjerno stablo) oblika $\{\mathbf{q}_i, \mathbf{q}_j\}$ [8]. Pošto imamo konfiguracije u višedimenzionalnim prostorima onda i oznaku za njih boldiramo u smislu da je matematski konfiguracija \mathbf{q}_i , n -dimenzionalni vektor odnosno:

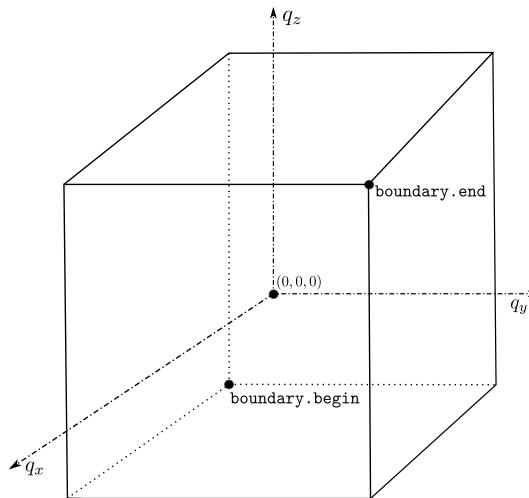
$$\mathbf{q}_i = [q_1 \quad \cdots \quad q_n]^T \quad (2.2)$$

2.1.1 Definisanje parametara RRT-a

Kako programiramo RRT algoritam onda ćemo koirstiti programersku notaciju klase, atributa i metoda za opis parametara (u programskom jeziku MATLAB) i to:

- Glavna klasa je RRT koja je ustvari stablo \mathcal{T} koje smo spominjali. Ova klasa sadrži brojne metode čiju ćemo implementaciju objasniti u poglavljju 7. Kako je stablo sačinjeno od čvorova i grana onda su i oni atributi unutar klase stabla i oni će biti također klase.
 - Atribut **maxIter** označava maksimalan broj iteracija tj. maksimalan broj čvorova koje možemo dodati (objekat tipa **int**).
 - Atribut **deltaQ** označava inkrement za koji širimo stablo odnosno udaljenost između dva čvora (objekat tipa **double**).
 - Atribut **minDistance** označava kolika je ϵ okolina oko finalnog čvora, koju, kada stablo dosegne, smatramo da smo pronašli rješenje (objekat tipa **double**).

- Atribut **boundary** je struktura koja sadrži attribute **begin** i **end** i ona predstavlja granice konfiguracijskog prostora kojeg predstavljamo kao hiperkocku (eng. *hypercube*) (mada je izvršena takva implementacija da prostor može biti i “hiperpravougaonik” (eng. *hyperrectangle*)) gdje je atribut **begin** koordinata čoška hiperkocke sa minimalnim koordinatama dok je atribut **end** koordinata čoška hiperkocke sa maksimalnim koordinatama. Na slici 6 je prikazana ideja iza **boundary** strukture.
- Atribut **obstacles** označava varijablu koja sadrži interno u sebi informacije o preprekama koje izbjegavamo u konfiguracijskom prostoru. Za sad ćemo smatrati da imamo funkciju koja prima ovu varijablu i novodobivenu granu i daje informaciju tipa **bool** o sudaru grana stabla sa preprekama pa ćemo u poglavljju broj 6 detaljno objasniti generisanje prepreka i funkciju za testiranje sudara.



Slika 6: **boundary** struktura za 3D hiperkocku.

- Klasa RRT sadrži još dvije varijable **nodes** (tipa **Node**) i **edges** (tipa **Edge**) koje objašnjavamo u nastavku.

Kako je stablo sačinjeno od čvorova i grana onda su i oni atributi unutar klase stabla RRT i oni će biti također klase:

- Klasa **Node** označava jedan čvor \mathbf{q} stabla \mathcal{T} .
 - Atribut **coordinates** označava vektor kolonu koordinata konfiguracije iz jednačine 2.2 (objekat tipa **vector<double>**).
 - Atribut **index** označava indeks čvora odnosno njegov redni broj (objekat tipa **int**).
 - Atribut **degree** označava *stepen* čvora odnosno broj grana koje su povezane na dati čvor (objekat tipa **int**).

- Klasa Edge označava i -tu granu \mathcal{V}_i stabla \mathcal{T} koja je veza dva čvora odnosno neuređeni par $\{\mathbf{q}_i, \mathbf{q}_j\}$.
 - Atribut `node1` označava prvi čvor grane (objekat tipa `Node`).
 - Atribut `node2` označava drugi čvor grane (objekat tipa `Node`).
 - Atribut `weight` označava težinu grane \mathcal{V}_i . U našem slučaju će predstavljati udaljenost između dva čvora trenutne grane i to će biti kriterij po kojem ćemo tražiti minimalni put (objekat tipa `double`).

Pošto smo definisali potrebne pojmove objasnimo sada način konstrukcije RRT-a sa prikladnim slikama radi bolje vizualizacije.

2.1.2 Opis algoritma

Pogledajmo prvo pseudokod za RRT algoritam nakon kojeg ćemo detaljno objasniti njegovu implementaciju.

```
Inicijaliziraj:  $\mathcal{T}, \mathbf{q}_s, \mathbf{q}_f, \text{maxIter}, \text{deltaQ}, \text{minDistance}, \text{boundary}, \text{obstacles}$ 
function getRRT( $\mathcal{T}, \mathbf{q}_s, \mathbf{q}_f$ )
    1  $\mathcal{T}.\text{addNode}(\mathbf{q}_s)$ 
    2 for  $i = 1$  to  $\mathcal{T}.\text{maxIter}$  do
    3      $\mathbf{q}_{rand} \leftarrow \mathcal{T}.\text{randConfig}();$ 
    4      $[\mathbf{q}_{near}, d_{\mathbf{q}.rand}] \leftarrow \mathcal{T}.\text{nearestNode}(\mathbf{q}_{rand});$ 
    5     if  $d_{\mathbf{q}.rand} < \mathcal{T}.\text{deltaQ}$  then
    6          $\mathbf{q}_{new} \leftarrow \mathbf{q}_{rand};$ 
    7     else
    8          $\mathbf{q}_{new} \leftarrow \mathcal{T}.\text{newConfig}(\mathbf{q}_{near}, \mathbf{q}_{rand});$ 
    9     end
   10    if  $\text{isSegmentInFreeSpace}(\mathcal{T}.\text{obstacles}, \mathbf{q}_{near}.\text{coordinates}, \mathbf{q}_{new}.\text{coordinates})$  then
   11         $\mathcal{T}.\text{addNode}(\mathbf{q}_{new});$ 
   12         $\mathcal{T}.\text{addEdge}(\mathbf{q}_{near}, \mathbf{q}_{new});$ 
   13    end
   14    if  $\text{distance}(\mathbf{q}_{new}.\text{coordinates}, \mathbf{q}_f.\text{coordinates}) < \mathcal{T}.\text{minDistance}$  then
   15         $\mathcal{T}.\text{addNode}(\mathbf{q}_f);$ 
   16         $\mathcal{T}.\text{addEdge}(\mathbf{q}_{new}, \mathbf{q}_f);$ 
   17        break;
   18    end
   19 end
   20 return  $\mathcal{T};$ 
end
```

Pseudokod 1: Funkcija getRRT

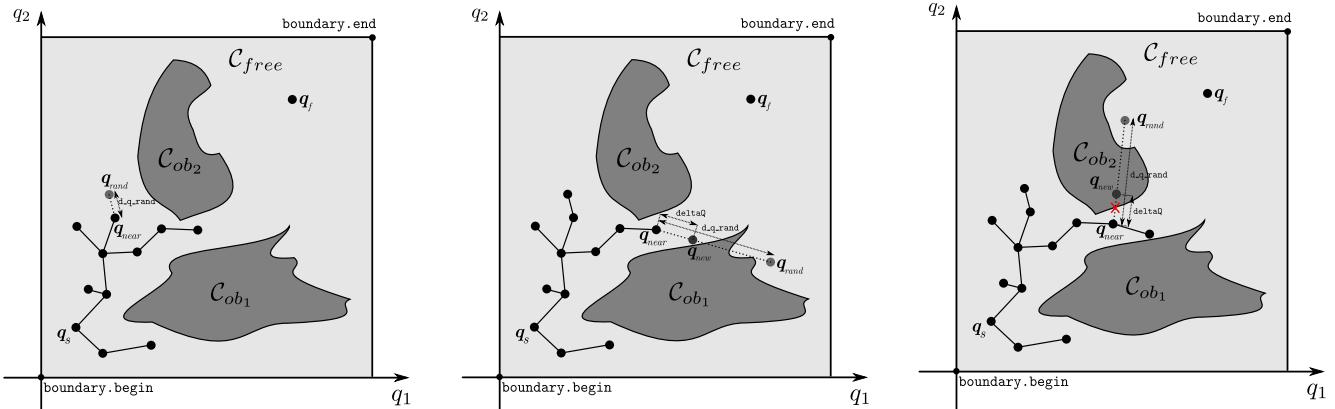
Inicijalizirajmo parametre stabla `maxIter`, `deltaQ`, `minDistance`, `boundary` i `obstacles`. Na početku je naše stablo \mathcal{T} (tipa RRT) prazno, tj. skup čvorova i grana je prazan skup ($\mathcal{T} = (\emptyset, \emptyset)$) tj. $\mathcal{T}.\text{nodes} = \text{Node.empty}()$ i $\mathcal{T}.\text{edges} = \text{Edge.empty}()$). Izaberimo početnu konfiguraciju (tipa `Node`) \mathbf{q}_s i dodajmo je u stablo sa $\mathcal{T}.\text{addNode}(\mathbf{q}_s)$. Generišimo finalnu konfiguraciju \mathbf{q}_f . Sada koristimo uniformnu distribuciju rada generisanja nasumične konfiguracije $\mathbf{q}_{rand} = \mathcal{T}.\text{randConfig}()$ unutar granica konfiguracijskog prostora određenim varijablom `boundary`.

Sada prolazimo kroz sve čvorove unutar stabla (trenutno imamo samo jedan čvor \mathbf{q}_s tako da ćemo uzeti njega na početku ali u i -toj iteraciji ćemo imati drugi broj čvorova) i pronalazimo onaj čvor u stablu \mathcal{T} koji je najbliži čvoru \mathbf{q}_{rand} tj. $[\mathbf{q}_{near}, d_{q_rand}] = \mathcal{T}.\text{nearestNode}(\mathbf{q}_{rand})$ gdje je \mathbf{q}_{near} nađeni najbliži čvor čvoru \mathbf{q}_{rand} , a d_{q_rand} je udaljenost između njih. Naglasimo da čvor \mathbf{q}_{rand} može biti i u \mathcal{C}_{free} i u \mathcal{C}_{ob} .

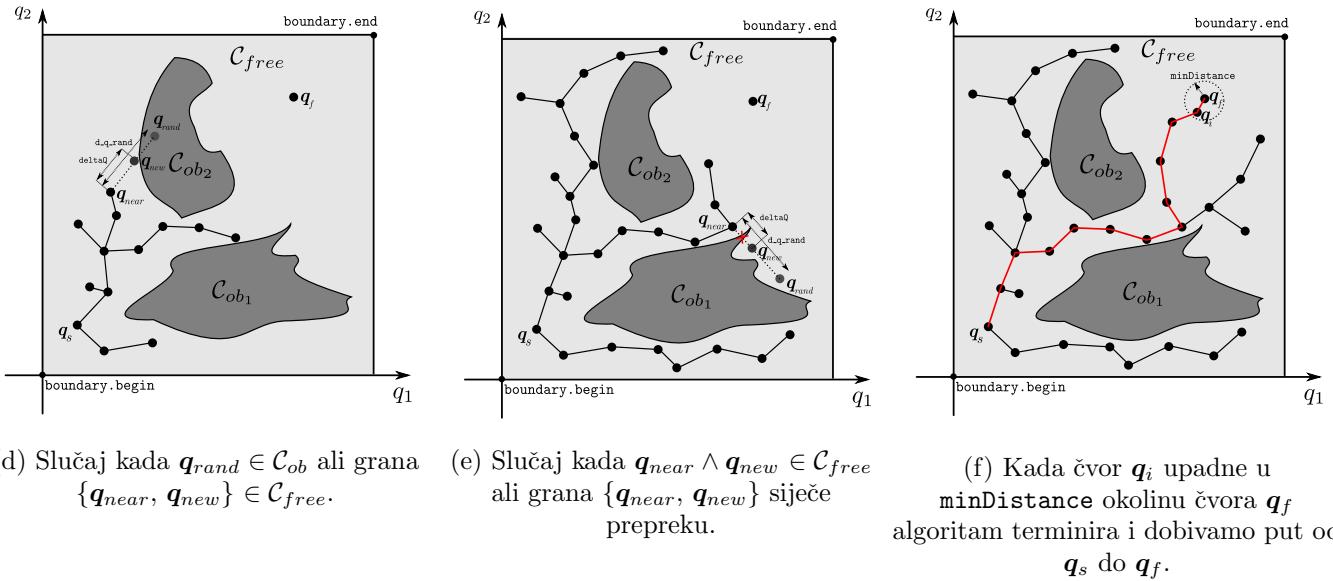
Ako je $d_{q_rand} < \text{deltaQ}$ onda dobiveni čvor \mathbf{q}_{rand} testiramo da li u spoju sa \mathbf{q}_{near} siječe neku od prepreka sa $\text{isSegmentInFreeSpace}(\mathcal{T}.\text{obstacles}, \mathbf{q}_{near}.\text{coordinates}, \mathbf{q}_{rand}.\text{coordinates})$ funkcijom koja vraća log. 0 ako sjećemo prepreke i log. 1 ako ne sjećemo. Ukoliko imamo presjek sa preprekama, idemo u novu iteraciju. Ukoliko ne sijećemo prepreke onda dodajemo u stablo čvor \mathbf{q}_{rand} sa $\mathcal{T}.\text{addNode}(\mathbf{q}_{rand})$ i dodajemo novonastalu granu sa $\mathcal{T}.\text{addEdge}(\mathbf{q}_{near}, \mathbf{q}_{rand})$.

Ako je $d_{q_rand} > \text{deltaQ}$, što će i najčešće biti slučaj, onda na duži koja spaja \mathbf{q}_{near} i \mathbf{q}_{rand} biramo novu konfiguraciju \mathbf{q}_{new} sa koordinatama koje se nalaze na udaljenosti deltaQ od \mathbf{q}_{near} u smjeru \mathbf{q}_{rand} tj. $\mathbf{q}_{new} = \mathcal{T}.\text{newConfig}(\mathbf{q}_{near}, \mathbf{q}_{rand})$. Sada ponavljamo isti postupak sa testiranjem presjeka sa preprekama sa $\text{isSegmentInFreeSpace}(\mathcal{T}.\text{obstacles}, \mathbf{q}_{near}.\text{coordinates}, \mathbf{q}_{new}.\text{coordinates})$. I ako imamo presjek onda idemo u novu iteraciju, a ako nemamo presjek onda dodajemo novi čvor \mathbf{q}_{new} u stablo sa $\mathcal{T}.\text{addNode}(\mathbf{q}_{new})$ i novonastalu granu sa $\mathcal{T}.\text{addEdge}(\mathbf{q}_{near}, \mathbf{q}_{new})$.

Ovaj proces ponavljamo u petlji dok ne dosegnemo maksimalni broj iteracija `maxIter` što znači da nismo pronašli rješenje, ili dok se ne desi da dobijemo čvor \mathbf{q}_{new} koji je na udaljenosti manjoj od `minDistance` od finalnog čvora \mathbf{q}_f što znači da smo pronašli rješenje, te terminiramo algoritam. Krajnji zadatak je dobiti vektor konfiguracija \mathbf{Q} koji sadrži sve konfiguracije od početne do finalne koje prave put (eng. *roadmap*) sa minimalnom težinom tj. $[\mathbf{Q}, L] = \mathcal{T}.\text{getShortestPath}(\mathbf{q}_s, \mathbf{q}_f)$ gdje je još L ukupna težina nađenog puta. Ovu je pretragu trivijalno izvesti Dijkstrinim algoritmom. Na slici 7 je prikazano generisanje i gradnja RRT stabla za razne slučajeve.

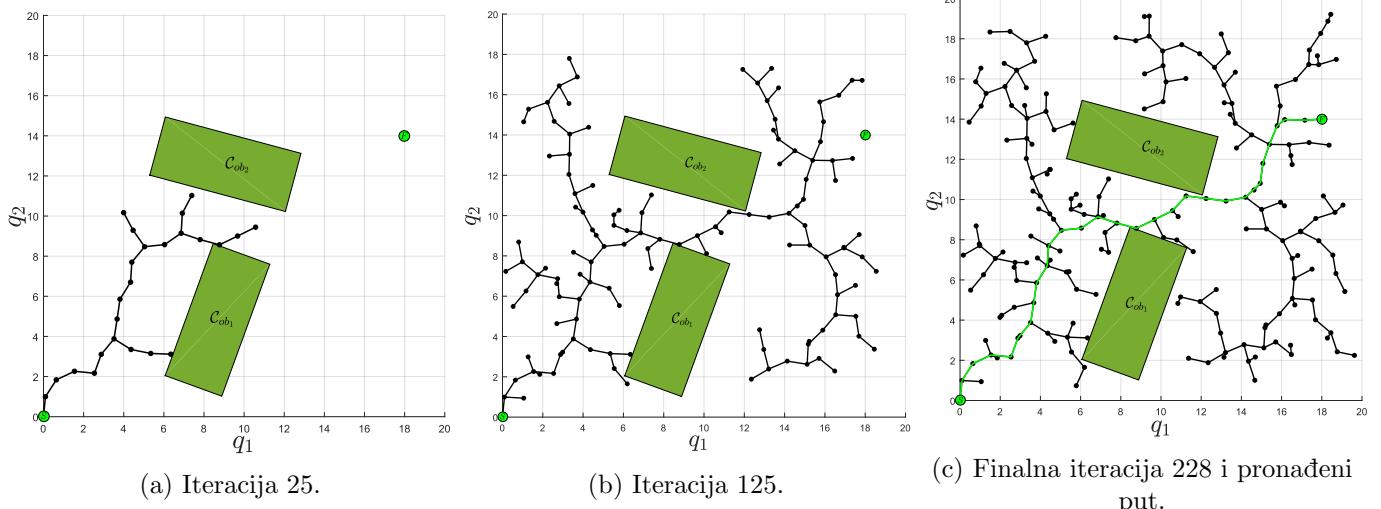


(a) Slučaj kada je $d_{q_rand} < \text{deltaQ}$. (b) Slučaj kada je $d_{q_rand} > \text{deltaQ}$. (c) Slučaj kada grana $\{\mathbf{q}_{near}, \mathbf{q}_{new}\}$ sijeće prepreku.

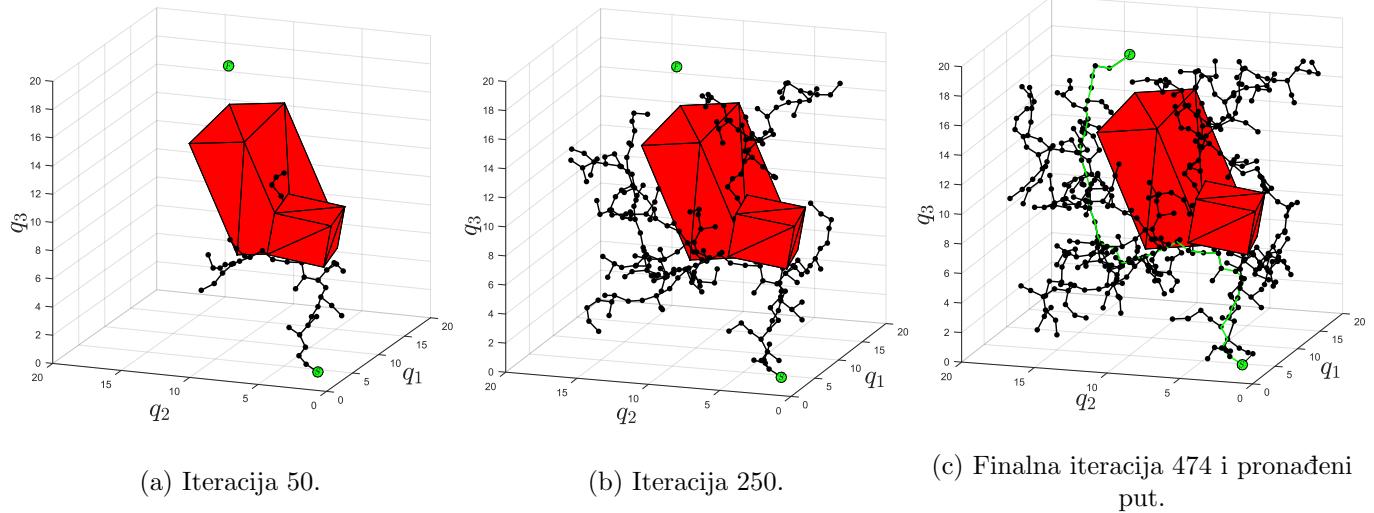


Slika 7: Pregled gradnje RRT stabla za razne slučajeve.

Pogledajmo sada kako izgleda primjena RRT algoritma na dva stvarna implementirana primjera. Na slici 8 je prikazano širenje RRT stabla za više iteracija u 2D konfiguracijskom prostoru $\mathcal{C} = \{(q_1, q_2) \in \mathbb{R}^2 : 0 < q_1 < 20, 0 < q_2 < 20\}$ sa preprekama dok je na slici 9 prikazan 3D konfiguracijski prostor $\mathcal{C} = \{(q_1, q_2, q_3) \in \mathbb{R}^3 : 0 < q_1 < 20, 0 < q_2 < 20, 0 < q_3 < 20\}$ sa preprekama.



Slika 8: Primjena RRT algoritma u 2D konfiguracijskom prostoru sa preprekama sa sljedećim parametrima: $\maxIter = 1000$, $\delta\text{taQ} = 1$, $\minDistance = 1$, $L = 27.3346$.



Slika 9: Primjena RRT algoritma u 3D konfiguracijskom prostoru sa preprekama sa sljedećim parametrima: `maxIter = 1000`, `deltaQ = 1`, `minDistance = 3`, $L = 42.6999$.

Za prostore čija je dimenzija veća od 3 ne možemo izvršiti ovakvu vizualizaciju ali bitan je izlaz algoritma - sekvenca konfiguracija koje čini put od početne do finalne konfiguracije.

2.2 BiRRT algoritam

Nakon pojave RRT algoritma prirodno su se i pojavile modifikacije istog. Jedna od najčešće implementiranih modifikacija RRT-a je RRT-Connect, poznat kao i BiRRT (eng. *Bidirectional RRT*) odnosno Dvosmjerni RRT algoritam [4]. Kako ćemo u nastavku vršiti modifikaciju RRT-Connect algoritma onda ćemo u dalnjem tekstu koristiti naziv BiRRT algoritam. BiRRT se zasniva na ideji gradnje dva stabla i to prvog sa korijenom u startnom čvoru \mathbf{q}_s , i drugog u korijenu u finalnom čvoru \mathbf{q}_f . U svakoj iteraciji algoritma, jedan od dva stabla prolazi kroz proces gradnje RRT-a kojeg smo opisali u prethodnom poglavljju. Kada čvorovi dva stabla dođu do neke predefinisane minimalne udaljenosti onda pokušamo spojiti dva stabla i ako ne sjećemo prepreke spajanje je uspješno.

Nekad nije pogodno definisati minimalnu udaljenost jer se može desiti da na pravolinijskom putu između čvorova dva stabla nema prepreka i mogli bismo ih direktno spojiti, međutim, to ne bismo mogli uraditi ako je udaljenost između tih čvorova veća od minimalne udaljenosti. Zbog toga je pogodnije testirati direktno, bez minimalne udaljenosti, mogući spoj čvorova dva stabla. Pogledajmo prvo pseudokod algoritma nakon čega ćemo dati detaljan opis istog.

```

Inicijaliziraj:  $\mathcal{T}_1, \mathcal{T}_2, q_s, q_f, \text{maxIter}, \text{deltaQ}, \text{boundary}, \text{obstacles}, \text{eta}$ 
function getBiRRT( $\mathcal{T}_1, \mathcal{T}_2, q_s, q_f, \text{eta}$ )
    1  $\mathcal{T}_1.\text{addNode}(q_s);$ 
    2  $\mathcal{T}_2.\text{addNode}(q_f);$ 
    3  $q_{last_1} \leftarrow q_s;$ 
    4  $\mathcal{T} \leftarrow \text{RRT}.\text{empty}();$ 
    5 for  $i = 1$  to  $\mathcal{T}_1.\text{maxIter}$  do
        6     if  $\text{rand}(0, 1) < \text{eta}$  then
            7          $q_{rand_1} \leftarrow \mathcal{T}_1.\text{randConfig}();$ 
            8          $[q_{near_1}, d\_q\_rand1] \leftarrow \mathcal{T}_1.\text{nearestNode}(q_{rand_1});$ 
            9         if  $d\_q\_rand1 < \mathcal{T}_1.\text{deltaQ}$  then
                10              $q_{new_1} \leftarrow q_{rand_1};$ 
                11             else
                12             |  $q_{new_1} \leftarrow \mathcal{T}_1.\text{newConfig}(q_{near_1}, q_{rand_1});$ 
                13             end
                14             if  $\text{isSegmentInFreeSpace}(\mathcal{T}_1.\text{obstacles}, q_{near_1}.\text{coordinates}, q_{new_1}.\text{coordinates})$ 
                15             then
                16                  $\mathcal{T}_1.\text{addNode}(q_{new_1});$ 
                17                  $\mathcal{T}_1.\text{addEdge}(q_{near_1}, q_{new_1});$ 
                18                  $q_{last_1} \leftarrow q_{new_1};$ 
                19             else
                20                 | continue;
                21             end
                22     else
                23          $q_{rand_2} \leftarrow \mathcal{T}_2.\text{randConfig}();$ 
                24          $[q_{near_2}, d\_q\_rand2] \leftarrow \mathcal{T}_2.\text{nearestNode}(q_{rand_2});$ 
                25         if  $d\_q\_rand2 < \mathcal{T}_2.\text{deltaQ}$  then
                26             |  $q_{new_2} \leftarrow q_{rand_2};$ 
                27             else
                28                 |  $q_{new_2} \leftarrow \mathcal{T}_2.\text{newConfig}(q_{near_2}, q_{rand_2});$ 
                29             end
                30             if  $\text{isSegmentInFreeSpace}(\mathcal{T}_2.\text{obstacles}, q_{near_2}.\text{coordinates}, q_{new_2}.\text{coordinates})$ 
                31             then
                32                  $\mathcal{T}_2.\text{addNode}(q_{new_2});$ 
                33                  $\mathcal{T}_2.\text{addEdge}(q_{near_2}, q_{new_2});$ 
                34             else
                35                 | continue;
                36     end
                37      $q_{try} \leftarrow \mathcal{T}_2.\text{nearestNode}(q_{last_1});$ 
                38     if  $\text{isSegmentInFreeSpace}(\mathcal{T}_1.\text{obstacles}, q_{last_1}, q_{try})$  then
                39         |  $\mathcal{T} \leftarrow \mathcal{T}_1.\text{mergeRRTs}(\mathcal{T}_2, q_{last_1}, q_{try});$ 
                40         | break;
                41     end
                42     return  $\mathcal{T};$ 
end

```

Pseudokod 2: Funkcija getBiRRT

Proglasimo čvor \mathbf{q}_{last_1} za zadnji uspješni dodani čvor u prvo stablo. Na početku algoritma će $\mathbf{q}_{last_1} = \mathbf{q}_s$. U narednim iteracijama će $\mathbf{q}_{last_1} = \mathbf{q}_{new_1}$ ali samo kada uspješno dodamo čvor \mathbf{q}_{new_1} u stablo. Pokušavanjem više mogućnosti kako birati dva čvora gdje izvršiti spoj dva stabla, najbrži način, je za \mathbf{q}_{last_1} čvor prvog stabla korištenjem funkcije `nearestNode` naći najbliži čvor (nazovimo ga \mathbf{q}_{try}) iz drugog stabla novododanom čvoru \mathbf{q}_{last_1} prvog stabla (koji je zadnji dodani čvor u prvo stablo) tj. $\mathbf{q}_{try} = \mathcal{T}_2.\text{nearestNode}(\mathbf{q}_{last_1})$. Zatim testiramo na koliziju granu $\{\mathbf{q}_{try}, \mathbf{q}_{last_1}\}$ sa preprekama funkcijom `isSegmentInFreeSpace`($\mathcal{T}_1.\text{obstacles}$, $\mathbf{q}_{try}.\text{coordinates}$, $\mathbf{q}_{last_1}.\text{coordinates}$) i ako nema sudara spajamo dva stabla granom $\{\mathbf{q}_{try}, \mathbf{q}_{last_1}\}$ i dobivamo jedno stablo. Spajanje vršimo komandom $\mathcal{T} = \mathcal{T}_1.\text{mergeRRTs}(\mathcal{T}_2, \mathbf{q}_{last_1}, \mathbf{q}_{try})$.

