Univerzitet u Beogradu
Matematički Fakultet

Navigacija i planiranje putanje autonomnog mobilnog robota

Master rad

Mentor:
prof. dr Miodrag Živković

Student:
Igor Petković
1022/2011

Komisija:
prof. dr Predrag Janičić
prof. dr Saša Malkov

Beograd 2018.

Sadržaj

Predgovor

Cilj ovog rada jeste predstavljanje algoritama za navigaciju i planiranje putanje autonomnih mobilnih robota. U prvom delu rada napravljen je pregled značajnih algoritama za navigaciju i planiranje putanje, dok se drugi deo bavi implementacijom nekih od njih i njihovim rezultatima u simuliranom okruženju.

Drugo i treće poglavlje fokusiraju se na tzv. "klasične" algoritme za navigaciju i planiranje putanje. Izbor opisanih algoritama za ova dva poglavlja načinjen je na osnovu knjige [1], koja je uvod u kompletnu oblast autonomnih mobilnih robota. Ta knjiga se pored samih algoritama navigacije bavi i drugim oblastima vezanim za autonomne mobilne robote, kao što su lokalizacija, percepcija, kinematika, itd.

U četvrtom poglavlju dat je opis inteligentnih kontrolnih sistema zasnovanih na neuronskih mrežama, fazi logici, genetskim algoritmima i tehnici optimizacije mravljom kolonijom, a poslednji deo rada bavi se implementacijom nekih od algoritama za navigaciju i planiranje putanje opisanih u prethodnim poglavljima.

1. Uvod

1.1 Pregled današnjeg stanja (mobilne) robotike

Robotika je do danas postigla odlične rezultate u svetu industrijske proizvodnje. Robotske ruke, ili manipulatori[1], sačinjavaju industriju vrednu dve milijarde dolara. Instalirana na specifičnoj poziciji na montažnoj traci robotska ruka može da se pomera velikom brzinom i sa visokom preciznošću može da obavlja ponavljajuće zadatke kao što su zavarivanje i farbanje. U elektronskoj industiriji manipulatori postavljaju izuzetno male komponente sa nadljudskom preciznošću, čineći mobilne telefone i laptop računare mogućim [1].

Ipak, pored svog tog uspeha, komercijalni roboti trpe nedostatak jedne ključne osobine: mobilnosti. Fiksirani manipulator ima ograničen opseg kretanja koji zavisi od toga gde je instaliran. U suprotnom, mobilni robot bi bio u mogućnosti da putuje kroz postrojenje, fleksibilno primenjivajući svoje veštine gde god je to najefikasnije [1].

Industrijske i tehničke aplikacije mobilnih robota neprestano dobijaju na značaju, pogotovo uzimajući u obzir pouzdanost, pristupačnost i cenu. Pouzdnost se ogleda u sposobnosti mobilnih robota da neprekidno i pouzdano izvrašavaju jednolične zadatke kao što je nadzor. Osobina pristupačnosti se može uvideti kod mobilnih robota koji vrše inspekciju oblasti koje nisu pristupačne čoveku, kao što su uski prostori, opasna okruženja ili udaljene oblasti. Što se tiče cene, u nekim zadacima mobilni roboti mogu biti jeftiniji izbor od standardnih sistema zasnovanih na trakama (*track-bound systems*). Mobilni roboti su već široko korišćeniji za nadzorne, inspekcijske i transportne zadatke [3]. Za aplikacije kao što su roboti za vođenje slepih ili mentalno hendikepiranih ljudi, koji čiste velike kancelarijeske zgrade ili robne kuće, koji pomažu ljudima u rekreacionim aktivnostima, i sl., prototipovi već postoje. [4].

Neki mobilni roboti operišu ne tamo gde ljudi ne mogu, već dele prostor sa njima u njihovim okruženjima. Ovakvi mobilni roboti su korisni ne samo zbog mobilnosti, već i zbog njihove autonomnosti, te su održavanje osećaja pozicije i mogućnost navigacije bez ljudske intervencije njihove glavne osobine [1].

Primer ovakvih robota, koji dele prostor sa ljudima, jesu *AGV* roboti (*autonomous guided vehicle* - autonomno vođeno vozilo) koji se mogu videti na slici 1.1. *AGV* roboti autonomno raznose delove između raznih stanica za montažu prateći specijalne električne žice za navođenje korišćenjem senzora. Helpmejt (*Helpmate*) servisni robot prenosi hranu i lekove kroz bolnice prateći poziciju sijalica na plafonu, koje su unapred manualno specifirane robotu (slika 1.2) [1].

Nekoliko kompanija su razvile autonomne robote za čišćenje velikih zgrada (slika 1.3). Jedan takav robot se koristi u pariskom metrou. Drugi roboti koji su specijalizovani za čišćenje iskorišćavaju pravilne geometrijske oblike hodnika supermarketa da bi olakšali lokalizaciju i zadatak navigacije [1]. Pored robota specijalizovanih za čišćenje velikih prostora, u poslednje vreme na tržištu su dosta popularni roboti koji su specijalizovani za čisćenje u domaćinstvima. Primer takvih robota su Rumba (*Roomba*) roboti usisivači (slika 1.4) [5].

Pored industrijske primene, mobilni roboti su široko korišćeni i u svrhe istraživanja. Istraživački mobilni roboti su razvijani da se prilagode laboratorijskim okruženjima i uglavnom su korišćeni u istraživanju spoznaje, lokalizacije i navigacije. Ovo je trenutno jedno od najvećih tržišta za mobilne robote. Razne mobilne robotske platforme su dostupne za programiranje i uglavnom variraju u veličini i mogućnosti terena. Najpopularniji roboti za istraživanje su roboti proizvođača poput *AcitvMedia Robotics*, *K-Team SA*, i *I-Robot* (slike 1.5, 1.6, 1.7) kao i veoma mali roboti kao što je Alis (*Alice*) iz Švajcarskog federalnog instituta za tehnologije u Lozani [1].

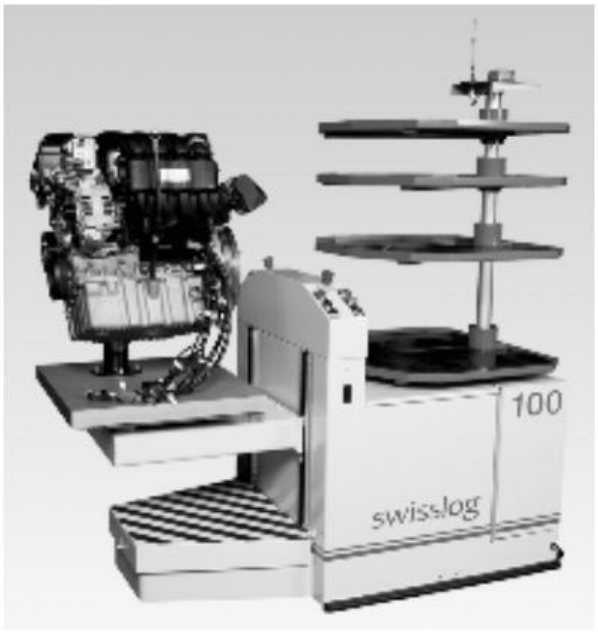
Mobilnost robota je besmislena bez mogućnosti navigacije. Nasumično kretanje, koje ne zahteva mogućnost navigacije, može biti korisno za određene nadzorne ili operacije čišćenja, ali za većinu naučnih i industrijskih aplikacija neophdno je da robot poseduje mogućnost kretanja u zahtevanom maniru.

Fokus ovog rada je upravo na metodama koje omogućavaju uređeno kretanje robota, odnosno na navigaciji. U prvom delu rada napravljen je pregled značajnih metoda navigacije, dok se drugi deo bavi implementacijom nekih od njih i njihovim rezultatima u simuliranom okruženju.

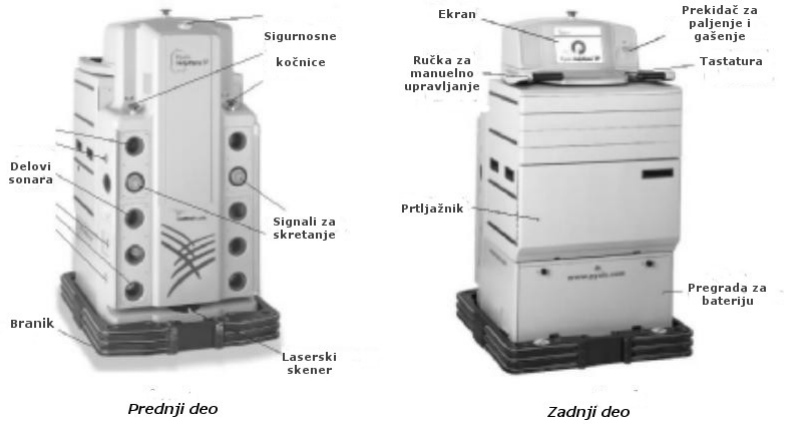
1.2 Navigacija i planiranje putanje

U slučaju mobilnih robota, specifičan aspekt kognitivnosti koji je direktno povezan sa robusnom mobilnošću jeste sposobnost navigacije. Za dato parcijalno znanje o svom okruženju i ciljnu poziciju ili skup pozicija, navigacija obuhvata sposobnost robota da se ponaša na osnovu svojih znanja i senzorskih vrednosti kako bi došao do ciljne pozicije na što efikasniji i što pouzdaniji način [1].

[1] Robot manipulator je mehanizam upravljan elektronskim putem, uglavnom sačinjen od više segmenata, koji izvršava razne zadatke u interakciji sa svojim okruženjem [2].



Slika 1.1: *SWISSLOG*-ov *AVG* robot korišćen za transport blokova motora od jedne montažne stanice do druge. Vođen je elektrčinom žicom ugrađenom u pod [1].



Slika 1.2: Helpmejt (*Helpmate*) robot korišćen za zadatke transporta u bolnicama. Ovaj robot ima ugrađene razne senzore koje koristi za autonomnu navigaciju kroz hodnike. Glavni senzor za lokalizaciju je kamera koja gleda u plafon i može da detektuje lampice, koje koristi kao smernice pri kretanju [1].

Planiranje kretanja autonomnih mobilnih robota je veoma kompleksno jer uključuje nekoliko zadataka. Za proizvoljnu lokaciju i cilj, robot mora da pronađe geometrijsku putanju do datog cilja. Problem postaje još komplikovaniji ukoliko robot mora da pronađe optimalnu putanju za neka data ograničenja ili objektnu funkciju kao što je najkraća putanja ili najkraće vreme. Takođe, robot može da naiđe na nekoliko prepreka, statičkih ili dinamičkih, pa je stoga zaobilaženje prepreka veoma bitno za mobilne robote [6]. Otuda je od autonomnog robota koji se kreće do ciljne tačke u nekom datom okruženju neophodno da planira optimalnu ili ostvarljivu putanju obilazeći prepreke odgovarajući na neka data ograničenja [7].

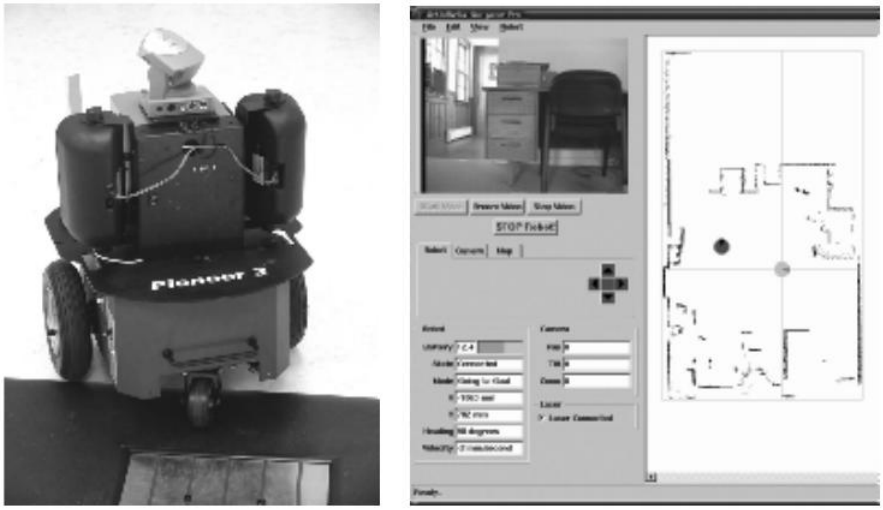
Planiranje putanje jedan je od najvitalnijih zadataka navigacije autonomnog mobilnog robota[8]. Za datu mapu i poziciju cilja, planiranje putanje uključuje identifikovanje putanje kretanja koja će učiniti da robot postigne ciljnu poziciju [1]. Planiranje putanje mobilnih robota se može podeliti na lokalno i globalno planiranje putanje. Lokalno planiranje putanje je zasnovano na senzorskim informacijama u nepoznatom okruženju gde su veličine, oblici i lokacije prepreka nepoznate, dok je globalno planiranje putanje zasnovano na unapred poznatim informacijama o kompletnom okruženju [7].

Lokalno ili reaktivno planiranje nema potrebu za apriori informacijama o okruženju. Mobilni robot reaguje na detektovane prepreke i menja pravac u realnom vremenu da bi ih izbegao. Ovi tipovi metoda

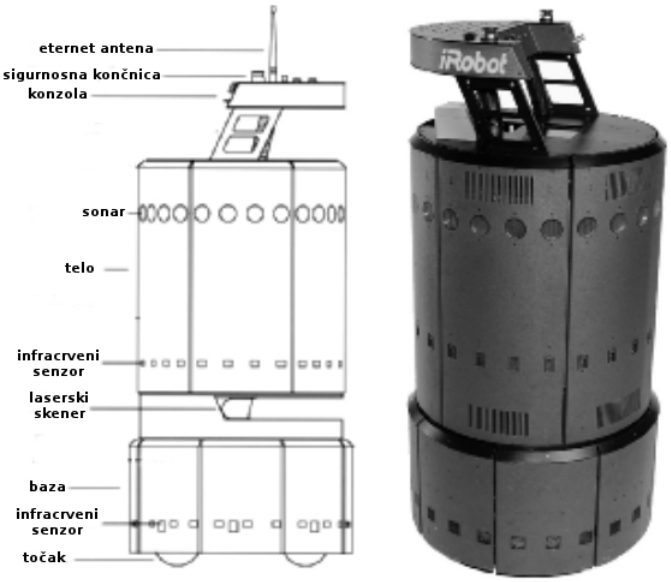
Slika 1.3: *BR 700* industrijski robot za čišćenje(levo) i *RoboCleaner RC 3000* potrošački robot koje je razvila *Alfred Kärcher Gmbh & Co.*, Nemačka. Navigacioni sistem *BR 700* je baziran na veoma sofisticiranom sonarnom sistemu i žiroskopu. *RoboCleaner RC 3000* prelazi preko uprljanih površina specijalnom strategijom vožnje sve dok one ne budu stvarno čiste [1].

(a) (b)

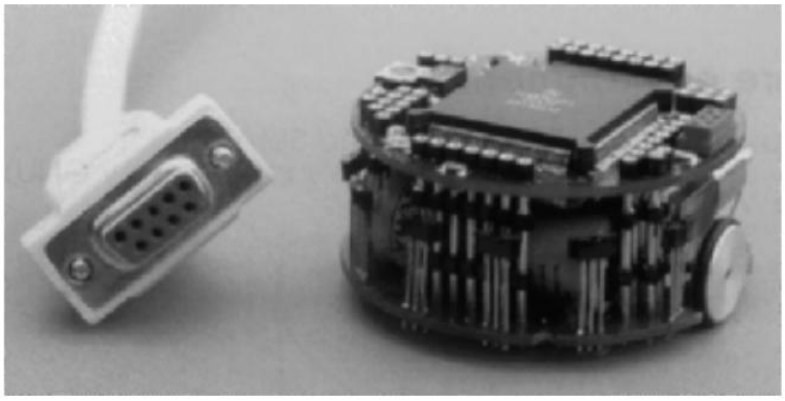
Slika 1.4: *Roomba* robot za čišćenje, proizvod *iRobot* kompanije [5]



Slika 1.5: *PIONEER* je modularni robot koji nudi razne opcije kao što su hvataljka ili ugrađena kamera. Opremljen je sofisticiranom bibliotekom za navigaciju koji su razvili *SRI, SA, Stanford* [1]



Slika 1.6: *B21* mobilni robot proizvod kompanije *iRobot*. Sadrži veliki broj različitih senzora koji mu omogućavaju izvršavanje zadataka navigacije sa visokim performansama[1]



Slika 1.7: *KHEPERA* je mali robot, kojeg je razvio *K-Team, SA*, Švajcakrska, korišćen za svrhe edukacije i iztraživanja [1].

navigacije su veoma pogodni za dinamička okruženja. Međutim, reaktivno planirananje je neefikasno kada je cilj veoma daleko i kada je okruženje "pretrpano" [8].

Globalno planiranje putanje uglavnom uključuje strategije koje generišu putanje bez opasnosti, bazirane na poznatoj mapi okruženja, kako bi se robot doveo do pre-definisane destinacije. U slučaju neočekivanih prepreka koje blokiraju unapred planiranu putanju, trebalo bi sprovesti replaniranje putanje bazirane na trenutnom ukruženju. Ovo rezultira u sporijem odzivu na neočekivane prepreke i takođe povećava cenu izračunavanja, pogotovo ako su prepreke dinamične [8].

1.3 Konfiguracioni prostor

Planiranje putanje za manipulator robote, kao i za većinu mobilnih robota, formalno se izvršava u reprezentaciji zvanoj konfiguracioni prostor. Pretpostavimo da robotska ruka ima k stepena slobode[2] [1].

Svako stanje ili konfiguracija robota može biti opisana sa nekih k realnih vrednosti: $q_1 \dots q_k$. k vrednosti se mogu shvatiti kao tačka p u k-dimenzionom prostoru zvanom konfiguracioni prostor C robota. Ovaj opis je pogodan jer omogućava da se kompleksni 3-D oblici robota opišu sa jednom k-dimenzionom tačkom [1].

[2]Stepeni slobode opisuje sve pokrete u prostoru koje robot može da izvede, odnosno opisuju njegovu slobodu kretanja u prostoru [9].

Neka se robotska ruka kreće u okruženju gde radni prostor (workspace), to jest njegov fizički prostor, sadrži poznate prepreke. Cilj planiranja putanje je da nađe putanju u fizičkom prostoru od inicijalne pozicije do ciljne, izbegavanjem svih kolizija sa preprekama. Ovaj problem je teško vizualizovati i rešiti u fizičkom prostoru, pogotovo ako je k mnogo veliko. Međutim, u konfguracionom prostoru ovaj problem je jednostavan. Ako defnišemo kofiguracioni prostor prepreka O kao podskup od C gde robotska ruka udara u nešto, onda slobodan prostor gde se robot može sigurno kretati računamo sa $F = C - O$ [1].

2. Lokalno planiranje (zaobilaženje prepreka)

Lokalno planiranje putanje ili zaobilaženje prepreka, se fokusira na oblikovanje robotove putanje na osnovu informacija dobijenih od senzora prilikom samog kretanja robota [10]. Zaobilaženje prepreka je najvažniji zadatak autonomne kontrole u navigaciji robota, pogotovo u potpuno nepoznatom okruženju jer igra ključnu ulogu u planiranju sigurne putanje[6]. Kod ovakvog planiranja, okruženje je potpuno nepoznato robotu, to jest ono je dinamičko i nestruktuirano ili prepreke nisu unapred poznate. U takvoj situaciji robot treba da sakupi informacije o okruženju u realnom vremenu i ažurira njegove zakone kontrole kako bi postigao cilj [11]. Algoritam za ove svrhe mora da bude dovoljno efikasan, tako da može da donese brzu odluku prilikom nailaska na prepreku bez ljudske intervencije [6].

Najosnovniji algoritmi detektuju prepreku i stopiraju robota u cilju izbegevanja kolizije, a nakon toga vode robota oko prepreke. Napredniji algoritmi, pored same detekcije, vrše i analizu vezanu za dimenzije prepreka. Nakon ove analize, algoritam za zaobilaženje prepreka vodi robota oko prepreke i nastavlja kretanje robota ka originalnom cilju [12].