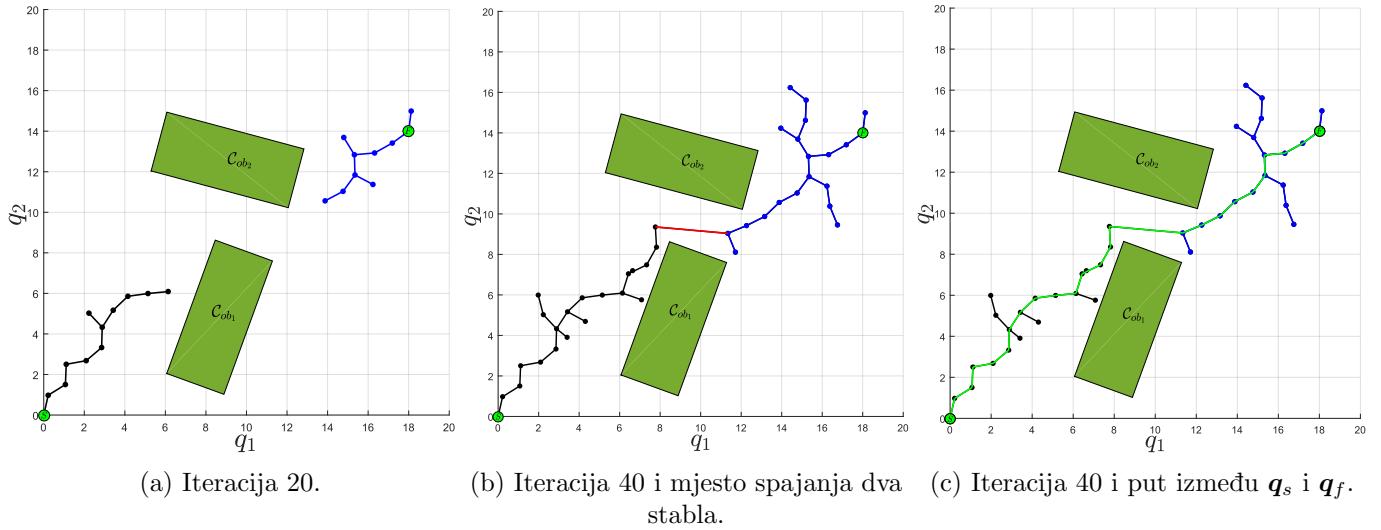
Napomenimo da oba stabla interno u sebi čuvaju iste prepreke tj. $\mathcal{T}_1.\text{obstacles} = \mathcal{T}_2.\text{obstacles}$ tako da nije bitno koje od ove dvije varijable šaljemo u `isSegmentInFreeSpace` funkciju. Također, primjetimo da ne trebamo incijalizirati `minDistance` varijablu i da će također varijable `maxIter` i `boundary` biti ista za oba stabla. Pošto su dva stabla simetrična onda bi ovaj način spajanja radio analogno ako bi imali da je $\mathbf{q}_{try} = \mathcal{T}_1.\text{nearestNode}(\mathbf{q}_{last_2})$.

Uvedimo parametar `eta` koji će za BiRRT algoritam biti $\text{eta} \equiv 0.50$. Ovaj parametar nam služi da, unutar petlje, širimo jedno pa drugo stablo na način da uzimamo uniformnu distribuciju između 0 i 1 kao `rand(0,1)` i poredimo je sa parametrom `eta`. Ako vrijedi da je `rand(0,1) < eta` onda generiramo čvorove za prvo stablo, u suprotnom za drugo. Pošto je $\text{eta} \equiv 0.50$ u prosjeku ćemo 50 % vremena širiti prvo i 50 % vremena drugo stablo što je i zamisao BiRRT algoritma. Napomenimo da originalni RRT-Connect algoritam deterministički alternira između dva stabla, ali za naše potrebe definicija alterniranja između dva stabla sa parametrom `eta` će biti pogodnija.

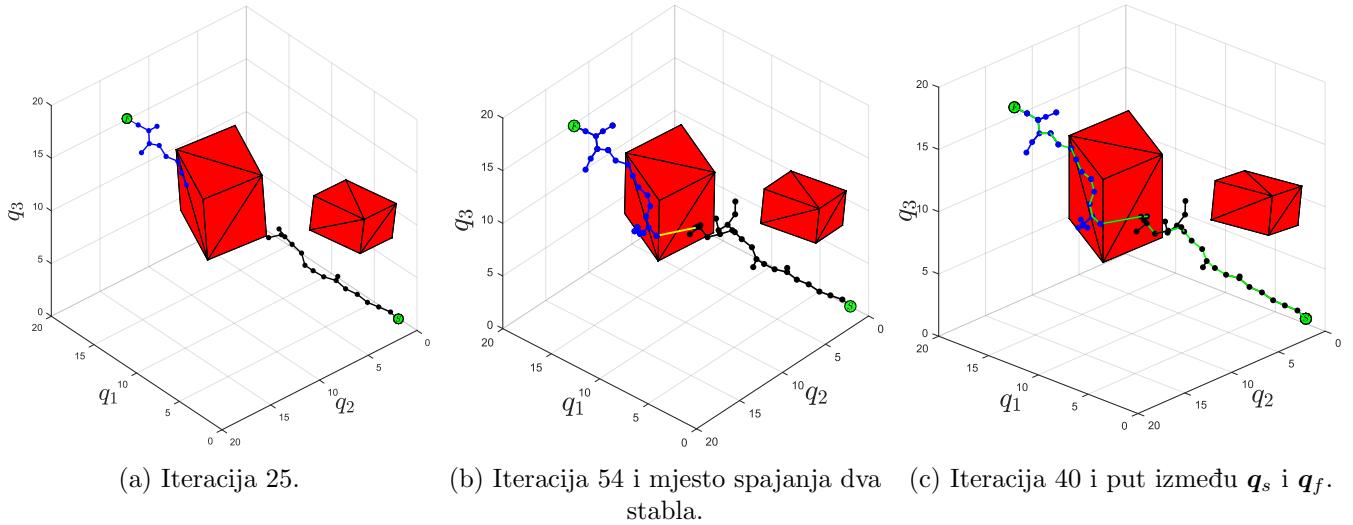
Opisano spajanje dva stabla nije optimalno rješenje tj. moguće je da postoji put koji je kraći ako napravimo vezu između druga dva čvora (ovo slijedi iz činjenice da spajamo drugo stablo sa čvorom \mathbf{q}_{last_1} koje može biti bilo gdje u prvom stablu). Optimalno rješenje bismo dobili kada bismo koristili funkciju $[\mathbf{q}_1, \mathbf{q}_2] = \text{connectRRTs}(\mathcal{T}_1, \mathcal{T}_2)$ koja prolazi dvostrukom petljom kroz sve čvorove dva stabla i trivijalnim poređenjem udaljenosti čvorova dva stabla traži ona dva koja su na najmanjoj udaljenosti. Očito je da je ovakvo traženje vremenski skupo jer moramo ovo raditi u svakoj iteraciji generisanja čvorova RRT stabla i kako broj čvorova raste broj prolazaka kroz petlje raste značajno (kao $n_{n_1} \cdot n_{n_2}$ gdje su n_{n_1} i n_{n_2} respektivni broj čvorova stabala).

Iako prvi način možda neće uvijek dati optimalno rješenje biti će i čest slučaj da imamo optimalno rješenje zbog širenja RRT stabala ka neistraženim područjima. Vremenski je efikasan način jer u svakoj iteraciji testiramo samo vezu dva čvora umjesto svih čvorova.

Pogledajmo sada kako izgleda primjena BiRRT algoritma na dva stvarna implementirana primjera. Na slikama 10 je prikazano širenje stabala za više iteracija u 2D konfiguracijskom prostoru sa preprekama dok je na slikama 11 prikazan 3D konfiguracijski prostor sa preprekama.



Slika 10: Primjena BiRRT algoritma u 2D konfiguracijskom prostoru sa preprekama sa sljedećim parametrima: `maxIter` = 1000, `deltaQ` = 1, `eta` = 0.50, L = 26.5909.



Slika 11: Primjena BiRRT algoritma u 3D konfiguracijskom prostoru sa preprekama sa sljedećim parametrima: `maxIter` = 1000, `deltaQ` = 1, `eta` = 0.50, L = 34.6349.

Primjetimo da je pronalaženje rješenja BiRRT algoritmom i do 10 puta brže nego običnim RRT algoritmom. Najveći razlog toga jeste kada imamo samo jedno stablo i ako je finalna konfiguracija skrivena među preprekama, potrebno je dosta iteracija da bi došli do te konfiguracije korištenjem jednog stabla. U slučaju kada imamo dva stabla onda se stablo sa korijenom u finalnoj konfiguraciji razgrana do slobodnijeg dijela prostora što olakšava spoj sa prvim stablom. Osim toga, čim se pojavi grana $\{q_{try}, q_{last_1}\} \in \mathcal{C}_{free}$ BiRRT algoritam odmah terminira (mada se ovo može i implementirati unutar RRT algoritma).

2.2.1 Upravljanje širenjem stabala

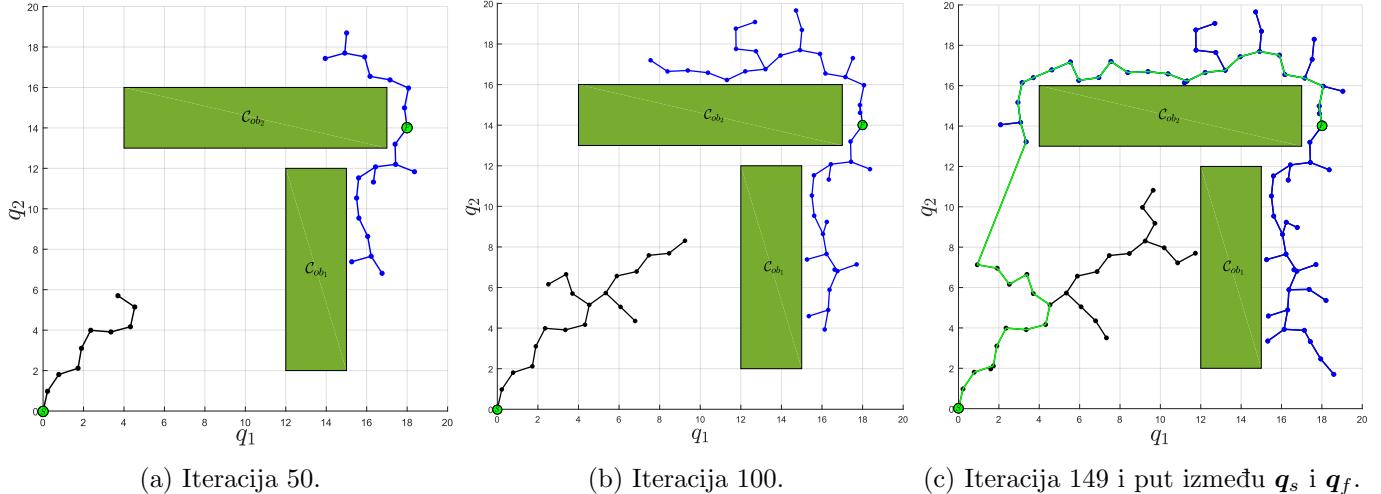
U prethodnoj implementaciji BiRRT algoritma imamo jedan jako zanimljiv parametar - `eta`. Smatrali smo da je on imao vrijednost 0.50 jer smo željeli da se oba stabla šire podjednako.

Postavlja se pitanje šta ako želimo da se jedno stablo brže širi od drugog.

Podešavanjem parametra `eta` možemo jednostavno upravljati širenjem stabala. Pošto nam je uslov generisanja čvorova stabala oblika $\text{rand}(0,1) < \text{eta}$ i ako je ovaj uslov tačan onda generiramo čvorove prvog stabla, u suprotnom čvorove drugog stabla. Iz ovog slijedi da je vjerovatnoća generiranja čvorova prvog stabla $p_1 = \text{eta}$, a za drugo stablo $p_2 = 1 - \text{eta}$ gdje $\text{eta} \in [0, 1]$.

Uzmimo primjer da je `eta` = 0.25 što znači da će u se prosjeku četvrtinu vremena generirati čvor prvog stabla, a tri četvrtine vremena čvor drugog stabla odnosno krajnji odnos broja čvorova stabala teži ka odnosu 1:4. U klasičnom slučaju kada je `eta` = 0.50 imamo da je odnos broja čvorova stabala 1:1 što smo i priželjkivali. Kada bi postavili `eta` = 1.00 to bi značilo da se širi samo prvo stablo tako da time dobivamo klasični RRT slučaj. Time možemo reći da nam parametar `eta` daje svojevrsnu generalizaciju spomenutih algoritama.

Razlog zbog kojeg bismo željeli da se jedno stablo brže širi od drugog ima najviše veze sa geometrijom prostora \mathcal{C} i \mathcal{C}_{obs} i koordinatama početne i finalne konfiguracije. Ako nam je unaprijed poznata pozicija prepreka i koordinata konfiguracija, možemo dati prednost širenja stabla one konfiguracije koja je više "zabačena" tako da to stablo može brže izaći iz takve nepovoljne pozicije. Ako bi imali klasični BiRRT onda bi se jedno stablo sa nepovoljnim korijenom polako širilo dok ne bi "ugledalo" drugo stablo koje se daleko brže širilo. Pogledajmo jedan primjer u 2D konfiguracijskom prostoru sa finalnom konfiguracijom koja je uklještena među preprekama na slikama 12.



Slika 12: Primjena BiRRT algoritma u 2D konfiguracijskom prostoru sa preprekama sa sljedećim parametrima: `maxIter` = 1000, `deltaQ` = 1, `eta` = 0.20, L = 42.1545.

2.3 Biased RRT/BiRRT algoritam

Zadnja modifikacija koju ćemo izvesti će biti vezana za samu probabilističku prirodu RRT algoritma. Kako smo već spomenuli, parametrom `eta` unutar BiRRT algoritma možemo generalisati i RRT i

BiRRT algoritam. Bez umanjenja opštosti, posmatrat ćemo samo BiRRT algoritam sa upravljanjem širenjem stabala.

Ideja iza Biased (bos. *pristrasan*) RRT/BiRRT algoritma je da manipulišemo nasumičnim uzorcima \mathbf{q}_{rand} na način da ih povremeno odabiramo deterministički. Uvedimo novi parametar $\mathbf{mi} = [\mathbf{mi}_1, \mathbf{mi}_2]$ gdje se \mathbf{mi}_1 odnosi na prvo stablo, a \mathbf{mi}_2 na drugo stablo. Ovi parametri predstavljaju vjerovatnoću stoga pripadaju intervalu $[0,1]$. Posmatrajmo slučaj kada ulazimo u iteraciju u kojoj generišemo čvor za drugo stablo. Imamo poznat zadnji novi čvor prvog stabla tj. \mathbf{q}_{new_1} . Prvi korak generisanja nasumičnog uzorka drugog stabla bi bio:

$$\mathbf{q}_{rand_2} \leftarrow \mathcal{T}_2.\text{randConfig}();$$

ali ćemo sada taj korak zamijeniti sa:

```
if rand(0, 1) < mi2 then
    | qrand2 ← qnew1;
else
    | qrand2 ← T2.randConfig();
end
```

Ono što smo uradili jeste da, zavisno od vrijednosti vjerovatnoće \mathbf{mi}_2 nasumična konfiguracija drugog stabla (\mathbf{q}_{rand_2}) postaje ono što je bila nova konfiguracija prvog stabla u prošloj iteraciji (\mathbf{q}_{new_1}). Ako se podsjetimo kako se generira nova konfiguracija stabla, znamo da je ona udaljena za δQ od najbližeg čvora (\mathbf{q}_{near_2}) u smjeru nasumičnog čvora (\mathbf{q}_{rand_2}). Kako je sad nasumični čvor drugog stabla ustvari novi čvor prvog stabla iz prošle iteracije onda to znači da se drugo stablo širi u smjeru širenja prvog stabla!

Ukoliko $\text{rand}(0, 1) > \mathbf{mi}_2$ onda samo generišemo obični nasumični čvor drugog stabla i drugo stablo se širi nasumično.

Sve rečeno vrijedi analogno i za prvo stablo:

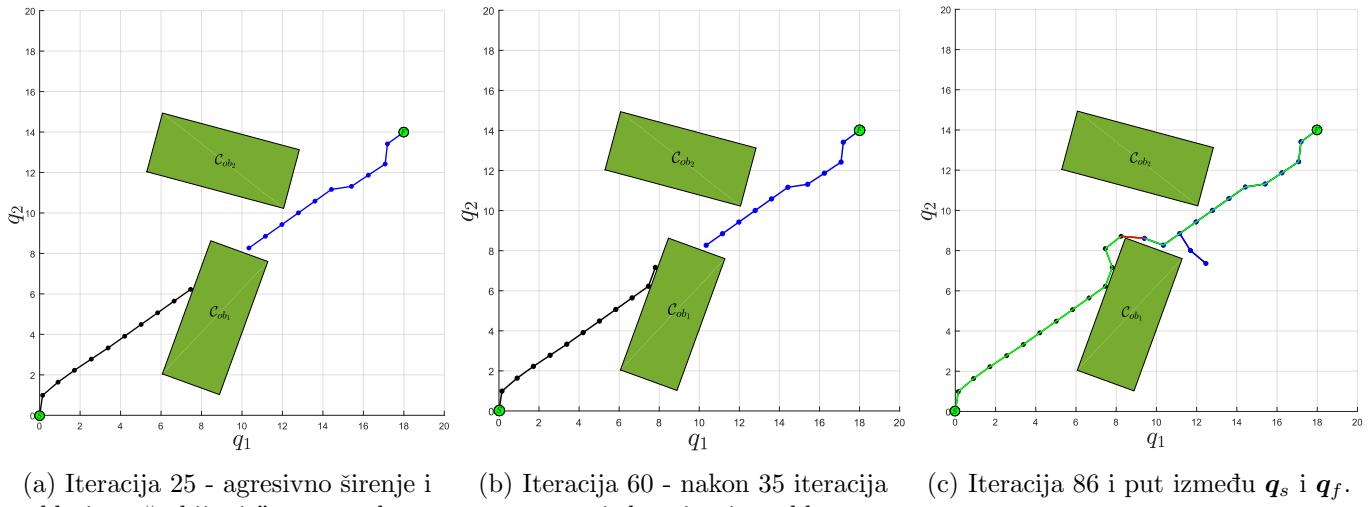
```
if rand(0, 1) < mi1 then
    | qrand1 ← qnew2;
else
    | qrand1 ← T1.randConfig();
end
```

Ono što smo ustvari uradili jeste da smo omogućili “pristrasno” širenje jednog stabla ka drugom i obratno. Ovo je jako moćan alat u smislu da će u pozadini stabla biti usmjerena jedan ka drugom tako da se neće trošiti bespotrebno vrijeme na širenje u prostor koji nije od zanačaja za pronalazak puta. Moramo također biti oprezni koje vrijednosti postavljamo za parametre \mathbf{mi}_1 i \mathbf{mi}_2 . Ako imamo mnogo prepreka ili konfiguraciju koja se nalazi u nepovoljnem položaju, onda nam nije u interesu stavljati prevelike vrijednosti za parametre \mathbf{mi}_1 i \mathbf{mi}_2 jer se može desiti da se “zabijemo” u prepreku i ne možemo je izbjegći jer većina novih konfiguracija ide u smjeru prepreke s obzirom da je drugo stablo u slobodnom prostoru iza prepreke. I dok se drugo stablo dovoljno ne razgrana, ostajemo “zabijeni” u prepreku. Stoga je logično da se za prostore sa dosta prepreka ili sa nepovoljnim kořijenom jednog od stabala stavi manja vjerovatnoća - do otprilike 0.25. Ukoliko imamo povoljniji prostor onda možemo ići i sa vjerovatnoćama i preko 0.80 što će rezultirati agresivnijim širenjem stabala jednog ka drugom. Opet napominjemo da to može u većini slučajeva biti korisno, ali nekad i

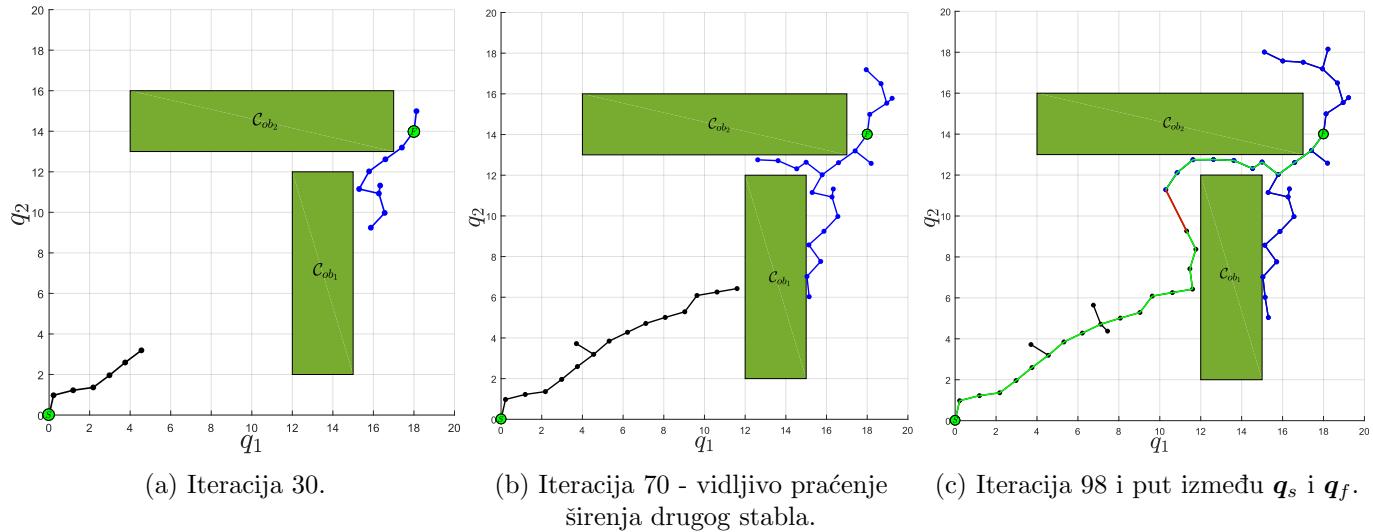
loša ako se "zabijemo" u prepreke. Uvijek je poželjno postaviti neku nenultu vrijednost vjerovatnoće tako da se stabla koliko - toliko šire jedno prema drugom. Ukoliko postavimo $mi = [0.00, 0.00]$ onda imamo opet obični BiRRT algoritam čime smo uveli dodatno generalisanje algoritma.

Ova ideja povezana sa idejom upravljanja širenjem stabala može biti efikasna kombinacija. Primjer dobre primjene ove kombinacije bi bio slučaj kada je korijen drugog stabla u nepovoljnem položaju, onda želimo da smanjimo parametar **eta** (jer je vjerovatnoća generisanja čvorova drugog stabla $1 - \text{eta}$) i smanjimo mi_2 dok za korijen prvog stabla koji je u povoljnijem položaju povećamo mi_1 tako da se brže širi prema "zarobljenom" drugom stablu. Iako generišemo manje čvorova prvog stabla, oni koji su generisani će se biti usmjereni prema smjeru širenja drugog stabla tako da se kompenzira manji broj čvorova. Ovakav rezon može dosta smanjiti vrijeme izvršavanja i dužinu nađenog puta uz gotovo nikakav drugi utrošak, ali puni potencijal se tek ogleda kada radimo u višedimenzionalnim prostorima koja imaju velika područja koja nije nužno istražiti ali u običnom RRT i BiRRT algoritmu na to ne možemo uticati. Međutim, sada možemo po volji smanjiti pretragu tog područja koje nije isplativo pretraživati. Pored svih pogodnosti ove modifikacije najveći uticaj imamo na smanjenje dužine puta što ćemo vidjeti u sljedećim primjerima.

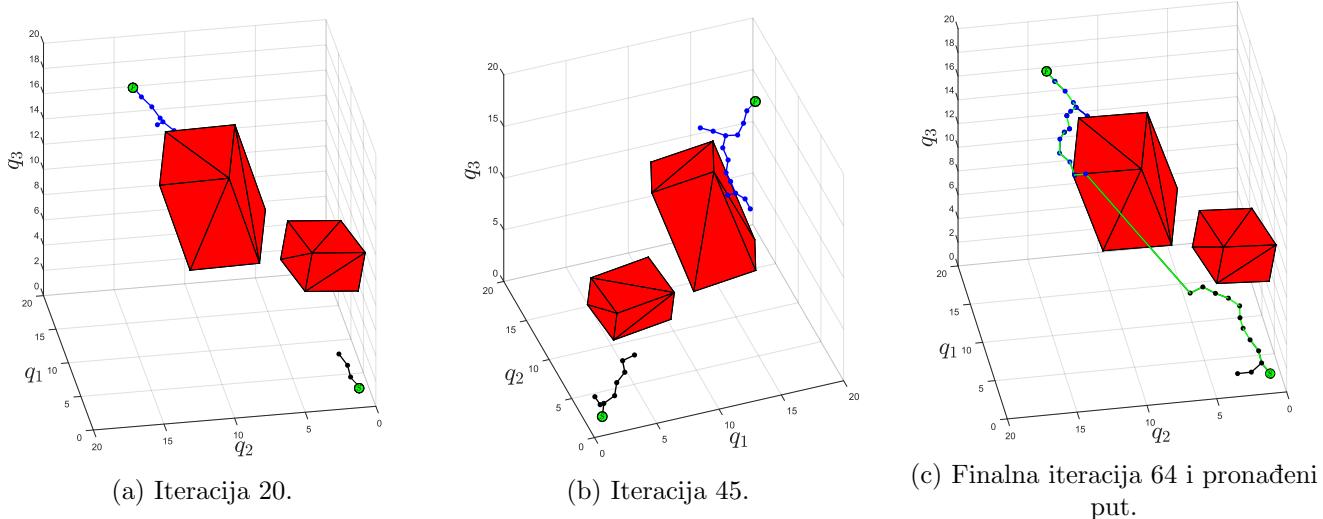
Pogledajmo sada tri primjera korištenja ovog algoritma i to dva 2D slučaja: sa povolnjim konfiguracijskim prostorom za oba korjena stabala (slika 13) i sa nepovolnjim konfiguracijskim prostorom za jedan od korjena (slika 14) i primjer u 3D konfiguracijskom prostoru (slika 15).



Slika 13: Primjena Biased BiRRT algoritma u 2D konfiguracijskom prostoru sa preprekama sa sljedećim parametrima: $\text{maxIter} = 1000$, $\text{deltaQ} = 1$, $\text{eta} = 0.50$, $mi = [0.80, 0.80]$, $L = 25.1631$.



Slika 14: Primjena Biased BiRRT algoritma u 2D konfiguracijskom prostoru sa preprekama sa sljedećim parametrima: `maxIter = 1000`, `deltaQ = 1`, `eta = 0.25`, `mi = [0.60, 0.30]`, $L = 28.8219$.