2.1 Bag algoritmi

Bag (*Bug*) algoritmi su jedni od najranijih i najjednostavnijih algoritama za zaobilaženje prepreka baziranih na senzorima sa dokazanim garancijama [13].

Cilj Bag algoritama jeste da vode robota od startne tačke S to destinacije T, a da on nema nikakvo znanje o okruženju. Robot bi trebalo da postigne ovaj cilj sa što manje globalnih informacija koliko je to moguće. U praktičnim terminima ovo znači da robot može da pamti nekoliko tačaka od interesa, ali ne može, na primer, da radi mapiranje. Ako takva putanja ne postoji, algoritam se zaustavlja i prijavljuje da cilj nije moguće postići. Ovo se zove objektivna terminacija [14].

Ovi algoritmi se mogu isprogramirati na svakom robotu sa dodirnim ili daljinskim senzorom i metodom lokalizacije kao sto je GPS, odometer, itd. Takođe, robot ne mora da gradi mapu, samo mora da čuva jednu tačku da bi terminacija bila zagarantovana. Ovo čini Bag algoritme veoma pogodnim za implementaciju u realnom vremenu [14].

Model Bug algoritma pravi tri jednostavne pretpostavke o robotu. Prvo, robot se posmatra kao tačka. Ovo znači da robot nema nikakvu veličinu i da može da stane u prazninu bilo kojih dimenzija. Ova pretpostavka prevazilazi problem postajanja praznine na mapi za koju je robot previše veliki. Drugo, robot ima savršenu mogućnost lokalizacije. Ovo znači da robot zna njegovu tačnu poziciju i orijentaciju relativnu u odnosu na početnu tačku u bilo kom trenutku. Ova pretpostavka dozvoljava da robot određuje precizna rastojanja i azimut ka cilju, što je veoma važno za garantovano zaustavljanje i za stizanje do cilja ako je to moguće. Treće, robot ima savršene senzore. U određenim algoritmima, robot zahteva od daljinskih senzora da pomognu pri navigaciji. Ovi algoritmi se oslanjaju značajno na senzorske podatke i senzori sa velikom nepreciznošću mogu negativno uticati na performanse [14].

Očigledno je da su ove pretpostavke nerealistične za prave robote i da se bag algoritmi ne mogu direktno primeniti za navigacione zadatke pravih robota, ali se mogu uzeti u obzir kao nadzorne komponente na visokom nivou sistema koji pripaja sve tri pretpostavke [14].

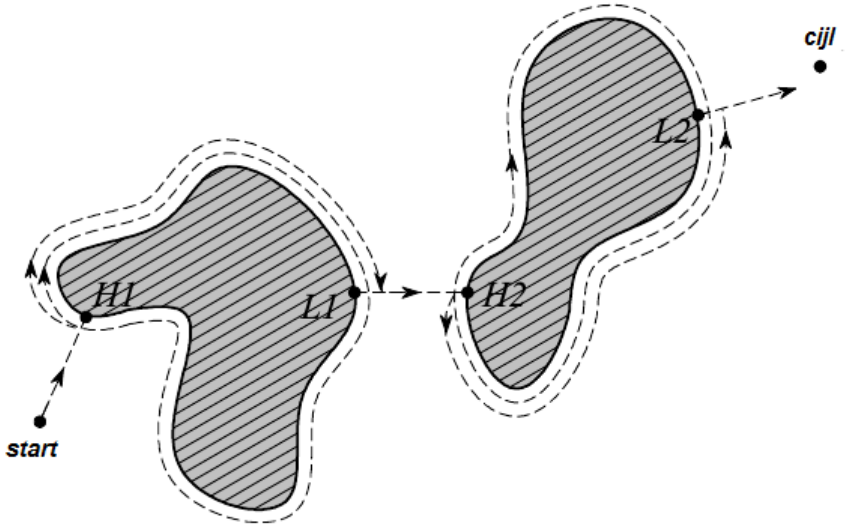
2.1.1 Bag1 algoritam

Bag1 (*Bug1*) algoritam je prvi algoritam iz familje Bag algoritama koji su razvili Lumkaski i Stepanov [14].

Sa Bag1, robot prvo u potpunosti obilazi objekat, a zatim odstupa od tačke sa najkraćim rastojanjem prema cilju (Slika 2.1). Ovaj pristup je, naravno, veoma neefikasan, ali garantuje da će robot postići bilo kakav dostupan cilj [1].

U suštini, Bag1 algoritam formalizuje "zdravorazumsku" ideju kretanja ka cilju i obilaženja prepreka. Pretpostavlja se da je robot tačka koja je savršeno pozicionirana (bez pozicione greške) sa kontakt senzorom koji može da detektuje ivice prepreka ako je tačka-robot đodiruje". Robot može meriti rastojanje $d(x, y)$ između bilo koje dve tačke x i y. Na kraju, pretpostavlja se da je prostor u kome robot može da se kreće ograničen. Neka $B_r(x)$ označava loptu poluprečnika r centriranog na x, odnosno $B_r(x) = \{y \in \mathbb{R}^2 | d(x, y) < r\}$. Činjenica da je prostor u kojem robot ima mogućnost kretanja ograničen podrazumeva da za sve $x \in W$ postoji $W \subset Br(x)$ [13].

Start i cilj su označeni sa q_{start} i q_{goal} respektivno. Neka je $q_0^L = q_{start}$ i neka je m-linija duž koja povezuje q_i^L do q_{goal}. Inicijalno $i = 0$, pa je na početku izvršavanja m-linija duž $\overline{q_0^L q_{goal}}$. Bug1 algoritam ima dva režima rada: kretanje do cilja i praćenje ivica. Tokom kretanja ka cilju, robot se kreće duž m-linije prema q_{goal}, dok ne naiđe na cilj ili prepreku. Ukoliko robot naiđe na prepreku, pamti se tačka q_1^H na



Slika 2.1: Bug1 algoritam sa $H1$, $H2$, tačkama udara, i $L1$, $L2$ tačkama napuštanja prepreke [1]

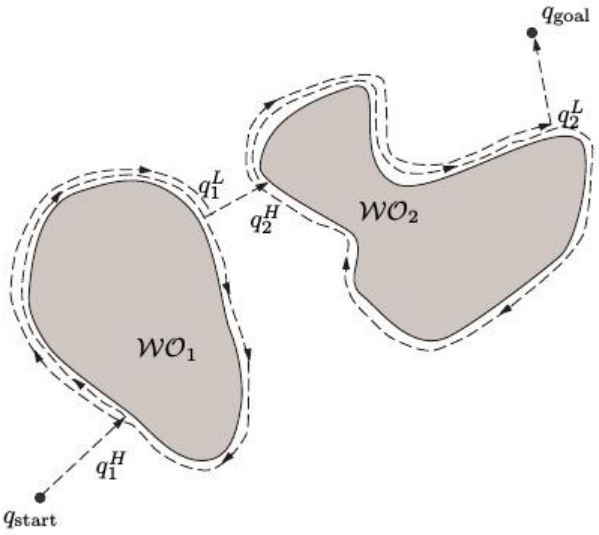
kojoj je robot prvi put naišao na nju. Ova tačka se naziva udarnom tačkom (*hit point*). Zatim, robot zaobilazi prepreku dok se ne vrati do q_1^H. Potom, robot određuje najbližu tačku do cilja na putanji kojom je zaobišao prepreku i prelazi do nje. Ova tačka se zove tačka napuštanja (*leave point*) i obeležena je sa q_1^L. Od q_1^L, robot se ponovo kreće pravo prema cilju. Ako linija koja povezuje q_1^L do cilja preseca prepreku koju je robot upravo napustio, onda algoritam ne može da pronađe putanju do cilja. Treba napomenuti da će se u ovakvoj situaciji, kada je cilj nedostižan, presek sa već zaobiđenom preprekom desiti neposredno nakon napuštanja q_1^L. U suprotnom se indeks i samo uvećava, a ovaj postupak se onda ponavlja za q_i^L i q_i^H dok se ne dođe do cilja ili dok planer ne utvrdi da robot ne može da stigne do cilja (slike 2.2, 2.3 ilustruju ova dva moguća ishoda). Na kraju, ukoliko linija do cilja samo dodiruje prepreku, robot ne mora da pređe u režim praćenja ivice, već može da nastavi kretanje napred ka cilju. Kôd Algoritam 1 opisuje algoritam Bag1 [13].

Algoritam 1 Bag1 algoritam [13]

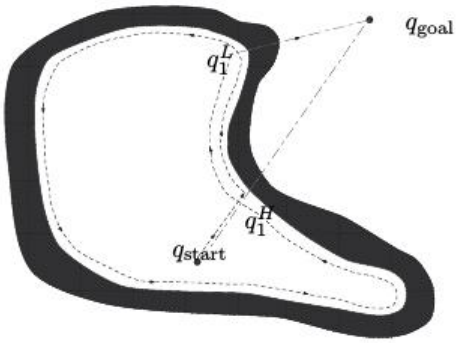
Data: Robot sa senzorom kontakta, početna tačka q_{start}, i krajnja tačka q_{goal}
Result: Putanja do q_{goal} ili zaključak da takva putanja ne postoji
$q_0^L := q_{start}$;
$i := 1$;
while *true* **do**
  repeat
    Od tačke q_{i-1}^L kreći se ka q_{goal};
  until *stiglo se do* q_{goal} ***or*** *naišlo se na prepreku u tački* q_i^H;
  if *stiglo se do* q_{goal} **then**
    Završi;
  repeat
    Prati ivice prepreke;
  until *stiglo se do* q_{goal} ***or*** *naišlo se ponovo na* q_i^H;
  Odredi tačku q_i^L na obodu prepreke koja ima najmanje rastojanje do cilja;
  Idi do q_i^L;
  if *ponovo naišlo na prethodno zaobiđenu prepreku* **then**
    Zaključi da se do q_{goal} ne može stići i završi;
end

2.1.2 Bag2 algoritam

Kao i Bag1 njegov srodnik Bag2(*Bug2*) algoritam ima dva režima rada: kretanje do cilja i praćenje granice prepreke. Tokom kretanja do cilja, robot se kreće ka cilju po m-liniji. Međutim, u Bag2 m-linija je duž koja povezuje q_{start} i q_{goal}, i tako ostaje fiksna. U režim rada praćenja ivica robot prelazi kada naiđe na prepreku. Režim rada praćenja ivica Bag2 algoritma se razlikuje od režima praćenja ivica Bag1 algoritma. Za Bag2 robot obilazi prepreku dok ne dostigne novu tačku na m-liniji bliže cilju od početne



Slika 2.2: Bag1 uspešno pronalazi cilj [13]



Slika 2.3: Bag1 prijavljuje da je cilj nedostižan [13]

tačke kontakta sa preprekom. U ovom trenutku, robot nastavlja ka cilju, ponavljajući ovaj proces ako naiđe na prepreku. Ukoliko robot ponovo naiđe na originalnu tačku napuštanja iz m-linije, onda nema puta do cilja (slike 2.4, 2.5). Neka je $x \in W_{free} \subset \mathbb{R}^2$ trenutni položaj robota, a $i = 1$, i q_0^L startna lokacija. Videti algoritam 2 za opis Bag2 pristupa [13].

Na prvi pogled, čini se da je Bug2 efikasniji algoritam nego Bug1 jer robot ne mora u potpunosti da zaobiđe prepreke. Međutim, to nije uvek slučaj, što se može videti poređenjem dužine putanja ova dva algoritma. Kod Bag1, kada se naiđe na i-tu prepreku, robot potpuno obilazi ivicu, a zatim se vraća na tačku napuštanja. U najgorem slučaju, robot mora preći pola perimetra, p_i, prepreke do tačke napuštanja [13].

Štaviše, u najgorem slučaju, robot nailazi na sve n prepreke. Ukoliko nema prepreka, robot mora preći rastojanje dužine $d(q_{start}, q_{goal})$. Dakle, dobija se:

$$L_{Bug_1} \leq d(q_{start}, q_{goal}) + 1.5 \sum_{i=1}^{n} p_i. \tag{2.1}$$

Za Bag2, dužina puta je malo komplikovanija. Neka linija kroz q_{start} i q_{goal} preseca i-tu prepreku n_i puta. Tu onda postoji najviše n_i tačaka napuštanja za ovu prepreku, jer robot može napustiti prepreku samo kada se vraća u tačku na ovoj liniji. Lako je uvideti da polovina ovih tačaka preseka nisu validne tačke napuštanja, jer one leže na "pogrešnoj strani" prepreke, odnosno, kretanje ka cilju bi izazvalo sudar. U najgorem slučaju, robot će preći skoro ceo perimetar prepreke za svaku tačku napuštanja. Dakle, dobija se:

$$L_{Bug_2} \leq d(q_{start}, q_{goal}) + \frac{1}{2} \sum_{i=1}^{n} n_i p_i. \tag{2.2}$$

Naravno, (2.2) je gornja granica - jer suma prelazi preko svih prepreka, nasuprot prelasku preko skupa prepreka na koje robot nailazi [13].

Letimični pregled (2.1) i (2.2), pokazuje da L_{Bug_2} može biti proizvoljno duže od L_{Bug_1}. To se može postići konstruisanjem prepreke čija granica ima mnogo preseka sa m-linijom. Tako, dok "složenost" prepreke raste, postaje sve verovatnije da Bag1 može nadmašiti Bag2 (Slika 2.5) [13].

U stvari, Bag 1 i Bag2 ilustruju dva osnovna pristupa za pretragu problema. Za svaku prepreku sa kojom se susreće, Bag1 vrši iscrpnu pretragu kako bi pronašao optimalnu tačku napuštanja. To zahteva da Bag1 prolazi ceo perimetar prepreka, ali pošto to uradi, izvesno je da je pronašao optimalnu tačku napuštanja. Suprotno, Bag2 koristi oportunistički pristup. Kad Bag2 pronađe tačku odsustva koja je bolja od svih koje je video ranije, on se posvećuje toj tački odsustva. Takav algoritam se naziva i "pohlepni", jer se opredeljuje za prvu obećavajuću opciju koja je pronađena. Kada su prepreke jednostavne, pohlepni pristup Bag2 daje brzi rezultat, ali kada su prepreke složene, konzervativniji pristup Bag1 često daje bolje performanse [13].

Algoritam 2 Bag2 algoritam [13]

Data: Robot sa senzorom kontakta, početna tačka q_{start}, i krajna tačka q_{goal}
Result: Putanja do q_{goal} ili zaključak da takva putanja ne postoji
$q_0^L := q_{start}$;
$i := 1$;
while *true* **do**
  repeat
    Od tačke q_{i-1}^L kreći se ka q_{goal};
  until *stiglo se do* q_{goal} ***or*** *naišlo se na prepreku u tački* q_i^H;
  Skreni levo (ili desno);
  repeat
    Od tačke q_{i-1}^L kreći se ka q_{goal};
  until *stiglo se do* q_{goal} ***or***
  ponovo se naišlo na q_i^H ***or***
  ponovo se naišlo na m-liniju u tački m *tako da je* $m \neq q_i^H$ *(robot nije došao do tačke udara),* $d(m, q_{goal}) < d(m, q_i^H)$ *(robot je bliže), i ako se robot kreće ka cilju, neće udariti u prepreku*;
  if *stiglo se do* q_{goal} **then**
    Završi;
  if *ponovo se naišlo na* q_i^H **then**
    Zaključi da se do q_{goal} ne može stići i završi;
  $q_{i+1}^H := m$;
  $i := i + 1$;
end

2.1.3 Tangentni Bag

Tangentni Bag(*Tangent Bug*) jedan je od brojnih proširenja Bag algoritma i služi kao poboljšanje Bug2 algoritma tako što određuje kraći put do cilja, koristeći senzor opsega od 360 stepeni rezolucije beskonačne orijentacije (azimuta) [1, 13].

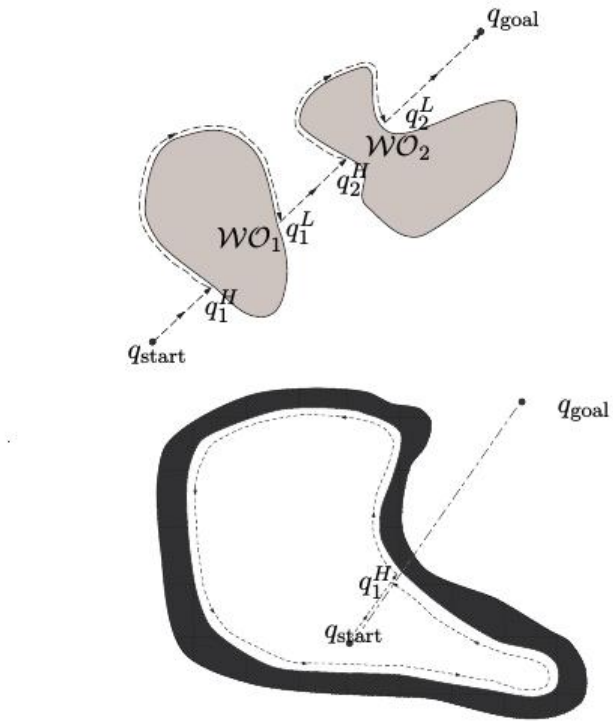
Senzor opsega se može modelirati sirovom funkcijom rastojanja $\rho : \mathbb{R}^2 \times \mathrm{S}^1 \to \mathbb{R}$. Posmatrajmo robota, predstavljenog kao tačka, koji se nalazi na $x \in \mathbb{R}$ sa zracima koji radijalno potiču od njega. Za svako $\theta \in S^1$, vrednost $\rho(x, \theta)$ je udaljenost do najbliže prepreke duž zraka iz x pod uglom θ. Izraženo formulom:

$$\rho(x,\theta) = \min_{\lambda \in [0,\infty]} d(x, x + \lambda[\cos\theta, \sin\theta]^T), \quad \text{takvo da} \quad x + \lambda[\cos\theta, \sin\theta]^T \in \bigcup_i WO_i \tag{2.3}$$

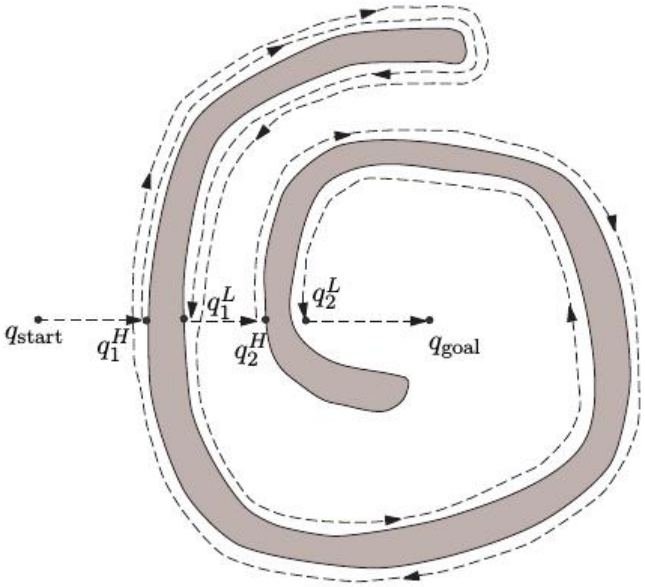
Može se primetiti da postoji beskonačno mnogo $\theta \in S^1$, a samim tim beskonačno mnogo rezolucija. Ova pretpostavka je usklađena sa ograničenim brojem dometnih senzora smeštenih duž obima kružnog mobilnog robota koji je oblikovan kao tačka [13].

Pošto pravi senzori imaju ograničen domet, definiše se zasićena sirova funkcija rastojanja, označena sa $\rho_R : \mathbb{R}^2 \times \mathrm{S}^1 \to \mathbb{R}$, koja preuzima iste vrednosti kao ρ kada je prepreka u dometu senzora i ima vrednost beskonačnosti kada je dužina zraka veća od opsega detekcije R što znači da su prepreke izvan opsega senzora. Izraženo formulom:

$$\rho_R(x,\theta) = \begin{cases} \rho(x,\theta) & \rho_R(x,\theta) < R \\ \infty, & \text{inače.} \end{cases} \tag{2.4}$$



Slika 2.4: (Gore)Bag2 uspešno pronalazi cilj. (Dole) Bug2 prijavljuje neuspeh [13].



Slika 2.5: Bag2 algoritam [13]

Skup tačaka u okviru raspona robotovog senzora se može označiti sa:

$$V_R(x) = \{y \in Q_{free} | d(x,y) < R \text{ i } \lambda x + (1-\lambda)y \in W_{free} \text{ za sve } \lambda \in [0,1]\} \tag{2.5}$$

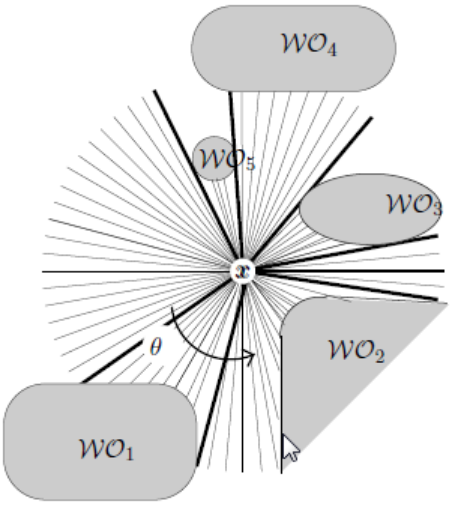
Planer Tangentnog Baga pretpostavlja da robot može da otkrije diskontinuitete u ρ_R kao što je prikazano na slici 2.6. Za fiksnu $x \in \mathbb{R}^2$, interval kontinuiteta je definisan kao povezan skup tačaka na $\partial V_R(x)$ gde je ρ_R konačano i varira kontinualno. Ovo su tačke $x + \rho(x,\theta)[\cos\theta, \sin\theta]^T$ na granici slobodnog prostora gde je ρ_R konačno i kontinualno u odnosu na θ [13].

Krajnje tačke ovih intervala javljaju se tamo gde $\rho_R(x,\theta)$ gubi kontinuitet, bilo kao rezultat toga što jedna od prepreka blokira drugu ili senzor dostiže svoju granicu opsega. Završne tačke su označene sa

O_i. Slika 2.7 sadrži primer gde ρ_R gubi kontinuitet. Tačke O_1, O_2, O_3, O_4, O_5, O_6, O_7 i O_8 odgovaraju gubicima kontinuiteta povezanih sa preprekama koje blokiraju druge delove W_{free}. Obratite pažnju na to da su ovde zraci tangente prepreka. Tačka O_5 je diskontinuitet, jer granica prepreke pada van dometa senzora. Skupovi tačaka na granici slobodnog prostora između O_1 i O_2, O_3 i O_4, O_5 i O_6, O_7 i O_8 su intervali kontinuiteta [13].

Baš kao i drugi Bag algoritmi, Tangentni Bag ima dva režima rada: kretanje do cilja i praćenje ivica. Međutim, režimi rada Tangentnog Baga se razlikuje od Bag1 i Bag2 pristupa. Iako kretanje do cilja usmerava robota do cilja, ovakvo ponašanje može da ima fazu u kojoj robot prati ivicu. Isto tako, režim rada kretanja po ivici može da ima fazu u kojoj robot ne prati ivicu [13].

Robot je na početku u režimu rada kretanja ka cilju, koji sam po sebi ima dva dela. Prvo, robot se kreće po pravoj liniji ka cilju dok ne oseti prepreku R nekoliko jedinica udaljenu i direktno između njega i cilja. To znači da duž koja povezuje robota i cilj mora da preseca interval kontinuiteta. Na primer, na slici 2.8 WO_2 je u dometu senzora, ali ne blokira cilj, ali WO_1 blokira. Kada robot prvobitno vidi prepreku, krug poluprečnika R postaje tangentan na prepreku. Neposredno nakon toga, ova tačka tangente se deli na dve O_i tačke, koje su krajnje tačke intervala. Ukoliko je prepreka ispred robota, onda njen interval preseca duž koja povezuje robota i cilj [13].



Slika 2.6: Tanke linije su vrednosti sirove funkcije rastojanja $\rho_R(x, \theta)$, za fiksirano $x \in \mathbb{R}^2$, dok debele linije predstavljaju diskontinuitet, koji se javlja ili zbog prepreke na putu ili zato što je opseg senzora dostignut. Treba još napomenuti da segmenti prekinuti u slobodnom prostoru predstavljaju beskonačne zrake [13]

Neka je O_i tačka takva da važi $d(O_i, q_{goal}) < d(x, q_{goal})$. Robot se zatim kreće prema jednoj od O_i koja maksimalno smanjuje heurističko rastojanje do cilja. Primer heurističkog rastojanja je zbir $d(x, O_i) + d(O_i, q_{goal})$. Heurističko rastojanje može da bude komplikovanije kada faktoriše u raspoloživim informacijama u vezi sa preprekama [13].

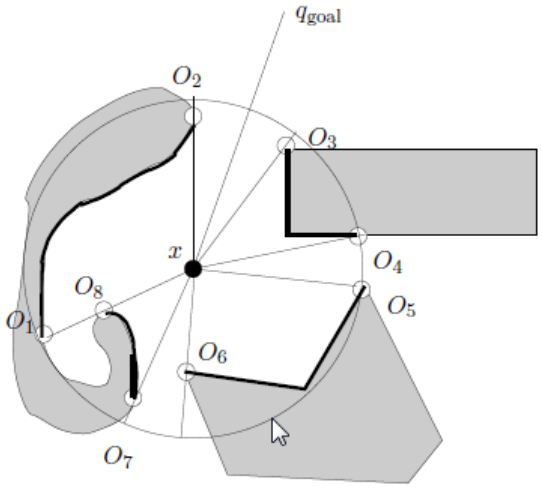
Na slici 2.9 (levo), robot vidi WO_1 i ide ka WO_2 jer $i = 2$ minimizira $d(x, O_i) + d(O_i, q_{goal})$. Kada se robot nalazi u x, on ne može da zna da WO_2 blokira put od O_2 do cilja. Na slici 2.9 (desno), kada se robot nalazi na x, ali je cilj drugačiji, ima dovoljno informacija senzora da zaključi da WO_2 zaista blokira put od O_2 do cilja, pa samim tim se kreće ka O_4. Dakle, iako kretanje prema O_2 u početku može minimizirati $d(x, O_i) + d(O_i, q_{goal})$ više od kretanja ka O_4, planer efektivno dodeljuje beskonačnu cenu na $d(O_2, q_{goal})$ jer ima dovoljno informacija da zaključi da će svaki put kroz O_2 biti suboptimalan [13].

Skup $\{O_i\}$ se stalno ažurira dok se robot kreće ka određenom O_i, koji se može videti na slici 2.10. Kada je $t = 1$, robot nije osetio prepreku, pa otuda kreće prema cilju, kada je $t = 2$, robot prvobitno registruje prepreku označenu debelom punom linijom. Robot nastavlja da se kreće ka cilju, ali sa strane prepreke idući prema diskontinuitetu u ρ. Za $t = 3$ i $t = 4$, robot registruje više prepreka i nastavlja da smanjuje razdaljinu do cilja, dok obgrljava ivicu [13].

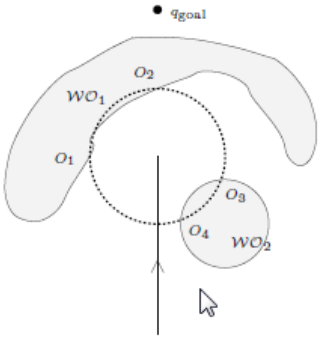
Robot je u režimu rada kretanja do cilja dok se više ne može smanjiti heurističko rastojanje do cilja. Drugim rečima, nalazi tačku koja je kao lokalni minimum $d(\cdot, O_i) + d(O_i, q_{goal})$ ograničena na putanju koje kretanje do cilja diktira [13].

Kada robot pređe u režim kretanja po ivici nailazi na tačku M na registrovanom delu prepreke koja ima najkraće rastojanje od prepreke do cilja. Može se primetiti da ako je opseg senzora nula, onda je M ista kao udarna tačka Bag1 i Bag2 algoritama. Ova registrovana prepreka se zove i praćena prepreka. Pravi se razlika između praćene prepreke i prepreke koja blokira [13].

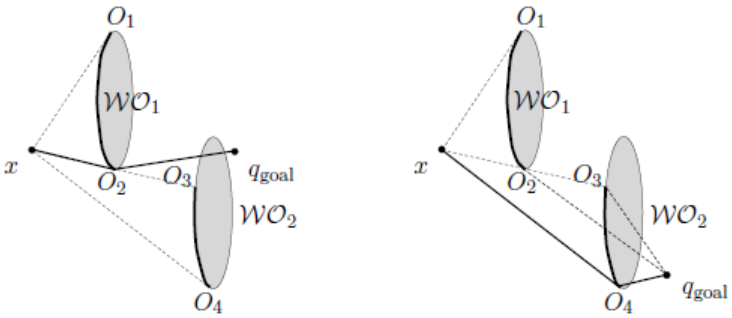
Drugum rečima, neka je x trenutni položaj robota. Prepreka koja blokira je najbliža prepreka unutar dometa senzora koju preseca duž $(1 - \lambda)x + \lambda q_{goal}, \forall \lambda \in [0, 1]$. U početku su blokirna prepreka i praćena



Slika 2.7: Tačke diskontinuiteta $\rho_R(x, \theta)$ odgovaraju tačkama O_i na preprekama. Debele pune krive predstavljaju povezane komponente opsega $\rho_R(x, \theta)$, to jest, intervale kontinuiteta. U ovom primeru robot veruje da postoji prava liniji do cilja [13].



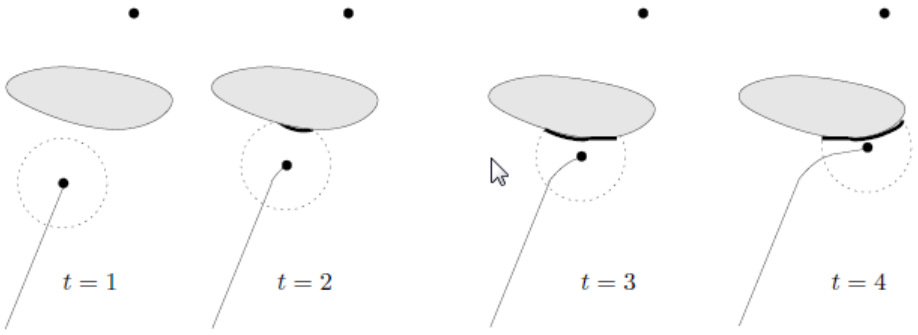
Slika 2.8: Vertikala predstavlja putanju robota, dok isprekidani krug predstavlja opseg njegovog senzora. Trenutno, robot je lociran na "vrhu" duži. Tačke O_i predstavljaju tačke diskontinuiteta zasićene sirove funkcije rastojanja. Primetimo da je robot zaobišao WO_2 [13]



Slika 2.9: (Levo) planer odabira O_2 kao podcilj robota. (Desno) planer odabira O_4 kao podcilj robota. Može se primetiti da duž od O_4 do q_{goal} prolazi kroz prepreku[13]

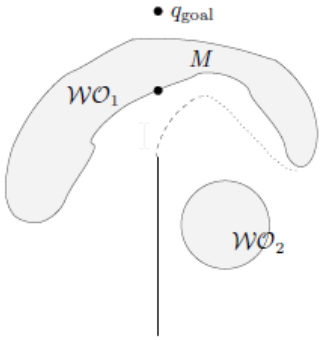
prepreka iste prepreke [13].

Sada se robot kreće u istom pravcu kao da je u režimu kretanja do cilja. Kontinuirano se kreće ka O i na praćenoj prepreci u odabranom smeru (Slika 2.11). Dok preuzima ovaj pokret, planer takođe ažurira dve vrednosti $d_{followed}$ i d_{reach}. Vrednost $d_{followed}$ je najkraće rastojanje između ivice koju je registrovao i cilja. Vrednost d_{reach} je najkraće rastojanje između cilja i bilo koje tačke u okruženju gde je robot lociran. Kada nema blokirne prepreke, neka T bude tačka oko koje je centriran krug na x sa poluprečnikom R koji



Slika 2.10: Demonstracija ponašanja kretanja ka cilju robota sa ograničenim opsegom senzora koji se kreće ka cilju koji se nalazi iznad svetlo sive prepreke [13].

preseca duž koja povezuje x i q_{goal}. Ovo je tačka na periferiji senzornog opsega koja je najbliža cilju, kada se robot nalazi na x. Dakle, kada nema prepreke $d_{reach} = d(T, q_{goal})$. U suprotnom, T je nedefinisana i d_{reach} se stalno ažurira da bude na najkraćoj udaljenosti između cilja i bilo koje tačke koja blokira ivicu prepreke. Kada je $d_{reach} < d_{followed}$, robot prekida režim praćenja ivice [13].



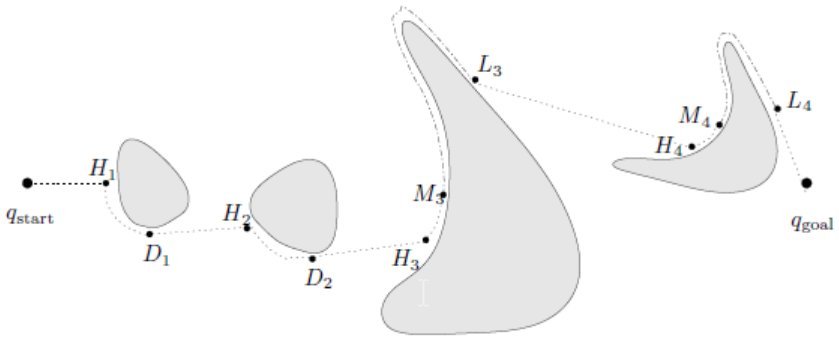
Slika 2.11: Okruženje je isto kao i na slici 2.8. Puna linija predstavlja putanju generisanu kretanjem ka cilju dok isprekidani segment predstavlja praćenje ivica prepreke. Primetimo da je M tačka "lokalnog minimuma"[13].

Slika 2.12 sadrži putanju robota sa nultim dometom senzora. Ovde je robot u režimu kretanja do cilja dok ne susretne prvu prepreku na udarnoj tački H_1. Za razliku od Bag1 i Bag2, nailaženje na udarnu tačku ne menja režim rada robota. Robot nastavlja u režimu kretanja do cilja skretanjem udesno i praćenjem ivica prve prepreke kretanjem do cilja. Robot je skrenuo desno, jer taj pravac minimizira svoju heurističku udaljenost do cilja. Robot prolazi ovu granicu na tački napuštanja (*depart point*) D_1. Robot nastavlja u režimu kretanja do cilja manevrisanjem oko druge prepreke dok ne naiđe na treću prepreku H_3. Tada robot skreće levo i nastavlja u režimu kretanja do cilja dok ne dostigne M_3, minimalnu tačku. Sada, planer prebacuje robota u režim kretanja po ivici, dok ne dostigne L_3. Treba navesti da zbog toga što robot ima nulti opseg senzora, d_{reach} predstavlja rastojanje između robota i cilja. Postupak se nastavlja sve dok robot ne dođe do cilja. Samo na M_i i L_i robot menja režim rada. Slike 2.13, 2.14 sadrže primere gde robot ima konačne i beskonačne opsege očitavanja, respektivno [13].

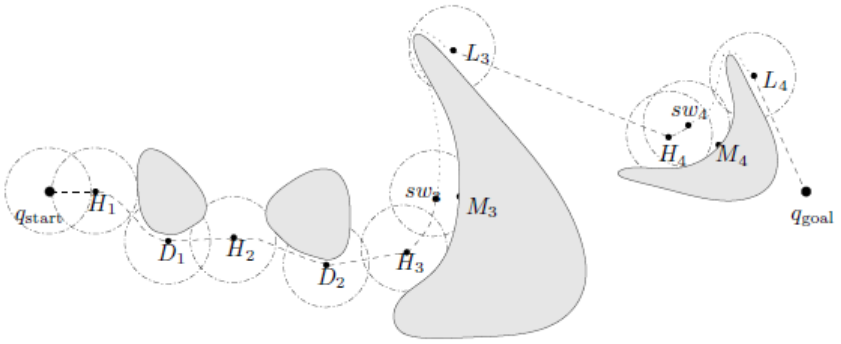
2.2 Histogram vektorskog polja (HVP)

Johan Borenstajn je, zajedno sa Joran Korenom, 1991. godine razvio Histogram vektorskog polja (HVP, eng. *Vector Field Histogram, VFH*) [1, 12]. Originalni HVP je bio zasnovan na Virtualnom polju sila (*Virtual Force Field*), njihovom ranije razvijenom algoritmu za lokalno planiranje putanje. Posle njegove kreacije 1991. godine, 1998. Ivan Urlih i Borenstajn su unapredili HVP i preimenovali ga u HVP+(*VHF+*)[15]. Algoritam su 2000. Urlih i Borenstajn ponovo unapredili i preimenovali u HVP*(*VHF**) [16].

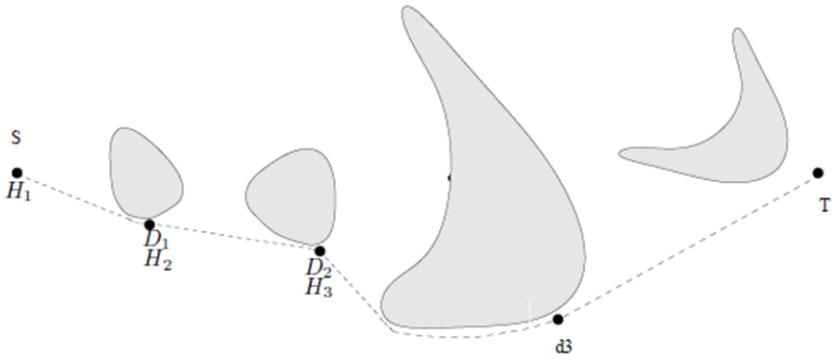
Jedna od glavnih kritika Bag algoritama je da ponašanje robota u svakoj instanci zavisi uskljućivo od najskorije učitanih podataka sa senzora. To može da dovede do nepoželjnih ali sprečivih problema u slučajevima kada trenutna očitavanja senzora robota ne pružaju dovoljno informacija za robusno izbegavanje prepreka [1].



Slika 2.12: Putanja generisana Tangetnim Bagom sa senzorom nultog opsega. Isprekidana linija predstavlja ponašanje kretanja ka cilju, dok tačkasta linija predstavlja praćenje ivica [13].



Slika 2.13: Putanja generisana Tangetnim Bagom sa senzorom konačnog opsega. Isprekidana linija predstavlja ponašanje kretanja ka cilju, dok tačkasta linija predstavlja praćenje ivica. Crta-tačka krugovi predstavljaju opseg robotovog senzora [13].



Slika 2.14: Putanja generisana Tangetnim Bagom sa senzorom beskonačnog opsega. Isprekidana linija predstavlja ponašanje kretanja ka cilju, dok praćena ivica u ovom slučaju nema [13].

HVP prevazilazi ovo ograničenje stvaranjem lokalnih mapa okruženja oko robota [1]. Ova loklana mapa se naziva aktivni prozor (*active window*) i konstantni je podskup globalne mape čitavog okruženja. Globalna mapa je dvodimenziolna mreža sačinjena od ćelija koje u sebi sadrže vrednost pouzdanosti (*certainty value*), vrednost koja predstavlja meru pouzdanosti da prepreka stvarno postoji u oblasti te ćelije. Ova mapa se naziva histogramska mreža (*histogram grid*) i predstavlja model sveta. Model sveta se neprestano ažurira podacima učitanim sa senzora.

Algortiam na podatke iz aktivnog prozora primenjuje redukciju iz dva koraka radi izračunavanja narednog koraka kretanja [12, 17, 18, 19].