Slika 15: Primjena RRT algoritma u 3D konfiguracijskom prostoru sa preprekama sa sljedećim parametrima: `maxIter = 1000`, `deltaQ = 1`, `minDistance = 3`, `eta = 0.30`, `mi = [0.25, 0.30]`, $L = 34.4868$.

Kako ova nadogradnja RRT algoritma daje najbolje rezultate i daje najgeneralniji algoritam, nju ćemo koristiti za T-RRT algoritam koji je tema ovog rada.

3 Težinske mape i težinske funkcije

Težinske mape (eng. *costmaps*) i težinske funkcije (eng. *cost functions*) su matematski koncepti korišteni u *optimizaciji*. U optimizacijskom problemu težimo da minimiziramo težinsku funkciju da bi ispunili neki kriterij. Da bismo konstruisali i objasnili težinske mape trebamo prvo objasniti koncept težinskih funkcija.

Težinska funkcija je neprekidna funkcija koja uzima vektor varijabli koje definišu dati problem i vraća realan broj koji reprezentira "težinu" (eng. *cost*) datog vektora varijabli. Iz ove definicije vidimo da težinska funkcija spada u vrstu *skalarnih* funkcija. Obično je praksa da težina bude pozitivan realan broj dok za svrhu problema kojeg mi obrađujemo u ovom radu, vektor varijabli će biti višedimenzionalna konfiguracija iz jednačine 2.2. Tako da, strogo matematski rečeno, težinska funkcija f_c mapira konfiguraciju \mathbf{q} iz konfiguracijskog prostora \mathcal{C} u realan broj c kojeg nazivamo težina:

$$f_c : \mathcal{C} \rightarrow \mathbb{R}_+ \quad (3.1)$$

$$f_c(\mathbf{q}) = c \quad (3.2)$$

Korištenjem ove definicije, ako mapiramo svaku konfiguraciju iz \mathcal{C} u \mathbb{R}_+ onda smo konstruisali *težinsku mapu* tako da svaka konfiguracija iz \mathcal{C} ima korespondentnu težinu u \mathbb{R}_+ . Primjetimo da prostor težinske mape \mathcal{C}_c ima dimenziju $\dim(\mathcal{C}_c) = \dim(\mathcal{C}) + 1$ gdje dodatna dimenzija dolazi od težine c težinske funkcije. Minimizacijom težinske funkcije odnosno težinske mape (koja je sačinjena od težina težinske funkcije), po nekom kriteriju, dobivamo optimalno rješenje problema. Sad se postavlja sljedeće pitanje:

Kako biramo težinsku funkciju f_c ?

Odgovor na ovo pitanje zavisi od problema kojeg pokušavamo riješiti. Jedna od najčešće korištenih težinskih funkcija je kvadratna težinska funkcija:

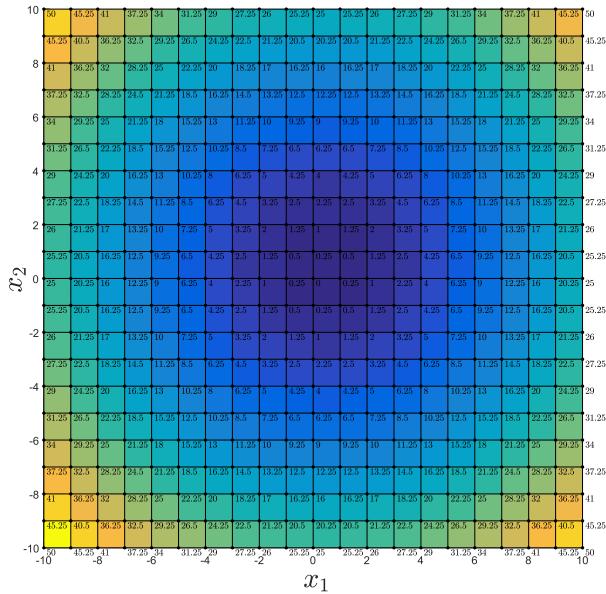
$$f_c(x) = A(t - x)^2$$

gdje je A pozitivna konstanta, x varijabla koju optimiziramo (u ovom slučaju smatramo da je x skalar) i t je konfiguracija kojoj težimo. Ona se primjenjuje u mnogim optimizacijskim algoritmima kao *metoda najmanjih kvadrata*, *metode regresije*, itd. Vidimo da bi kriterij po kojem vršimo optimizaciju bio minimalna udaljenost od date konfiguracije x do ciljane konfiguracije t .

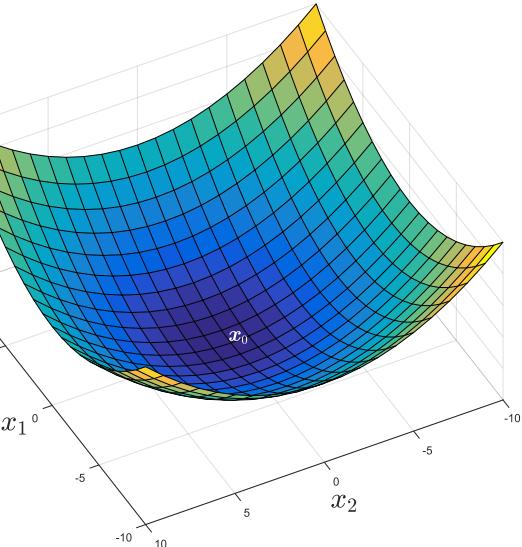
Pogledajmo primjer generisanja težinske mape za kvadratnu težinsku funkciju. Smatrajmo da imamo tačku $\mathbf{x}_0 = [0, 0]$ u 2D konfiguracijskom prostoru kojoj želimo prići, onda bi na ugled na kvadratnu težinsku funkciju imali da je ekvivalentna funkcija kada je x vektor:

$$f_c(\mathbf{x}) = A \cdot \|\mathbf{x}_0 - \mathbf{x}\|^2$$

Uzmimo, konfiguracijski prostor \mathcal{C} , kvadrat u 2D prostoru definisan kao $\mathcal{C} = \{(x_1, x_2) \in \mathbb{R}^2 : -10 < x_1 < 10, -10 < x_2 < 10\}$. Pošto je $\dim(\mathcal{C}) = 2$ onda će dimenzija prostora težinske mape \mathcal{C}_c biti $\dim(\mathcal{C}_c) = 3$. Finalna težinska mapa je prikazana na slici 16.



(a) Pogled na težinsku mapu iz z ose. Svaka tačka ima odgovarajuću težinu c .



(b) Puni pogled na težinsku mapu.

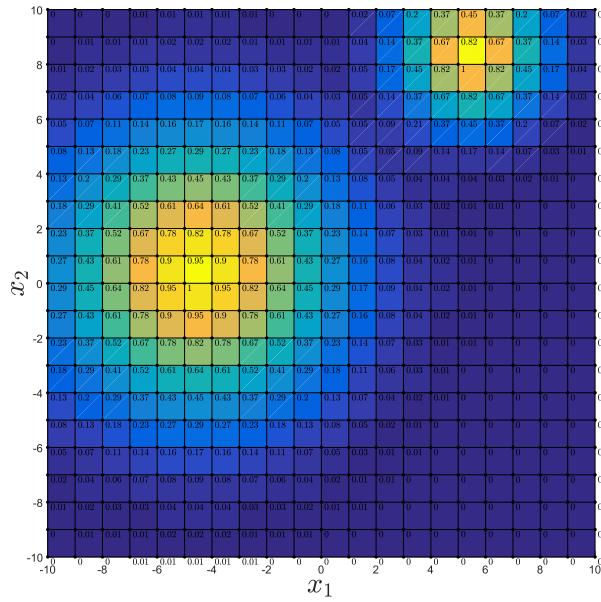
Slika 16: Težinska mapa generisana težinskom funkcijom: $f_c(\mathbf{x}) = 0.25 \cdot \|\mathbf{x}_0 - \mathbf{x}\|^2$.
Kriterij težinske funkcije je minimalna udaljenost do ciljane konfiguracije \mathbf{x}_0 .

Dakle, vidimo da ciljana konfiguracija mora imati minimalnu težinu i gdje god da se nalazimo u konfiguracijskom prostoru moramo se kretati u smjeru koji ima što manju težinu da bi došli do ciljane konfiguracije. U slučaju ovakve kvadratne težinske funkcije, kriterij po kojem minimiziramo težinsku mapu je minimalna udaljenost do ciljne konfiguracije.

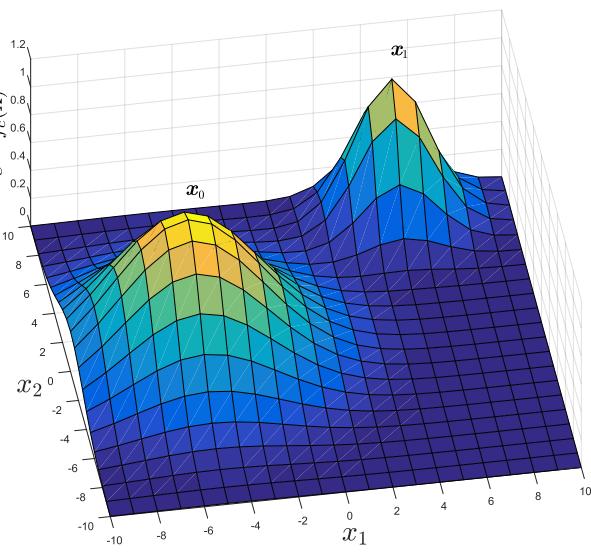
Ako bismo željeli suprotno - da se nalazimo što je dalje moguće od neke konfiguracije ili više njih jedna od pogodnih težinskih funkcija bi bila:

$$f_c(\mathbf{x}) = A \cdot \exp(-B \cdot \|\mathbf{x}_0 - \mathbf{x}\|^2) + C \cdot \exp(-D \cdot \|\mathbf{x}_1 - \mathbf{x}\|^2),$$

gdje pokušavamo izbjegavati dvije konfiguracije \mathbf{x}_0 i \mathbf{x}_1 . Poprilično je trivijalno generalizirati izbjegavanje bilo koji broj konfiguracija. Uz biranje pogodnih konstanti unutar težinske funkcije možemo dati veći ili manji "prioritet" odgovarajućim konfiguracijama. Težinska mapa izbjegavanja dvije konfiguracije i prostora oko njih je prikazana na slici 17.



(a) Pogled na težinsku mapu iz z ose. Svaka tačka ima odgovarajuću težinu c .



(b) Puni pogled na težinsku mapu.

Slika 17: Težinska mapa generisana težinskom funkcijom:

$$f_c(\mathbf{x}) = \exp(-0.05 \cdot \|\mathbf{x}_0 - \mathbf{x}\|^2) + \exp(-0.2 \cdot \|\mathbf{x}_1 - \mathbf{x}\|^2).$$

Kriterij težinske funkcije je što veća udaljenost od konfiguracija \mathbf{x}_0 i \mathbf{x}_1 .

U ovom primjeru vidimo da konfiguracije koje pokušavamo izbjegići i prostor blizak njima imaju visoku težinu tako da prateći isti koncept da idemo u smjeru gdje se male težine uspjevamo ispuniti kriterij izbjegavanja konfiguracija.

Najveći izazov sada postaje biranje pogodne težinske funkcije u zavisnosti od problema s kojim radimo. U ovom radu se najviše baziramo na probleme povezane sa udaljenošću tako da će dva prethodna kriterija koja smo predstavili biti sasvim dovoljna da postignemo zamišljeno.

Postoje razne metode koje se koriste za traženje minimalnog puta od početne do krajnje konfiguracije u prostoru težinske mape kao što su *Metod gradijentnog spusta*, *Newton-ov metod*, *Lagrange-Newton-ov metod*, itd.

Koncepti težinskih mapa u kombinaciji sa prethodnim RRT algoritmima daju jako moćan alat za traženje optimalnih putanja. Sada ćemo se sa RRT stablima širiti po težinskoj mapi gdje će konfiguracije q_i dobiti dodatnu dimenziju koja će čuvati informaciju o težini pojedinih čvorova. Sa pogodnim težinskim funkcijama i pogodnim algoritmima za minimiziranje puta od početne do finalne konfiguracije ćemo biti u mogućnosti rješavati jako interesantne probleme iz oblasti robotike. Upravo je ova ideja jezgro T-RRT algoritma, kojeg objašnjavamo u nastavku.

4 T-RRT algoritam

T-RRT (eng. *Transition-based RRT*) algoritam je modifikacija RRT algoritma koja na osnovu korištenja konfiguracijskog prostora definisanog težinskim mapama pronalazi niskotežinske puteve koji prate "prevoje", "kanjone" i "doline" na težinskoj mapi od početne do finalne konfiguracije [1]. Novi, potencijalni uzorci se biraju na osnovu *prelaznih testova* (eng. *transition tests*) čiji se interni parametri modifikuju u zavisnosti od uspješnosti generiranja novih uzoraka. Prelazni testovi, na osnovu poslatog uzorka vraćaju izlaz tipa `bool` o tome da li je novodobiveni uzorak legitiman ili ne, tako da je izmjena pseudokoda minimalna. Uz provjeru da li grana $\{q_{near}, q_{new}\}$ siječe prepreke imat ćeemo još dvije funkcije T-RRT algoritma: `transitionTest` i `minExpandControl` koje daju log. 1 ako možemo dodati čvor u stablo odnosno log. 0 ako ne možemo. Uslov dodavanja čvora q_{new} i grane $\{q_{near}, q_{new}\}$ u stablo će biti:

```
if isSegmentInFreeSpace( $\mathcal{T}.obstacles$ ,  $q_{near}.coordinates$ ,  $q_{new}.coordinates$ )
and  $\mathcal{T}.transitionTest(q_{near}, q_{new})$ 
and  $\mathcal{T}.minExpandControl(q_{near}, q_{rand})$  then
|    $\mathcal{T}.addNode(q_{new});$ 
|    $\mathcal{T}.addEdge(q_{near}, q_{new});$ 
else
|   continue;
end
```

Kako se nalazimo na težinskoj mapi, onda stabla moraju ostati "priljepljena" za mapu jer zadnja koordinata čvorova čuva težinu što je ujedno i vrijednosti zadnje dimenzije težinske mape. Pošto koristimo BiRRT algoritam u kombinaciji sa T-RRT algoritmom, onda nećemo moći direktno vezati stabla kada se pojave dva čvora koja se mogu spojiti bez prepreka kao što smo radili prije. Razlog je što moramo ostati vezani za mapu. Kada bismo povezali direktno dva stabla koja su na većoj udaljenosti onda bi grana koja veže dva stabla ili "lebdjela" na težinskoj mapi ili "probila" težinsku mapu. Zbog ovog opet moramo incijalizirati atribut `minDistance` čime ograničavamo minimalnu udaljenost na kojoj dva stabla moraju biti da bi se mogla povezati. Dodatna izmjena koda će biti spoj uslova za terminiranje RRT i BiRRT algoritma prikazano ispod:

```
 $q_{try} \leftarrow \mathcal{T}_2.nearestNode(q_{last_1});$ 
if distance( $q_{last_1}.coordinates$ ,  $q_{try}.coordinates$ ) <  $\mathcal{T}_1.minDistance$ 
and isSegmentInFreeSpace( $\mathcal{T}_1.obstacles$ ,  $q_{last_1}.coordinates$ ,  $q_{try}.coordinates$ ) then
|    $\mathcal{T} \leftarrow \mathcal{T}_1.mergeRRTs(\mathcal{T}_2, q_{last_1}, q_{try});$ 
|   break;
end
```

Vidimo da je minimalna izmjena unutar RRT algoritma da bi dobili T-RRT algoritam. Objasnimo sada kako rade prelazni testovi.

4.1 Implementacija funkcije `testTransition`

Prelazni testovi su konstruisani na bazi principa stohastičke optimizacije. Pošto imamo velik broj novih varijabli koje su inherentne za funkcije `transitionTest` i `minExpandControl` onda ćemo naslijediti iz klase `RRT` klasi `TRRT` koja će sadržavati spomenute dvije funkcije i njihove parametre. Pogledajmo prvo pseudokod za funkciju `tranistionTest` pa ćemo zatim pokazati konstrukciju

klasa, nakon toga objasniti značaj parametara i na kraju kako radi algoritam. Pseudokod za funkciju **transitionTest** (koja je metoda TRRT klase, tako da se stablo inherentno može pozivati u njoj (kao i njegove metode) iako ga nismo direktno poslali kao parametar) je:

```
Inicijaliziraj: T, K, alpha, c_max, nFails, maxFails
function testTransition( $q_i, q_j$ )
    1 c_i = costFunction( $q_i$ );
    2 c_j = costFunction( $q_j$ );
    3 d_ij = distance( $q_i.coordinates, q_j.coordinates$ );
    4 if  $c_j > T.c\_max$  then
        5 | return false;
    6 end
    7 if  $c_j < c_i$  then
        8 | return true;
    9 end
    10 if  $rand(0,1) < T.getTranProbability(c_i, c_j, d_ij)$  then
        11 |  $T.T = T.T/\alpha$ ;
        12 |  $T.nFails = 0$ ;
        13 | return true;
    14 else
        15 | if  $T.nFails > T.maxFails$  then
            16 | |  $T.T = T.T * \alpha$ ;
            17 | |  $T.nFails = 0$ ;
        18 | else
            19 | |  $T.nFails = T.nFails + 1$ ;
        20 | end
        21 | return false;
    22 end
end
function getTranProbability( $c_i, c_j, d_ij$ )
    1 p = 1;
    2 deltaC =  $c_j - c_i$ ;
    3 if  $\delta C > 0$  then
        4 | p =  $\exp(-\delta C / (d_{ij} * T.K * T.T))$ ;
    5 end
    6 return p;
end
```

Pseudokod 3: Funkcija `testTransition` i pomoćna funkcija `getTranProbability`

Pogledajmo parametre korištene u prethodnom pseudokodu pa ćemo zatim objasniti kako algoritam radi.

- Naslijedena klasa TRRT ima sljedeća atributte koji su korišteni prilikom izvršavanja funkcije `testTransition`:
 - Atribut T se naziva *temperatura* i to je parametar koji kontroliše nivo vjerovatnoće dodavanja konfiguracije q_2 na osnovu prirasta težine grane $\{q_1, q_2\}$.
 - Atribut K je konstanta koja se koristi radi normalizacije vjerovatnoće dodavanja novih konfiguracija. Njena vrijednost se uzima kao prosjek težina početne i finalne konfiguracije

kako su to jedine težine poznate na početku pretrage.

- Atribut `alpha` je konsanta koja upravlja sa adaptacijom temperature u zavisnosti uspješnosti dodavanja novih konfiguracija.
- Atribut `c_max` je maksimalna dopuštena težina kojoj smijemo pristupiti u pretrazi. Time bi se prostor koji ima težinu višu od `c_max` ponašao kao prepreka jer tom prostoru ne možemo prići.
- Atribut `nFails` je broj uzastopnih neuspješnih dodavanja novih konfiguracija.
- Atribut `maxFails` je maksimalan broj uzastopnih neuspješnih dodavanja `nFails`. Ako prekoračimo parametar `maxFails` onda parametar `nFails` resetiramo.

Iz pseudokoda sa početka ovog poglavlja, koji pokazuje koje izmjene uvesti radi adaptacije T-RRT algoritma, vidimo da funkcija `transitionTest` prima čvorove \mathbf{q}_{near} i \mathbf{q}_{new} . Slijedi da će u pseudokodu za `transitionTest` funkciju $\mathbf{q}_1 = \mathbf{q}_{near}$ i $\mathbf{q}_2 = \mathbf{q}_{new}$. Vrijednosti `c_i` i `c_j` predstavljaju respektivne težine dva ulazna čvora dobivene unaprijed definisanim težinskim funkcijom `costFunction`. Vrijednost `d_ij` je udaljenost između dva čvora.

`if` naredba iz linije 4 testira da li je težina `c_j` čvora \mathbf{q}_{new} veća od maksimalne dozvoljene težine `c_max`. Ako jeste vraćamo log. 0 jer tom prostoru ne smijemo pristupiti čime ograničavamo pristup nepovoljnim konfiguracijama. Tako da se sav prostor koji ima težinu veću od `c_max` efektivno ponaša kao prepreka.

`if` naredba iz linije 7 testira da li je težina čvora \mathbf{q}_{new} manja od težine čvora \mathbf{q}_{near} . Ako jeste onda vraćamo log. 1 što ima smisla jer dodajemo granu u stablo koje ima negativan nagib jer vežemo \mathbf{q}_{new} sa \mathbf{q}_{near} koji se već nalazi u stablu čime se širimo prema niskotežinskim regijama.

Vjerovatnoća prelaza (eng. *transition probability*) p_{ij} (koju dobivamo pomoćnom funkcijom `getTranProbability`) je definisana kao:

$$p_{ij} = \begin{cases} \exp(-\frac{\Delta c_{ij}}{K \cdot T}) & \text{ako } \Delta c_{ij} > 0, \\ 1 & \text{inače,} \end{cases} \quad (4.1)$$

gdje je $\Delta c_{ij} = \frac{c_j - c_i}{d_{ij}}$, nagib (eng. *slope*) grane $\{\mathbf{q}_{near}, \mathbf{q}_{new}\}$, odnosno razlika težina podjeljenja sa udaljenošću dvije konfiguracije. Temperaturu T i konstantu K smo već objasnili. Ovakvom definicijom vjerovatnoće prelaza, potencijalne grane koje imaju negativan nagib ($\Delta c_{ij} < 0$) imaju vjerovatnoću prelaza 1 jer su usmjerene ka niskotežinskim konfiguracijama. U protivnom, kada imamo grane sa pozitivnim nagibom vjerovatnoća prelaza eksponencijalno opada sa strminom nagiba.

Vjerovatnoća prelaza se poredi sa nasumičnim brojevima uniformne raspodjele iz intervala $[0,1]$ u `if` naredbi iz linije 10. U slučaju uspješnog prelaza, temperaturu smanjujemo `alpha` puta čime indirektno utičemo na smanjenje vjerovatnoće prelaza u sljedećoj iteraciji jer kako se temperatura smanjuje tako se vjerovatnoća prelaza smanjuje prema jednačini 4.1. Ovo bi fizički značilo da ulazimo u visokotežinsku regiju prostora \mathcal{C}_c jer odobravamo prelaze koji imaju pozitivan nagib (iz 4.1) pa se nije pogodno širiti u takvom regionu. Dalje resetiramo broj uzastopnih neuspješnih prelaza `nFail` na nulu i šaljemo log. 1 da je odabrana konfiguracija sa pozitivnim nagibom povoljna za širenje stabla. U slučaju neuspješnog prelaza, poredimo da li smo prekoračili maksimalni broj neuspješnih prelaza `maxFails` i ako jesmo, onda temperaturu povećavamo `alpha` puta što indirektno u sljedećoj iteraciji

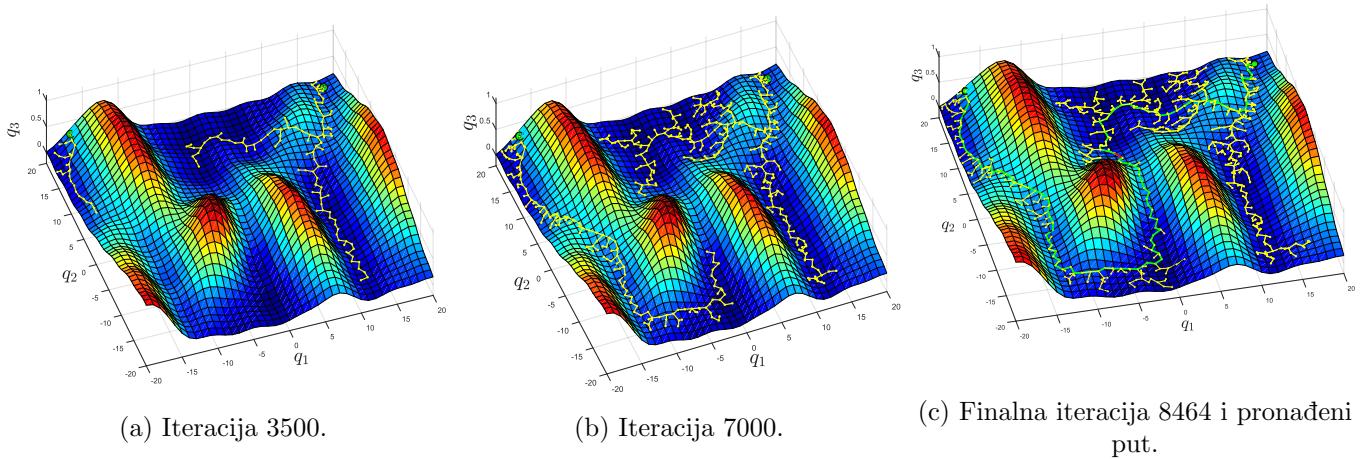
povećava vjerovatnoću prelaza. Ovo fizikalno znači da povećavamo vjerovatnoću prelaza jer nismo već `maxFails` iteracija mogli napraviti prelaz čime odobravamo kretanje ka regiji sa višim nagibom. Nakon adaptacije temperature resetiramo broj uzastopnih neuspješnih prelaza `nFails` na nulu. U slučaju da nismo prekoračili `maxFails` onda `nFails` inkrementiramo za jedan i nakon izlaska iz `if` naredbe šaljemo log. 0.

Temperatura, uz nagib Δc_{ij} , je ključni faktor uspješnosti prelaza. Na početku postavljamo niske vrijednosti temperature čime većinski dopuštamo prelaze sa negativnim nagibom i one sa blagim pozitivnim nagibom. Za svaki prelaz sa pozitivnim nagibom, temperatura se smanjuje `alpha` puta čime se smanjuje vjerovatnoća prelaska. U određenom momentu ćemo dosegnuti maksimalan broj uzastopnih neuspješnih prelaza i temperatura će se adaptirati (množenjem sa `alpha`) tako da se poveća vjerovatnoća prelaska. Povećanjem vjerovatnoće prelaska omogućavamo da istražimo i regije sa višim težinama tako da ne budmo zarobljeni u lokalnim minimumima. Primjer bi bio "prevoj" na mapi koji povezuje dvije niskotežinske regije. Stablo će težiti ka širenju u niskotežinskim regijama tako da sa samoadaptacijom temperature dajemo šansu i da se popnemo uz prevoj i povežemo niskotežinske regije.

Ideja iza ove funkcije i samog T-RRT algoritma se bazira na kriteriju *puta minimalnog rada* (eng. *minimal work path*) (MW). Ovo je kriterij koji tjera stablo da se širi kroz niskotežinske regije prostora. Zamisao je da pozitivni nagibi grana djeluju kao sile koje djeluju u suprotnom smjeru prirodnog kretanja koje je nizbrdo i time se generiše *mehanički rad* da se prevaziđu pozitivni nagibi što je i koncept iza kinetičke i potencijalne energije u fizici. U slučaju negativnih nagiba ne generišemo mehanički rad jer se krećemo u prirodnom smjeru i definišemo da sistem ne gubi energiju. Pojednostavljeni rečeno, cilj nam je dobiti put sa što manjim ukupnim mehaničkim radom [1]. Jednačina 4.1 za vjerovatnoću prelaza dosta podsjeća na formulu za Boltzmann-ovu raspodjelu i može se pokazati da će upravo putevi sa najmanjim mehaničkim radom imati najvišu vjerovatnoću dostizanja cilja što je i prirodno jer stanja sistema u stvarnom svijetu teže ka stanju minimalne energije. Dokaz ovog principa i detaljan opis MW koncepta se može naći u [1].