Prvi korak redukcije podatka prodrazumava mapiranje podataka aktivnog prozora dimenzija $ws \times ws$ na polarni histogram H, i to na sledeći način:

Sadržaj svake ćelije aktivnog prozora se posmatra kao vektor prepreke, čiji se pravac β određuje na

osnovu pravca koji se proteže od ćejile do centralne tačke vozila (*Vehicle Center Point*):

$$\beta_{i,j} = \tan^{-1} \frac{(y_0 - y_j)}{(x_i - x_0)} \tag{2.6}$$

gde su x_0 i y_0 trenutne koordinate centralne tačke vozila, a x_i i y_j koordinate aktivne ćelije $c_{i,j}$. Intenzitet vektora aktivne ćelije se dobija na sledeći način:

$$m_{i,j} = c*^2_{i,j} \cdot (a - b * d_{i,j}) \tag{2.7}$$

gde je $c*_{i,j}$ vrednost pouzdanosti aktivne ćelije $c_{i,j}$, $d_{i,j}$ rastojanje od aktivne ćelije $c_{i,j}$ do centralne tačke vozila, gde su a i b pozitivne konstante.
Iz 2.7 se može primetiti da je:

a) $c*_{i,j}$ kvadrirana što pokazuje da ponavljajuće senzorske vrednosti stvarno predstavljaju prereku, za raziliku od vrednosti koji se samo jednom pojavljuju i koje mogu da izazovu šum.

b) $m_{i,j}$ je proporcionalno sa $-d$, što pokazuje da okupirane ćelije daju vektore sa velikim intenzitetom ako se nalaze u neposrednoj blizini robota, a vektore sa malim intezitetom ako se nalaze znatno dalje od robota. Posebno, a i b su uzete tako da je $a - b \cdot d_{max} = 0$, gde je $d_{max} = \sqrt[2]{\frac{(ws-1)}{2}}$ rastojanje između najdalje aktivne ćelije i centralne tačke robota. Na ovaj način za najdalju aktivnu ćeliju vrednost $m_{i,j}$ je nula i povećava se linearno za bliže ćelije.

Histogram H ima proizvoljnu ugaonu rezoluciju α takvu da je $n = 360/\alpha$ ceo broj (npr. za $\alpha = 50$ $n = 72$). Svaki sektor odgovara diskretnom uglu ρ, takvom da je $\rho = k\alpha$, gde je $k = 0, 1, 2, ..., n - 1$. Povezanost između $c*_{i,j}$ i sektora k je omogućena kroz:

$$k = \int \frac{\beta_{i,j}}{\alpha} \tag{2.8}$$

Za svaki sektor k, polarna gustina prepreke h_k se računa na sledeći način:

$$h_k = \sum_{i,j} m_{i,j} \tag{2.9}$$

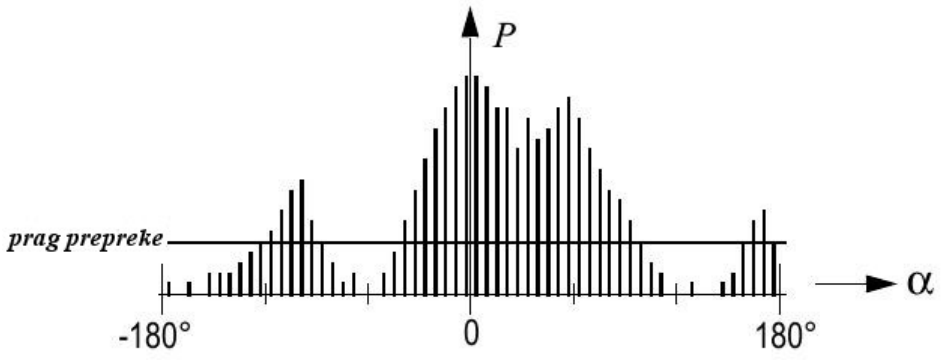
Svaka aktivna ćelija je povezana sa određenim sektorom na osnovu jednačina 2.6 i 2.8.

Zbog diskretne prirode histogramske mreže, rezultat mapiranja može da ispadne "hrapav" što može da dovede do grešaka u selekciji sledećeg pravca kretanja. Zbog toga se na histogram H primenjuje funkcija ublaživanja (*smoothing function*), definisana sa:

$$h_k^{'} = \frac{h_k - l + 2h_k - l + 1 + ... + lh_k + ... + 2h_k + l - 1 + h_k + l}{2l + 1} \tag{2.10}$$

gde je h_k^2 poravnata gustina prepreke, a l proizvoljna konstanta [12].

Nakon određivanja polarnih gustina sektora dobija se polarni histogram nalik na histogramu prikazanom na slici2.15 [12, 17, 18, 19].



Slika 2.15: Polarni histogram [1]

Iks-osa predstavlja ugao α na kome je prepreka pronađena, a ipsilon-osa predstavlja verovatnoću P da zaista postoji prepreka u tom pravcu [1]. U drugom koraku redukcije podataka algoritam računa pravac narednog kretanja θ. U ovom koraku algoritam prvo odabira sve potencijalne prolaze (*valleys*), delove histograma koji su dovoljno veliki da kroz njih robot može da prođe i čije su verovatnoće postojanja prepreke manje od unapred definisanog praga (*threshold*). Ovi prolazi se dalje klasifikuju na široke i uske na osnovu toga koliko uzastopnih sektora histograma sa verovatnoćom prepreke manjom od praga prepreke sadrže. Ako je broj takvih sektora veći ili jednak od S_{max} prolaz je širok, u suprotnom prolaz je uzak.

Parametar S_{max} se određuje na osnovu dimenzija i kinematičkih ograničenja robota. Kod prolaza koji su klasifikovani kao uski, robot ima samo jednu moguću putanju, pravo uz prepreku, u cilju održavanja prihvatljive distanace od okolnih prepreka. Kod širokih prolaza robot ima do 3 opcije kretanja: uz desnu granicu prolaza, uz levu granicu prolaza i ako se pravac ka cilju nalazi u prolazi, onda i kretanje u pravcu cilja postaje opcija. Od svih kandidata algoritam izabira onaj koji se najviše poklapa sa pravcem cilja k_{tar}. Kada je određeni prolaza izabran, neophodno je izabrati odgovarajući sektor koji se nalazi u tom prolazu. Sektor koji je najbliži k_{tar} i koji je ispod praga prepreke se označava sa k_n i predstavlja blisku granicu (*near border*) prolaza. Dalja granica (*far border*) se označava sa k_f i definiše kao $k_f = k_n + S_{max}$. Željeni pravac narednog kretanja θ se definiše kao $\theta = \frac{(k_n + k_f)}{2}$ [12, 17, 18, 19].

Kao što je gore napomenuto, jedan od kritrijuma za odabir kandidata prolaza jeste prag prepreke. Dok je odabir odgovarajućeg praga kritičan slučaj za mnoge sisteme bazirane na senzorima, uticaj dobro odabranog praga na performanse HVP-a je uglavnom krajnje neosetljiv. Situacije u kojima vrednost praga utiče na performanse sistema jesu sledeće:

a) Ako je prag previše visok, robot može previše blizu prići prepreci (pogotovo ako se kreće velikom brzinom) i sudariti se sa objektom.

b) Ako je prag previše nizak, neke od potencijalnih prolaza (uglavnom uskih) HVP neće uopšte uzeti u razmatranje.

Može se zaključiti da je dobro odabran prag potreban HVP sistemu samo za neke naprednije aplikacije (kada se robot kreće velikom brzinom i kada je okruženje gusto popunjeno preprekama); u manje zahtevnim uslovima algoritam se ponaša dobro i sa nepreciznim pragom. Jedan način da se optimizuju performanse jeste korišćenjem adaptivnog praga iz nekog višeg nivoa hijerarhije, na primer, funkcije globalnog plana. Tako je, tokom normalnog kretanja prag veoma sigurne, niske vrednosti. Ako globalni plan poziva na prolazak kroz uzak prolaz, prag se privremeno podiže i brzina se privremeno smanjuje [12].

2.2.1 HVP+

HVP+(*VFH+*) je unapređeni metod koji nudi nekoliko poboljšanja u odnosu na HPV koja rezultuju u glatkijim putanjama robota i većoj pouzdanosti [15].

HVP+ metod eksplicitno uzima u obzir širinu robota, tako da ovaj metod se može lako implementirati na robote različitih veličina. Ovo poboljšanje takođe eliminiše podešavanje niskofrekventong filtera (*low-pass filter*) koje je prisutno kod HVP metoda 2.10. Još jedno poboljšanje HPV+ metoda jesta bolje aproksimacija putanje robota, što rezultira u većoj pouzdanosti [15].

Koncept HPV+ algoritma je dosta sličan konceptu HVP algoritma. HVP+ metod uključuje redukciju podataka od četiri koraka u cilju izračunavanja novog pravca kretanja robota. U prva tri koraka, dvodimenzionalna histogramska mreža se redukuje u jednodimenzioni polarni histogram. U četvrtom koraku, algoritam izabira najbolji pravac na osnovu maskiranog polarnog histograma i funkcije cene [15].

Originalni HVP metod ne uzima u obzir eksplicitno širinu robota. Umesto širine uzima empirijski determinisan niskofrekventni filter da nadoknadi širinu robota i ublaži polarni histogram 2.10. Podešavanje ovog filtera je najkomplikovanija stvar kod implementacije originalnog HVP. Međutim, čak i sa dobro podešenim filterom, robot ima tendecije da preseca uglove [15].

HVP+ metod, za razliku od HVP, koristi teoretski određen niskofrekventni filter da nadoknadi širinu robota. Ćelije prepreka se uvećavaju za radijus robota r_r, koji je definisan kao rastojanje od centra robota do najdalje tačke perimetra. Za dalju sigurnost, ćelije prepreka se uvećavaju za radijus $r_{r+s} = r_r + d_s$ gde je d_s minimlano rastojanje između robota i prepreke [15].

Sa preprekama uvećanim za r_{r+s}, robot se može tretirati kao tačka. Ovaj metod radi dobro za mobilne robote čiji se oblik može aproksimirati diskom. Ako je robotov oblik previše asimetričan, ćelije prepreka će biti uvećane u odnosu na dimenzije trenutne orijentacije robota [15].

Ovaj metod nadoknade širine se implementira vrlo efikasno uvećanjem prepreka u toku izgradnje primarnog polarnog histograma. Umesto ažuriranja samo jednog sektora histograma za svaku ćeliju kao sto je rađeno u originalnom HVP, svi sektori histograma kome odgovaraju uvećene ćelije su ažurirani. Za svaku ćeliju, ugao uvećanja je $\gamma_{i,j}$ i definisan je sa:

$$\gamma_{i,j} = \arcsin \frac{r_{r+s}}{d_{i,j}} \tag{2.11}$$

Za svaki sektor k, polarna gustina prepreke se izračunava na sledeći način:

$$H_k^p = \sum_{i,j} m_{i,j} h'_{i,j} \tag{2.12}$$

gde je:

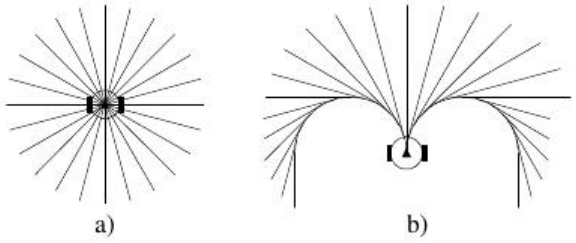
$$\begin{aligned} h'_{i,j} &= 1 \text{ ako } k * \alpha \in [\beta_{i,j} - \gamma_{i,j}, \beta_{i,j} + \gamma_{i,j}] \\ h'_{i,j} &= 0, \text{ inače} \end{aligned} \tag{2.13}$$

Rezultat ovog procesa je polarni histograma takav da uzima u obzir širinu robota. h' funkcija takođe služi kao niskofrekventni filter i ublažava polarni histogram. Još jedno važno poboljšanje jeste što ovaj proces eliminiše poteškoću podešavanja niskofrekventnog filtera HVP [15].

Za većinu aplikacija, poželjna je glatka putanja, dok oscilacije u komandi upravljanja bi trebalo izbegavati. Originalni HVP metod uglavnom prikazuje veoma glatku putanju. Međutim, fiksiran prag prepreke τ korišćen u HVP može dovesti do problema u okruženjima sa nekoliko uskih prolaza, tako što odgovarajući prolaz u histogramu može da alternira nekoliko puta između otvorenog i blokiranog stanja u toku par sempliranja rezultata senzora. U takvoj situaciji, pravac robota može da alternira nekoliko puta između ovog uskog prolaza i nekog drugog prolaza. Ovo rezultuje neodlučno ponašanje, tokom kojeg se robot može previše približiti prepreci. Problem se može rešiti histerezom zasnovanom na dva praga, τ_{low} i τ_{hig}. Na osnovu primarnog polarnog histograma H^p i dva praga, gradi se binarni polarni histogram H^b. Umesto prikazivanja vrednosti polarne gustine, sektori H^b su ili slobodni (0) ili blokirani (1). Ovaj polarni histogram prikazuje koji su pravci slobodni za robota koji trenutno može da promeni svoj pravac kretanja. Binarni polarni histogram se ažurira na osnovu sledećih pravila:

$$\begin{aligned} H^b_{k,i} &= 1 \quad \text{ako} \quad H^p_{k,i} > \tau_{hig} \\ H^b_{k,i} &= 0 \quad \text{ako} \quad H^p_{k,i} < \tau_{low} \\ H^b_{k,i} &= H^b_{k,i-1} \quad \text{inače} \end{aligned} \tag{2.14}$$

Originalni HVP ne uzima u obzir dinamiku i kinematiku robota, nego implicitno podrazumeva da je robot u mogućnosti da promeni svoj pravac kretanja u svakom trenutku kao što je prikazano na slici (Slika 2.16, model a). Osim ako se robot zaustavlja svaki put kada semplira rezulate senzora, ova pretpostavka je očigledno prekršena. HVP+ metod koristi jednostavniju, ali bližu aproksimaciju putanje kod većine mobilnih robota. HVP+ podrazumeva da je robotova putanja zasnovana na kružnim lukovima - konsantna zakrivljena krivina (*constant curvature curves*) i pravih linija (Slika 2.16, model b). Zakrivljenje krivine je definisano sa $\kappa = 1/r$ [15].



Slika 2.16: Aproksimacija putanja: a) bez dinamike b) sa dinamikom [15]

Maksimalno zakrivljenje putanje mobilnog robota je često funkcija robotove brzine. Što robot brže putuje, manje je maksimalno zakrivljenje. U nekim specijalnim slučajevima maksimalno zakrivljenje može biti različito za leva i desna skretanja [15].

Vrednosti minimalnog radijusa upravljanja (*minimum steering radius*) izraženog kao funkcija robotove brzine se lako mogu izmeriti. Minimalni radijus upravljanja za levu i desnu stranu se može definisati sa $r_r = 1/\kappa_r$ i $r_l = 1/\kappa_l$ [15].

Sa ovim parametrima i histogramskom mrežom, možemo odrediti koji sektori su blokirani preprekama. Primer sa dve prepreke je prikazan na slici 2.17. Kao što je već navedeno, potrebno je uzeti u obzir širinu robota, sto znači da su ćelije uvećane za r_{r+s}. Ako se krug putanje i prepreka poklapaju, svi pravci od prepreke do pravca kretanja unazad su blokirani. U primeru sa slike 2.17, prepreka A blokira sve pravce sa njene leve strane zbog dinamike robota. Sa druge strane, prepreka B ne blokira nijedan pravac sa desne strane [15].

Originalni HVP metod bi sve pravce levo od A uzeo u obzir kao pogodne putanje kretanja. Ako bi željeni pravac kretanja bio levo, originalni HVP algoritam bi naveo robota levo pravo u prepreku A [15].

Sa druge strane, HVP+ metod bi robota proveo korektno, između prepreka A i B i skrenuo levo posle zaobilaženja prepreke A [15].

Pozicije levog i desnog centra putanje relativne u odnosu na trenutnu poziciju robota definisane su sa:

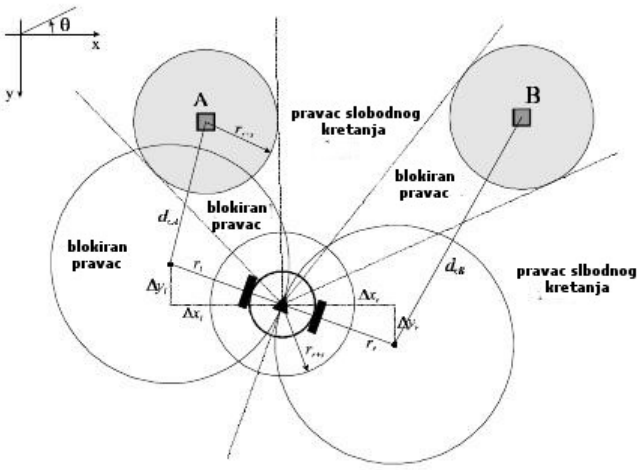
$$\begin{aligned} \Delta x_r &= r_r * \sin\theta \quad \Delta y_r = r_r * \cos\theta \\ \Delta x_l &= -r_l * \sin\theta \quad \Delta y_l = -r_r * \cos\theta \end{aligned} \tag{2.15}$$

Distance od aktivne celije $c_{i,j}$ do ta dva centra putanje data su sa:

$$\begin{aligned} d_r^2 &= (\Delta x_r - \Delta x(j))^2 + (\Delta y_r - \Delta y(i))^2 \\ d_l^2 &= (\Delta x_l - \Delta x(j))^2 + (\Delta y_l - \Delta y(i))^2 \end{aligned} \tag{2.16}$$

Prepreka blokira pravce sa njene desne strane ako:

$$d_r^2 < (r_r + r_{r+s}) \quad \text{[uslov 1]} \tag{uslov 1}$$



Slika 2.17: Primer blokiranih pravaca [15]

i prepreka blokira sve pravce sa njene leve strane ako:

$$d_l^2 < (r_l + r_{l+s}) \quad \text{[uslov 2]} \tag{uslov 2}$$

Proveravanjem svake aktivne ćelije sa ova dva uslova, dobijamo dva granična ugla, φ_r za desne uglove i φ_l za leve uglove. Takođe definišemo i $\varphi_b = \theta + \pi$ za pravac kretanja unazad od trenutnog pravca kretanja [15].

Ovaj metod se može implementirati veoma efikasno sledećim algoritmom koji uzima u obzir ćelije koje imaju uticaj na φ_r ili na φ_l:

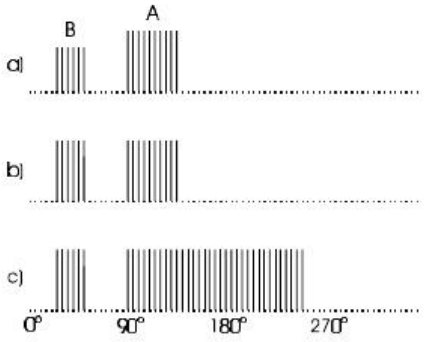
1. Odrediti φ_b. Postaviti φ_r i φ_l da budu jednaki φ_b.
2. Za svaku ćeliju $c_{i,j}$ iz aktovnog prozora sa $c^*_{i,j} > \tau$:
   (a) Ako je $\beta_{i,j}$ sa desne strane θ i sa leve strane φ_r, proveri uslov 1. Ako je uslov zadovoljen postavi φ_r na $\beta_{i,j}$
   (b) Ako je $\beta_{i,j}$ sa leve strane θ i sa desne strane φ_l, proveri uslov 2. Ako je uslov zadovoljen postavi φ_l na $\beta_{i,j}$

Sa vrednostima za φ_r i φ_l i binarnim polarnim histogramom, može se kreirati maskirani polarni histogram:

$$\begin{aligned} H_k^m &= 0 \quad \text{ako} \quad H_k^b \quad \text{i} \quad (k * \alpha) \in \{[\varphi_r, \theta], [\theta, \varphi_l]\} \\ H_k^m &= 0 \quad \text{inače} \end{aligned} \tag{2.18}$$

Maskirani polarni histogram pokazuje koji su svi pravci kretanja mogući sa trenutnom brzinom [15].

Ako su svi sektori blokirani, robot ne može da prođe trenutnom brzinom. Tada robot mora da odredi novi skup vrednosti φ_r i φ_l zasnovanih na sporijoj brzini. Ako je maskirani polarni histogram i dalje blokiran u svim pravcima, robot bi trebao da se u momentu zaustavi. Pomoću ovog maskiranog polarnog histograma se onda može odrediti da li je robot zaglavljen u ćorsokaku [15].