Sve rečeno ćemo najbolje prikazati na primjeru na slici 18 korištenjem ovog algoritma na težinskoj mapi $\mathcal{C}_c = \{(q_1, q_2, q_3) \in \mathbb{R}^3 : -20 < q_1 < 20, -20 < q_2 < 20, q_3 = f_c(q_1, q_2)\}$ definisanoj sljedećom težinskom funkcijom:

$$\begin{aligned} f_c(q_1, q_2) = & 0.03 \cdot \sin(q_1) + 0.02 \cdot \cos(q_2) + \exp(-0.05 \cdot ((q_1 + 6)^2 + 0.5 \cdot (q_1 - q_2)^2)) \\ & + 0.4 \cdot \exp(-0.05 \cdot ((q_1 - 12)^2 + 0.5 \cdot (q_1 - q_2)^2)) + \exp(-0.05 \cdot ((q_1 + 20)^2 + 0.05 \cdot (q_1 - q_2 - 1)^2)) \\ & + \exp(-0.05 \cdot ((q_1 - 20)^2 + 0.1 \cdot (q_1 - q_2 - 20)^2)) + \exp(-0.05 \cdot ((q_1 + 12)^2 + 0.1 \cdot (q_1 + q_2)^2)) \\ & + \exp(-0.08 \cdot ((q_1 - 5)^2 + 0.1 \cdot ((q_1 + q_2 + 2)^2))) \end{aligned}$$



Slika 18: Primjena T-RRT algoritma (bez `minExpandControl`) na težinskoj mapi u 2D konfiguracijskom prostoru. Parametri T-RRT algoritma: `maxIter` = 10000, `deltaQ` = 1, `minDistance` = 2, `eta` = 0.50, `mi` = [0.10, 0.10], $T = 10^{-6}$, $K = 0.26$, `alpha` = 1.25, `c_max` = 0.38, `maxFails` = 15, $L = 118.3633$.

4.2 Implementacija funkcije `minExpandControl`

Dodatna funkcija koja je uvedena u T-RRT algoritmu je `minExpandControl`. Razlog zbog kojeg uvodimo ovu funkciju je problem kada prilikom adaptacije temperature, iako ona opada, čime opada i vjerovatnoća prelaza, dobivamo nove čvorove ali ne istražujemo nove regije prostora, već samo usitnjavamo (eng. *refine*) niskotežinske regije koje smo već istražili. Ovim gušimo algoritam na brdovitim težinskim mapama usporavanjem širenja stabla ka neistraženim regijama. Iako ovu pojavu nismo eksplicitno vidjeli na primjeru sa slike 18 zbog korištenja BiRRT-a na maloj težinskoj mapi, za slučaj korištenja RRT-a ili slučaj većih težinskih mapa sa brdovitim reljefom, ova pojava dosta dobiva na značaju.

Na ovaj problem možemo gledati na način koju vrstu čvorova dodajemo u stablo. Imamo čvorove koje usitnjavaju već istraženi prostor (eng. *refiner nodes*) i čvorove na periferiji stabla koji su odgovorni za širenje stabla (eng. *frontier nodes*). Analogija bi bila drvo sa glavnim granama koje su odgovorne za širenje stabla i te grane imaju grančice koje usitnjavaju prostor kojeg su već dosegnule glavne grane. Pogledajmo sada pseudokod za `minExpandControl` funkciju koja ublažava ovaj problem.

```

Inicijaliziraj: rho, refinerNodes
function minExpandControl( $q_i, q_j$ )
    1 if  $distance(q_i.coordinates, q_j.coordinates) > T.deltaQ$  then
    2     return true;
    3 else
    4     if  $(T.refinerNodes+1)/(T.getNodeNum()+1) > T.rho$  then
    5         return false;
    6     else
    7          $T.refinerNodes = T.refinerNodes + 1;$ 
    8         return true;
    9     end
    10 end
end

```

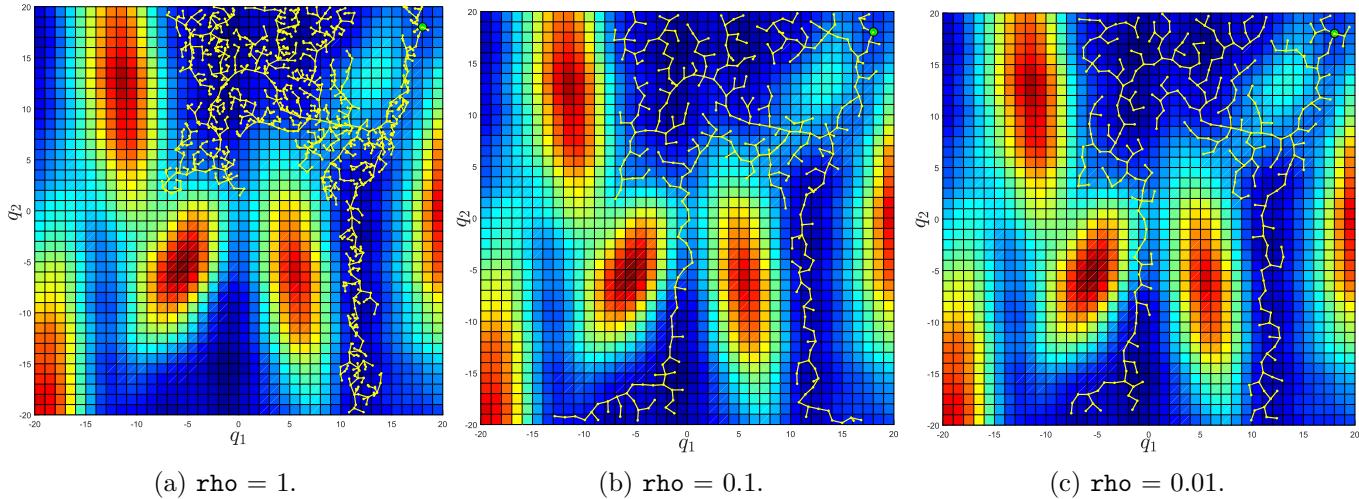
Pseudokod 4: Funkcija `minExpandControl`

Dva parametra koja konfigurišu u ovoj funkciji su `rho` i `refinerNodes` koji će biti atributi TRRT klase. Ako pogledamo pseudokod sa početka ovog poglavlja o izmjeni uslova za dodavanje novih čvorova u stablo, vidimo da funkcija `minExpandControl` prima čvorove \mathbf{q}_{near} i \mathbf{q}_{rand} . U glavnoj `if` naredbi poredimo da li je udaljenost čvorova \mathbf{q}_{near} i \mathbf{q}_{rand} veća od udaljenosti širenja `deltaQ` jer za velike udaljenosti između ova dva čvora, čvor \mathbf{q}_{near} ima veću šansu da bude periferalni čvor što implicira da i \mathbf{q}_{new} ima veću da bude periferalni čvor. U suprotnom slučaju, šanse su da će čvor \mathbf{q}_{new} učestvovati u usitnjavanju prostora.

Drugi dio ove funkcije “tjera” algoritam da istražuje novi prostor upravljanjem odnosa čvorova usitnjavanja i svih čvorova. Ideja je da moramo imati i određen broj čvorova za usitnjavanje ali da držimo taj broj prema broju svih čvorova ispod nekog praga. Taj prag definiramo atributom `rho` dok broj čvorova koji učestvuju u ustinjaivanju čuva atribut `refinerNodes`, a funkcija `getNodeNum` daje trenutni broj čvorova u stablu.

Ukoliko je udaljenost između \mathbf{q}_{near} i \mathbf{q}_{rand} manja od `deltaQ`, imamo potencijalni čvor usitnjavanja, što je uredu jer nam trebaju i takvi čvorovi. Ukoliko je to slučaj, ulazimo u drugu `if` naredbu i provjeravamo da li je odnos broja čvorova usitnjavanja naprema ukupnom broju čvorova ispod praga `rho`. Ako smo prešli prag šaljemo log. 0 čime spriječavamo daljnje usitnjavanje. Ako smo ispod praga, onda imamo mjesta za dodatno usitnjavanje i inkrementiramo broj čvorova usitnjavanja za jedan i šaljemo log. 1 čime usitnjavamo prostor kojeg istažuje stablo.

Iz ovog vidimo da je ključni parametar za kontrolisanje usitnjavanja prostora prag `rho`. Zaključujemo da za veći prag imamo veće usitnjavanje, dok za manji prag imamo manje usitnjavanje. U zavisnosti od terena težinske mape podešavamo ovaj parametar. Primjetimo da ako postavimo `rho = 1`, funkcija `minExpandControl` je neaktivna jer odnos čvorova može najviše biti 1 tako da će funkcija uvijek vraćati log. 1. Pogledajmo sada na slikama 19, širenje stabla u zavisnosti od praga `rho`.

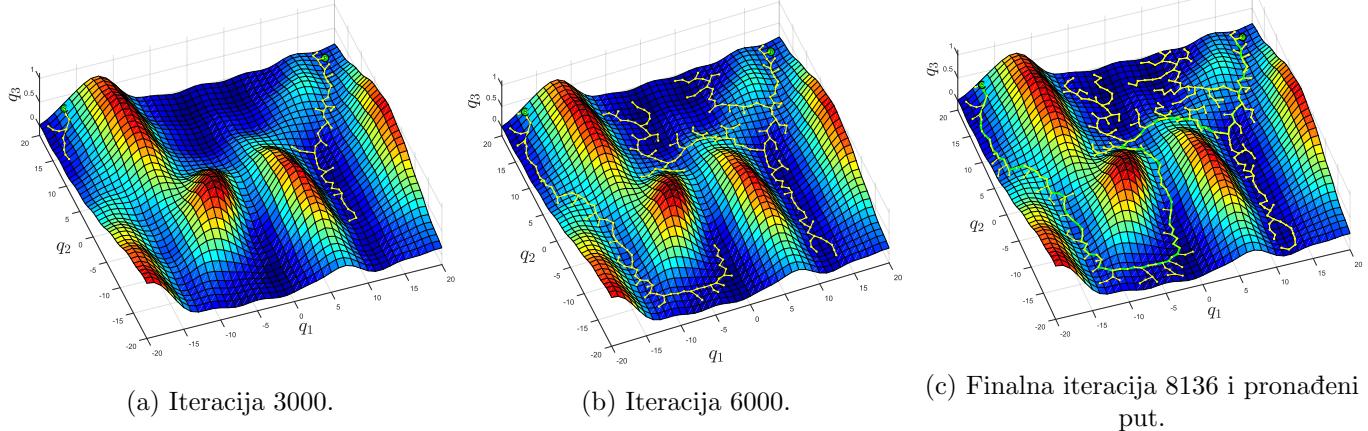


Slika 19: Uticaj `minExpandControl` funkcije unutar T-RRT algoritma nakon 10000 iteracija.

Vidimo na slici 19a da se stablo ne može proširiti u donju regiju preko prevoja čime se regija u kojoj se nalazi stablo počinje usitnjavati što efektivno usporava algoritam i novi prostor ostaje neistražen ili sporo istražen. Na slikama 19b i 19c vidimo uticaj `minExpandControl` za vrijednosti praga 0.1 i 0.01. Ograničili smo usitnjavanje prostora čime efektivno tjeramo algoritam da pretražuje

nove regije prostora i time prelazimo prevoj i istražujemo nove regije. Testiranjem raznih vrijednosti za prag ρ , najbolje performanse pretraživanja prostora daju vrijednosti između 0.01 i 0.1.

Pogledajmo na slici 20, sada potpuni, T-RRT algoritam na težinskoj mapi koju smo koristili na slici 18.



Slika 20: Primjena T-RRT algoritma na težinskoj mapi u 2D konfiguracijskom prostoru. Parametri T-RRT algoritma: $\text{maxIter} = 10000$, $\text{deltaQ} = 1$, $\text{minDistance} = 2$, $\text{eta} = 0.50$, $\text{mi} = [0.10, 0.10]$, $T = 10^{-6}$, $K = 0.26$, $\text{alpha} = 1.25$, $c_{\text{max}} = 0.38$, $\text{maxFails} = 15$, $\rho = 0.05$, $L = 110.1681$.

Sa ovim primjerom vidimo da je T-RRT algoritam zaista pogodan za traženje puteva u problemima za koje moramo naći putanju u zavisnosti od nekog kriterija definisanog težinskom funkcijom. Sa ovim završamo priču oko RRT tipova algoritama i nastavljamo sa njihovom primjenom na sisteme robotskih manipulatora.

5 Robotski manipulatori i njihov matematski model

U ovom poglavlju ćemo pokazati korištenje RRT baziranih algoritama na primjeru planiranja kretanja robotskih manipulatora u prostoru sa preprekama. Sada ćemo dati kratki uvod bitnih pojmove za robotske manipulatore.

Robotski manipulator je mehanička struktura sačinjena od niza *segmenata* (eng. *segments, links*) koji su spojeni *zglobovima* (eng. *joints*). Kretanje manipulatora kao strukture se postiže integracijom pojediničnih kretanja segmenata u odnosu na one prethodne. Ovo se postiže transformacijom koordinata prethodnog segmenta da bi se dobole koordinate trenutnog segmenta i time njegova pozicija u odnosu na prethodni segment. Ovako možemo ići do baznog zgloba i dobiti koordinate vrha manipulatora u odnosu na bazu manipulatora. Ovo je metoda homogenih transformacija koje se sastoje iz uzastopnih transformacija koordinata koje mogu biti rotacija i/ili translacija u prostoru. Ovim se dobiva matrična jednačina koja opisuje poziciju i orijentaciju trenutnog segmenta.

Kod manipulatora možemo samo upravljati zglobovima tako da su oni varijable sistema tj. pozicija i orijentacija segmenata i vrha manipulatora zavise od trenutne konfiguracije zglobova manipulatora. Ovo je upravo definicija *direktne kinematike* koja ima zadatku da odredi poziciju i orijentaciju vrha manipulatora u zavisnosti od varijabli zglobova. Odavde slijedi da su zglobovi glavni elementi kojim upravljamo da bismo doveli vrh manipulatora tamo gdje želimo. Zglobovi koji spajaju dva susjedna segmenta manipulatora mogu biti *rotacioni* (eng. *revolute*) ili *translacijski* (eng. *translational, prismatic*). Sa rotacionim zglobovima imamo mogućnost da rotiramo željeni segment oko ose zgloba dok sa translacionim zglobovima imamo mogućnost da “uvučemo” ili “izvučemo” segment. U ovom radu ćemo se baviti samo sa rotacionim zglobovima. Varijable zglobova su uglovi za koje zakrećemo zglove da bi pomjerali cijelu strukturu manipulatora. Dakle, trenutni položaj robota nam daju uglovi zglobova odnosno konfiguracija \mathbf{q}_i uglova zglobova manipulatora. Slijedi da je i -ta konfiguracija robota data kao:

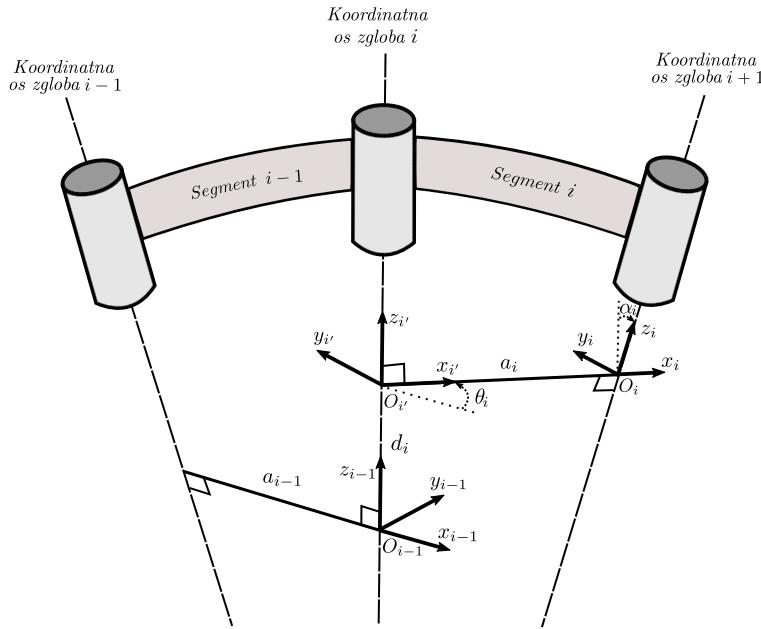
$$\mathbf{q}_i = [\theta_1 \ \dots \ \theta_n]^T \quad (5.1)$$

gdje smo sa θ označili uglove zakreta zglobova. Radi konzistentnosti sa ranijom notacijom konfiguracija, ostatak ćemo pri oznakama iz jednačine 2.2 za uglove. Smatrat ćemo da je ugao rotacije zgloba ograničen na intervalu $[-\pi, \pi]$ čime spriječavamo da segment može napraviti punu rotaciju. Broj zglobova manipulatora odnosno broj zglobovskih varijabli nam daje broj *stepeni slobode* (eng. *degrees of freedom*) manipulatora. Svaki novi zglob nam daje dodatni stepen slobode.

Detaljniji opis svega spomenutog kao i matrične jednačine koje opisuju model manipulatora se može naći u [6].

5.1 DH parametri

DH (Denavit-Hartenberg) parametri su veličine dobivene iz istoimenog postupka kojim generaliziramo dobivanje matrice homogene transformacije \mathbf{T} za bilo koji manipulator. Postupak se bazira na činjenici da svaki zglob povezuje najviše dva segmenta, tako da opisom veze dva susjedna segmenta možemo rekurzivnim putem definisati veze svih segmenata manipulatora. Na slici 21 su prikazani DH parametri za generalni slučaj dva segmenta.



Slika 21: Prikaz generalnog slučaja DH parametara za dva segmenta.

DH parametri su definisani kao:

- a_i – udaljenost između O_i i $O_{i'}$,
- d_i – udaljenost između $O_{i'}$ i O_{i-1} ,
- α_i – ugao rotacije oko osi x_i mjerjen od osi z_{i-1} prema osi z_i ,
- θ_i – ugao rotacije oko osi z_{i-1} mjerjen od osi x_{i-1} prema osi x_i .

Cilj je da dobijemo vezu između koordinatnih sistema $O_{i-1} - x_{i-1}y_{i-1}z_{i-1}$ i $O_i - x_iy_iz_i$ dva susjedna zgloba preko pomoćnog koordinatnog sistema $O_{i'} - x_{i'}y_{i'}z_{i'}$.

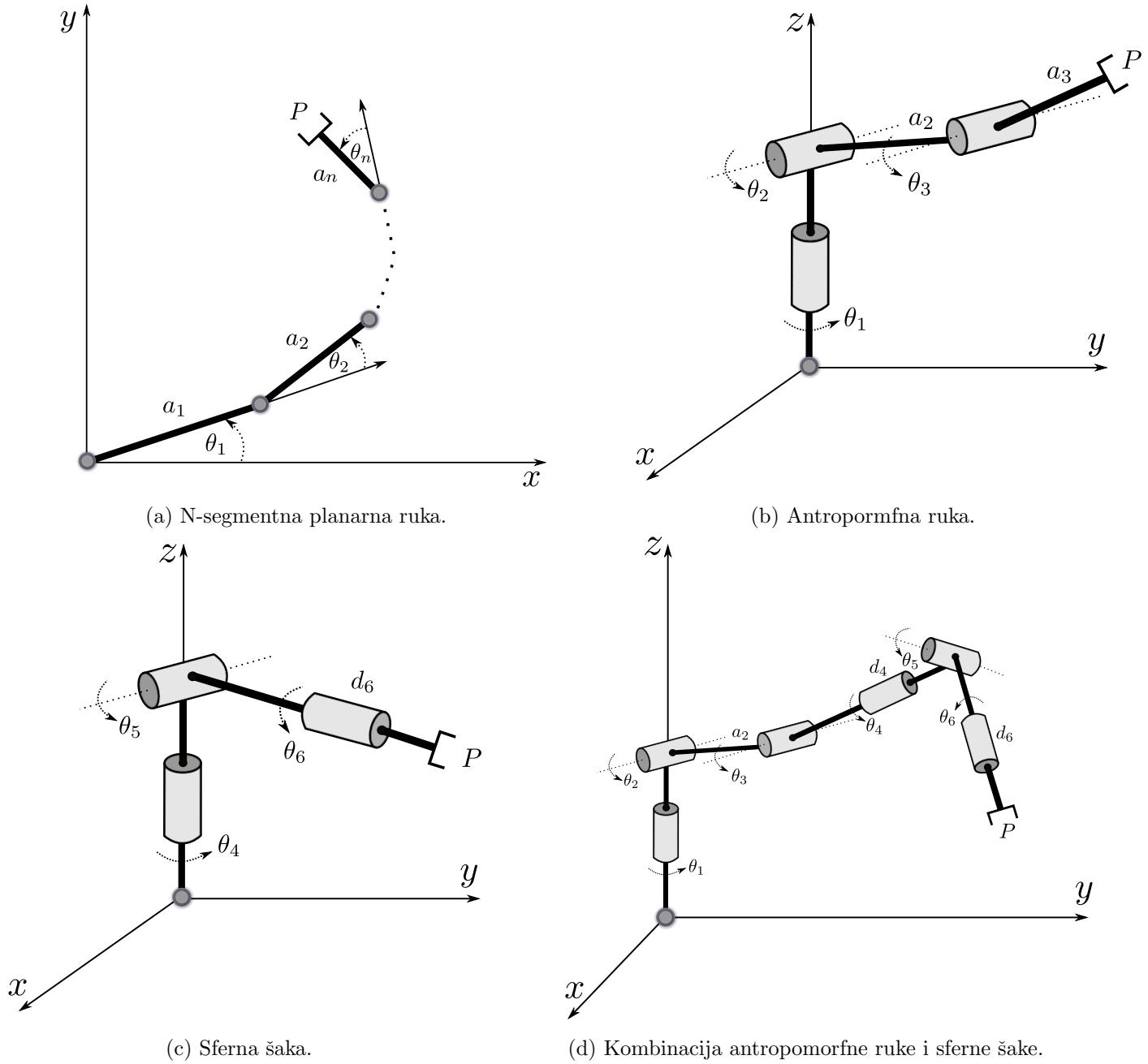
Moramo koordinatni sistem $O_{i-1} - x_{i-1}y_{i-1}z_{i-1}$ translirati za d_i duž z_{i-1} osi i zatim ga zarotirati za ugao θ_i oko $z_{i'}$ osi čime se podudaraju koordinatni sistemi $O_{i-1} - x_{i-1}y_{i-1}z_{i-1}$ i $O_{i'} - x_{i'}y_{i'}z_{i'}$. Drugi korak je translirati koordinatni sistem $O_{i'} - x_{i'}y_{i'}z_{i'}$ za a_i duž $x_{i'}$ osi i zatim ga zarotirati za ugao α_i oko osi x_i čime se podudaraju koordinatni sistemi $O_{i'} - x_{i'}y_{i'}z_{i'}$ i $O_i - x_iy_iz_i$ što znači da imamo direktnu vezu ciljanih koordinatnih sistema. Sve opisano se može izraziti sljedećom matricom homogene transformacije:

$$\mathbf{A}_i^{i-1}(q_i) = \begin{bmatrix} c_{\theta_i} & -s_{\theta_i}c_{\alpha_i} & s_{\theta_i}s_{\alpha_i} & a_i c_{\theta_i} \\ s_{\theta_i} & c_{\theta_i}c_{\alpha_i} & -c_{\theta_i}s_{\alpha_i} & a_i s_{\theta_i} \\ 0 & s_{\alpha_i} & c_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5.2)$$

gdje je $q_i = \theta_i$ za rotacione zglove, a $q_i = d_i$ za translacione zglove. Poznavanjem DH parametara možemo konstruisati matricu homogene transformacije i dobiti opis pozicije i orientacije segmenata. Kako radimo samo sa rotacionim zglovima onda su varijable sistema samo uglovi θ_i dok su ostali parametri konstante i zavise od geometrije manipulatora. U slučaju translacionog zglova varijabla sistema bi bila d_i . U nastavku ćemo pretstaviti neke robotske manipulatore i njihove DH parametre.

5.2 Tipične strukture robotskih manipulatora

U ovom radu ćemo raditi sa tri vrste manipulatora i jednom njihovom kombinacijom. Već smo kazali da će manipulatori imati samo rotacione zglobove. Bez gubljenja na opštosti smatrat ćemo da je baza manipulatora postavljena u koordinatni početak. Na slici 22 su predstavljene šeme spomenutih manipulatora (vrh manipulatora označen za P).



Slika 22: Razni robotski manipulatori.

(a) **N-segmentna planarna ruka** (eng. *N-segment planar arm*)

Ovaj manipulator je ograničen na 2D prostor. Sastoji se od n segmenata gdje i -ti segment ima dužinu a_i i ugao zakreta i -og zglobo u odnosu na prethodni je θ_i . Ovaj manipulator ima n stepeni slobode. U nastavku ćemo koristiti primjere za dvosegmentnu i trosegmentnu planarnu ruku. Na slici 22a je prikazana šema N-segmentne planarne ruke dok su DH parametri prikazani u tabeli 1.

Tabela 1: DH parametri N-segmentne planarne ruke.

<i>Segment</i>	a_i	α_i	d_i	θ_i
1	a_1	0	0	θ_1
:	:	:	:	:
n	a_n	0	0	θ_n

(b) **Antropomorfna ruka** (eng. *Anthropomorphic arm*)

Ovaj manipulator operira u 3D prostoru. Ima tri zgloba što mu daje tri stepena slobode. Ima dva segmenta dužine a_2 i a_3 uz mogućnost rotacije baze oko neke od osa. Mogao bi se dodati i početni segment koji bi samo podigao manipulator iz ravni ali bi funkcionalnost ostala u suštini ista. Naziv dobiva zbog slične funkcionalnosti sa ljudskom rukom. Na slici 22b su prikazani mogući pokreti manipulatora dok su DH parametri prikazani u tabeli 2.

Tabela 2: DH parametri antropomorfne ruke.

<i>Segment</i>	a_i	α_i	d_i	θ_i
1	0	$\pi/2$	0	θ_1
2	a_2	0	0	θ_2
3	a_3	0	0	θ_3

(c) **Sferna šaka** (eng. *Spherical wrist*)

Ovaj manipulator također operira u 3D prostoru. Ima tri zgloba što mu daje tri stepena slobode. Naziv sferna šaka dolazi od toga što se struktura manipulatora kreće po sfernoj ljusci. Primjetimo da prva dva zgloba imaju istu funkcionalnost kao oni u antropomorfnoj ruci. Imamo samo jedan segment sa dužinom d_6 i to je segment koji opisuje sferu. Prva dva segmenta se mogu dodati ali opet će samo traslirati vrh manipulatora u prostoru tako da u suštini struktura ostaje ista ako bi bazu pomjerili u mjesto gdje smo translirali manipulator.

Zadnji zglob rotira na način da se struktura efektivno ne pomjera već imamo samo pojavu "zavrtanja" segmenta. Primjer korištenja ovog bi bio zavrtanje šarafa ili bilo čega drugog što treba zavrnuti ili odvrnuti. Primjetimo da su oznake za uglove zglobova i segmenta numerisane od broja 4. Razlog je što se najčešće sferna šaka montira na neki manipulator koji predstavlja ruku. Na slici 22c su prikazani mogući pokreti manipulatora dok su DH parametri prikazani u tabeli 3.

Tabela 3: DH parametri sferne šake.

Segment	a_i	α_i	d_i	θ_i
4	0	$-\pi/2$	0	θ_4
5	0	$\pi/2$	0	θ_5
6	0	0	d_6	θ_6

(d) **Kombinacija antropomorfne ruke i sferne šake**

Ovaj manipulator predstavlja upravo spomenuto montiranje sferne šake na neki manipulator koji predstavlja ruku gdje je taj manipulator antropomorfna ruka. Ovaj manipulator ima šest zglobova što mu daje šest stepeni slobode čime možemo u potpunosti definiati i poziciju i orijentaciju vrha manipulatora. Primjetimo da smo na mjesto spajanja dva manipulatora uveli segment dužine d_4 za sfernu šaku, a odstranili segment dužine a_3 za antropomorfnu ruku. Na slici 22d su prikazani mogući pokreti manipulatora dok su DH parametri prikazani u tabeli 4.

Tabela 4: DH parametri kombinacije antropomorfne ruke i sferne šake.

Segment	a_i	α_i	d_i	θ_i
1	0	$\pi/2$	0	θ_1
2	a_2	0	0	θ_2
3	0	$\pi/2$	0	θ_3
4	0	$-\pi/2$	d_4	θ_4
5	0	$\pi/2$	0	θ_5
6	0	0	d_6	θ_6

5.3 Radni i konfiguracijski prostor robotskih manipulatora

Na početku ovog rada smo kratko govorili o radnom prostoru dok smo o konfiguracijskom prostoru govorili naširoko kroz cijeli rad. Sada ćemo vidjeti kakva je veza ova dva prostora u smislu robotskih manipulatora.

Označimo oznakom \mathcal{B} robotski manipulator koji se sastoji od niza segmenata povezanih zglobovima. Prostor u kojem se nalazi i u kojem operira manipulator \mathcal{B} se naziva *radni prostor* (eng. *workspace*) kojeg ćemo označiti sa \mathcal{W} . Radni prostor manipulatora može biti 2D ili 3D odnosno $\mathcal{W} \in \mathbb{R}^n$, gdje $n = 2$ ili $n = 3$. Unutar radnog prostora se nalaze prepreke koje zauzimaju prostor \mathcal{W}_{ob} dok prostor kojeg ne zauzimaju prepreke je slobodni prostor \mathcal{W}_{free} odakle slijedi da je:

$$\mathcal{W} = \mathcal{W}_{free} \cup \mathcal{W}_{ob} \quad (5.3)$$

Ukoliko imamo p prepreka koje nisu fizičke spojene onda slijedi da je ukupni prostor kojeg zauzima tih p prepreka:

$$\mathcal{W}_{ob} = \bigcup_{i=1}^p \mathcal{W}_{ob_i} \quad (5.4)$$

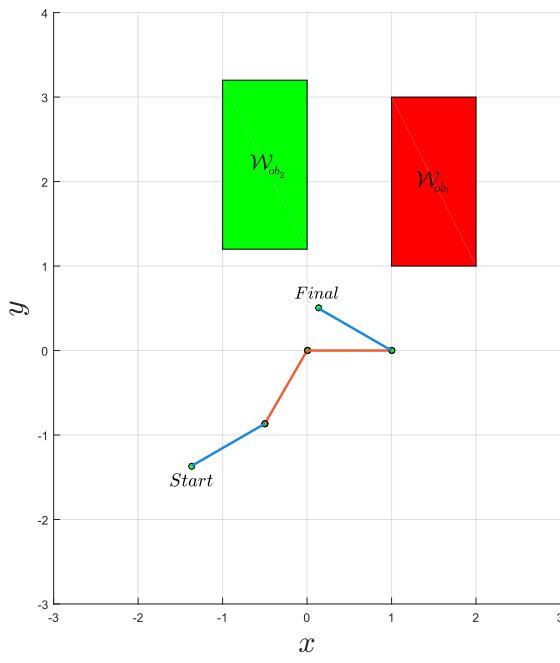
Već smo spomenuli da zglobovske varijable \mathbf{q} određuju položaj manipulatora tj. položaj svih segmenata $\mathcal{B}(\mathbf{q}) \in \mathcal{W}_{free}$. Zadatak koji se postavlja prilikom planiranja putanje je ukoliko imamo početni položaj manipulaotra $\mathcal{B}(\mathbf{q}_s) \in \mathcal{W}_{free}$ i ukoliko imamo željeni krajnji položaj manipulatora $\mathcal{B}(\mathbf{q}_f) \in \mathcal{W}_{free}$ da li postoji put \mathcal{P}_W od početne do krajnje konfiguracije manipulatora takav da ne sječe prostor prepreka \mathcal{W}_{ob} tj. mora vrijediti $\mathcal{P}_W \in \mathcal{W}_{free}$. Napomenimo još jednom da su spomenute konfiguracije manipulatora ustvari uglovi zakreta zglobova manipulatora iz jednačine 5.1. Način kojim upravljamo manipulatorom jeste preko zakreta zglobova tako da je prirodan način rješavanja ovog problema iz radnog prostora, prelazak u konfiguracijski prostor \mathcal{C} , pronalazak sekvene konfiguracija (uglova zakreta zglobova) i slanje tih konfiguracija kao naredbi zglobovima manipulatora koji se kreće u radnom prostoru.

Konfiguracijski prostor \mathcal{C} manipulatora je skup svih konfiguracija koje manipulator \mathcal{B} može poprimiti. Konfiguracijski prostor će u slučaju rotacionih zglobova generalno biti definisan kao:

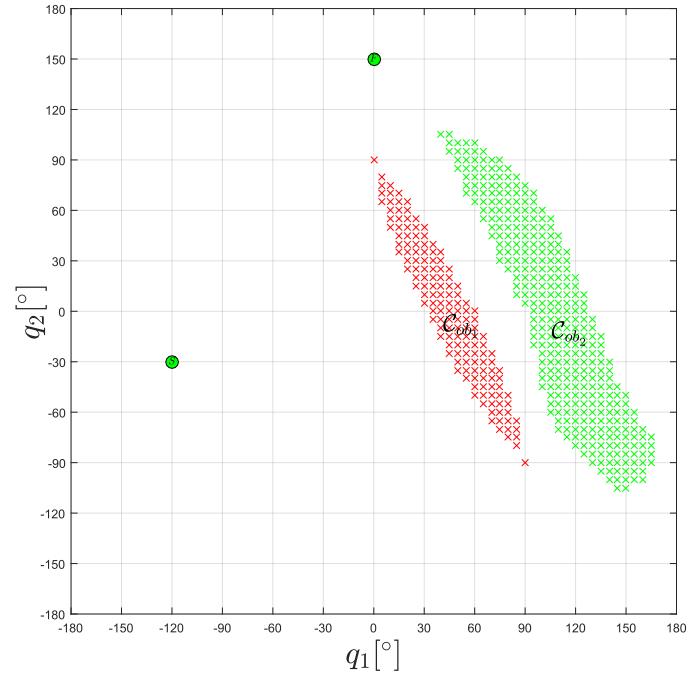
$$\mathcal{C} \in \{\mathbf{q} \in \mathbb{R}^n, -\pi < q_1 < \pi, \dots, -\pi < q_n < \pi\} \quad (5.5)$$

gdje koristimo pretpostavku da se svaki zglob rotira u intervalu $[-\pi, \pi]$ mada se i to može podešavati po volji. Glavno pojednostavljenje koje dobivamo predstavljanjem manipulatora u konfiguracijskom prostoru je što cijelog robota možemo predstaviti kao jednu tačku \mathbf{q} dok je manipulator u radnom prostoru predstavljen kao niz segmenata i zglobova. Kako smo predstavili položaj manipulatora u prostoru \mathcal{W} kao jednu tačku u prostoru \mathcal{C} onda možemo koristiti punu moć RRT baziranih algoritama za rješavanje problema pronalaska putanje. Kako znamo direktnu kinematiku manipulatora \mathcal{B} onda za svaku pronađenu konfiguraciju \mathbf{q} puta $\mathcal{P}_C \in \mathcal{C}_{free}$ možemo dobiti položaj manipulatora u \mathcal{W}_{free} . Prelazak iz \mathcal{C} u \mathcal{W} je generalisano za bilo koji manipulator preko DH parametara iz direktne kinematike ali prelaz iz \mathcal{W} u \mathcal{C} se vrši inverznom kinematikom i nemamo generalni opis ovog prelaza. Dobra stvar je ta što nam uopšte neće biti potreban direktan prelaz iz \mathcal{W} u \mathcal{C} . Naime, mi tražimo put u konfiguracijskom prostoru \mathcal{C} i problem koji nastaje je kako dobiti pozicije prepreka $\mathcal{C}_{ob} \in \mathcal{C}$ na osnovu pozicija prepreka $\mathcal{W}_{ob} \in \mathcal{W}$. Ali, kako mi generišemo konfiguracije u \mathcal{C} onda možemo za svaku generisanu konfiguraciju $\mathbf{q} \in \mathcal{C}$ na osnovu direktne kinematike provjeriti da li vrijedi $\mathcal{B}(\mathbf{q}) \in \mathcal{W}_{free}$. Ovako, bez eksplicitnog poznавања inverzne kinematike, možemo dobiti skup konfiguracija koje čine pronađeni put.

Ako bismo željeli da generišemo cijeli prostor \mathcal{C} na osnovu \mathcal{W} onda moramo uz određenu diskretizaciju konfiguracija proći kroz sve moguće položaje u radnom prostoru i za svaku od njih provjeriti da li siječemo prepreku u radnom prostoru. Na slikama 23 je prikazan prelaz iz radnog u konfiguracijski prostor dvosegmentne planarne ruke.



(a) Radni prostor.



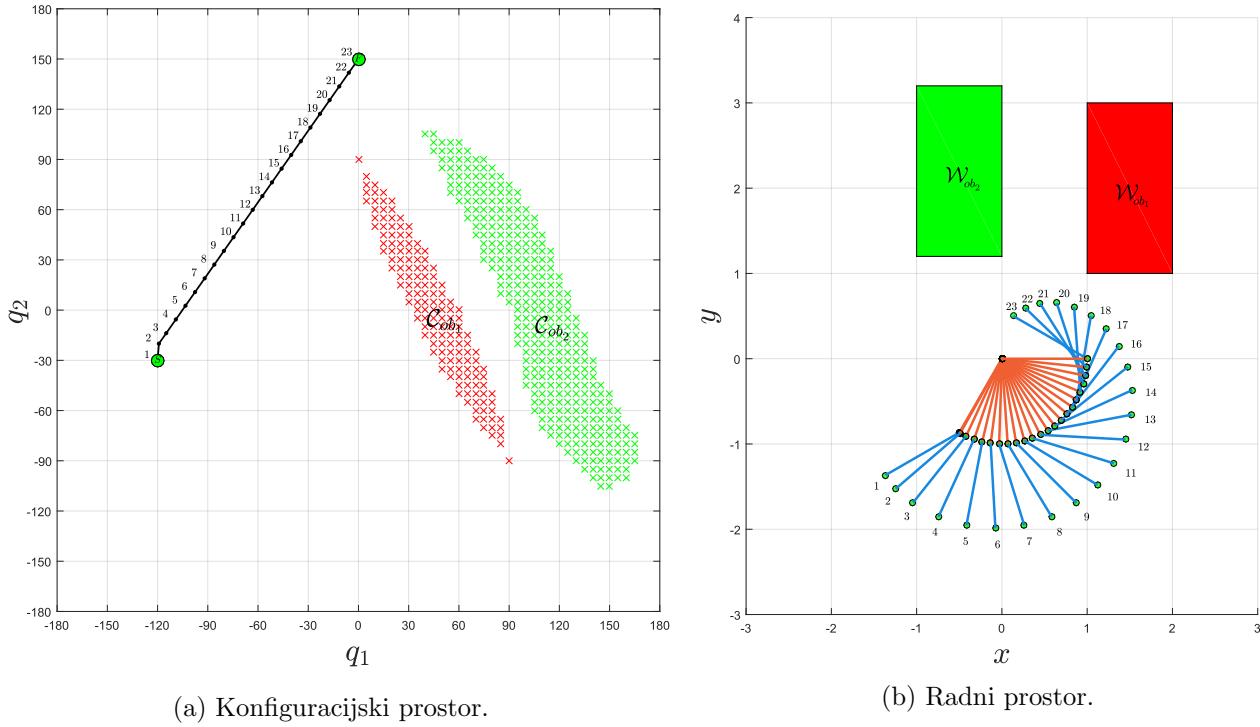
(b) Konfiguracijski prostor.

Slika 23: Početni i krajnji položaj dvosegmentne planarne ruke u radnom i konfiguracijskom prostoru.

Parametri manipulatora: $\mathbf{a} = [1 \ 1]^T$, $\mathbf{q}_s = [-30^\circ \ -120^\circ]^T$ i $\mathbf{q}_f = [0^\circ \ 150^\circ]^T$.

Da bismo dobili punu mapu konfiguracijskog prostora, moramo ispitati sve moguće položaje manipulatora u radnom prostoru tj. moramo pronaći $\mathcal{B}(\mathbf{q})$ za svako $\mathbf{q} \in \{(q_1, q_2) \in \mathbb{R}^2 : -\pi < q_1 < \pi, -\pi < q_2 < \pi\}$ i provjeriti da li je $\mathcal{B}(\mathbf{q}) \in \mathcal{W}_{free}$. Diskretiziranjem navedenog intervala dobivamo mapu konfiguracijskog prostora sa slike 23b. Ugao na grafovima u konfiguracijskom prostoru ćemo mjeriti u stepenima radi lakše interpretacije položaja manipulatora u radnom prostoru. Geometrija prepreka u konfiguracijskom prostoru zavisi od geometrije prepreka u radnom prostoru i geometrije manipulatora i često se dobivaju zanimljivi izgledi prepreka u konfiguracijskom prostoru. Za manipulator sa dva stepena slobode možemo lako konstruisati i prikazati konfiguracijski prostor (slika 23b). Za manipulator sa tri stepena slobode vremenski je dosta duže konstruisati konfiguracijski prostor jer moramo proći kroz tri intervala konfiguracija i zavisno od koraka diskretizacije generisanje može dugi potrajati. Za manipulator sa više stepeni slobode ne možemo ni prikazati konfiguracijski prostor tako da ćemo za takve manipulatori samo prikazivati radni prostor.

Primjetimo da ugao zakreta zglobo u startnom i finalnom položaju iz radnog prostora odgovaraju tačkama iz konfiguracijskog prostora. Slijedi da se korištenjem RRT baziranih algoritama u konfiguracijskom prostoru bilo kakav put u stablu manifestira kao kretanje u realnom svijetu radnog prostora manipulatora. Ovo je upravo prikazano na slici 24 za kretanje od početne do krajnje konfiguracije.



Slika 24: Korištenje BiRRT algoritma u konfiguracijskom prostoru se manifestira kao kretanje manipulatora u relanom svijetu tj. radnom prostoru.

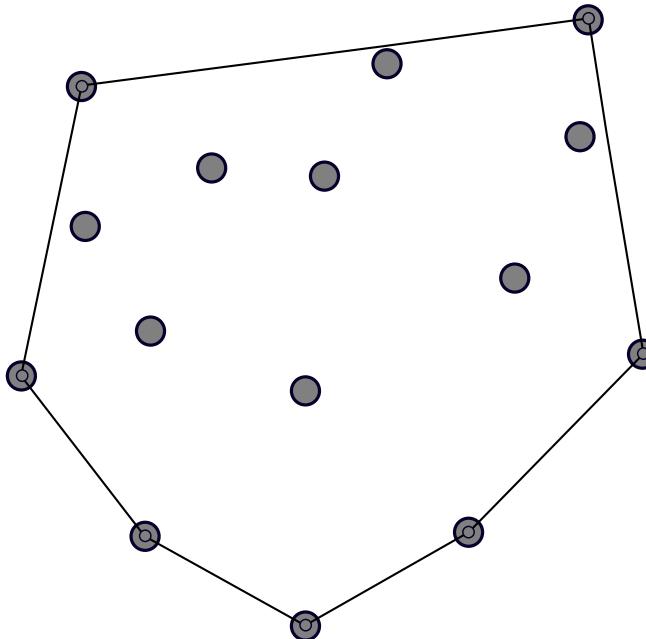
Primjetimo da na slici 23 korištenjem radnog prostora konstruišemo konfiguracijski prostor. Zatim na slici 24 korištenjem radnog prostora kontruišemo stablo i put koji se manifestira kao kretanje manipulatora u radnom prostoru. Svakom čvoru stabla odgovara jedan i samo jedan položaj manipulatora. U primjeru sa slikom 24 smo numerisali korespondentne čvorove sa položajima manipulatora. Već smo kazali da je inkrement za koji se širi stablo u konfiguracijskom prostoru definisan parametrom δQ . Interpretacija ovog parametra u radnom prostoru je maksimalni inkrement za koji se zglob rotira. Uzimanje $\delta Q = 5$ ima značenje da zglobovi manipulatora u radnom prostoru rotiraju segmente za maksimalno 5° jer se δQ rastavlja na komponente (kao vektor) u konfiguracijskom prostoru. Moramo obraćati pažnju na odabir ovog parametra jer za veće vrijednosti je moguće da između dva položaja manipulatora koja su u \mathcal{W}_{free} nakon interpolacije (radi motora zgloba) dobijemo položaj kojim bi zakačili ivicu neke prepreke.

6 Modeliranje prepreka u radnom i konfiguracijskom prostoru

U prethodnom poglavlju smo vidjeli da na osnovu prepreka \mathcal{W}_{ob} u radnom prostoru generišemo prepreke u \mathcal{C}_{ob} u konfiguracijskom prostoru poznavanjem direktnе kinematike manipulatora. Pseudokod za ovaj postupak ћemo pokazati u sljedećem poglavlju jer prepreke \mathcal{C}_{ob} slijede iz prepreka \mathcal{W}_{ob} tako da ћemo se sad bazirati na generisanje \mathcal{W}_{ob} .

6.1 Prepreka kao konveksni omotač

Modeliranje prepreka u radnom prostoru smo izvršili na bazi *konveksnih omotačа* (eng. *convex hull*) skupa tačaka u n -dimenzionalnom prostoru. Konveksni omotač skupa tačaka \mathcal{X}' je najmanji konveksni skup¹ \mathcal{X} koji je sadrži \mathcal{X}' . Slijedi da je $\mathcal{X} \subseteq \mathcal{X}'$. Na slici 25 je prikazan primjer konveksnog omotačа \mathcal{X} skupa tačaka \mathcal{X}' .



Slika 25: Konveksni omotač skupa tačaka u 2D prostoru.

Postoje brojni algoritmi [9] kojima možemo dobiti \mathcal{X} iz \mathcal{X}' i taj algoritam ћemo uzeti kao poznat:

$$\mathcal{X} \leftarrow \text{convhull}(\mathcal{X}')$$

tako da smatramo da znamo tačke koje čine konveksni omotač poredane u smjeru suprotnom od kazaljke na satu oko omotačа. Razlog zbog kojeg uzimamo kao model prepreka konveksni omotač je zbog jasnog opisa prepreke, zbog brojnih algoritama koji nam daju informaciju o tome da li se

¹Konveksan skup je podskup Euklidskog prostora koji je takav da između bilo koje dvije tačke možemo povući linijski segment koji ih povezuje.

proizvoljna tačka nalazi unutar ili van omotača i informacije o minimalnoj udaljenosti proizvoljne tačke od omotača. Ove informacije su ključne za provjeru presjeka segmenata sa preprekama i generisanja težinskih funkcija.

Prepostavimo da imamo m tačaka koje čine omotač u n -dimenzionalnom prostoru onda će omotač \mathcal{X} biti definisan kao:

$$\mathcal{X} = \begin{bmatrix} x_{11} & \dots & x_{1j} & \dots & x_{1n} \\ \vdots & \ddots & \vdots & & \vdots \\ x_{i1} & \dots & y_{ij} & \dots & x_{in} \\ \vdots & & \vdots & \ddots & \vdots \\ x_{m1} & \dots & x_{mj} & \dots & x_{mn} \end{bmatrix} \quad (6.1)$$

gdje je svaki red jedna tačka omotača, a svaka kolona predstavlja odgovarajuću koordinatu tačke. Ovo predstavlja samo jednu prepreku. Ako grupišemo p prepreka u jedan niz, onda dobivamo našu varijablu **obstacles**:

$$\text{obstacles} = [\mathcal{X}_1 \ \dots \ \mathcal{X}_p]^T \quad (6.2)$$

gdje ustvari vrijedi $\mathcal{X}_i = \mathcal{W}_{ob_i}$.

Dodatni stepen opštosti možemo uvesti generisanjem *rotacionih matrica* (eng. *rotation matrix*) kojima možemo rotirati dobiveni omotač za proizvoljni ugao oko proizvoljne tačke. Rotaciona matrica $\mathbf{R}_i(\theta)$ je matrica dimenzija $n \times n$ koja rotira proizvoljnu tačku oko i -te koordinatne ose za ugao θ . Uvedimo funkciju **rotateObstacle** koja prima kao parametar konveksni omotač, rotacionu matricu i tačku oko koje želimo da rotiramo omotač, a vraća rotirani omotač. Slijedi da je novi, rotirani, omotač dat kao:

$$\mathcal{X}_{new} \leftarrow \text{rotateObstacle}(\mathcal{X}, \text{R_i}, \text{pR})$$

gdje je **R_i** rotaciona matrica, a **pR** tačka oko koje rotiramo omotač. Ovo interno vršimo preko:

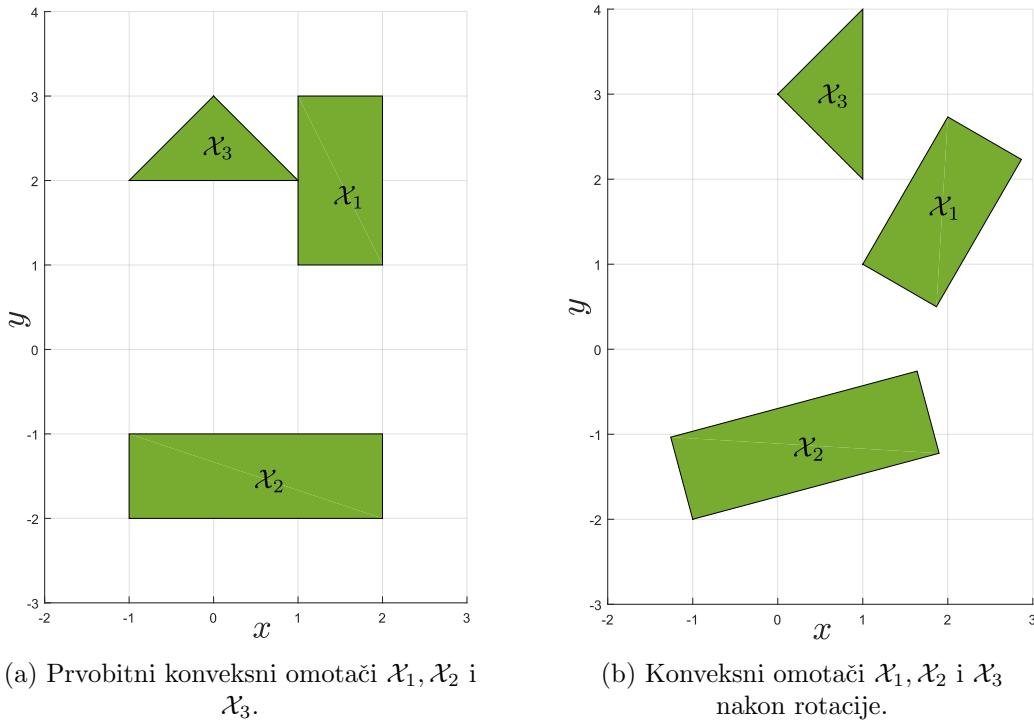
$$\mathcal{X}_{new} \leftarrow (\mathcal{X} - \text{P}) * \text{R_i} + \text{P}$$

gdje je $\text{P} = ([\text{pR} \ \dots \ \text{pR}]^T)_{m \times n}$.

Ukoliko želimo da rotiramo omotač oko neke od njegovih tačaka onda prema gornjoj jednačini moramo translirati omotač od te tačke u koordinatni početak, rotirati ga množenjem sa rotacionom matricom, i translirati ga nazad u prvobitnu poziciju. Nekad je povoljnije rotirati omotač oko koordinatnog početka dok je nekad povoljnije oko neke od njegovih tačaka. U budućnosti ćemo naglasiti tačku oko koje ćemo rotirati omotač.

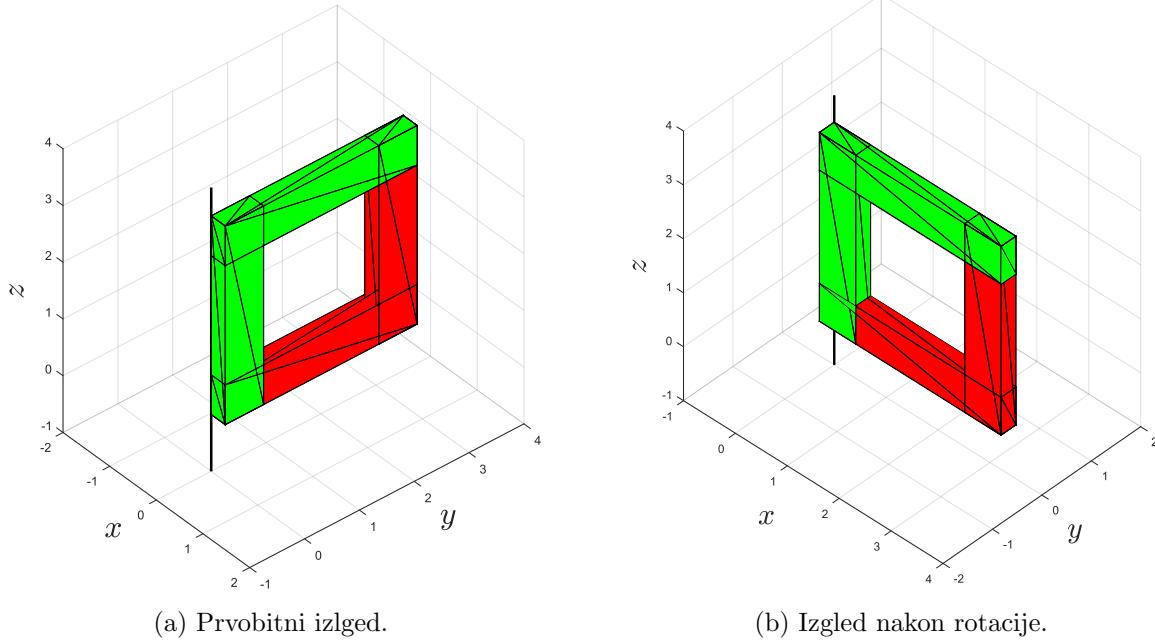
Pogledajmo primjer sa tri prepreke u 2D prostoru sa slike 26 čiji su omotači definisani sa:

$$\mathcal{X}_1 = \begin{bmatrix} 1 & 1 \\ 2 & 1 \\ 2 & 3 \\ 1 & 3 \end{bmatrix} \quad \mathcal{X}_2 = \begin{bmatrix} -1 & -2 \\ 2 & -2 \\ 2 & -1 \\ -1 & -1 \end{bmatrix} \quad \mathcal{X}_3 = \begin{bmatrix} 0 & 3 \\ -1 & 2 \\ 1 & 2 \end{bmatrix}$$



Slika 26: Prikaz konveksnih omotača u prvobitnom i rotiranom položaju. Uglovi rotacije odgovarajućih omotača: $\alpha_1 = 30^\circ$, $\alpha_2 = -15^\circ$ i $\alpha_3 = -90^\circ$ oko prve tačke omotača.

Iako smo se ograničili na konveksne prepreke, pažljivom kombinacijom konveksnih prepreka možemo gotovo napraviti bilo kakvu komplikovaniju prepreku. Na slici 27 je prikazan prozor u 3D prostoru koji je sačinjen od četiri konveksna omotača i rotacija tog prozora oko z ose.



Slika 27: Prikaz prozora u 3D prostoru, sačinjenog od četiri konveksna omotača, u prvobitnom položaju i nakon rotacije oko z ose za ugao $\alpha = 90^\circ$.

6.2 Testiranje sudara i dobivanje minimalne udaljenosti od prepreka

Način kojim provjeravamo da li linijski segment, koji može biti grana u grafu ili segment manipulatora, siječe prepreku je *brute force* taktikom. Prepostavimo da imamo algoritam naziva `inhull` koji prima matricu od k tačaka $\text{points} \in \mathbb{R}^{k \times n}$ (imamo k redova gdje svaki red sadrži jednu tačku u n -dimenzionalnom prostoru) i jedan konveksni omotač $\mathcal{X}_i \in \mathbb{R}^{m \times n}$ defnisan jednačinom 6.1, a vraća kao rezultat broj tačaka koje su unutar omotača \mathcal{X}_i :

$$s \leftarrow \text{inhull}(\text{points}, \mathcal{X}_i)$$

Slijedi da je $s \leq k$.