Slika 2.18: a) Primarni polarni histogram b) Binarni polarni histogram c) Maskirani polarni histogram [15]

Kao i HVP, HVP+ iz finalnog histograma pronalazi sve moguće kandidate (prolaze) za naredni korak kretanja. Za raziliku od HVP, koji vrši izbor kandidata na osnovu njegove blizine cilju, HVP+ vrši izbor kandidata tako što primenjuje funkciju cene (*cost function*). Funkcija cene uzima u obzir više nego samo razliku između kandidata pravca c i pravca cilja k_{tar}, koja se primenjuje na sve moguće kandidate novog pravca. Ova funkcija se definiše kao funkcija g kandidata pravca c, i ima sledeći oblik:

$$g(c) = \mu_1 \cdot \Delta(c, k_{tar}) + \mu_2 \cdot \Delta(c, \frac{\theta_i}{\alpha}) + \mu_3 \cdot \Delta(c, k_{n,i-1}) \tag{2.19}$$

gde je $\Delta(c_1, c_2)$ funkcija koja izračunava apsolutnu razliku ugla između dva sektora c_1 i c_2 , tako da je rezultat $\leq n/2$. Jedna od mogućih implementacija je:

$$\Delta(c_1, c_2) = \min\{|c_1 - c_2|, |c_1 - c_2 - n|, |c_1 - c_2 + n|\} \tag{2.20}$$

Prvi član funkcije cene predstavlja cenu vezanu za razliku kandidata pravca i pravca cilja. Što je veća razlika, više će ponuđeni pravac voditi robota dalje od pravca cilja, a samim tim će i trošak biti veći.

Drugi član predstavlja trošak vezan za razliku kandidata pravca i orijentacije točkova robota. Što je veća razlika, veća je potrebna promena pravca kretanja.

Treći član predstavlja troškove povezane sa razlikom kandidata pravca i prethodno izabranog pravca kretanja. Što je veća razlika, veća je promena nove upravljačke komande [15].

Ukratko, prvi član je odgovoran za ciljno orijentisano ponašanje, dok drugi i treći član čine da se mobilni robot posveti pravcu. Drugi i treći člana obezbeđuju robota sa nekim vidom kratkoročnog pamćenja. Drugi član je sličan mehaničkoj memoriji. Uz pomoć trećeg člana, robot se obavezuje na pravac čak i pre nego što mu je orijentacija promenjena [15].

Što je koeficijent μ_1 veći, kretanje robota je više ciljno orijentisano. Što je koeficijent μ_2 veći, robot više pokušava da izvrši efikasnu putanju sa minimalnom promenom pravca kretanja. Što je koeficijent μ_3 veći, robot više pokušava da krene prema prethodno izabranom pravcu, što rezultuje glađu putanju [15].

Samo odnos između ova tri parametra je važan, a ne njihove veličine. Kako bi se garantovalo ciljno orijentisano ponašanje, sledeći uslov mora biti zadovoljen:

$$\mu_1 > \mu_2 + \mu_3 \tag{2.21}$$

Ukoliko je bitnija efikasnost putanje od varijacija u komandama upravljanja, onda bi μ_2 trebao da bude veći od μ_3. Ako je glatkost komandi upravljanja bitnija od efikasnosti robotove putanje, onda bi μ_3 trebao da bude veći od μ_2 [15].

Takođe je moguće dodati i druge članove funkciji cene. Na primer, može se načiniti da mobilni robot izbegava uske prolaze dodajući član koji uzima u obzir širinu samog prolaza [15].

Funkcija cene omogućava korisniku da implementira suptilnije ponašanje od onog u prvobitnom pristupu HVP metoda. Funkcija cena je dosta bitna prilikom odbira pravog kandidata pravca, pogotovo u situacijama kada robot prilazi objektu koji se tačno nalazi na njegovom putu. Bez funkcije cene robot bi zapravo mogao da udari direktno u ovaj objekat "premišljajući" se da li da ga obiđe sa njegove leve ili desne strane. Još jedna prednost funkcije cene je u tome što se ponašanje mobilnog robota može lako izmeniti promenom parametra funkcije cene ili promenom same funkcije [15].

2.2.2 HVP*

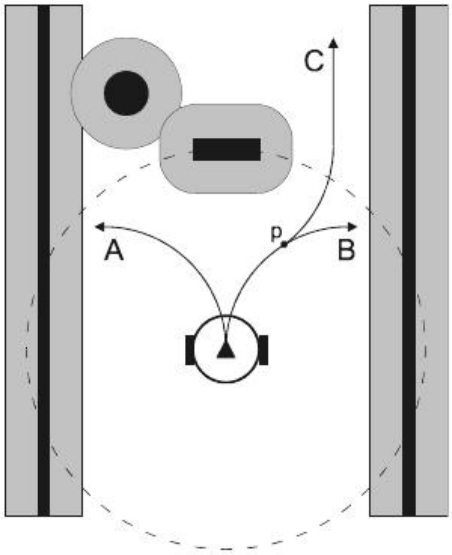
HVP+ ponekad, zbog svoje lokalne prirode, ne uspeva da pronađe odgovarajući pravac kretanja. Inherentan problem čistog lokalnog algoritma za zaobilaženje prepreka se najbolje može objasniti sa primerom. Slika 2.19 prikazuje situaciju gde se mobilni robot kreće kroz hodnik i nailazi na dve prepreke na njegovoj putanji. Prepreke su predstavljene crnom bojom, dok je konfiguracioni prostor predstavljen sivom. Iako HPV+ koristi koncept konfiguracionog prostora implicitno, konfiguracioni prostor je nacrtan eksplicitno zbog bolje vizualizacije [16].

Veliki krug nacrtan isprekidanom linijom predstavlja aproksimaciju rastojanja na kojem prepreka izaziva manevar zaobilaženja. Na poziciji prikazanoj u primeru, HVP+ detektuje dva prolaza. Dok prolaz sa leve strane rezultuje u nepoželjnoj putanji A, prolaz sa desne rezultuje u putanju B, koja u tački p eventualno prelazi u poželjnu putanju C. Nažalost, obe putanje A i B HVP+ posmatra kao jednako odgovarajuće. U problematičnim situacijama, kao što je ova, HVP+ bi izabrao odgovarajući pravac u proseku samo 50% vremena [16].

Takođe je bitno napomenuti da se isti problem očekuje i kod drugih čisto lokalnih algoritama za zaobilaženje prereka. Osnovni problem je što sistemi koji su čisto lokalni uzimaju u obzir samo neposredne efekte izabrane putanje bez provere njenih posledica [16].

HVP* (*VFH**) bi zaobilazio ovakve problematične situacije u većini vremena kombinovanjem HVP+ algoritma sa A* algoritmom pretrage. HVP* uspeva to tako što projektuje putanju robota u napred nekoliko koraka i izračunava posledice. Iako HVP* više nije čisto lokalan, još uvek je lokalan algoritam i kao takav može da radi u realnom vremenu [16].

Za razliku od HVP+, HVP* analizira posledice kretanja u napred za svakog primarnog kandidata pravca, pre finalnog izbora za novi pravac kretanja. Za svakog primarnog kandidata pravca, HVP* računa novu poziciju i orijentaciju koju bi robot imao posle kretanja za projektovani korak rastojanja d_s. Za



Slika 2.19: Problematična situacija za čisto lokalne algoritme zaobilaženja prepreka [16]

svaku projektovanu poziciju, HVP+ je ponovo upotrebljen za konstrukciju polarnog histograma na osnovu informacija iz mape. Analiziranjem ovog histograma se pronalaze kandidati za novi pravac, koji se nazivaju projektovani kandidati pravca (*projected candidate directions*). Ponavljanjem ovog procesa n_g puta, gradi se stablo pretrage sa dubinom n_g, gde krajnji čvorovi (ciljevi) odgovaraju celokupnoj projektovanoj putanji $d_t = n_g * d_s$ [16].

Cilj ovog procesa pretarge je da se nađe odgovarajuća projektovana putanja rastojanja d_t. Čvorovi u stablu pretrage predstavljaju projektovane pozicije i orijentacije mobilnog robota. Grane predstavljaju kandidate pravca koji vode od jedne pozicije do druge. Za svakog kandidata pravca c, cena $g(c)$ se računa slično kao funkcija cene koja je korišćena u HVP+ algoritmu. Cena asocirana sa jednim čvorom je jednostavno suma svih cena grana koje vode nazad do početnog čvora. Primarni kandidat pravca koji vodi do krajnjeg čvora sa najmanjom ukupnom cenom je izabran za novi pravac kretanja φ_d [16].

Za lakše razumevanje ovog koncepta, HVP* je u prethodnom pasusu opisan kao da je baziran na BFS algoritmu. U stvari, HVP* zapošljava A* metod pretrage i koristi heurističnu funkciju $h(c)$ koja je slična funkciji cene VFH+. Prioritet vrednosti za svaki čvor je onda definisan sa $f(c) = g(c) + h(c)$. S obzirom da je mnogo manje vremena potrebno za izračunavanje heurističke funkcije nego za proširivanje čvora, A* je mnogo brži od BFS za aplikaciju HVP*. Kao BFS, A* je optimalan i kompletan, zato što heuristična funkcija nikad ne precenjuje cenu postizanja stanja cilja [16].

HVP* je lokalni algoritam za zaobilaženje prepreka koji koristi verifikaciju gledanja unapred (*look ahead verification*), to jest uzima u obzir više nego samo trenutno robotovo okruženje. Dok HVP* ima iste performanse za zaobilaženje prepreka kao HVP+ za regularne prepreke, HVP* je sposoban da se suoči sa problematičnim sutacijama koji bi zahtevali od robota da znatno uspori ili čak da se zaustavi [16].

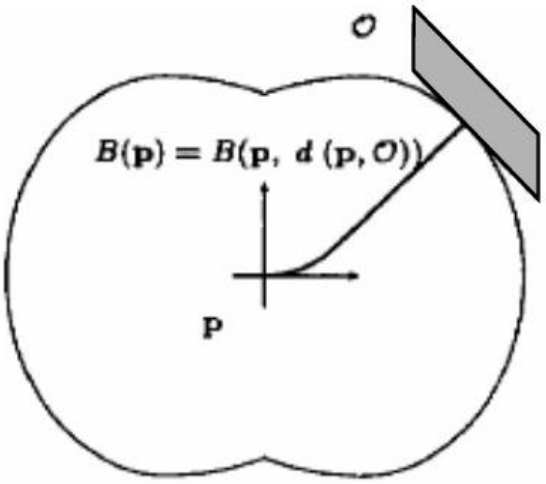
2.3 Tehnika trake mehura

Tehnika trake mehura je proširenje koncepta elastične trake (*elastic band concept*) koji su predložili O. Khatib i S. Kuinlan. Originalni koncept elastične trake primenjen je samo na holonomična vozila[1], dok je tehnika trake mehura, koju su predložili M. Khatiba, H. Jaounija, R. Šatila i J.P. Laumoda, proširila ovaj koncept i na neholomična vozila.

Mehur se definiše kao maksimalni lokalni podskup slobodnog prostora oko date konfiguracije robota kojim se može putovati u bilo kom pravcu bez sudara. Mehur se generiše korišćenjem pojednostavljenog modela robota zajedno sa informacijama koje su dostupne iz mape robota. Čak i na pojednostavljenom modelu geometrije robota, moguće je da se uzme u obzir i stvaran oblik robota prilikom izračunavanja veličine mehura (Slika 2.20) [1].

Imajući u vidu takve mehure, traka ili niz mehura može se koristiti zajedno sa putanjom od početne do ciljne pozicije robota da bi se pokazao očekivani slobodan prostor tokom puta robota (Slika 2.21). Jasno je, izračunavanje trake mehura zahteva globalnu mapu i globalni planer putanje [1].

[1]Holonomičnost predstavlja odnos između svih stepena slobode robota i stepena slobode koji se mogu kontrolisati. Ako je broj stepena slobode robota koji se mogu kontrolisati jednak ukupnom broju stepena slobode, onda se za takvog robota kaže da je holonomičan. U suprotnom, ako je broj stepena slobode koji mogu da se kontrolišu manji od ukupnog broja stepena slobode robot je neholonomičan [20].

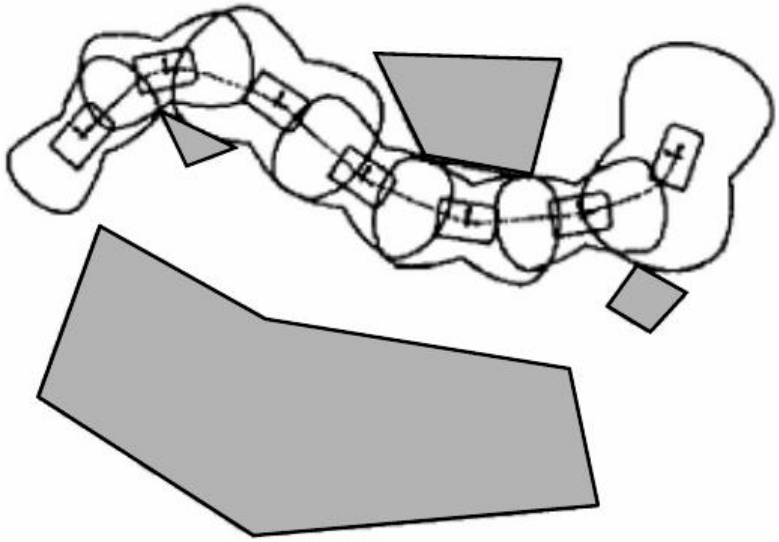


Slika 2.20: Oblik mehura oko vozila [1]

Nakon izračunavanja inicijalne putanje planera i trake mehura, sledi modifikacija planirane putanje. Traka mehura uzima u obzir sile modeliranih objekta i unutrašnje sile. Ove unutrašnje sile pokušavaju da minimiziraju "popuštanje" (energiju) između susednih mehurova [1].

Ovaj proces, plus konačna operacija ublažavanja, čini putanju glatkom u smislu da će se slobodan prostor robota promeniti glatko koliko je moguće tokom izvršenja putanje. Naravno, sada ovo više liči na optimizaciju puta nego zaobilaženje prepreka. Aspekt zaobilaženja prepreka strategije trake mehura ulazi u igru tokom kretanja robota [1].

Kako robot nailazi na nepredviđene vrednosti senzora, model trake mehura se koristi da skrene robota iz njegove, prvobitno namenjene, putanje na način koji minimizira napetost trake mehura. Prednost ove tehnike je da se može uzeti u obzir stvarna dimenzija robota. Međutim, metoda se najviše primenjuje samo kada je konfiguracija okruženja unapred dobro poznata [1].



Slika 2.21: Tipična traka mehura [1]

2.4 Metod brzina zakrivljenja

Metod brzina zakrivljenja (krivina) (*curvature velocity method*) ili MBZ(CVM) je algoritam za zaobilaženje prepreka koji je razvio Rejd Simons. Ovaj metod uzima u obzir stvarna kinematička i neka

dinamička ograničenja robota prilikom zaobilaženja prepreka, što je prednost u odnosu na neke primitivnije tehnike. MBZ počinje dodavanjem fizičkih ograničenja robota i okoline prostoru brzina. Prostor brzina se sastoji od rotacionih brzina t_r i translacionih brzina t_v, pod pretpostavkom da robot putuje samo lukovima krugova sa zakrivljenjem (krivinom) $c = t_r/t_v$ [1].

Identifikuju se dve vrste ograničenja: ona koja su izvedena iz ograničenja robotovog ubrazanja i brzine; i ograničenja nastala od prepreka koja zbog njihovih pozicija blokiraju određene t_r i t_v vrednosti [1].

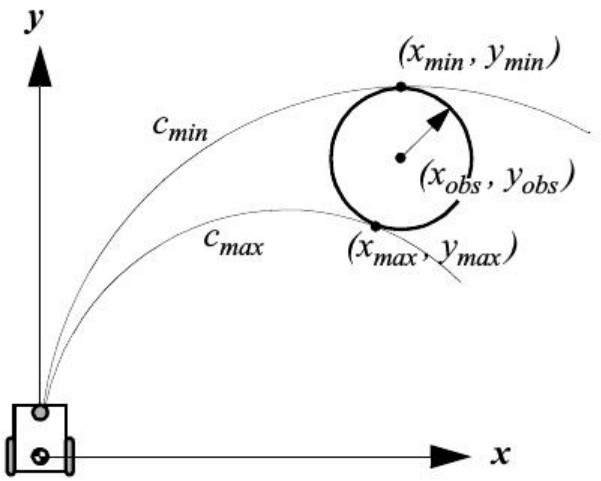
Ograničenja ubrzanja i brzine koja održavaju kretanje robota u granicama njegovih fizičkih osobina su sledeća:

$$\begin{aligned} &0 \leq t_v \leq t_{v_{max}}, -r_{v_{max}} \leq r_v \leq r_{v_{max}} \\ &r_v \geq r_{v_{cur}} - (r_{a_{max}} \cdot T_{accel}) \\ &r_v \geq r_{v_{cur}} - (r_{a_{max}} \cdot T_{accel}) \\ &t_v \geq t_{v_{cur}} + (t_{a_{max}} \cdot T_{accel}) \end{aligned} \tag{2.22}$$

Ovim uslovima ograničava se robotova translaciona brzina, rotaciona brzina, translaciono ubrazanje i rotaciono ubrzanje, korišćenjem njihovih maksimalnih vrednosti $t_{v_{max}}, r_{v_{max}}, t_{a_{max}} ir_{a_{max}}$. Uslov $0 \leq t_v$ ne dozvoljava robotu kretanje u nazad. T_{accel} predstavlja vremenski interval sa kojim su komande izdate [21].

Prepreke se na početku posmatraju kao objekti u Dekartovskoj mreži, ali se kasnije transformišu u prostor brzina računanjem rastojanja od pozicije robota do prepreke praćenjem nekog konstantnog zakrivljenja (krivine) robotove putanje (Slika 2.22). Samo zakrivljenja koja leže unutra c_{min} i c_{max} se uzimaju u obzir pošto prostor krivina sadrži samo dozvoljene (legalne) putanje [1].

Da bi postigao performanse u realnom vremenu prepreke su aproksimirane kao kružni objekti i konture objekta su podeljene u nekoliko intervala. Rastojanje od krajnje tačke intervala do robota se izračunava, dok se za rastojanja između krajnih tačaka podrazumeva da su konstantna [1].



Slika 2.22: Tangentna zakrivljenja za prepreku [1]

Konačna odluka neke nove brzine (t_v i r_v) donosi objektna funkcija koja ima sledeći oblik:

$$\begin{aligned} &f(t_v, r_v) = \alpha_1 \cdot dist(t_v, r_v) + \alpha_2 \cdot head(r_v) + \alpha_3 \cdot speed(t_v) \\ &dist(t_v, r_v) = d(t_v, r_v, OBS)/L \\ &head(r_v) = 1 - |\theta_c - r_v * T_c|/\pi \\ &speed(t_v) = t_v/t_{v_{max}} \end{aligned} \tag{2.23}$$

α_1, α_2, α_3 su pozitivne konstante. $d(t_v, r_v, OBS)$ je rastojanje do skupa prepreka OBS do kojeg robot može da dođe krećući se zakrivljenjem (krivinom) $c = t_r/t_v$. Rastojanje $d(t_v, r_v, OBS)$ se normalizuje graničnim rastojanjem L. $head(r_v)$ je normalizovana greška pravca cilja. Definisana je kao razlika zapovedanog pravca θ_c i pravca koji će robot ispuniti ukoliko skrene za r_v za neku vremensku konstantu T_c. Drugim rečima, objektna funkcija pokušava da dostigne visoku brzinu kretanja na pravcu bliskom zapovedanom pravcu kretanja, mnogo pre nego što naiđe na prepreku [21].

MBZ pokušava da maksimizira vrednost ove funkcije i izračunava je samo kao deo prostora brzine koji ispunjava kinematička i dinamička ograničenja, kao i ograničenja usled nailaska na prepreke [21].