Neka je linijski segment u n -dimenzionalnom prostoru definisan sa početnom i krajnjom tačkom p_1 i p_2 . Da bismo provjerili da li linijski segment siječe omotač, dovoljno je podijeliti segment na k tačaka i sa funkcijom `inhull` provjeriti da li je $s > 0$, ako jeste onda segment siječe omotač.

Ovo je upravo opis rada funkcije `isSegmentInFreeSpace` koju smo koristili radi provjere sudara grane grafa i prepreka u konfiguracijskom prostoru.

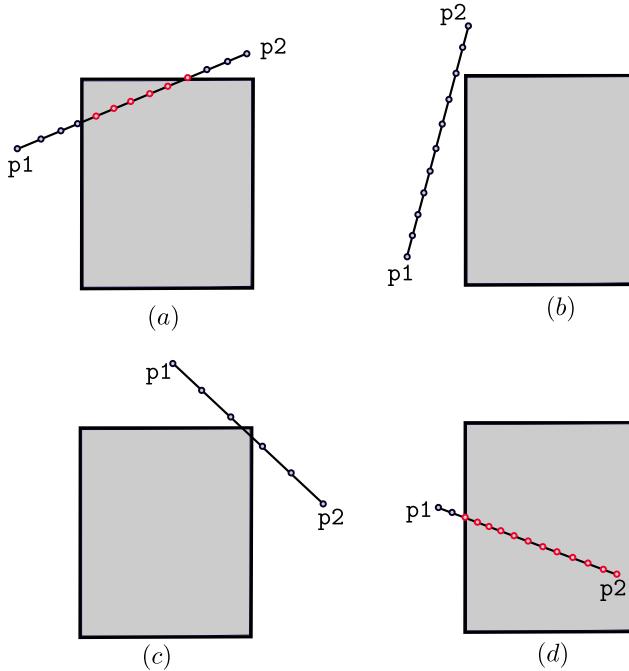
Neka varijabla `disect` sadrži broj k (na koliko tačaka dijelimo segment). Neka funkcija `linspace` prihvata početnu i krajnju tačku segmenta (p_1 i p_2) i broj podjela `disect`, a vraća matricu `points` gdje će prvi red biti prva tačka segmenta p_1 , zadnji red druga tačka segmenta p_2 , a $k-2$ ostalih tačaka će biti ravnomjerna podjela između p_1 i p_2 . Slijedi da je pseudokod funkcije `isSegmentInFreeSpace`:

```
Inicijaliziraj: disect
function isSegmentInFreeSpace(obstacles,  $p_1$ ,  $p_2$ )
    1 points  $\leftarrow$  linspace( $p_1$ ,  $p_2$ , disect);
    2 for  $i = 1$  to length(obstacles) do
        3 | if inhull(points, obstacles( $i$ , 1))  $> 0$  then
            4 | | return false;
            5 | end
        6 end
        7 return true;
end
```

Pseudokod 5: Funkcija `isSegmentInFreeSpace`

Nećemo ulaziti kako interno funkcija `inhull` provjerava da li je tačka unutar omotača ili ne ali se objašnjenje i kod ove funkcije implementirane u MATLAB-u se može naći u [10].

Vrijednosti broja podjela `disect` može jako ubrzati ili usporiti algoritam, pogotovo za velik broj poziva `isSegmentInFreeSpace` funkcije. Ako povećamo podjelu usporavamo algoritam, ali ako smanjimo podjelu možemo doći u situaciju da između dvije tačke podjele se nalazi prepreka što dovodi do pogrešnog izlaza, tako da u zavisnosti od izgleda radnog prostora i dužine segmenata biramo vrijednost varijable `disect`. Pogledajmo na slici 28 par primjera za moguće slučajeve testiranja sudara sa preprekama.



Slika 28: Tačke p_1 i p_2 su u slobodnom prostoru ali cijeli segment nije (a). Cijeli segment u slobodnom prostoru (b). Nedovoljna podjela dovodi do netačne interpretacije da je cijeli segment u slobodnom prostoru (c). Tačna interpretacija ali velik broj podjela segmenta što usporava algoritam (d).

Računanje minimalne udaljenosti proizvoljne tačke p do omotača \mathcal{X}_i je nimalo lagahan zadatak. Ova će nam informacija biti potrebna prilikom konstrukcije težinske funkcije koja će se bazirati na udaljenosti vrha manipulatora od najbliže prepreke jer ćemo sami vrh manipulatora tretirati kao tačku p . Ovaj problem se može uraditi za generalni slučaj (koji ćemo koristiti nadalje za 3D manipulator) i dat ćemo i objašnjenje kako bi se ovaj problem rješio za 2D slučaj (koji ćemo koristiti za planarne manipulatore jer je algoritam dosta brži nego kad bi se koristio onaj za generalni slučaj).

1. 2D slučaj

Posmatrajmo segment omotača između vrhova i i $i + 1$. Povucimo pravu l između ta dva vrha. Ako projekcija tačke p pada na segment prave l između vrhova i i $i + 1$ onda je to najmanja distanca od tačke p do tog segmenta. U suprotnom slučaju, najmanja distanca do segmenta je manja od udaljenosti između tačke p i vrhova i i $i + 1$. Onda je najmanja udaljenost tačke p do cijelog omotača, najmanja od udaljenosti do pojedinačnih segmenata. Funkcija koja ovo obavlja je

$$d_{\min} \leftarrow p_poly_dist(xp, yp, xv, yv)$$

gdje su xp i yp koordinate tačke p , a vektori xv i yv su koordinate tačaka koje čine omotač. Implementacija ove funkcije u MATLAB-u se može naći u [11].

2. Generalni slučaj

Konveksni omotač se može predstaviti kao konačan skup vrhova omotača što smo dosad i radili. Drugi način predstavljanja konveksnog omotača (ili u općem slučaju poligona) je kombinacija

linearnih uslova tipa jednakosti i nejednakosti oblika:

$$\mathbf{Ax} \leq \mathbf{b} \quad (6.3)$$

$$\mathbf{A}_{eq}\mathbf{x} = \mathbf{b}_{eq} \quad (6.4)$$

gdje je omotač presjek regija koje čine nejednakosti i jednakosti. Pošto imamo linearne jednakosti i nejednakosti onda one čine n -dimenzionalne ravni koje svojim presjekom definišu poliedar. Funkcija kojom možemo dobiti matrice koje čine ove uslove iz predstave omotača kao skupa vrhova je:

$$[\mathbf{A}, \mathbf{b}, \mathbf{A}_{eq}, \mathbf{b}_{eq}] \leftarrow \text{vert2lcon}(\mathcal{X}_i)$$

Detaljniji opis ovakve predstave konveksnog omotača sa primjerima i implementacija ove funkcije u MATLAB-u se može naći u [12].

Sada se može koristiti optimizaciona *metoda najmanjih kvadrata* sa uslovima jednakosti i nejednakosti, definisana funkcijom `lsqlin`, da se nađu koordinate tačke na omotaču koja ima najmanju udaljenost od proizvoljne tačke p:

$$\mathbf{pC} \leftarrow \text{lsqlin}(\mathbf{I}(n), \mathbf{p}, \mathbf{A}, \mathbf{b}, \mathbf{A}_{eq}, \mathbf{b}_{eq})$$

gdje je \mathbf{pC} tražena tačka na omotaču, a $\mathbf{I}(n)$ jedinična matrica dimenzije $n \times n$. Konačnu distancu nalazimo kao udaljenost tačaka \mathbf{p} i \mathbf{pC} tj. `distance(p1, pC)`. Demonstracija i objašnjenje `lsqlin` funkcije u MATLAB-u se može naći u [13].

Ovim smo definisali sve potrebne funkcionalnosti prepreka tako da možemo sada pokazati integraciju prethodnih algoritama sa modelom manipulatora i implementaciju bitnih funkcija za algoritme u ovom radu.

7 Integracija RRT baziranih algoritama sa modelom robotskih manipulatora

U ovom poglavlju ćemo ujediniti dosad obrađene algoritme sa modelom robotskog manipulatora. Dodatno ćemo prikazati implementaciju bitnijih funkcija i generisati težinsku funkciju kojom ćemo izbjegavati prepreke u radnom prostoru.

7.1 Implementacija modela robotskih manipulatora u MATLAB-u

Neka klasa `Robot` predstavlja robotski manipulator \mathcal{B} o kojem smo pričali s tim da će ova klasa imati puno više ulaznih parametara i funkcionalnosti od \mathcal{B} . Atributi ove klase su:

- Atribut `dof` čuva informaciju u broju stepeni slobode manipulatora (objekat tipa `int`).
- Atribut `dim` čuva informaciju o dimenziji prostora u kojem se robot nalazi (objekat tipa `int`).
- Atribut `a` čuva informaciju o a_i DH parametru (objekat tipa `double`).
- Atribut `d` čuva informaciju o d_i DH parametru (objekat tipa `double`).
- Atribut `initConfig` čuva koordinate početne konfiguracije \mathbf{q}_s manipulatora tj. koordinate prvog čvora stabla (objekat tipa `vector<double>`).
- Atribut `finalConfig` čuva koordinate krajnje konfiguracije \mathbf{q}_f manipulatora tj. koordinate zadnjeg čvora stabla (objekat tipa `vector<double>`).
- Atribut `currConfig` čuva koordinate trenutne konfiguracije \mathbf{q}_c manipulatora (objekat tipa `vector<double>`).
- Atribut `configVector` čuva pronađeni put, od početne do krajnje konfiguracije, za kretanje manipulatora u radnom prostoru (objekat tipa `vector<vector<double>>`).
- Atribut `obstacles` čuva informaciju o preprekama u radnom prostoru koje smo definisali u prethodnom poglavlju.
- Atribut \mathcal{T}_1 je prvo stablo BiRRT algoritma koje može biti tipa `RRT` ili `TRRT`.
- Atribut \mathcal{T}_2 je drugo stablo BiRRT algoritma koje može biti tipa `RRT` ili `TRRT`.
- Atribut `eta` je koeficijent kojim upravljamo širenjem stabala (objekat tipa `double`).
- Atribut `mi` je koeficijent kojim upravljamo vjerovatnoćom generisanja nasumičnih čvorova stabala (objekat tipa `vector<double>`).

Pošto radimo sa četiri vrste manipulatora onda radi boljeg pregleda naslijedit ćemo četiri klase iz klase `Robot` sa imenima manipulatora: `PlanarArm`, `AnthroArm`, `SphericalWrist` i `AnthroSphericalArmWrist`. Ove klase neće imati posebne atribute ili funkcije, već samo odgovarajuće konstruktore.

Što se tiče DH parametara, direktno smo uključili a_i i d_i parametre jer oni definišu dužinu segmenata koju ćemo mijenjati za pojedine manipulatore (dok je α_i fiksan za pojedine manipulatore). Ovi atributi su uključeni u funkciju `getDHparameters` koja će vraćati DH parametre, zavisno o manipulatoru s kojim radimo, u obliku matrice dimenzija $dof \times 4$ kao što smo prikazivali u tabelama DH parametara. Jedna od pogodnosti uvođenja naslijedenih klasa je što možemo u zavisnosti od imena manipulatora vratiti odgovarajuće DH parametre.

Iz definicije početne, trenutne i krajnje konfiguracije manipulatora zaključujemo da vrijedi veza prema čvorovima stabla: `initConfig = qs.coordinates`, `currConfig = qc.coordinates` i `finalConfig = qf.coordinates`. Kada budemo radili sa T-RRT algoritmom onda nećemo uključivati zadnju koordinatu čvorova stabla u navedene jednakosti jer ona sadrži težinu čvora koja fizički ne igra ulogu za varijable zglobova.

Atribut `configVector` je vektor konfiguracija \mathbf{Q} , o kojem smo već pričali da sadrži sve konfiguracije od početne do finalne koje prave put sa minimalnom težinom tj. $[\mathbf{Q}, \mathbf{L}] = \mathcal{T}.\text{getShortestPath}(\mathbf{q}_s, \mathbf{q}_f)$. Dimenzija ovog atributa će biti $dof \times r$ gdje je r broj čvorova koji čine pronađeni put.

Iz liste atributa vidimo da klasa `Robot` interno u sebi čuva prvo i drugo stablo BiRRT algoritma. Stabla moramo inicijalizirati unaprijed sa njihovim parametrima i takve poslati kao atribut `Robot` klase. To podrazumijeva inicijaliziranje parametara običnog RRT-a i ako koristimo T-RRT algoritam inicijalizacija parametara ovog algoritma. Parametar `obstacles` ne moramo inicijalizirati unutar stabala jer informaciju o sudari dobivamo iz radnog prostora manipulatora (zato klasa `Robot` ima atribut `obstacles`).

7.2 Implementacija bitnih funkcija

Sada ćemo predstaviti pseudokodove bitnijih funkcija koje smo koristili i koje ćemo koristiti.

1. Krenimo sa funkcijom, `randConfig`, za generisanje nasumičnih konfiguracija. Kako je ona metoda `RRT` klase onda podrazumijevamo da interno možemo pristupiti atributima stabla i ostalim metodama unutar klase.

Inicijaliziraj: boundary

function `randConfig()`

```
1 randCoor ← double.empty();
2 for  $i = 1$  to  $\text{length}(\mathcal{T}.\text{boundary}.begin)$  do
3 | randCoor( $1, i$ ) ← rand( $\mathcal{T}.\text{boundary}.begin(1, i)$ ,  $\mathcal{T}.\text{boundary}.end(1, i)$ );
4 end
5 return Node(randCoor);
```

end

Pseudokod 6: Funkcija `randConfig`

Ova funkcija na osnovu geometrije prostora definisanog sa `boundary` strukturom, o čemu smo već pričali, generiše čvor sa koordinatama `randCoor`. Funkcijom `rand` generišemo nasumične brojeve između početka i kraja i -te koordinate prostora u kojem se nalazi stablo. Ovim smo omogućili da prostor u kojem se nalazi stablo ne mora strogo biti hiperkocka. Primjetimo

da vrijedi $\text{dof} = \text{length}(\mathcal{T}.\text{boundary}.\text{begin})$. Opet napominjemo da prilikom korištenja T-RRT algoritma, moramo dodati koordinatu koja će u sebi čuvati težinu čvora. Interval iz kojeg ćemo generisati ovu koordinatu zavisi od intervala definisanosti težinske funkcije.

2. Funkcija `nearestNode` traži najblizi čvor unutar stabla čvoru kojeg smo poslali kao parametar te funkcije. Uz to smo vraćali i udaljenost ta dva čvora. Ovo je također metoda RRT klase tako da imamo pristup ostalim atributima i metodama spomenute klase.

Inicijaliziraj: \mathcal{T}

```
function nearestNode( $q$ )
    1 min  $\leftarrow$  Inf;
    2 index  $\leftarrow$  1;
    3 for  $i = 1$  to  $\mathcal{T}.\text{getNodeNum}()$  do
        4 |  $d \leftarrow \text{distance}(q.\text{coordinates}, \mathcal{T}.\text{nodes}(i,1).\text{coordinates})$ ;
        5 | if  $d < min$  then
        6 | | min  $\leftarrow d$ ;
        7 | | index  $\leftarrow i$ ;
        8 | end
    9 end
    10 return [ $\mathcal{T}.\text{nodes}(index,1)$ , min];
end
```

Pseudokod 7: Funkcija `nearestNode`

Ova funkcija je implementirana kao standardna funkcija za traženje minimalnog elementa niza. Napomenimo da su čvorovi unutar $\mathcal{T}.\text{nodes}$ sortirani po indeksu.

3. Funkcija `newConfig` na osnovu čvorova q_{near} i q_{rand} generiše novi čvor udaljen za `deltaQ` od q_{near} prema q_{rand} . Pseudokod je prikazan ispod.

Inicijaliziraj: \mathcal{T}

```
function newConfig( $q_{near}$ ,  $q_{rand}$ )
    1  $q_{new} \leftarrow \text{Node}()$ ;
    2  $v \leftarrow q_{rand}.\text{coordinates} - q_{near}.\text{coordinates}$ ;
    3  $\text{unitV} \leftarrow v / \text{distance}(q_{rand}.\text{coordinates} - q_{near}.\text{coordinates})$ ;
    4  $q_{new}.\text{coordinates} \leftarrow q_{near}.\text{coordinates} + \mathcal{T}.\text{deltaQ} * \text{unitV}$ ;
    5  $q_{new}.\text{index} \leftarrow q_{rand}.\text{index}$ ;
    6 return  $q_{new}$ ;
end
```

Pseudokod 8: Funkcija `newConfig`

Ova funkcija je elegantno realizirana pomoću vektorskog računa iz matematike (i fizike). Objekat v čuva vektor u prostoru usmjeren od q_{near} prema q_{rand} . Ako takav vektor podijelimo sa distancom između q_{near} i q_{rand} dobivamo jednični vektor sa očuvanim smjerom. Onda takav jednični vektor možemo skalirati za `deltaQ` što je potrebna udaljenost novog čvora od q_{near} i dodavanjem takvog vektora na koordinate čvora q_{near} dobivamo željeni novi čvor. Kako q_{rand} više ne igra ulogu onda njegov indeks dodijelimo novom čvoru i vraćamo ga iz funkcije.

4. Unutar implementacije BiRRT algoritma smo koristili `mergeRRTs` funkciju koja spaja stablo nad kojim je pozvana (\mathcal{T}_1) sa drugim stablom (\mathcal{T}_2) između dva poslana čvora (q_{last_1}, q_{try}).

Pseudokod ove funkcije je prikazan ispod.

Inicijaliziraj: \mathcal{T}

```
function mergeRRTs( $\mathcal{T}_2$ ,  $q_1, q_2$ )
    1  $\mathcal{T}$ .addEdge( $q_1, q_2$ );
    2  $\mathcal{T}_2$ .shiftNodeIndexes( $\mathcal{T}$ .getNodeNum());
    3 for  $i = 1$  to  $\mathcal{T}_2$ .getNodeNum() do
        4 |  $\mathcal{T}$ .addNode( $\mathcal{T}_2$ .nodes( $i, 1$ ));
    5 end
    6 for  $i = 1$  to  $\mathcal{T}_2$ .getEdgeNum() do
        7 |  $\mathcal{T}$ .addEdge( $\mathcal{T}_2$ .edges( $i, 1$ ));
    8 end
end
```

Pseudokod 9: Funkcija mergeRRTs

Na početku ove funkcije dodajemo na prvo stablo granu $\{q_1, q_2\}$ koja spaja dva stabla. Zatim pomjerimo indekse čvorova drugog stabla (funkcijom `shiftNodeIndexes`) za broj čvorova prvog stabla tako kad ih budemo spajali indeksi konačnog stabla će opet biti sortirani. Zatim kroz dvije petlje dodamo čvorove i grane (čiji su indeksi sada pomjereni za broj čvorova prvog stabla) drugog stabla u prvo stablo. Kako je ovo interna funkcija prvog stabla onda ne moramo ništa vraćati već samo pozvati ovu metodu nad stablom i nakon pozivanja ono će u sebi sadržavati drugo stablo koje smo poslali.

5. Funkcija kojom dobivamo direktnu kinematiku robota preko DH parametara i matrica homogene transformacije je `getIthTransMatrix`. Funkcija je implementirana tako da vraća matricu transformacije i -tog zglobo. Ukoliko želimo da dobijemo matricu homogene transformacije vrha manipulatora onda je dovoljno postaviti $i = \text{dof}$. Razlog ovakve implementacije je što će nam često biti potrebne koordinate vrha i -tog segmenta koje se čitaju iz četvrte kolone matrice homogene transformacije. Uvest ćemo i pomoćnu funkciju `getIthSegmentTipCoordinates` koja vadi tu informaciju iz matrice homogene transformacije. Pseudokod za ove dvije funkcije je prikazan ispod.

Inicijaliziraj: robot

```
function getIthTransMatrix(i)
    1 T  $\leftarrow$  I(4);
    2 DH  $\leftarrow$  robot.getDHparameters();
    3 for  $j = 1$  to  $i$  do
        4 | T  $\leftarrow$  T * A(DH( $j, :$ ));
    5 end
    6 return T;
end
function getIthSegmentTipCoordinates(i)
    1 T  $\leftarrow$  robot.getIthTransMatrix(i);
    2 return T(1:3,4);
end
```

Pseudokod 10: Funkcija getIthTransMatrix

gdje je A matrica homogene transformacije koju smo definisali u jednačini 5.2 i ona prima DH parametre (j -ti red DH matrice) kao ulaz.

6. Funkcija kojom ćemo provjeravati da li manipulator siječe prepreke u radnom prostoru je `isRobotInFreeSpace` koja se bazira na već objašnjenoj funkciji `isSegmentInFreeSpace`. Ono što moramo uraditi jeste pozvati funkciju `isSegmentInFreeSpace` za svaki segment, čije koordinate vrha dobivamo iz funkcije `getIthSegmentTipCoordinates`, i provjeriti da li segment siječe prepreku. Napomenimo da podrazumijevamo da se baza manipulatora nalazi u koordinatnom početku. Prva tačka prvog segmenta će biti koordinatni početak, a druga tačka će biti izlaz funkcije `getIthSegmentTipCoordinates(1)`. Onda će za sljedeći segment prva tačka biti vrh prethodnog segmenta, a druga tačka će biti izlaz funkcije `getIthSegmentTipCoordinates(2)` i tako dalje sve do zadnjeg segmenta. Sve navedeno je prikazano u pseudokodu ispod.

```
Inicijaliziraj: robot
function isRobotInFreeSpace(robot)
    1 p1 ← zeros(robot.dim,1);
    2 for i = 1 to robot.dof do
        3     p2 ← robot.getIthSegmentTipCoordinates(i);
        4     if !isSegmentInFreeSpace(robot.obstacles, p1, p2) then
        5         | return false;
        6     end
        7     p1 ← p2;
    8 end
    9 return true;
end
```

Pseudokod 11: Funkcija `isRobotInFreeSpace`

7. Funkcijom `isRobotMoveInFreeSpace` ćemo provjeriti da li manipulator prilikom kretanja od konfiguracije i do konfiguracije $i + 1$ siječe prepreke. Ova informacija će biti korisna prilikom spajanja dva stabla, jer na mjestu spajanja ne znamo da li bi manipulator kretanjem od prvog do drugog čvora te grane sijekao prepreku. Ovo mjesto spajanja ima dužinu `minDistance` i u većim prostorima poželjno je povećati `minDistance` zbog obimnosti prostora, ali povećanjem `distance` spajanja se povećava mogućnost sudara na mjestu spoja. Ovo se može lagano riješiti `linspace` funkcijom koju smo koristili ranije tako što ćemo podijeliti interval između konfiguracija gdje vršimo spoj na `disect` dijelova i za svaki dio pozvati `isRobotInFreeSpace` funkciju i provjeriti postoji li presjek sa preprekom. Pseudokod je prikazan ispod.

```
Inicijaliziraj: robot, dissect
function isRobotMoveInFreeSpace(robot, config1, config2)
    1 configs ← linspace(config1, config2, dissect);
    2 for i = 1 to dissect do
        3     robot.currConfig ← configs(:,i);
        4     if !isRobotInFreeSpace(robot) then
        5         | return false;
        6     end
    7 end
    8 return true;
end
```

Pseudokod 12: Funkcija `isRobotMoveInFreeSpace`

8. U prethodnom poglavlju smo govorili o algoritmima kojima možemo računati minimalnu udal-

jenost proizvoljne tačke p od konveksnog omotača. Ova informacija će nam biti potrebna prilikom konstrukcije težinske funkcije. Funkcija `getRobotTipToObstaclesDistance` vraća udaljenost vrha manipulatora do najbliže prepreke (možemo imati više prepreka). Vrh manipulatora se tretira kao tačka p čije koordinate dobivamo iz funkcije `getIthSegmentTipCoordinates`. Pseudokod ove funkcije je prikazan ispod.

```
Inicijaliziraj: robot
function getRobotTipToObstaclesDistance (robot)
    1 if !isRobotInFreeSpace(robot) then
    2     | return 0;
    3 end
    4 dMin ← Inf;
    5 p ← robot.getIthSegmentTipCoordinates(robot.dof);
    6 for i = 1 to length(robot.obstacles) do
    7     |  $\mathcal{X}_i \leftarrow \text{robot.obstacles}(i,1)$ ;
    8     | if robot.dim == 2 then
    9         |     | currD ← p.poly_dist(p(1,1), p(2,1),  $\mathcal{X}_i(:,1), \mathcal{X}_i(:,2)$ );
   10     | else if robot.dim == 3 then
   11         |     | [A, b, A_eq, b_eq] ← vert2lcon( $\mathcal{X}_i$ );
   12         |     | pC ← lsqlin(I(3,3), p, A, b, A_eq, b_eq);
   13         |     | currD ← distance(p, pC);
   14     | else
   15         |     | return error;
   16     | end
   17     | if currD < dMin then
   18         |     | dMin ← currD;
   19     | end
   20 end
   21 return d;
end
```

Pseudokod 13: Funkcija `getRobotTipToObstaclesDistance`

Na početku provjeravamo da li neki od segmenata manipulatora siječe prepreke sa funkcijom `isRobotInFreeSpace`. Ukoliko siječe, onda smatramo da je udaljenost vrha manipulatora od prepreka jednaka nuli. Ovaj rezultat će dobiti značaj kada budemo konstruisali težinsku funkciju. U funkciju `getNthSegmentTipCoordinates` šaljemo stepen slobode manipulatora čime dobivamo koordinate vrha manipulatora za koje provjeravamo udaljenost. Ostatak pseudokoda je trivijalan uz prezentirano objašnjenje, iz prošlog poglavlja, kako se određuje udaljenost između proizvoljne tačke i konveksnog omotača.

7.3 Integrisani algoritam za planiranje putanje robotskog manipulatora

Kako smo definisali potrebne funkcije i algoritme sada možemo konačno integrisati traženje putanje sa kretanjem manipulatora u radnom prostoru. Glavna funkcija `computeRobotPath` prima objekat `robot` i korištenjem RRT baziranih algoritama pronađi skup konfiguracija koje čine put robota. Naglasimo da se objekat `robot` šalje po referenci tako da, koje god mu se promjene dese u funkciji, to se i odražava na taj objekat nakon što je funkcija nad njim pozvana. Ovo je bitno zbog činjenice da, kada pronađemo konačno stablo nakon pozivanja funkcije, možemo direktno pozivanjem `getShortestPath`

funcije nad stablom objekta **robot** dobiti **configVector** vektor konfiguracija koje čine finalni put.

```
Inicijaliziraj: dof, dim, a, d, initConfig, finalConfig, currConfig, configVector,  
obstacles,  $\mathcal{T}_1$ ,  $\mathcal{T}_2$ , eta, mi
 $\mathcal{T}_1$ .addNode(initConfig, 1);
 $\mathcal{T}_2$ .addNode(finalConfig, 1);
function computeRobotPath(robot)
    1  $q_{last_1} \leftarrow \text{robot.}\mathcal{T}_1\text{.nodes}(1,1)$ ;
    2 for  $i = 1$  to  $\text{robot.}\mathcal{T}_1\text{.maxIter}$  do
        3     if  $\text{rand}(0,1) < \text{robot.eta}$  then
            4         if  $\text{rand}(0,1) < \text{robot.mi}_1$  then
                5              $q_{rand_1} \leftarrow q_{new_2}$ ;
            6         else
                7              $q_{rand_1} \leftarrow \text{robot.}\mathcal{T}_1\text{.randConfig}()$ ;
            8     end
            9      $q_{rand_1} \leftarrow \text{robot.}\mathcal{T}_1\text{.randConfig}()$ ;
        10    [ $q_{near_1}$ ,  $d_{-q\_rand1}$ ]  $\leftarrow \text{robot.}\mathcal{T}_1\text{.nearestNode}(q_{rand_1})$ ;
        11    if  $d_{-q\_rand1} < \text{robot.}\mathcal{T}_1\text{.deltaQ}$  then
            12         $q_{new_1} \leftarrow q_{rand_1}$ ;
        13    else
            14         $q_{new_1} \leftarrow \text{robot.}\mathcal{T}_1\text{.newConfig}(q_{near_1}, q_{rand_1})$ ;
        15    end
        16    if  $\text{isa}(\text{robot.}\mathcal{T}_1, \text{'TRRT'})$  then
            17        con  $\leftarrow \text{robot.}\mathcal{T}_1\text{.transitionTest}(q_{near_1}, q_{new_1})$  and
            18             $\text{robot.}\mathcal{T}_1\text{.minExpandControl}(q_{near_1}, q_{rand_1})$ ;
        19    else
            20        robot.currConfig  $\leftarrow q_{new_1}$ .coordinates;
            21        con  $\leftarrow \text{isRobotInFreeSpace}(\text{robot})$ ;
        22    end
        23    if con then
            24         $\mathcal{T}_1$ .addNode( $q_{new_1}$ );
            25         $\mathcal{T}_1$ .addEdge( $q_{near_1}$ ,  $q_{new_1}$ );
            26         $q_{last_1} \leftarrow q_{new_1}$ ;
        27    else
            28        continue;
        29    end
        30    Analogno ponoviti postupak prvog stabla za drugo stablo
    end
    32     $q_{try} \leftarrow \text{robot.}\mathcal{T}_2\text{.nearestNode}(q_{last_1})$ ;
    33    if  $\text{distance}(q_{last_1}\text{.coordinates}, q_{try}\text{.coordinates}) < \mathcal{T}_1\text{.minDistance}$ 
    34    and  $\text{isRobotMoveInFreeSpace}(\text{robot}, q_{last_1}\text{.coordinates}, q_{try}\text{.coordinates})$  then
        35         $\text{robot.}\mathcal{T}_1\text{.mergeRRTs}(\mathcal{T}_2, q_{last_1}, q_{try})$ ;
        36        break;
    37    end
    38 end
    39 return;
end
```

Pseudokod 14: Funkcija **computeRobotPath**

Unutar ovog pseudokoda su sjedinjene ideje:

- Običnog RRT algoritma,
- BiRRT algoritma,
- BiRRT algoritma sa upravljanjem širenja stabala sa parametrom `eta`,
- BiRRT algoritma sa upravljanjem smjera širenja stabala sa parametrom `mi`,
- T-RRT algoritma,

ćime dobivamo dosta fleksibilnosti prilikom testiranja jer jednom funkcijom pokrivamo sve slučajeve. Stvar koju treba naglasiti je uslov dodavanja novog čvora i nove grane u stablu. Kako T-RRT algoritam ima dva dodatna uslova za dodavanje čvora i grane u stablo onda moramo ispitati funkcijom `isA` tip stabla koje je poslato. Primjetimo da prilikom konstrukcije uslova `con` za dodavanje novog čvora T-RRT stablu ne testiramo direktno da li manipulator siječe prepreke sa funkcijom `isRobotInFreeSpace` već samo pravimo logičko “i” sa dva uslova T-RRT algoritma. Razlog za ovo je optimizacijske prirode jer unutar funkcije `transitionTest` pozivamo težinsku funkciju `costFunction`, a unutar `costFunction` ćemo pozivati funkciju `getRobotTipToObstacleDistance` (što ćemo vidjeti ukratko), a unutar te funkcije pozivamo funkciju `isRobotInFreeSpace`. Tako da pozivanjem `transitionTest` indirektno pozivamo `isRobotInFreeSpace` ćime se može uštedjeti dosta vremena ako to radimo na ovakav, indirektni način.

Što se tiče RRT stabla, tu direktno testiramo da li manipulator siječe prepreke u radnom prostoru. Ostali dio funkcije je kroz prethodnu diskusiju indirektno objašnjen.

7.4 Generisanje težinske funkcije manipulatora

Nakon što smo definisali sve potrebne pojmove, algoritme i funkcije zadnji korak prije testiranja je generisati težinsku funkciju na osnovu koje ćemo izbjegavati prepreke. Kriterij koji želimo uspostaviti je da vrh manipulatora bude što dalje od prepreka. Sličan kriterij smo imali kada smo izbjegavali dvije tačke sa slike 17. Samo će sada tačke koje ćemo izbjegavati biti konfiguracije \mathbf{q} za koje manipulator siječe prepreke u radnom prostoru odnosno one konfiguracije za koje funkcija `isRobotInFreeSpace` vraća log. 0. Pošto moramo izbjegavati prepreke, težinsku funkciju možemo potražiti u obliku:

$$f_c(\mathbf{q}) = A \cdot \exp(-B \cdot f_d(\mathbf{robot})), \quad (7.1)$$

gdje funkcija f_d za trenutnu konfiguraciju manipulatora vraća udaljenost vrha manipulatora od prepreka. Ovo je upravo funkcija koju smo već implementirali `getRobotTipToObstaclesDistance` tako da je tražena težinska funkcija:

$$f_c(\mathbf{q}) = A \cdot \exp(-B \cdot \text{getRobotTipToObstaclesDistance}(\mathbf{robot}(\mathbf{q}))), \quad (7.2)$$

s tim da imamo međukorak u kojem postavljamo manipulator u konfiguraciju težinske funkcije:

```
robot.currConfig ← q.coordinates
```

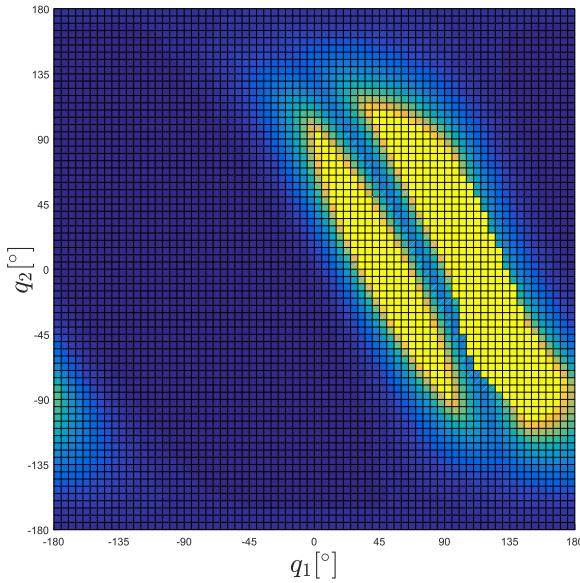
tako da je funkciji `getRobotTipToObstaclesDistance` poznat trenutni položaj manupulatora u radnom prostoru. Nakon nekoliko testiranja, konstante A i B koje su dale zadovoljavajuće težinske mape su $A = 1$ i $B = 3$ tako da ćemo od sada koristiti sljedeću težinsku funkciju:

$$f_c(\mathbf{q}) = \exp(-3 \cdot \text{getRobotTipToObstaclesDistance}(\text{robot}(\mathbf{q}))) \quad (7.3)$$

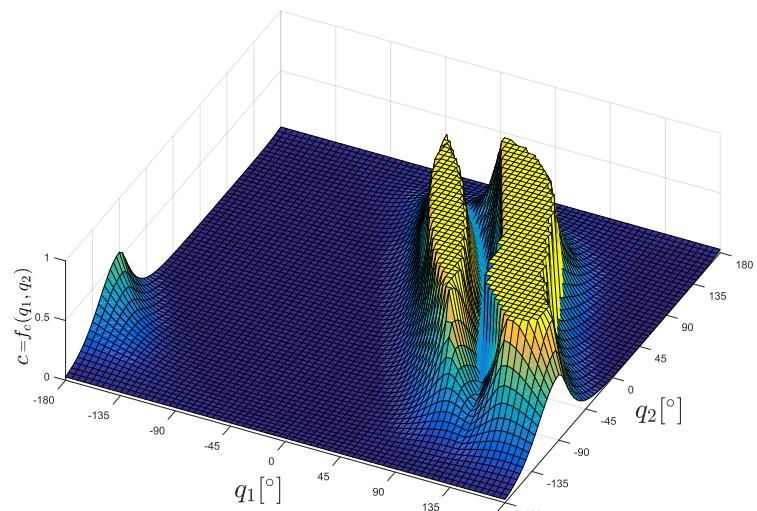
Primjetimo da konstanta A samo skalira težinsku funkciju tako da biranjem da je $A = 1$ normaliziramo težinsku mapu tako da je $0 < f_c(\mathbf{q}) \leq 1$ gdje za $f_c(\mathbf{q}) \rightarrow 0$ imamo najdalju poziciju vrha manipulatora od prepreka, a kada $f_c(\mathbf{q}) \rightarrow 1$ onda smo blizu prepreka. Zbog eksponencijanog oblika težinske funkcije, činjenica koju smo spomenuli, da će izlaz `getRobotTipToObstaclesDistance` funkcije biti nula kada smo pozivali `isRobotInFreeSpace` (prva linija koda pseudokoda 13) garantuje da će za konfiguracije koje dovode do sudara, težina biti 1 tako da ograničavanjem $c_{\max} < 1$ ova regija težinske mape se de facto ponaša kao prepreka.

Drugi parametar B određuje koliko je strma težinska mapa, gdje za strmije mape (veće B) možemo prići bliže preprekama, a za blaže mape smo dalji od prepreka (manje B), ali ovim možemo usporiti širenje stabla jer onda težina cijele mape raste. Odabriom $B = 3$ pravimo malo strmije težinske mape, ali omogućujemo veću brzinu širenja stabla.

Težinska mapa konstruisana na bazi težinske funkcije f_c za primjer dvosegmentne planarne ruke sa slike 23a je prikazan na slici 29.



(a) Pogled na težinsku mapu iz z ose.



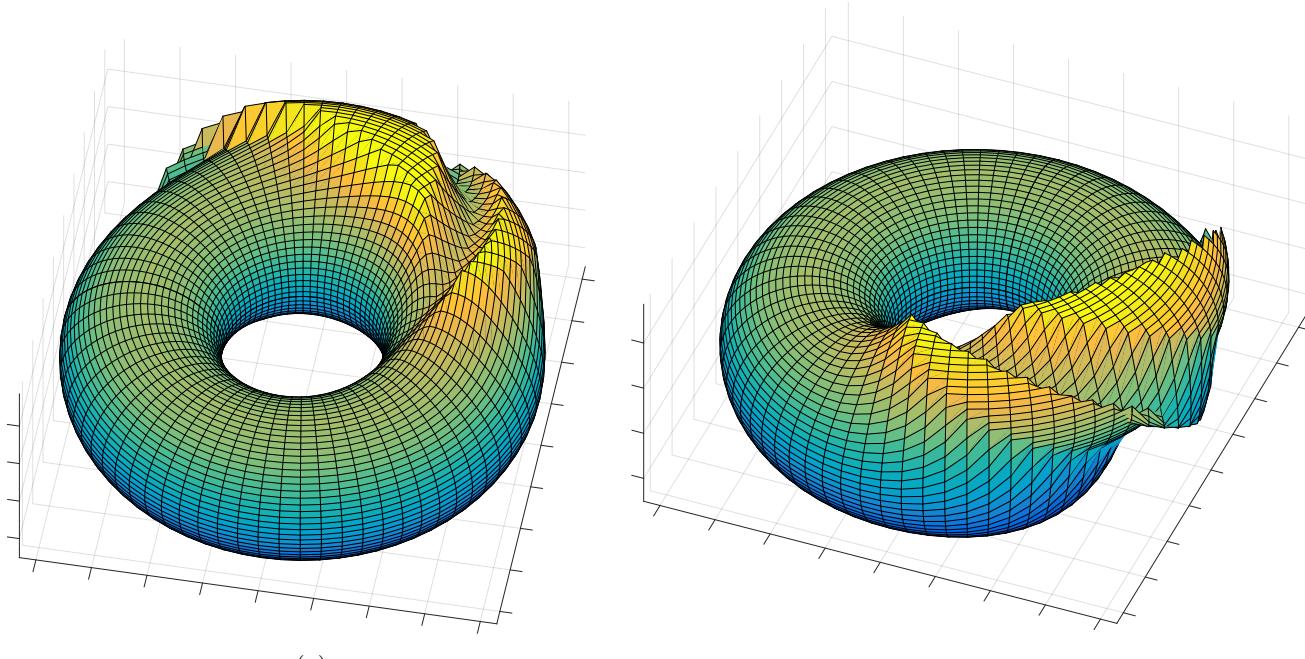
(b) Puni pogled na težinsku mapu.

Slika 29: Težinska mapa u konfiguracijskom prostoru manipulatora. Generisano težinskom funkcijom $f_c(\mathbf{q}) = \exp(-3 \cdot \text{getRobotTipToObstaclesDistance}(\text{robot}(\mathbf{q})))$ koju ćemo koristiti za buduće testiranje.

Ovakvim posmatranjem težinske mape, kanjon između dvije prepreke u konfiguracijskom prostoru odgovara konfiguracijama koje dovede manipulator između dvije prepreke u radnom prostoru.

Možemo primjetiti interesantnu pojavu na slici 29, da malo uzvišenje lijeve strane gornjih slika upravo odgovara odsječenom dijelu mape sa desne strane. Kako smo ograničili zakret ugla zglobova

na interval $[-\pi, \pi]$ onda ne može spojiti ova dva regiona. Preciznije, iako su dvije konfiguracije manipulatora u radnom prostoru bliske, u konfiguracijskom prostoru su daleko. U slučaju kada bismo imali punu slobodu zakreta, mogli bismo spojiti te regije i dva bi prostora imala sličnu topološku reprezentaciju. Da bismo dobili takvu "spojenu" mapu konfiguracijskog prostora, morali bismo "urolati" mapu u oblik cilindra, ali to nije sve. Isto vrijedi i za gornju i donju stranu konfiguracijskog prostora. Tako da bismo sad morali "urolati" i cilindar čime dobivamo topološki oblik poznat kao *torus*. Na slikama 30 vidimo težinsku mapu sa slike 29 mapiranu na površinu torusa.



Slika 30: Reprezentacija konfiguracijskog prostora kao torus.

Sa ovakvom mapom imamo punu slobodu kretanja u konfiguracijskom prostoru.

8 Testiranje, simulacije, rezultati i diskusija

U ovom poglavlju ćemo intenzivno testirati dosad opisane algoritme na raznim manipulatorima, nakon toga ćemo analizirati dobivene rezultate. Kako težinske mape možemo vizualizirati samo za manipulatore sa dva stepena slobode onda ćemo težinsku mapu prikazati samo za dvosegmentnu planarnu ruku. Što se tiče prikazivanja konfiguracijskog prostora, i njega ćemo prikazati samo za dvosegmentnu planarnu ruku. Iako ga možemo prikazati i za manipulatore sa tri stepena slobode, vremenski je intenzivno dobiti egzaktnu sliku prepreka u konfiguracijskom prostoru na osnovu radnog prostora. Tako da ćemo za svoje ostale manipulatore samo prikazati sliku radnog prostora. Kako smo već navodili, struktura `boundary` će imati oblik:

$$\text{boundary.begin} = [-\pi \cdots -\pi]_{1 \times \text{dof}}$$

$$\text{boundary.end} = [\pi \cdots \pi]_{1 \times \text{dof}}$$

zbog ograničavanja ugla rotacije zglobova na intervalu $[-\pi, \pi]$ tako da će to ograničiti radni prostor na gore navedeni interval za `boundary` strukturu ma da zbog načina implementacije ove varijable možemo ograničiti proizvoljni zglob na proizvoljni interval.

Što se tiče prikazivanja simulacija i parametara simulacija odlučili smo da prvo prikažemo u tabelama parametre manipulatora, parametre BiRRT/T-RRT algoritma i rezultate simulacija. Nakon toga slijede slike simulacija tako da se čitaoc lahko može snaći zbog količine informacija. Uveli smo još dodatno označke za manipulatore u tabelama: 2SPR (dvosegmentna planarna ruka), 3SPR (trosegmenata planarna ruka), AR (antropomorfna ruka), SŠ (sferna šaka), AS (kombinacija AR i SŠ).

Što se tiče tabele rezultata, tu ćemo prikazati dužinu puta vrha manipulatora u radnom prostoru (RP), dužinu nađenog puta u konfiguracijskom prostoru (KS), prosječnu težinu puta koju definišemo kao sumu težinu konfiguracija koje čine put podjeljenu sa brojem konfiguracija koje čine taj put, broj iteracija algoritma, broj čvorova spojenog stabla i vrijeme koje je potrebno da algoritam terminira mjereno u sekundama. Svi primjeri će biti urađeni na istoj pseudoslučanoj sekvenci brojeva (rečeno drugim riječima, svi primjeri imaju isti *random seed*) radi komparabilnosti rezultata između pojedinih primjera.

Na kraju ćemo za trosegmentnu planarnu ruku uraditi nekoliko testova za parametre T-RRT algoritma radi shvatanja kako oni utiču na performanse ovog algoritma. Radi bolje vizualizacije ovi testovi će biti prikazani u uslovno-formatiranim tabelama tako da je moguće vidjeti jasna veza između parametara i rezultata.

8.1 Testiranje robotskih manipulatora

Tabela 5: Informacije o parametrima manipulatora.

Slika	Alg.	Man.	dof	dim	a	α	d	initConfig	finalConfig
31	BiRRT	2SPR	2	2	$[1 \ 1]^T$	$[0 \ 0]^T$	$[0 \ 0]^T$	$[-166^\circ \ 0^\circ]^T$	$[64^\circ \ 28^\circ]^T$
32	T-RRT								
33a	BiRRT	3SPR	3	2	$[1 \ 0.5 \ 0.5]^T$	$[0 \ 0 \ 0]^T$	$[0 \ 0 \ 0]^T$	$[-170^\circ \ 0^\circ \ 0^\circ]^T$	$[61^\circ \ 28^\circ \ 2^\circ]^T$
33b	T-RRT								
34a	BiRRT	AR	3	3	$[0 \ 1.5 \ 1.5]^T$	$[\pi/2 \ 0 \ 0]^T$	$[0 \ 0 \ 0]^T$	$[45^\circ \ 135^\circ \ -45^\circ]^T$	$[55^\circ \ 0^\circ \ -45^\circ]^T$
34b	T-RRT								
35a	BiRRT	SŠ	3	3	$[0 \ 0 \ 0]^T$	$[-\pi/2 \ \pi/2 \ 0]^T$	$[0 \ 0 \ 0.5]^T$	$[-90^\circ \ 90^\circ \ 0^\circ]^T$	$[0^\circ \ -90^\circ \ 120^\circ]^T$
35b	T-RRT								
36a	BiRRT	AS	6	3	$[0 \ 1 \ 0 \ 0 \ 0 \ 0]^T$	$[\pi/2 \ 0 \ \pi/2 \ -\pi/2 \ \pi/2 \ 0]^T$	$[0 \ 0 \ 0 \ 0.5 \ 0 \ 0.5]^T$	$[45^\circ \ 90^\circ \ 90^\circ \ 0^\circ \ 0^\circ \ 0^\circ]^T$	$[90^\circ \ -20^\circ \ 20^\circ \ 90^\circ \ 60^\circ \ 45^\circ]^T$
36b	T-RRT								

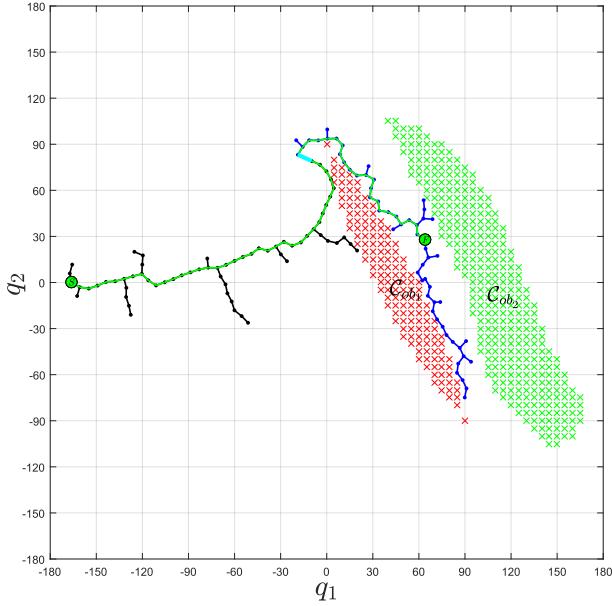
Tabela 6: Informacije o parametrima BiRRT/TRRT algoritma.

Slika	Algoritam	maxIter	deltaQ	minDitance	eta	mi	T	K	alpha	c_max	maxFails	rho
31	BiRRT	1000	6	10	0.33	$[0.500 \ 20]$	—	—	—	—	—	—
32	T-RRT	10000					$7 \cdot 10^{-5}$	0.15	2.69	0.35	15	0.05
33a	BiRRT	1000	8	10	0.2	$[0.25 \ 0.15]$	—	—	—	—	—	—
33b	T-RRT	10000					$5 \cdot 10^{-5}$	0.15	3.15	0.37	20	0.08
34a	BiRRT	5000	8	10	0.28	$[0.65 \ 0.30]$	—	—	—	—	—	—
34b	T-RRT	10000					$9 \cdot 10^{-5}$	0.04	2.15	0.42	9	0.12
35a	BiRRT	5000	10	10	0.35	$[0.30 \ 0.18]$	—	—	—	—	—	—
35b	T-RRT	10000					$2 \cdot 10^{-6}$	0.58	4.19	0.75	18	0.05
36a	BiRRT	5000	10	10	0.05	$[0.50 \ 0.30]$	—	—	—	—	—	—
36b	T-RRT	10000					$1 \cdot 10^{-6}$	0.36	1.2	0.75	5	0.15

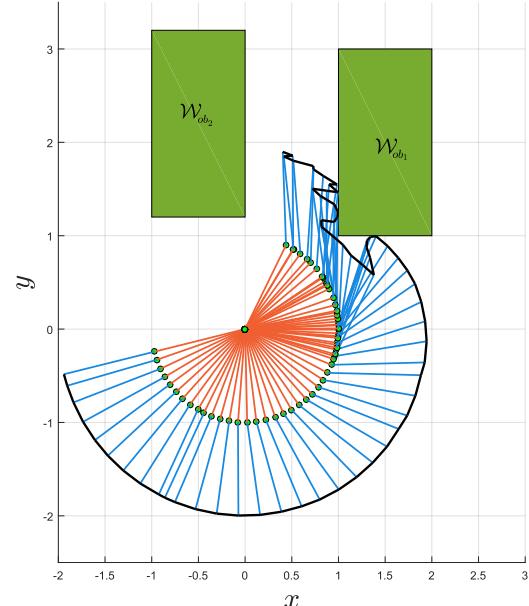
Tabela 7: Informacije o rezultatima simulacija.

Slika	Manipulator	Algoritam	Dužina puta (RP)	Dužina puta (KP)	Prosječna težina puta	Br. iteracija	Br. čv.	Vrijeme [s]
31	2SPR	BiRRT	10.25	375.81	0.326	243	122	1.092
32	2SPR	TRRT	7.229	388.86	0.097	3245	216	11.759
33a	3SPR	BiRRT	10.18	660.00	0.301	387	225	3.337
33b	3SPR	TRRT	7.709	621.26	0.080	2235	177	8.042
34a	AR	BiRRT	17.20	514.63	0.191	1205	248	4.759
34b	AR	TRRT	10.08	363.61	0.095	8664	373	89.249
35a	SŠ	BiRRT	1.98	307.71	0.553	39	38	0.606
35b	SŠ	TRRT	2.14	334.30	0.513	1853	207	32.220
36a	AS	BiRRT	8.03	526.80	0.381	346	129	3.632
36b	AS	TRRT	12.02	870.073	0.111	7045	440	170.781

8.1.1 Dvosegmentna planarna ruka

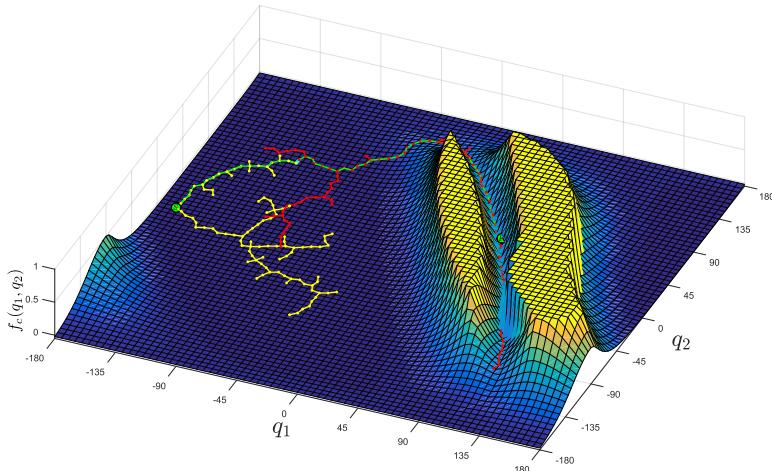


(a) Konfiguracijski prostor.

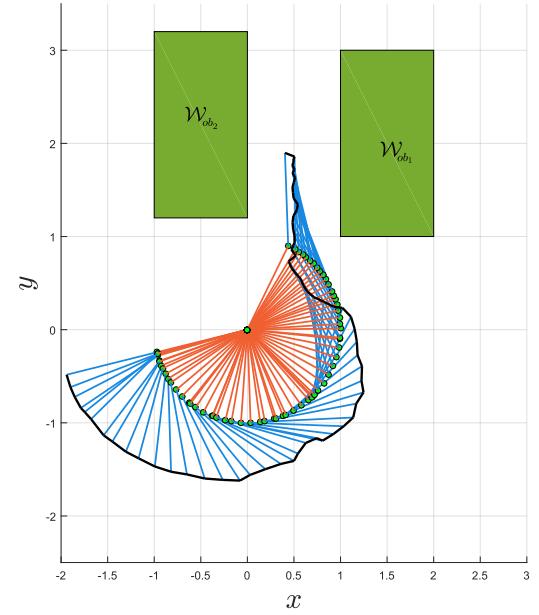


(b) Radni prostor.

Slika 31: Planiranje kretanja dvosegmentnom planarnom rukom sa BiRRT algoritmom.



(a) Konfiguracijski prostor.



(b) Radni prostor.

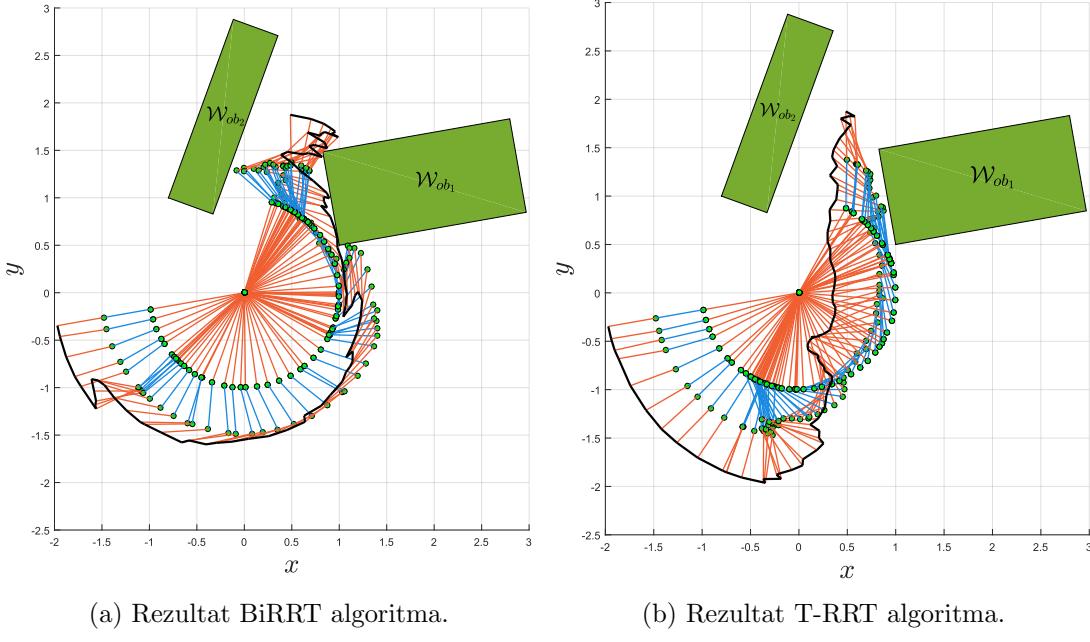
Slika 32: Planiranje kretanja dvosegmentnom planarnom rukom sa T-RRT algoritmom.