Iako proizvodi pouzdane glatke putanje ovaj metod ima neke propuste. Često, na preseku hodnika, ne uspeva da usmeri robota do otvorenog hodnika koji vodi ka cilju. Algoritam ponekad ne uzima u obzir neke putanje koje su na pravom uglu od trenutne robotove orijentacije. Takođe nekada pušta da robot ide pravo ka prepreci sve dok ne priđe previše blizu, čak i ako postoji čist prolaz oko nje. Svi ovi problemi

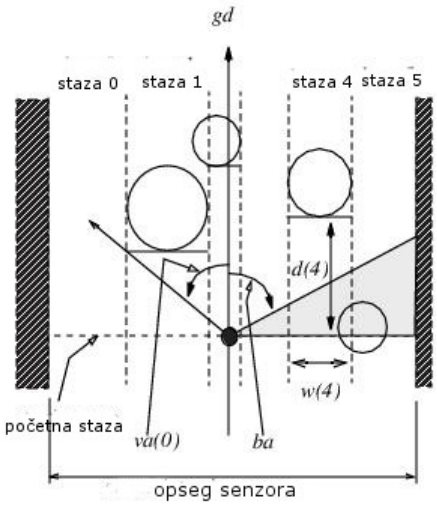
uzrok su toga što MBZ bira komande na osnovu dužine lukova bez kolizija za koje pretpostavlja da su putanje robota. Metod ne uzima u obzir da robot nekada može preći samo kratku distancu luka i onda ponovo skrenuti [21].

Još jedno ograničenje ovog metoda je u cirkularnoj simplifikaciji oblika prepreke. U nekim okruženjima ovo je prihvatljivo, u drugim, ovo pojednostavljivanje može da izazove ozbiljne probleme [1].

2.4.1 Metod staze zakrivljenja

Metod staze zakrivljenja (*Lane-Curvature Method*) ili MSZ(LCM), unapređuje pristup prostora brzina uzimajući u obzir pravce bez kolizija koliko i dužine lukova bez kolizija. Ovaj metod koristi pristup navigaciji koji se sastoji iz dva koraka. Prvo, za dati ciljni pravac, pristup pravca, zvani Metod staze (*Lane Method*) bira "stazu" u kojoj će robot biti, uzimajući u obzir zaobilaženje prepreka, efikasnost kretanja i usmerenje cilja. Zatim, metod staze računa lokalni pravac koji će voditi robota ili u tu ili tom stazom. Pošto Metod staze nije u mogućnosti da uračuna i fižicka ograničenja robota u svoju proceduru, lokalni pravac se prosleđuje MBZ-u. Na osnovu prosleđenog pravca, MBZ daje translacionu i rotacionu brzinu komande, uzimajći u obzir fižicka ograničenja robota [21].

Staze se konstruišu određivanjem maksimuma rastojanja bez kolizija do prepreka putem željenog ciljnog pravca. Susedne staze sa sličnim rastojanjem bez kolizija se spajaju. Da bi se olakšalo određivanje staze i da bi se one poklopile sa implementacijom MBZ-a, prepreke se aproksimiraju kao krugovi, predstavljeni njihovim lokacijama i poluprečnicima. Parametri koji opisuju k-tu stazu su širina staze $w(k)$, rastojanje bez kolizija $d(k)$ i pravac pogleda $v_a(k)$, koji predstavlja ugao pod kojim linija konstruisana iz robota prolazi stazom samo kroz oblasti u kojima nema kolizija sa preprekama. Ovi parametri su prikazani grafički na slici 2.23 [21].



Slika 2.23: Staze i njihovi parametri [21]

Na slici 2.23, broj staza N_l je šest. Radna oblast je podeljena na staze u okviru maksimalnog raspona senzora. Prilikom određivanja staza, ignorišu se prepreke koje su iza robota. Pošto se robot neprekidno kreće unapred, da bi se odredilo koje su prereke iza njega koriste se neke predifinisane ugaone granice. Prepreke čije ugaono rastojanje od željenog pravca kretanja prelazi ove ugaone granice smatraju se kao prepreke koje se nalaze iza robota. Ugaone granice za kretanja u smeru skazaljke na satu i smeru suprotnom skazaljke na satu se određuju individualno. Blokirajuća ugaona granica ba, za svaki pravac je definisana kao:

$$|ba| = \begin{cases} 90^\circ & \text{ako nema ni jedne prepreke na početnoj stazi,} \\ 55^\circ & \text{ako postoji prepreka na početnoj stazi.} \end{cases} \tag{2.24}$$

Rastojanje bez kolizija k-te staze, $d(k)$ je definisano kao rastojanje koje robot može da prođe, od početne staze, k-tom stazom pre udaranja u prepreku. Ugao pogleda k-te linije, $v_a(k)$ je minimalni ugao od poželjnog ciljnog pravca gd do pravca bez kolizija k-te linije. U određivanju $v_a(k)$, podrazumeva se da je k-ta staza blokirana na rastojanju $d(k)$ od početne staze [21].

Staza koja je veoma uska spaja se sa komšijskom stazom po sledećem pravilu: Ako $w(h) \leq w_{min}$ i $d(h) > Min(d(h-1), d(h+1))$ za neko $1 \leq h \leq N_l - 2$, onda integriši h-tu stazu u komšijsku sa manjim rastojanjem bez kolizije. Takođe, dve staze sa sličnim rastojanjima bez kolizije spajaju se zajedno po sledećem pravilu: ako $|d(h) - d(h+1)| \leq \Delta_{min}, 0 \leq h \leq N_l - 2$, onda integriši stazu sa većim rastojanjem bez kolizija u drugu stazu. Δ_{min} se postavlja eksperimentalno [21].

Kada su staze konstruisane, Metod staze zakrivljenja uzima najbolje staze za efikasno kretanje bez kolizija. Za bezbedna, kretanja bez kolizija, poželjno je ići stazama sa što dužim rastojanjem bez kolizija i što većom širinom. Za efikasno upravljanje, manje promene pravca su poželjne. Za što brža i efikasna kretanja, preferira se komanda kretanja sa orijentacijom bliskoj trenutnoj orijentaciji o_r. Izbor staze se može opisati funkcijom $f_s(k)$:

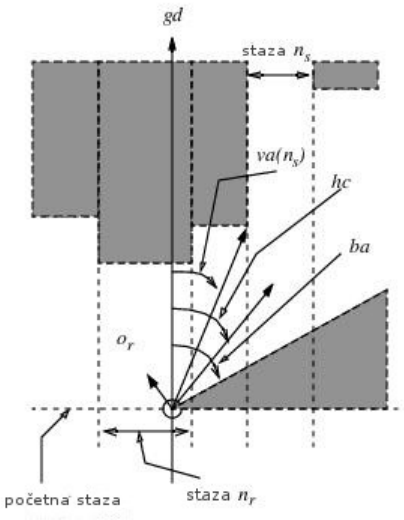
$$
\begin{aligned}
& f_s(k) = \beta_1 * \overline{d}(k) + \beta_2 * \overline{w}(k) - \beta_3 * \overline{ad}_{va,c}(k) - \beta_4 * \overline{ad}_{va,o}(k) \\
& \overline{d}(k) = \min\{d(k), D_{limit}\}/D_{limit} \\
& \overline{w}(k) = \min\{w(k), W_{limit}\}/W_{limit} \\
& \overline{ad}_{va,c}(k) = \min\{va(k) - c_p, C_{limit}\}/C_{limit} \\
& \overline{ad}_{va,o}(k) = \min\{va(k) - o_r, O_{limit}\}/O_{limit} \\
& c_p \quad : \text{trenutna komanda pravca} \\
& o_r \quad : \text{trenutna orijentacija robota}
\end{aligned}
\tag{2.25}
$$

Pošto je ugao pogleda $va(k)$ minimum ugaone devijacije bez kolizija od ciljnog pravca ad, koristi se kao pravac navođenja ka k-toj stazi u funkciji selekcije. Svaki parametar u $f_s(k)$ je ograničen i normalizovan uzimajući u obzir maksimalne vrednosti D_{limit}, W_{limit}, C_{limit} i O_{limit}. Parametar $\overline{ad}_{va,c}(k)$ pokazuje težnju ka manjim promenama komande pravca. Slično, parametar $\overline{ad}_{va,o}(k)$ ukazuje na težnju ka izboru komande kretanja bliskoj trenutnoj orijentaciji. β_1, β_2, β_3, β_4 vrednosti su težine koje su date svakom parametru i sve su pozitivne [21].

Ukoliko je robot već u najboljoj stazi, MBZ-u se šalje originalno poželjno ciljno kretanje, koje MBZ koristi za novu komandu kretanja robota. U suprotnom, računa se lokalni pravac koji će da izazove MBZ da promeni staze. Neka je n_s-ta staza izabrana kao najbolja. Pošto je ugao pogleda $va(n_s)$ minimalan ugao bez kolizije do n_s-te staze, lokalni pravac hc bi trebao biti $|va(n_s)| \leq |hc|$. Takođe, lokalni pravac se ograničava da bude unutar blokirajućeg ugla ba, odnosno za lokalni pravac bi trebalo da važi $|hc| \leq |b_a|$. Na osnovu ovih ograničenja lokalni pravac postaje:

$$
\begin{aligned}
& hc = va(n_s) + \delta * (ba - va(n_s)) \\
& \text{gde,} \\
& 0 \leq \delta \leq 1.0
\end{aligned}
\tag{2.26}
$$

Vrednost δ određuje koliko je lokalni pravac daleko od ugla pogleda do izabrane staze. Ako je $\delta = 0$, onda je komanda kretanja samo ugao pogleda, i ne postoji čist prolaz za sigurno kretanje. Ako je $\delta = 0$ komanda kretanja uvek navodi ka ekstremnoj levoj ili desnoj strani. Odnos između ugla pogleda, blokirajućeg ugla i komande pravca prikazan je na slici 2.24 [21].



Slika 2.24: Određivanje lokalnog pravca [21]

2.5 Pristupi dinamičkog prozora

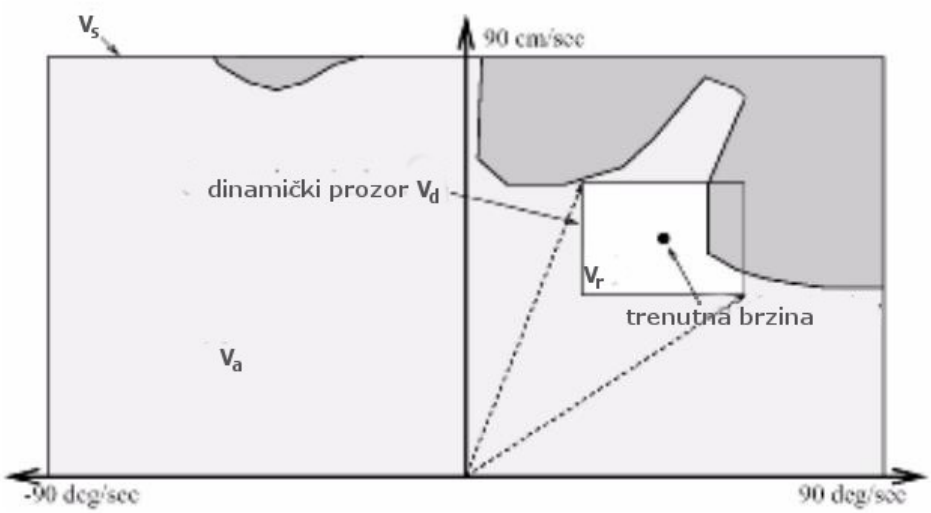
Još jedna tehnika koje može uzeti u obzir ograničenja robotove kinematike jeste metod dinamičkih prozora za zaobilaženje prepreka. Jednostavan, ali veoma efikasan dinamički model daje ime ovom pristupu.

U daljem tekstu izdvojene su dve varijacije ove tehnike, pristup dinamičkog prozora (*dynamic window aproach*) ili pristup lokalnog dinamičkog prozora (*local dynamic window aproach*) i pristup globalnog dinamičkog prozora (*global dynamic window aproach*) [1].

2.5.1 Pristup lokalnog dinamičkog prozora

U lokalnom pristupu dinamičkih prozora kinematika robota se uzima u obzir pretragom pažljivo izabranog prostora brzina (*velocity space*). Prostor brzina je skup svih mogućih torki (v, w) gde je v brzina, a w ugaona brzina. Pristup podrazumeva da se robot kreće samo kružnim lukovima koje predstavlja svaka torka, tokom makar jednog vremenskog perioda [1].

Uz pomoć date trenutne brzine robota, algoritam prvo bira dinamički prozor torki (v, w) koji se može dostići u toku sledećeg perioda sempliranja, uzimajući u obzir mogućnosti ubrzanja robota i vreme ciklusa. Sledeći korak je redukcija dinamičkog prozora čuvanjem samo onih torki koje nam obezbeđuju da će vozilo stati pre nailaska na (udaranja u) prepreku. Na slici 2.25 je prikazan tipičan dinamčki prozor. Može se primetiti da je oblik dinamičkog prozora pravougaoni, što proizilazi iz aproksimacije da su dinamičke sposobnosti za translacije i rotacije nezavisne [1].



Slika 2.25: **Pristup dinamičkog prozora.** Pravougaoni prozor prikazuje sve moguće brzine (v, w) i preklapanje sa preprekama u konfiguracionom prostoru [1]

Prepreke blizu robota nameću ograničenja translacionim i rotacionim brzinama robota. Translacione i rotacione brzine u okviru ograničenja ovih prepreka se nazivaju prihvatljive brzine (*admissible velocities*). Maksimalna prihvatljiva brzina, za dato zakrivljenje, zavisi od rastojanja robota do sledećeg objekta preko tog zakrivljenja. Skup prihvatljivih brzina (V_a) se dobija funkcijom $Dist(v, w)$ koja računa rastojanje do najbliže prepreke za dato zakrivljenje. Ovo se može izraziti kao:

$$Dist(v, w) = \min_{obs \in OBS} dist(v, w, obs), \tag{2.27}$$

gde je *obs* element iz skupa prepreka OBS.
Skup V_a se može izraziti kao :

$$V_a = \left\{ (v, w) \middle| \begin{array}{c} v \leq \sqrt{2 \cdot Dist(v, w) \cdot \dot{v}_{max}}, \\ w \leq \sqrt{2 \cdot Dist(v, w) \cdot \frac{\dot{w}_{max}}{c}} \end{array} \right\} \tag{2.28}$$

gde su, $\dot{v}_{max}$ i $\dot{w}_{max}$ maksimalna translaciona i rotaciona ubrzanja [22].

Sa druge strane, skup brzina, koji se naziva skup mogućih brzina (*reachable velocities*), ukazuje na one brzine koje robot može da dostigne tokom kontrolne petlje - brzine koje definišu dinamčiki prozor V_d. Skup V_d se izražava kao:

$$V_d = \left\{ (v, w) \middle| \begin{array}{c} \frac{v - v_c}{\Delta t} \in [-\dot{v}_{max}, \dot{v}_{max}], \\ \frac{w - w_c}{\Delta t} \in [-\dot{w}_{max}, \dot{w}_{max}] \end{array} \right\} \tag{2.29}$$

gde su, v_c i w_c trenutne translacione i rotacione brzine, i Δt trajanje kontrolne petlje. Sumirajući, prostor pretrage kontrolnih komandi je redukovan na tri vrste ograničenja: (i) kružne putanje, (ii) prihvatljive

brzine, (iii) brzine koje se mogu dostići. Od ograničenja koju su nametnuta robotovim brzinama, rezultujući prostor pretarage (*a resulting search space*) (V_r) se može definisati kao:

$$V_r = V_p \cap V_a \cap V_d, \tag{2.30}$$

gde V_p predstavlja ceo prostor mogućih brzina robota. Defininsan je sa:

$$V_p = \left\{ (v, w) | \begin{matrix} v \in [0, v_{max}], \\ w \in [-w_{max}, w_{max}] \end{matrix} \right\} \tag{2.31}$$

primetimo da je v, definisano samo za pozitivne vrednosti, sto znači da robot ne može da se kreće unazad [22].

Konačno od rezultujućeg prostora pretrage V_r, PDP bira par brzina koji maksimiziraju objektnu funkciju. Objektna funkcija uključuje parametre koji daju kretanje visokom brzinom, orijentisanost ka cilju i udaljenost od prepreka: brzina, usmerenost ka cilju i sigurnost. Dakle, objektna funkcija se definiše kao:

$$G(v, w) = \mu_1 \cdot Speed(v) + \mu_2 \cdot Goal(w) + \mu_2 \cdot Dist(v, w), \tag{2.32}$$

gde su $\mu_i > 0$, $i = 1, 2, 3$, i $\sum_i \mu_i = 1$ težinski faktori za svaki od parametara objektne funkcije.

Funkcija brzine *Speed* se koristi da postigne navigaciju sa visokom brzinom.

$$Speed(v) = v/v_{max}. \tag{2.33}$$

Ipak, korišćenje ove funkcije može dovesti do izbora pogrešne akcije pod nekim određenim uslovima. Na primer, kada robotova orijentacija ima visok raskorak sa ciljem ($\alpha > 90°$), kretanje velikom brzinom može odvesti robota dalje od cilja. Sa druge strane, *Speed* funkcija ne uzima u obzir blizinu cilja, dakle kada je robot blizu cilja ova funkcija promoviše pogrešnu akciju: brzu navigaciju [22].

Funkcija cilja *Goal* meri poravnanje robota sa orijentacijom cilja, definisanom sa parametrom α. Ova funkcija računa orijentacionu grešku, pretpostavljajući da se robot kreće konstantnom brzinom w tokom intervala vremena kontrolne petlje Δt. Ova funkcija se može definisati kao:

$$Goal(w) = 1 - |\alpha - w \cdot \Delta t|/\pi. \tag{2.34}$$

Može se primetiti da funkcija *Goal* zbog izostavljanja translacione brzne u izračunavanju ne uzima u obzir ugoanu blizinu. Ovaj propust dolazi do izražaja u onim slučajevima gde je robot blizu cilja, a ima pogrešnu orijentaciju (visoko α) [22].

Funkcija rastojanja *Dist*, predstavlja rastojanje do najbliže prepreke preko cirkularne putanje sa zakrivljenjem datim brzinama (v, w).

Dokazano je da su metode, kao što je PDP, bazirane na prostoru kontrolnih komandi pogodne za implementaciju strategija navigacije. One uključuju ograničenja okruženja i robotove dinamike. Dodatno, one uključuju i objektnu funkciju koja nameće pogodna ponašanja: brzinu, usmerenje ka cilju i sigurnost. Međutim, druge konvergirajuće kriterijume takođe treba uzeti u obzir radi izračunavanja dolaska do cilja [22].

2.5.2 Pristup globalnog dinamičkog prozora

O. Brok i O. Katib su uveli pristup globalnog dinamičkog prozora u cilju eliminasanja problema lokalnog minimuma koji se javljao kod običnog pristupa dinamičkog prozora zbog njegove čiste lokalne prirode [23]. Ovaj pristup dodaje, kao što ime sugriše, globalno razmišljanje algoritmu dinamičkog prozora. Ovo je urađeno dodavanjem funkcije navigacije $NF1$, ili grasfajera (*grassfire*), objektnoj funkciji (2.32) [1]. Funkcija navigacije $NF1$ je skalarna i ima jedinstveni minimum na poziciji cilja. Objektna funkcija proširena funkcijom navigacije dobija sledeći oblik:

$$\begin{aligned} \Omega_g(p, v, a) = \alpha \cdot nf1(p, v) \\ + \beta \cdot vel(v) \\ + \gamma \cdot goal(p, v) \\ + \delta \cdot \Delta nf1(p, v, a), \end{aligned} \tag{2.35}$$

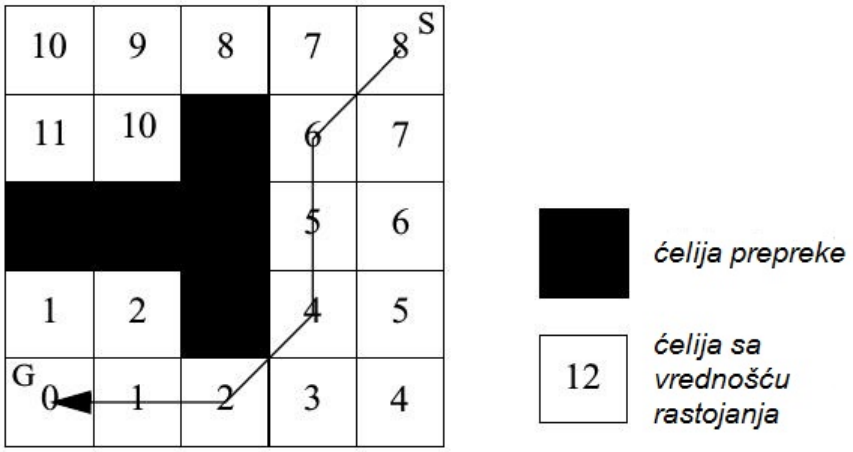
gde je p trenutna pozicija, v željena brzina a a željeno ubrzanje. Funkcija $nf1(p, v)$ daje prednost brzinama poravnatim sa gradijentom funkcije navigacije, $vel(b)$ preferira brzine sa višom translacionom brzinom. Funkcija $goal(p, v)$ je binarna funkcija sa vrednostima 1, ako putanja prolazi kroz ciljnu poziciju i 0 inače. Vrednost $\Delta nf1(p, v, a)$ ukazuje na očekivano smanjenje vrednosti funkcije navigacije uz putanju [23].

$NF1$ obeležava ćelije u mreži popunjenosti (*occupancy grid*) sa ukupnim rastojanjem L do cilja. Da bi se ovaj postupak ubrzao, globalni prozor računa $NF1$ samo za izabrani pravougaoni region koji je usmeren ka cilju. Širina ovog regiona se uvećava i preračunava ako cilj ne može da se dostigne u okviru ograničenja ovog izabranog regiona [1].

Ovo dozvoljava da globalni prozor postigne neke prednosti globalnog planiranja putanje bez potpunog apriori znanja. Mreža popunjenosti se ažurira sa merama rastojanja kako se robot pomera u okruženju.

$NF1$ se računa za svaku ažuriranu verziju mreže popunjenosti. Ako $NF1$ ne može da se izračuna zbog činjenice da je robot okružen preprekama, metod se degradira u običan pristup dinamičkog prozora. Ovo održava kretanje robota tako da se mogući izlaz može naći i izračunavanje $NF1$ se može nastavi [1].

Globalni dinamički prozor obećava izvršavanje u realnom vremenu, dinamička ograničenja, globalno razmišljanje i zaobilaženje prepreka sa minimalnom slobodom velikom brznom [1].



Slika 2.26: Primer transformisanja rastojanja i rezultujuća putanja generisana $NF1$ funkcijom. S, start; G, cilj [1].

2.6 Šlegelov pristup zaobliaženja prepreka

Šlegel je u svom radu predstavio pristup koji uzima u obzir dinamiku koliko i stvarni oblik robota. Ovaj pristup je usvojen za sirovo lasersko merenje podatka i fuziju senzora koristeći dekartovsku mrežu za predstavljanje prepreka u okruženju. Performanse u realnom vremenu su dostignute korišćenjem unapred izračunatih lukap tabela [1].

Kao i kod prethodnih metoda, osnovna pretpostavka je da se robot kreće putanjama oblika kružnih lukova, definisanih zakrivljenjem i_c. Za dato zakrivljenje i_c Šlegel računa rastojanje l do kolizije između jedne tačke $[x, y]$ objekta u Dekartovskoj mreži i robota (Slika 2.27). Pošto je robotu dozvoljeno da bude bilo kakvog oblika ovo izračunavanje troši dosta vremena pa se zbog toga rezultat izračunava unapred i smešta u lukap tabelu [1].

Na primer, prozor prostora pretrage V_s je definisan za vozno-diferencijalne robote kao sve moguće brzine levog i desnog točka, v_r, v_l. Dinamička ograničenja robota su uzeta u obzir "pročišćenjem" V_s da samo uzima one vrednosti koje se mogu dostignuti u sledećem koraku izvršavanja, u odnosnu na trenutno kretanje robota. Konačno, objektna funkcija bira najbolju brzinu i pravac na osnovu kompromisa između pravca cilja, brzine i rastojanja do kolizije [1].

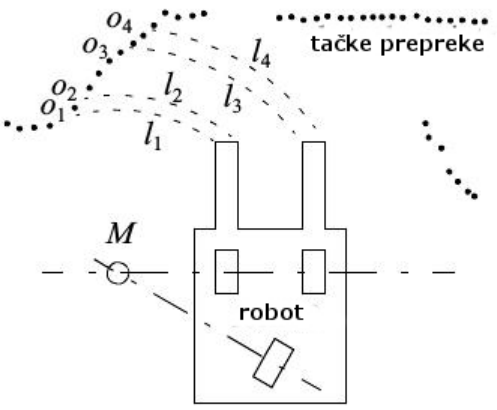
Nedostatak ovog pristupa je potencijalna potreba za memorijom za čuvanje lukap tabele. U njihovim eksperimentima, autori su koristili lukup tabelu veličine do 2.5 Mb koristeći 6×6 dekartovsku mrežu sa rezolucijom od 10 cm i 323 različitih zakrivljenja (krivina) [1].

2.7 ASL pristup

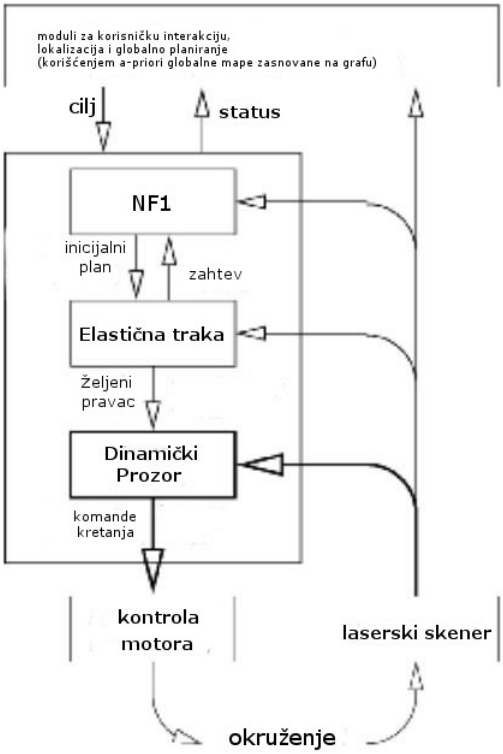
Autonomna Sistemska Laboratorija (ASL) Švedskog federalnog instituta za tehnologije razvila je metod za zaobilaženje prepreka za masivno izlaganje. Izlaganje je zahtevalo da se mobilni robot kreće kroz guste gužve i da se obezbedi određeni protok posetioca. Ovaj pristup spaja tri pristupa u cilju stvaranja sistema koji omogućava robotu da se kreće glatko bez zaustavljanja radi ponovnog planiranja, i koji je sposoban da pažljivo "izgura" robota kroz gužve kada je to bezbedno [1].

ALS pristup je sačinjen od lokalnog planera putanje i metoda za zaoblaženje prepreka koji dobija ulaz u formi tačaka prolaza (*waypoints*) sa viših nivoa. Pregled ASL pristupa je dat na slici 2.28 [1].

Lokalno planiranje putanje se izvršava uz pomoć $NF1$. Rezultujuća putanja se prevodi u elastičnu traku koja ne uzima u obzir kinematiku, iskorišćavajući činjenicu da oktagonalni robot korišćen u izlaganju može da se okrene u mestu u većini vremena. Ovo čini ažuriranje putanje jednostavnim koliko je to moguće. Unapređeni dinamički prozor onda preuzima odgovornost vođenja robota putanjom [1].



Slika 2.27: Rastojanja l_i nastala zakrivljenjem i_c, posle rotacije robota oko tačke M [1]



Slika 2.28: Dijagram toka ASL pristupa[1]

2.8 Dijagram blizine

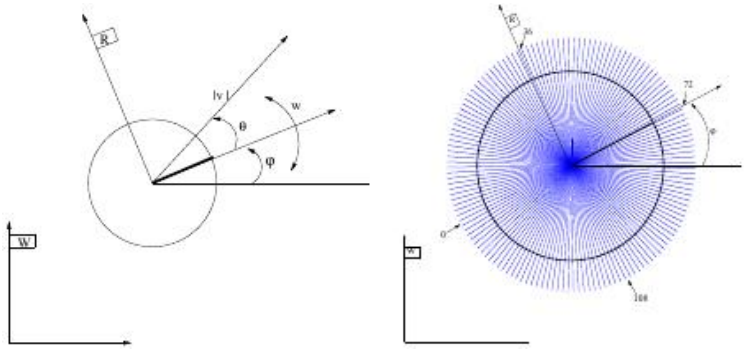
Dijagram blizine (*Nearnes Diagram*) je metod za zaobilaženje prepreka koji koristi opis visokog nivoa informacija okruženja da generiše nove komande kretanja, na osnovu pet različitih zakona [24]. Za dijagram blizine se može smatrati da je u nekoj meri sličan HVP algoritmu, ali i da reševa nekoliko njegovih propusta, posebno u veoma gustom prostoru [1]. Ovaj metod je pogodan za navigaciju u nepoznatim, nestruktuiranim i dinamičkim okruženjima jer uzima u obzir precizna geometrijska, kinematička i dinamička ograničenja [1, 24]. Ovo je postignuto razbijanjem problema u generisanje najobećavajućeg pravca kretanja sa čistim kružnim robotom, i onda prilagođavanjem ovoga u kinematička i dinamička ograničenja robota, praćena korekcijama za robotov oblik, ukoliko nije cirkularan (u originalnoj publikaciji bili su podržani pravougaoni roboti) [1].

Predloženi metod navigacije izvršava ekstrakciju informacija iz okruženja u tri koraka. Prvo, od raspoloživih informacija, konstruiše dva dijagrama blizine - dijagram blizine od centralne tačke PND i dijagram blizine od robota RND. U sledećem koraku, PND se analizira da bi se identifikovali regioni i

izabrao jedan od njih. Nakon ovoga, analizira se RND da bi se izračunla sigurna situacija robota (*robot safety situation*). Zatim se ova informacija koristi da identifikuje jednu od pet generalnih situacija [24].

Reprezentacija okruženja kod Dijagrama blizine je podeljena na sektore (Slika 2.29). Kako bi se predstavile informacije o blizni prepreka od okruženja se konstruišu gore pomenuti dijagrami (RNB i PNB). Primeri RND i PND dijagrama su prikazani na slici 2.30 [24].

Neka je $b \in A$ tačka robota koja se koristi kao centar sektora, i koja se nalazi u centru robota i naziva centralna tačka. Neka je n broj sektora i φ orijentacija robota u globalnom smislu. Neka je δ funkcija, koja za datu konfiguraciju robota q i tačku b računa vektor takav da je $\delta_i(q, b)$ najmanje rastojanje do prepreke u sektoru i, dok je φ ugao koji odgovara bisektoru od $n/2$ sektora (Slika 2.29b).



(a) (b)

Slika 2.29: (a) Referenca robota i komande kretanja (v, w) (b) Distribucija sektora u referenci robota[24]

Dijagram blizine od centralne tačke (PND) se definiše na sledeći način:

$$\begin{aligned} PND : C_{free} \times A &\rightarrow (\mathbb{R}^+ \cup \{0\})^n \\ (q, b) &\rightarrow (D_1, \ldots, D_n) \end{aligned} \tag{2.36}$$

$$\begin{aligned} \text{Ako,} \quad \delta_i(q, b) > 0 \quad & D_i = PND_i(q, b) = d_{max} + l - delta_i(q, b) \\ \text{inače} \quad & D_i = PND_i(q, b) = 0 \end{aligned}$$

Gde je:

- d_{max} maksimum vrednosti δ (predstavlja maksimalni domet senzora)
- l maksimalno rastojanje između dve takče robota (predstavlja prečnik kružnog robota)

Dijagram blizine od robota (RND) se definiše na sledeći način:

$$\begin{aligned} RND : C_{free} \times A &\rightarrow (\mathbb{R}^+ \cup \{0\})^n \\ (q, b) &\rightarrow (D_1, \ldots, D_n) \end{aligned} \tag{2.37}$$

$$\begin{aligned} \text{Ako,} \quad \delta_i(q, b) > 0 \quad & D_i = RND_i(q, b) = d_{max} + E_i(b) - delta_i(q, b) \\ \text{inače} \quad & D_i = RND_i(q, b) = 0 \end{aligned}$$

Gde je:

- E data uvećavajuća funkcija koja zavisi od geometrije robota. Vrednost ove funkcije u svakom sektoru $E_i(b)$ je polupreč nik robota.

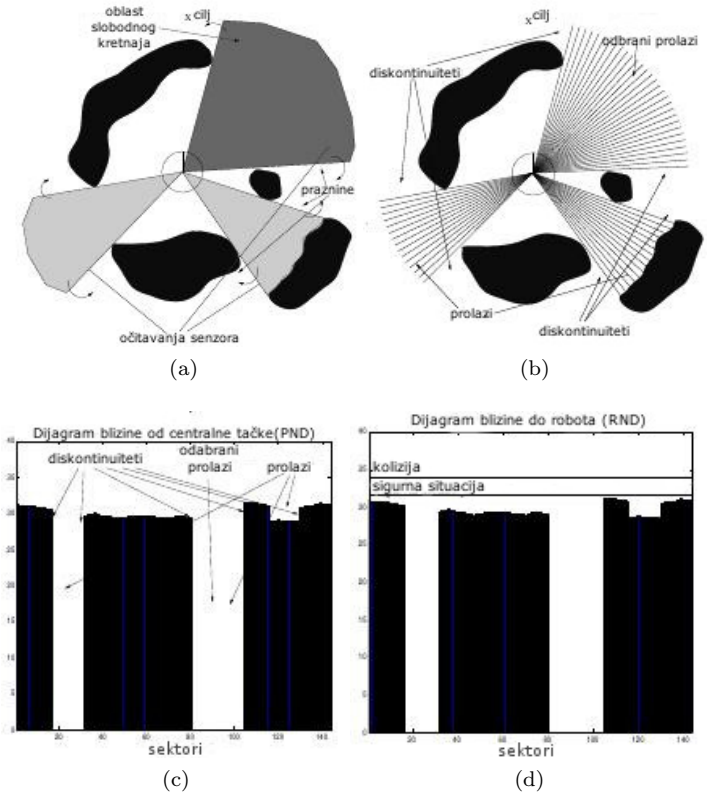
PND predstavlja blizinu prepreka od centralne tačke, dok RND predstavlja bliznu prepreka do robotovih granica (Slika 2.30). Neka je $PND_i \equiv PND_i(q, b)$ i $RND_i \equiv RND_i(q, b)$ [24].

PND predstavlja blizinu od prepreke do centralne tačke. Topologija okruženja ne varira u ovom dijagramu, pa se zato koristi za ekstrakciju informacija o karakteristikama okruženja. PND analiza se odvija u tri koraka. Prvo se u okruženju traže praznine (*gaps*). Od ovih praznina se dobijaju regioni slobodnog prostora, od kojih se na kraju odabira jedan na osnovu jednog od kriterijuma (Slika 2.30a) [24].

Između dva susedna sektora postoji diskontinuitet, ako je njihova razlika visina u PND veća od l (predstavljena u definiciji PND-a (2.36)) (Slika 2.30c). Diskontinuitet predstavlja praznine među preprekama ili praznine nakon završetka prepreke [24].

Skup sektora $S = \{s_i\}, i = 1, \ldots, k$ predstavlja prolaz (*valley*) (gde su s_l i s_r levi, odnosno desni ekstremni sektor) ako zadovoljava sledeće dve premise:

1. Svi sektori u S su susedni ako nema diskontinuiteta između njih



Slika 2.30: (a) Praznine, regioni i izabrana oblast slobodnog kretanja (b) Diskontinuiteti, prolazi i izabrana praznina mapirana na okruženje (c) PND generisan (d) RND generisan [24]

2. Neka su s_{nl} i s_{nr} susedni sektori s_l odnosno s_r, koji ne pripadaju S (nazvani susedni sektori prolaza). Postoje dva diskontinuiteta između (s_l, s_{nl}) i (s_r, s_{nr}) i oba sektora zadovoljavaju: $PND_{s_{nl}} - PND_{s_l} > 1$ ili $PND_{s_{nr}} - PND_{s_r} > 1$.

Specijalan slučaj se javlja kada se lokacija cilja nalazi između prepreke i robota. Tada, sektor koji sadrži lokaciju cilja (s_{goal}) ne može da pripada prolazu. Kada se ova situacija detektuje, $PND_{s_{goal}}$ se postavlja na nulu, stvarajući veštački prolaz u sektoru cilja [24].

Postoje dva tipa diskontinuiteta (praznina) u zavisnosti na premisu 2, rastući i opadajući diskontinuitet [24].

Što se tiče prolaza, postoje široki i uski prolazi. Prolaz je širok ako je broj sektora veći od s_{max} (na primer $s_{max} = n/2$), a uzak inače. Prolazi predstavljaju regione među preprekama [24].

Postoje dve sigurnosne situacije u kojima robot može biti: situacija niske sigurnosti **LS**(*Low Safety*) i situacija visoke sigurnosti **HS**(*High Safety*). Robot je u situaciji niske sigurnosti ako u RND-u najmanje jedan sektor premašuje bezbednu blizinu. Ovo znači da je rastojanje između granica robota i prepreka niže od bezbednog (sigurnosnog) rastojanja, što predstavlja potencijalni rizik za robota (nešto je unutar sigurnosne zone - Slika 2.31). Inače, situacija je visoke sigurnosti (Slike 2.32 i 2.33) [24].

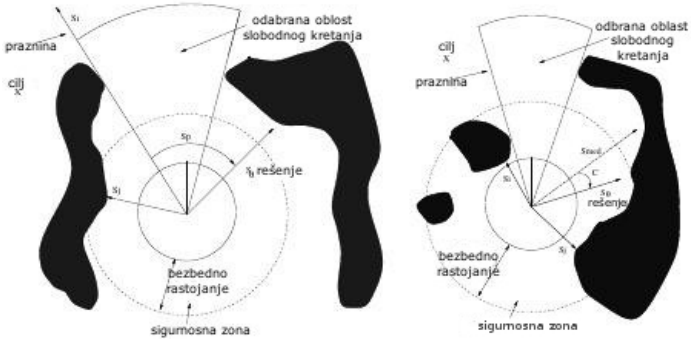
Na osnovu prethodno opisanih koraka RND analize dobijaju se neke informacije o regionima okruženja i opisu robotove sigurnosti. Ove informacije se koristi za definisanje pet generalnih situacija robota. One obuhvataju sve mogućnosti robotovih lokacija, lokaciju cilja i konfiguraciju prepreka, i moraju se proveriti u rigoroznom redosledu [24].

Postoje dve generalne situacije niske sigurnosti:

1. **LS1**: robot je u **LS1**, ako postoji makar jedan sektor koji premašuje bezbednu blizinu u RND-u, samo sa jedne strane rastućeg diskontinuiteta, koji je bliži sektoru cilja, nego sektoru izabranog prolaza. Ova situacija se dešava kada je robot previše blizu prepreke samo sa jedne strane praznine, koja je bliža cilju izabrane oblasti slobodnog kretanja (Slika 2.31a).
2. **LS2**: robot je u **LS2**, ako postoji makar jedan sektor koji premašuje bezbednu blizinu u RND-u, sa obe strane rastućeg diskontinuiteta, koji je bliži sektoru cilja izabranog prolaza. Ova situacija se dešava kada je robot previše blizu prepreke sa obe strane praznine, koja je bliža cilju izabrane oblasti slobodnog kretanja (Slika 2.31b).