Krajnja konfiguracija koju moramo postići u radnom prostoru dvosegmentne planarne ruke je poprilično zahtijevna. Ova se činjenica još bolje vidi u konfiguracijskom prostoru gdje je krajnja konfiguracija

zarobljena između dvije prepreke. U slučaju BiRRT algoritma nismo ograničeni na blizinu kojoj smijemo prići preprekama tako da se stabla šire u blizini prepreka jer je krajnja konfiguracija u nepovoljnem položaju. Ovo rezultira oblikom putanja manipulatora u radnom prostoru sa slike 31b gdje je vrh manipulatora jako blizu prepreka što u realnom svijetu može izazvati potencijalno oštećenje. Tamo gdje BiRRT algoritam kompenzira ovu manu je vrijeme izvršavanja, koje je oko 1.1 [s].

Na težinskoj mapi konfiguracijskog prostora krajnja konfiguracija je "uklještena" između prepreka i postoji uski "kanjon" koji vodi u niskotežnisku regiju mape. T-RRT algoritam prati ovaj "kanjon" i nakon dolaska u niskotežnisku regiju stabla se spajaju. Na slici radnog prostora 32b vidimo da je kriterij, koji je bio definisan težinskom funkcijom, ispunjen. Vrh manipulatora je na maksimalnoj mogućoj udaljenosti od prepreka i njegovo kretanje u radnom prostoru je sigurno. Ovakav povoljan rezultat je plaćen dužim vremenom izvršavanja od 11.8 [s]. Možemo vidjeti i jasnu razliku BiRRT i T-RRT algoritma u prosječnoj težini koja ide mnogostruko u korist T-RRT algoritma što se i ogleda na obliku putanja manipulatora.

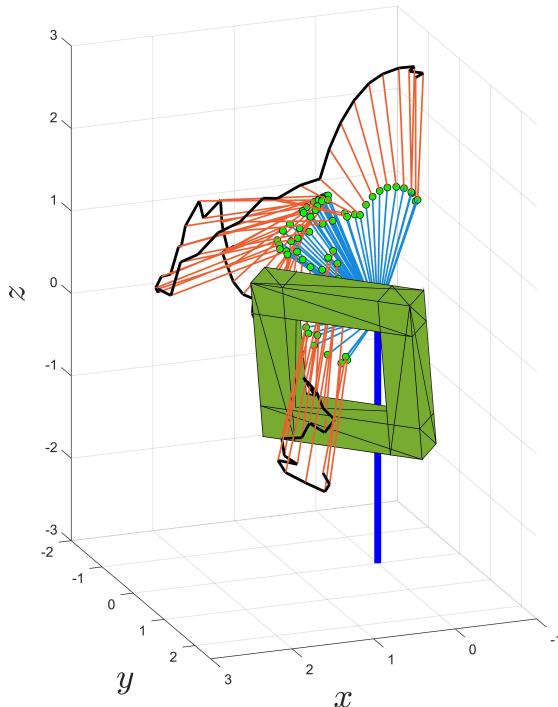
8.1.2 Trosegmentna planarna ruka



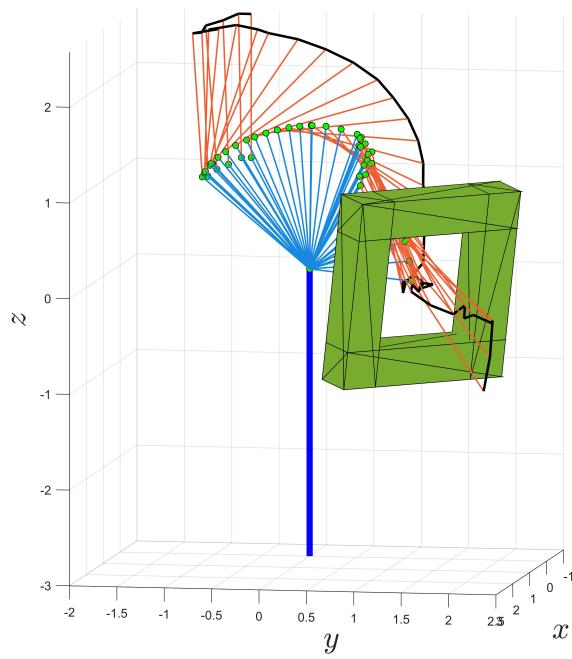
Slika 33: Planiranje kretanja trosegmentnom planarnom rukom sa BiRRT/T-RRT algoritmima u radnom prostoru.

Iz radnog prostora trosegmentne planarne ruke vidimo da je krajnja konfiguracija u povoljnijem položaju nego u prethodnom primjeru. Iako je porasla dimenzija konfiguracijskog prostora vrijeme izvršavanja T-RRT algoritma se smanjilo jer "kanjon" koji smo vidjeli u prošlom primjeru je sada "širi" što daje više mjesta za širenje stabla. Vidimo jasnu razliku u obliku putanja za dva algoritma gdje BiRRT algoritam dovodi manipulator u opasne regije gdje može doći do sudara dok T-RRT algoritam garantira da će vrh manipulatora biti na sigurnoj udaljenosti od prepreka. Rezultati u tabeli 7 su očekivani za ovaj primjer.

8.1.3 Antropomorfna ruka



(a) Rezultat BiRRT algoritma.

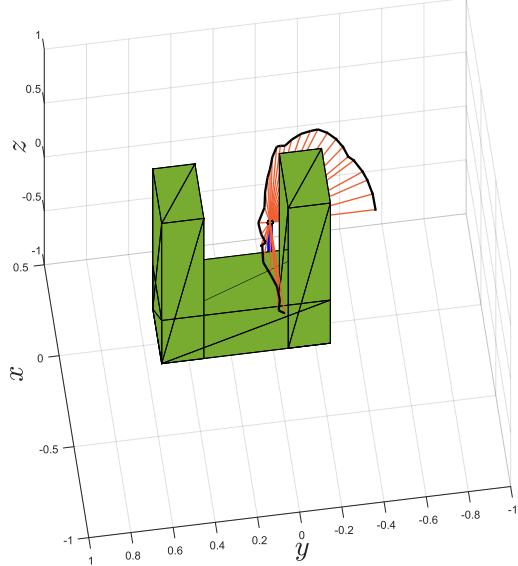


(b) Rezultat T-RRT algoritma.

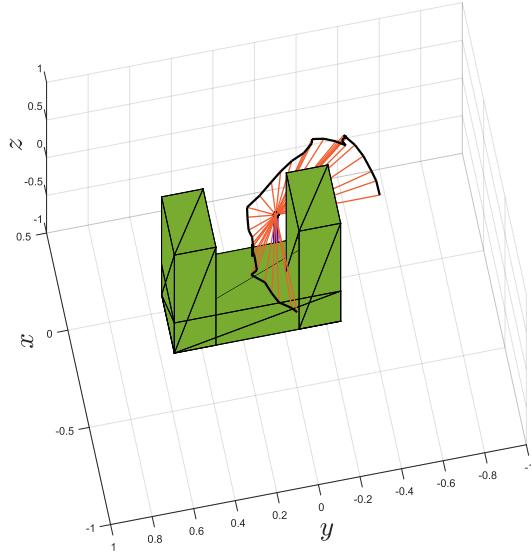
Slika 34: Planiranje kretanja antropomorfnom rukom sa BiRRT/T-RRT algoritmima u radnom prostoru.

Antropomorfna ruka je manipulator koji se kreće u 3D prostoru. Za prepreku u 3D prostoru smo izabrali rotirani prozor o kojem smo već pričali. Krajnja konfiguracija je izabrana tako da vrh manipulatora mora proći kroz otvor prozora gdje obavlja neku operaciju. Opet možemo primjetiti šablon izgleda putanje u radnom prostoru za dva korištena algoritma gdje BiRRT algoritam dovodi manipulator u željeni položaj ali opet je ugrožena sigurnost dok T-RRT algoritam provodi vrh manipulatora tačno kroz sredinu otvora prozora. U konfiguracijskom prostoru bi sredina otvora prozora bila opet "kanjon" koji spaja dvije niskotežinske regije (slobodni prostor prije otvora i poslije otvora) tako da T-RRT algoritam upravo prati taj "kanjon". Ovo možemo vidjeti i na podatuču o srednjoj težini puta. Sve dobre odlike su kompenzirane sa podugačkim vremenom izvršavanja dok BiRRT algoritam za ovako zahtijevan radni prostor rješenje nalazi u rekordnom vremenu.

8.1.4 Sferna šaka



(a) Rezultat BiRRT algoritma.



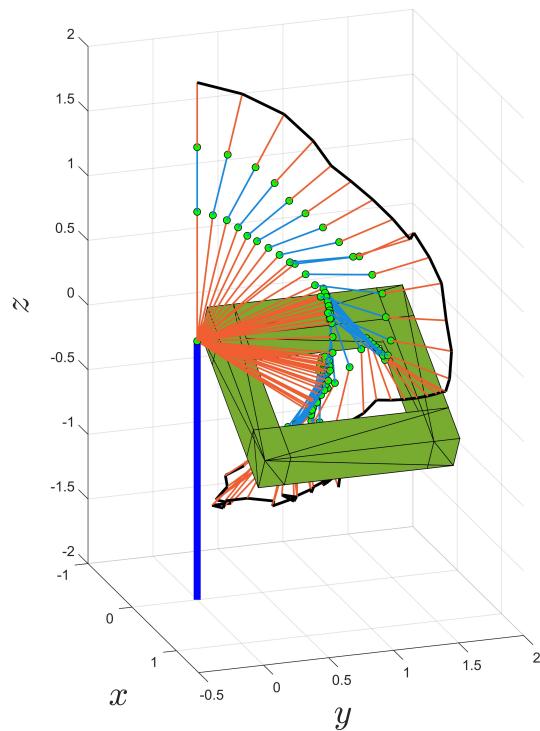
(b) Rezultat T-RRT algoritma.

Slika 35: Planiranje kretanja sfernom šakom sa BiRRT/T-RRT algoritmima u radnom prostoru.

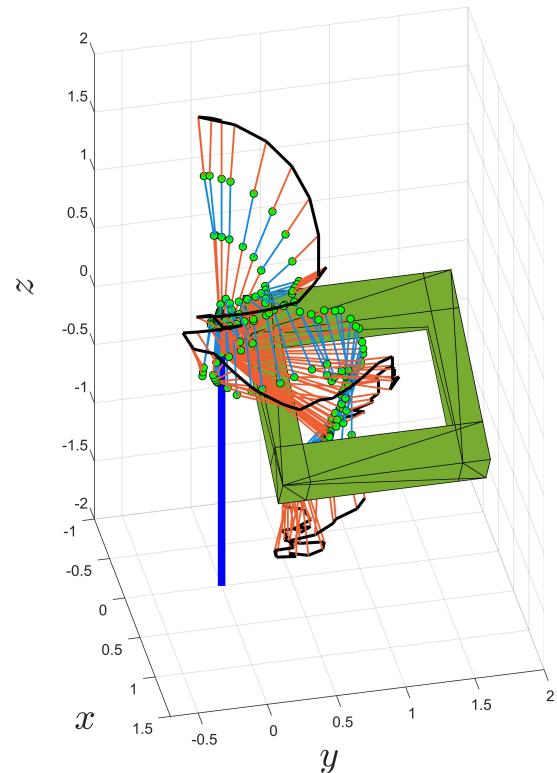
Sferna šaka je manipulator koji se generalno ne koristi samostalno već se montira na antropomorfnu ili sfernu ruku ali radi kompletnosti prikazat ćemo i primjer sa ovim manipulatorom. Sa slika 35 jasno vidimo da putanja dobivena BiRRT algoritmom je opasno blizu desne stranice prepreke dok putanja dobivena T-RRT algoritmom teži ka sredini otvora što ispunjava postavljeni kriterij.

8.1.5 Kombinacija antropomorfne ruke i sferne šake

Kombinacija antropomorfne ruke i sferne šake je najzahtijevniji manipulator jer ima šest stepeni slobode što implicira visoku dimenzionalnost konfiguracijskog prostora, kojeg treba pretražiti. U radnom prostoru smo izabrali opet prepreku oblika prozora gdje vrh manipulatora mora obaviti neki zadatak unutar otvora prozora. Za razliku od antropomorfne ruke koja ima dva segmenta nenule duljina, sada imamo tri takva segmenta, tako da je zahtijevnije dovesti manipulator u željeni položaj i mogućnost sudara je dosta veća. U smislu rezultata vidimo sličnu pojavu kao kod antropomorfne ruke gdje kod BiRRT algoritma vrh manipulatora teži da prođe kroz otvor prozora na bilo koji način, a u slučaju T-RRT algoritma vidimo da vrh manipulatora teži da prođe kroz otvor na što većoj udaljenosti od stranica prozora. Zbog veličine prostora i položaja početne i krajnje konfiguracije vrijeme izvršavanja T-RRT algoritma je dosta veliko. Ovdje se vidi brzina BiRRT algoritma koji ovako zahtijevan zadatak obavlja ispod četiri sekunde. Ostali rezultati i parametri se mogu naći u tabelama.



(a) Rezultat BiRRT algoritma.



(b) Rezultat T-RRT algoritma.

Slika 36: Planiranje kretanja kombinacijom antropomorfne ruke i sferne šake sa BiRRT/T-RRT algoritmima u radnom prostoru.

8.2 Uticaj ključnih parametara T-RRT algoritma na njegovu performansu

U predstavljanju T-RRT algoritma zaključili smo da su parametri sa najviše uticaja `alpha` koji adaptira temperaturu, `maxFails` koji upravlja vjerovatnoćom dodavanja grana sa pozitivnim nagibom i `rho` koji upravlja sa ustinjašanjem prostora kojeg smo već pretražili. Najboljni način predstave uticaja ovih parametara na performanse je promatrati uslovno-formatirane tabele rezultata za kombinaciju promjene parametara. Tople boje označavaju visoke vrijednosti dok hladne boje označavaju niske vrijednosti rezultata. Kako parametri `alpha` i `maxFails` konfigurišu u `testTransition` funkciji onda ćemo promatrati uticaj veze tih parametara na performanse algoritma, zatim ćemo zasebno promatrati uticaj `rho` parametra. Primjere ćemo predstaviti na kretanju trosegmentne planarne ruke iz primjera sa slike 33.

Tabela 8: Uticaj parametara **alpha** i **maxFails** na predeni put vrha trosegmentne planarne ruke.

		maxFails					
		2	5	10	20	35	70
alpha	2	8.67	8.23	10.46	8.4	8.03	7.2
	5	8.82	9.29	6.95	8.98	7.01	8.11
	10	9.75	10.19	9.12	8.64	8.24	7.49
	20	11.59	9.77	7.7	8.95	8.37	6.62
	50	9.61	8.04	8.51	8.78	8.39	8.78
	100	9.42	9.05	8.75	9.45	8.72	8.11

U tabeli 8 pratimo uticaj parametara na pređeni put vrha manipulatora. Vidimo da generalno za niske vrijednosti **maxFails** dužina puta raste jer dodajemo velik broj grana sa raznim nagibima što dovodi da putanja vijuga tj. nema jasan smjer da prati niskotežinske regije. Kako povećavamo ovaj paramtar vidimo da dobivamo kvalitetnije puteve što i očekivamo. Što se tiče parametra **alpha** vrijednosti do 20 daju generalno kvalitetnije puteve.

Tabela 9: Uticaj parametara **alpha** i **maxFails** na dužinu puta u konfiguracijskom prostoru trosegmentne planarne ruke.

		maxFails					
		2	5	10	20	35	70
alpha	2	581.09	587.96	754.71	690.09	648.67	616.24
	5	560.51	689.92	536.98	653.12	521.41	659.03
	10	651.58	664.65	659.46	656.12	685.92	614.57
	20	676.21	680.09	581.76	675.56	663.89	501.45
	50	525.75	554.46	611.46	660.18	605.6	665.83
	100	583.31	515.83	622.36	651.37	692.82	750.49

U tabeli 9 pratimo dužinu puta u konfiguracijskom prostoru. Iako često ovaj prametar nije presudan, on igra ulogu u količni zakretanja zglobova manipulatora jer za duže puteve motori u zglobovima moraju duže raditi i zglobovi rotiraju segment manipulatora duže vremena. Za male vrijednosti **maxFails** imamo kraće puteve u konfiguracijskom prostoru zbog iste činjenice da dopuštamo dodavanje velikog broja grana pozitivnih nagiba, što znači da lakše prevazilazimo prepreke ali time povećavamo težinu puta. Parametar **alpha** u ovom primjeru nije od primjetnog značaja.

Tabela 10: Uticaj parametara **alpha** i **maxFails** na prosječnu težinu puta na težinskoj mapi trosegmentne planarne ruke.

		maxFails					
		2	5	10	20	35	70
alpha	2	0.089	0.094	0.07	0.066	0.07	0.077
	5	0.115	0.079	0.095	0.082	0.092	0.067
	10	0.092	0.08	0.078	0.071	0.07	0.072
	20	0.09	0.089	0.088	0.077	0.079	0.106
	50	0.123	0.102	0.089	0.079	0.098	0.081
	100	0.112	0.119	0.089	0.081	0.077	0.066

U tabeli 10 pratimo prosječnu težinu puta na težinskoj mapi. Ovdje vidimo jasan uticaj parametara. Za male vrijednosti **maxFails** ne filtriramo dovoljno grana sa pozitivnim nagibom što dovodi do generiranja puteva sa visokom težinom. Kako se povećava ovaj parametar tako se smanjuje prosječna težina puta jer filtriramo grane sa pozitivnim nagibom. Za ovaj primjer bolje performanse dobivamo za vrijednosti parametra **alpha** ispod 10.

Tabela 11: Uticaj parametara **alpha** i **maxFails** na broj potrebnih iteracija T-RRT algoritma za trosegmentnu planarnu ruku.

		maxFails					
		2	5	10	20	35	70
alpha	2	2358	2646	3660	6230	6488	7071
	5	1200	1932	3416	2945	5948	7008
	10	1587	1620	1634	3104	6759	6818
	20	1339	1557	1648	3052	5031	4741
	50	915	2674	1983	3341	2577	3995
	100	1080	1342	2726	2846	6495	6249

Tabela 12: Uticaj parametara **alpha** i **maxFails** na vrijeme izvršavanja T-RRT algoritma za trosegmentnu planarnu ruku.

		maxFails					
		2	5	10	20	35	70
alpha	2	7.597	10.374	14.54	24.046	24.491	25.911
	5	4.937	7.797	13.329	11.365	22.652	26.861
	10	6.295	6.656	6.835	12.576	29.914	25.399
	20	5.43	6.016	6.922	12.128	19.171	17.02
	50	3.925	10.561	7.524	12.806	10.223	15.109
	100	5.235	5.635	10.095	11.638	25.115	23.211

Tabele 11 i 12 su direktno povezane jer kako raste broj iteracija tako i raste potrebno vrijeme da algoritam terminira. U primjerima ove dvije tabele vidimo najveći uticaj parametara. Obe tabele daju očekivane rezultate da će broj iteracija i vrijeme izvršavanja rasti kako povećavamo **maxFails** jer odbijamo više potencijalnih čvorova čime dobivamo na generiranju niskotežinskih puteva ali gubimo na brzini algoritma. Što se tiče parametra **alpha**, možemo zaključiti da sa njegovim blagim porastom ubrzavamo algoritam jer se temperatura može brže adaptirati.

Tabela 13: Uticaj parametra ρ na performanse T-RRT algoritma za trosegmentnu planarnu ruku.

ρ	Dužina puta (RP)	Dužina puta (KP)	Prosječna težina puta	Br. iteracija	Br. čvorova	Vrijeme [s]
0.001	9.434	695.629	0.075	3675	222	14.063
0.005	9.434	695.629	0.075	3675	222	13.912
0.01	8.630	643.291	0.080	3524	198	13.461
0.05	8.630	643.291	0.08	3525	199	12.810
0.1	8.630	643.291	0.08	3525	199	12.797
0.5	8.630	643.291	0.08	3525	199	12.770

Zanimljivo je da gotovo nikakvog uticaja nema promjena praga ρ na performanse T-RRT algoritma za trosegmentnu ruku. Odgovor na ovu pojavu možemo naći u težinskoj mapi konfiguracijskog prostora. Ako konfiguracijski prostor ima izražene niskotežinske regije onda, zbog samog svojstva širenja RRT stabla, većina novih čvorova koje dodajemo u stablo će biti periferalni čvorovi. Ono malo brdovite regije što imamo je većinom oblika kanjona tako da tu stablo također nema puno mjesta za usitnjavanje. Zbog ovakvog izgleda težinske mape, RRT stablo samo po sebi će imati veliki broj periferalnih čvorova čime prag ρ gubi na značaju. Tamo gdje dobiva na značaju je kada imamo *brdovitu* težinsku mapu kao onu sa slike 19 gdje imamo velik broj lokalnih minimuma i prevoja gdje vidimo punu moć praga ρ . Radi potpunosti analize parametara T-RRT algoritma pokazat ćemo u sljedećoj tabeli uticaj praga ρ na performanse na brdovitoj težinskoj mapi sa slike 18.

Tabela 14: Uticaj parametra ρ na performanse T-RRT algoritma za brdovitu težinsku mapu.

ρ	Dužina puta (KP)	Prosječna težina puta	Br. iteracija	Br. čvorova	Vrijeme [s]
0.001	112.838	0.206	9243	312	5.048
0.01	112.838	0.206	9243	314	5.148
0.05	120.212	0.196	16070	382	10.544
0.1	122.450	0.202	13622	402	8.402
0.5	116.623	0.179	19506	667	18.329
1	117.156	0.193	16569	1442	25.880

Tek sada vidimo punu moć praga ρ na brdovitoj težinskoj mapi. Dužina i težina puta ostaju manje-više konstantne. Vidimo da, kako povećavamo prag tako se i usporava algoritam jer dodajemo nove čvorove u stablo, a ne istražujemo nove regije težinske mape. Kao što smo već kazali, najbolje performanse imamo za interval $\rho \in [0.01, 0.1]$. Za $\rho > 0.5$ algoritam je znatno usporen i broj potrebnih iteracija, a samim time i vrijeme, mnogostruko raste.

9 Zaključak

Predstavili smo T-RRT algoritam zasnovan na uzorkovanju prostora koji interno radi na bazi RRT algoritma uz dodane modifikacije za kretanje po težinskoj mapi konfiguracijskog prostora. Pokazali smo konstrukciju prelaznih testova sa kojima upravljamo širenje stabla na težinskoj mapi ka niskotežinskim prostorima čime ispunjavamo kriterij koji smo prethodno zadali u obliku težinske funkcije. Vidjeli smo ogromni potencijal ove vrste algoritama koji rješavaju problem traženja putanje manipulatora u konfiguracijskom prostoru i takvo rješenje mapiraju u realni radni prostor manipulatora. Veliki dodatak ovakovom pristupu je što ne moramo znati inverznu kinematiku manipulatora već samo geometriju radnog prostora na osnovu koje upravljamo širenjem stabla u konfiguracijskom prostoru.

Predstavili smo veliki broj primjera kojim smo pokazali efikasnost i RRT algoritma i T-RRT algoritma. T-RRT algoritmom ispunjavamo predefinirani kriterij što smo pokazali u primjerima raznih manipulatora koji drže maksimalnu moguću udaljenost od prepreka u radnom prostoru što osigurava njihovu sigurnost. Promjena koja je morala biti uvedena radi nadogradnje RRT u T-RRT algoritam je bila minimalna - dodavanje dva uslova `testTransition` i `minExpandControl` u uslov dodavanja novih čvorova u stablo. Uz bilo kakvu modifikaciju koja doprinosi nečemu imamo i faktor usporenja algoritma što smo također vidjeli u primjerima gdje je RRT algoritam i za red veličine brži. Iako ovo vrijedi, RRT algoritam nije u mogućnosti ispuniti kriterij koji zadamo i dovodimo manipulator u potencijali sudsar sa preprekama u radnom prostoru. Ovo smo mogli potvrditi prosječnom težinom nađenog puta u rezultatima.

Simulacije koje smo prikazali su uključivali primjere nad dvosegmentnom i trosegmentnom planarnom rukom, antropomorfnom rukom, sfernom šakom i kombinacijom antropomorfne ruke i sferne šake. Također smo prikazali na slikoviti način uticaj `alpha`, `maxFails` i `rho` parametara na performanse T-RRT algoritma i odredili generalne intervale za odabir ovih parametara u zavisnosti zadatka koji želimo obaviti.

Zbog načina traženja putanja na težinskoj mapi T-RRT algoritma, njegovo korištenje se može u punom smislu generalizirati i na druge probleme pored one traženja putanje robotskih manipulatora. Dovoljno je samo definisati težinsku funkciju kojom želimo ispuniti kriterij i pustiti algoritam da nađe rješenje tako da u duhu ove opštosti bilo bi interesantno vidjeti rad ovog algoritma u problemima drugih naučnih disciplina.

10 Literatura

- [1] Jailllet, Léonard, Juan Cortés, and Thierry Siméon. “*Transition-based RRT for path planning in continuous cost spaces.*” 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, 2008.
URL: <https://hal.laas.fr/hal-01986342/document>
- [2] LaValle, Steven M. “*Rapidly-exploring random trees: A new tool for path planning*”. Computer Science Department, Iowa State University (TR 98-11), October 1998.
URL: <http://mssl.cs.uiuc.edu/~lavalle/papers/Lav98c.pdf>
- [3] Kavraki L. E., Svestka P., Latombe J.C., Overmars M. H. “*Probabilistic roadmaps for path planning in high-dimensional configuration spaces.*”. IEEE Transactions on Robotics and Automation, 1996.
URL: <http://dspace.library.uu.nl/handle/1874/17328>
- [4] Kuffner, James J., and Steven M. LaValle. “*RRT-connect: An efficient approach to single-query path planning.*”, Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065). Vol. 2. IEEE, 2000.
URL: http://kuffner.org/james/papers/kuffner_icra2000.pdf
- [5] LaValle, Steven M. “*Planning Algorithms*”. Cambridge University Press. ISBN 978-1-139-45517-6, May 2006.
- [6] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo. “*Robotics: Modelling, Planning and Control*”. Springer, London, UK, 2009.
- [7] Kroese D. P., Brereton T., Taimre T., Botev Z. I. “*Why the Monte Carlo method is so important today*”, 2014.
- [8] Jurić Ž. “*Diskretna matematika za studente tehničkih nauka*”. ETF Sarajevo, UNSA, 2017.
- [9] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. “*Introduction to Algorithms*”, Second Edition. MIT Press and McGraw-Hill. ISBN 0-262-03293-7. Section 33.3: *Finding the convex hull*, pp. 947–957, 2001.
- [10] John D’Errico (2020). Efficient test for points inside a convex hull in n dimensions. MATLAB Central File Exchange. Retrieved December 11, 2020.
URL: <https://www.mathworks.com/matlabcentral/fileexchange/10226-inhull>
- [11] Michael Yoshpe (2020). Distance from points to polyline or polygon. MATLAB Central File Exchange. Retrieved December 11, 2020.
URL: <https://www.mathworks.com/matlabcentral/fileexchange/12744-distance-from-points-to-polyline-or-polygon>

- [12] Matt J. (2020). Analyze N-dimensional Polyhedra in terms of Vertices or (In)Equalities.
MATLAB Central File Exchange. Retrieved December 10, 2020
URL: <https://www.mathworks.com/matlabcentral/fileexchange/30892-analyze-n-dimensional-polyhedra-in-terms-of-vertices-or-in-equalities>
- [13] lsqlin function description. Solve constrained linear least-squares problems.
URL: <https://www.mathworks.com/help/optim/ug/lsqlin.html>