Može se primetiti da obe situacije niske sigurnosti zavise od relativne lokacije između robota, prepreka i oblasti slobodnog kretanja (lokacija cilja se koristi samo radi izbora oblasti slobodnog kretanja). Postoje tri generalne situacije visoke sigurnosti:

3. **HSGV**: robot je u visokoj sigurnosti cilja u prolazu (*High safety goal in Valley*) ako sektor cilja pripada prolazu. Ova situacija se javlja kada je lokacija cilja unutar izabrane oblasti slobodnog kretanja (Slika 2.32).
4. **HSWV**: robot je u visokoj sigurnosti širokog prolaza (*High safety Wide Valley*) kada je izabrani prolaz širok. Ova situacija se dešava kada cilj nije unutar izabrane oblasti slobodnog kretanja, ali je ova oblast široka (Slika 2.33a).
5. **HSNV**: robot je u visokoj sigurnosti uskog prolaza (*High safety Narrow Valley*) ako je izabran prolaz uzak. Ova situacija se dešava kada cilj nije unutar izabrane oblasti slobodnog kretanja, ali je oblast uska (Slika 2.33b).



(a) (b)

Slika 2.31: (a) Primer **LS1** situacije (b) Primer **LS2** situacije [24]

Može se primetiti da situacije visoke sigurnosti zavise samo od pozicije cilja, u odnosu na izabrane oblasti slobodnog kretanja ili njenog oblika [24].

Strategija navigacije kod Dijagrama blizine se bazira na dve bitne pretpostavke: Prva, veoma je teško rešiti problem lokalne navigacije sa jedinstvenom heuristikom kretanja, zbog strukturalne kompleksnosti koja može da predstavlja okruženje (čak i u lokalnom pogledu). Druga, direktno korišćenje ciljne lokacije u heuristici kretanja ima pretenciozne efekte (osim u očigledim situacijama bez očigledne kompleksnosti [24]).

Dijagram blizine koristi pet zakona o kretanju, izvedenih od pet situacija dobijenih u analizi i koraku interpretacije, u kojima je ciljna lokacija direktno korišćena samo u jednoj od njih. Algoritam računa translacionu i rotacionu brzinu (v, w) u svakom vremenskom periodu kao komandu kretanja za holonomične mobilne robote (Slika 2.29a) [24].

Za svaku situaciju sektor rešenja (*solution sector*) $s_\theta \in \mathbb{R}$ se izračunava. Pravac kretanja θ se dobija iz bisektora od s_θ. Zbog činjenice da su $s_\theta \in \mathbb{R}$, beskonačni virtaulni sektori koji su kreirani oko robota da bi on mogao da izabere rešenje, svaki pravac kretanja može biti dodeljen, gde je $\theta \in [-\pi, \pi]$ [24].

Za realističnu implementaciju ovog metoda, poželjno je popraviti $\theta \in [-\frac{\pi}{2}, \frac{\pi}{2}]$ da je $\theta \in [\frac{n}{4}, \frac{3 \cdot n}{4}]$ kako bi se zabranilo momentalno kretanje unazad [24].

U navigaciji u situaciji niske sigurnosti robot je u opasnosti od sudaranja, pa rešenje treba da dovede robota u bezbednu situaciju.

Navigacija u situacijama niske sigurnosti se odvija na sledeći način:

Navigacija u LS1 : U **LS1** postoje prepreke koje su bliže od bezbednog rastojanja, sa samo jedne strane praznine, koja je bliža cilju izabrane oblasti slobodnog kretanja. Rešenje treba da da kretanje koje odvodi prepreke iz sigurnosne zone, dok se kreće unapred izabranom prazninom (Slika 2.31a). Rešenje se računa na sledeći način:

$$\begin{aligned} s_p &= Abs(s_i - s_j) \cdot p + \frac{s_{max}}{2} \\ s_\theta &= s_i + sign(s_i - s_j) \cdot s_p \end{aligned} \tag{2.38}$$

Gde je:

s_i : sektor koji odgovara rastućem diskontinuitetu (praznini) izabranog prolaza (oblasti slobodnog kretanja).

s_j : sektor sa najvećom vrednošću u RND, koja premašuje sigurnosnu blizinu, sa strane rastućeg diskontinuiteta izabranog prolaza. Odgovara bližoj prepreci.

p : eksperimentalno podešen parametar, čije vrednosti zavise od tranzicije između generalnih situacija sigurnosti, i obezbeđuje glatko ponašanje u njima.

U **LS1** navigaciji, glavni cilj je da se "pogura" robot dalje od najbliže prepreke, dok se kreće prazninom oblasti slobodnog kretanja [24].

Navigacija u LS2 : U **LS2** postoje prepreke koje su bliže od bezbednog rastojanja, sa obe strane praznine, koja je bliža cilju izabrane oblasti slobodnog kretanja. Rešenje treba da proizvede kretanje koje će da centrira robota između dve prepreke, dok se kreće izabranom prazninom (Slika 2.31b). Sektor rešenja se računa na sledeći način:

$$s_{med} = \frac{s_i + s_j}{2}$$
$$s_\theta = s_{med} \pm c \tag{2.39}$$

Gde su:

s_i, s_j : sektori sa najvećim vrednostima u RND, koja premašuju sigurnosnu blizinu, sa obe strane rastućeg diskontinuiteta izabranog prolaza. Oni odgovaraju dvema najbližim preprekama.

c : je korekciona vrednost korišćenja za centriranje robota između dve bliske prepreke. Zavisi od rastojanja bliže prepreke i razlike rastojanja dve najbliže prepreke. Kvantitet c se dodaje ili oduzima u funkciji sektora koji sadrži bližu prepreku od dve najbliže prepreke, u cilju postavljanja robota na istom rastojanju od obe (najsigurnijem rastojanju).

U **LS2** navigaciji, glavni cilj je održavanje robota na istoj udaljenosti od dve najbliže prepreke, dok se kreće kroz izabranu prazninu [24].

U situacijama visoke sigurnosti robot nije u opasnosti sudaranja, pa se solucija bira unutar oblasti slobodnog kretanja [24].

Navigacija u svim situacijama visoke sigurnosti je sledeća:
Navigacija u HSGV : U **HSGV** lokacija cilja je unutar oblasti slobodnog kretanja. Rešenje je da se robot pomeri unapred ka cilju. Sektor rešenja se računa sa $s_\rho = s_{goal}$ [24].

Ovo je situacija koja direktno koristi lokaciju cilja da izračuna komande kretanja. Primetimo da u ovoj situaciji robot nije u opasnosti sudaranja, i cilj je unutar oblasti slobodnog kretanja (ova situacija nije opasna za robota i nema očiglednu kompleksnost) [24].

Na slici 2.32 prikazana su dva primera **HSGV** situacije. U 2.32a robot se kreće direktno ka cilju (koji je unutar oblasti slobodnog kretanja). Slika 2.32b je specijalan slučaj gde je veštački prolaz napravljen. Navigacija u ovoj situaciji vodi robota direktno ka cilju [24].

Navigacija u HSWV: U **HSWV** sektor sa ciljem nije unutar oblasti slobodnog kretanja, ali je širok. Situacija bi trebala da ima ponašanje praćenja konture (Slika 2.33a). Sektor rešenja se računa na sledeći način:

$$s_\theta = s_i \pm \frac{s_{max}}{2} \tag{2.40}$$

Gde je:

s_i : sektor koji odgovara rastućem diskontinuitetu, koji je bliži ciljnom sektoru, izabranog prolaza. Odgovara praznini bližoj cilju izabrane oblasti slobodnog kretanja. Ova navigacija daje rezultate kretanja duž jedne strane prepreke.

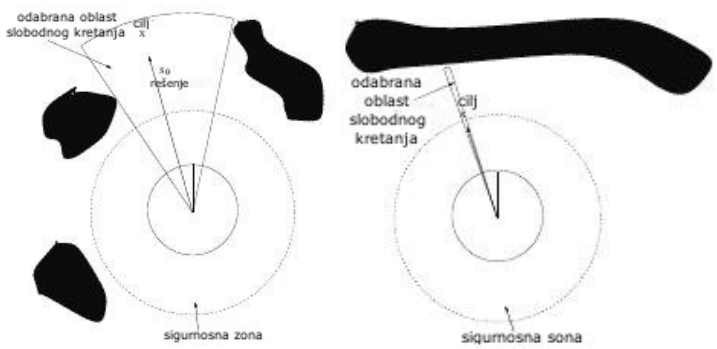
Navigacija u HSNV: U **HSNV** sektor sa ciljem nije unutar oblasti slobodnog kretanja, ali je uzak. Rešenje je da se kreće ka centru oblasti slobodnog kretanja (Slika 2.33b). Sektor rešenja se računa na sledeći način:

$$s_\theta = \frac{s_i + s_j}{2} \tag{2.41}$$

Gde su:

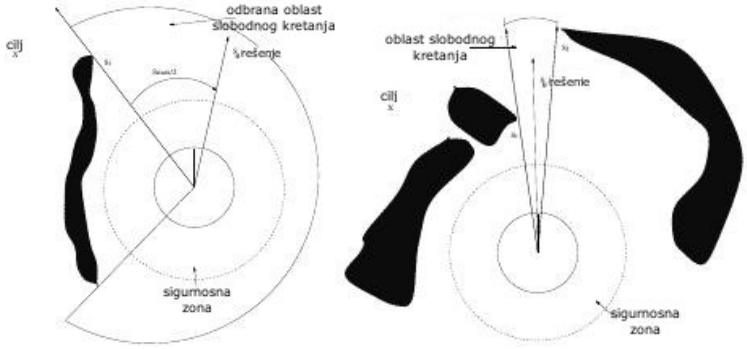
s_i, s_j : sektori dva diskontinuiteta izabranog prolaza. I odgovaraju dvema prazninama oblasti slobodnog kretanja.

Ova navigacija usmerava robota između prepreka.



(a) (b)

Slika 2.32: (a) Primer **HSGV** situacije (b) Primer **HSGV** situacije (slučaj veštačkih prolaza) [24]



(a) (b)

Slika 2.33: (a) Primer **HSWV** situacije (b) Primer **HSNV** situacije [24]

Model brzine se računa u zavisnosti od sigurnosti robota. Neka je v_{max} maksimum translacione brzine. Neka je d_{obs} rastojanje do najbliže prepreke od robotovih granica, i d_s bezbedno (sigurnosno) rastojanje. Neka je $\theta \in [-\pi/2, \pi/2]$ izračunat pravac translacione brzine.

1. Visoka sigurnost: $v = v_{max} \cdot Abs(1 - \theta/\frac{\pi}{2})$
2. Niska sigurnost: $v = v_{max} \cdot \frac{d_{obs}}{d_s} \cdot Abs(1 - \theta/\frac{\pi}{2})$

Sa ovom kontrolom brzine, robot se pomera maksimalnom brzinom sve dok jedna od prepreka ne upadne u sigurnosnu zonu, onda robot smanjuje brzinu proporcionalno rastojanju najbliže prepreke, dok sigurnosna zona nije očišćena od prepreka. Štaviše, velike promene u pravcu smanjuju modul translacione brzine, dok robot skreće pravac ka trenutnom pravcu kretanja [24].

Rotaciona brzina se računa iz pravca translacione brzine. Poželjno je da robot bude poravnat sa trenutnim pravcem kretanja. Neka je w_{max} maksimalna rotaciona brzina. Neka je $\theta \in [-\pi/2, \pi/2]$ izračunat pravac translacione brzine.

$$w = w_{max} \cdot \theta/\frac{\pi}{2} \tag{2.42}$$

Ovo daje nagla skretanja robota, kada su nagle promene ugla u translacionoj brzini (robot se okreće što je pre moguće pravcu kretanja) i blaga skretanja kada su promene male [24].

Kako bi se balansirala podela zakona kretanja i povećala glatkost tranzicija između nekih situacija, modifikacija Dijagrama blizine, DB+ (*ND+*) uvodi šestu generalnu situaciju **LSGR** (*Low Security Goal in Reagon*) [25, 26]. **LSGR** predstavlja situaciju niske sigurnosti u kojoj se može postići cilj. Prepreke su u ovoj situaciji u okviru sigurnosne zone, dok je cilj unutar oblasti slobodnog kretanja. Navigacija u **LSGR** udaljava robota od prepreke i usmerava ga direktno ka cilju [26].

Još jedna od modifikacija metoda Dijagrama blizine jeste Globalni dijagram blizine. Ova modifikacija daje globalno rezonovanje pristupu, nešto slično kao globalno proširenje dinamičkog prozora, ali zasnovana na reprezentaciji okruženja (umesto prostora konfiguracije) i ažuriranjem slobodnog prostora dodavanjem informacija o preprekama [1].

3. Globalno planiranje putanje (Globalna navigacija)

Kod globalnog planiranja putanje, okruženje je poznato unapred i teren je statičan ili su prepreke unapred poznate. Otuda je algoritam za planiranje putanje u mogućnosti da napravi kompletnu mapu okruženja od početne do ciljne tačke pre nego što samo kretanje počne [11].

Reprezentacija robotovog okruženja može varirati od kontinualnog geometrijskog opisa do geometrijske mape zasnovane na dekompoziciji ili čak topološke mape. Prvi korak bilo kog sistema planiranja putanje je da transformiše ovaj eventualno kontinualni model okruženja u diskretnu mapu pogodnu za odabrani algoritam planiranja putanje. Planeri putanja se razlikuju po tome kako utiču na ovu diskretnu dekompoziciju. Mogu se identifikovati tri opšte strategije dekompizicije:

1. Mapa puteva: identifikuje skup pravaca (ruta) unutar slobodnog prostora.
2. Ćelijska dekompozicija: razlikuje slobodne i zauzete ćelije.
3. Potencijalno polje: nameće matematičku funkciju nad prostorom.

Tekst u ovom poglavlju se zasniva na knjizi [1].

3.1 Metod mape puteva

Pristupi mapa puteva obuhvataju povezanost slobodnog prostora robota u jedno-dimenzionalnoj mreži krivih ili mreži linija, koja se naziva mapa puta. Kada je mapa puta izgrađena, ona se koristi kao mreža segmenata puta (putanje) za planiranje kretanja robota. Planiranje putanje je tako redukovano na povezivanju početne i ciljne pozicije robota na putnoj mreži, a zatim na pretragu niza puteva od početne pozicije robota do ciljne pozicije [1].

Mapa puteva je dekompozicija prostora konfiguracije robota zasnovana konkretno na geometriji prepreka. Izazov je da se izgradi skup puteva koji zajedno omogućavaju robotu da se kreće bilo gde u njegovom slobodnom prostoru, uz minimiziranje ukupnog broja puteva. Generalno, potpunost je sačuvana u takvim dekompozicijama sve dok je pravi stepen slobode robota obuhvaćen sa odgovarajućom tačnošću [1].

U daljem tekstu izdvojena su dva pristupa mapi puteva koja postižu taj rezultat sa drastično različitim vrstama puteva. U slučaju grafa vidljivosti (*visibility graph*), putevi su bliži preprekama koliko je to moguće, što kao rezultat daje putanje sa minimalnom dužinom. U slučaju Voronoi dijagrama, putevi ostaju što je dalje moguće od prepreka [1].

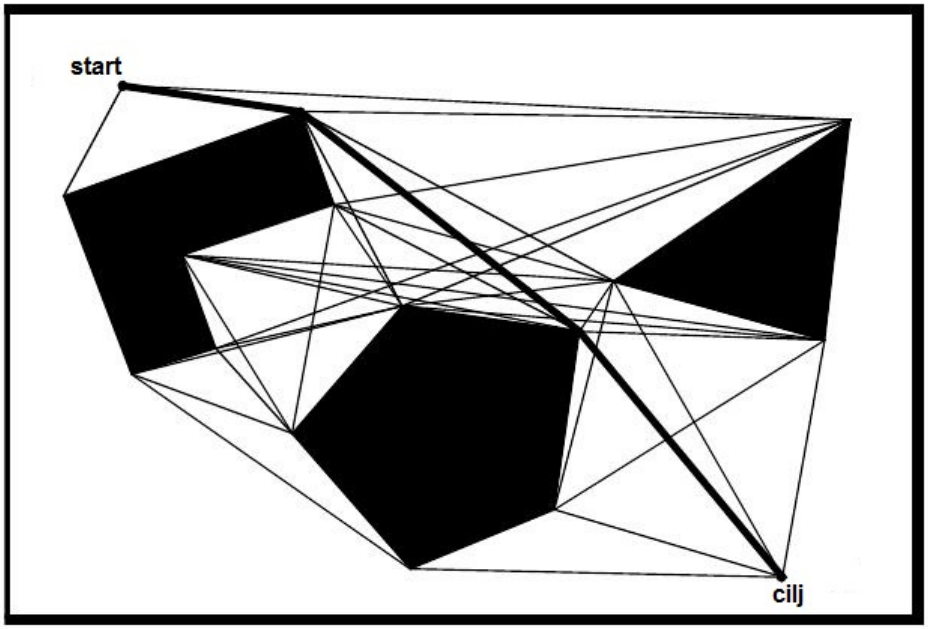
3.1.1 Graf vidljivosti

Graf vidljivosti (*Visibility graph*) za poligonalnu konfiguraciju prostora C se sastoji od ivica koje povezuju sve parove temena koji mogu da vide jedni druge (uključujući i početnu i ciljnu poziciju kao i temena). Nesmetane prave linije (putevi) koje spajaju ta temena su očigledno najkraća rastojanja između njih. Zadatak planera putanje je da pronađe najkraću putanju od početnog položaja do cilja duž puteva definisanih u grafu vidljivosti (Slika 3.1) [1].

Planiranje putanje grafom vidljivosti je umereno popularano u svetu mobilne robotike, delimično zbog prilično jednostavne implementacije. Posebno kada reprezentacija okruženja opisuje objekte u okruženju kao poligone bilo u kontinualnom ili diskretnom prostoru, jer su poligonski opisane prepreke dosta pogodne za pretragu grafa vidljivosti [1].

Postoje dve stvari na koje treba obratiti pažnju prilikom izvršavanja pretrage grafa vidljivosti. Prvo, veličina reprezentacije i broj ivica i čvorova se povećava sa brojem poligonskih prepreka. Stoga je metod veoma brz i efikasan u proređenim okruženjima, ali može biti spor i neefikasan u odnosu na druge tehnike, kada se koristi u gustim okruženjima [1].

Druga stvar na koju treba obratiti pažnju je mnogo ozbiljniji potencijalni propust: Rezultujuće putanje planiranja grafom vidljivosti imaju tendenciju da odvedu robota što je bliže moguće preprekama na putu do cilja. Formalnije, može se dokazati da je planiranje grafom vidljivosti optimalno u smislu dužine rezultujuće putanje. Ovaj moćni rezultat takođe znači da je osećaj sigurnosti, u smislu razumne udaljenosti od prepreka, žrtvovan za ovu optimalnost. Najprirodnije rešenje je da se prepreke uvećaju za znatno višu vrednost od poluprečnika robota, ili, alternativno, da se rezultujaća putanja modifikuje posle planiranja tako da distancira putanju od prepreka kada je to moguće. Naravno, takve akcije žrtvuju optimalne dužine dobijene planiranjem grafom vidljivosti [1].



Slika 3.1: **Graf vidljivosti.** Čvorovi grafa su inicijalne i ciljne tačke i temena prepreka konfiguracionog prostora (poligona). Svi čvorovi koji su međusobno vidljivi su povezani pravom linijom, definišući mapu puteva. To znači da postoje grane koje su ujedno i ivice poligona [1].

3.1.2 Voronoi dijagram

Metod Voronoi dijagrama, za razliku od pristupa grafa vidljivosti, je kompletan metod mape puteva koji teži da maksimizuje rastojanje između robota i prepreka na mapi. Na početku metod za svaku tačku u slobodnom prostoru izračunava udaljenost do najbliže prepreke. Nakon toga, metod gradi Voronoi dijagram tako što uzima tačke iz slobodnog prostora koje imaju ekvidistantnu udaljenost između jedne ili više prepreka. Primer Voronoi dijagrama prikazan je na slici 3.2. Kada su prepreke prostora konfiguracije poligoni, Voronoi dijagram se sastoji od ravnih i paraboličnih segmenata. Algoritmi koji pronalaze putanje na Voronoi mapi puteva su kompletni kao metode grafa vidljivost, jer postojanje putanje u slobodnom prostoru podrazumeva postojanje jedne na Voronoi dijagramu (tj. obe metode garantuju kompletnost). Međutim, putanja u Voronoi dijagramu je obično daleko od optimalne, u smislu ukupne dužine rezultujuće putanje [1].

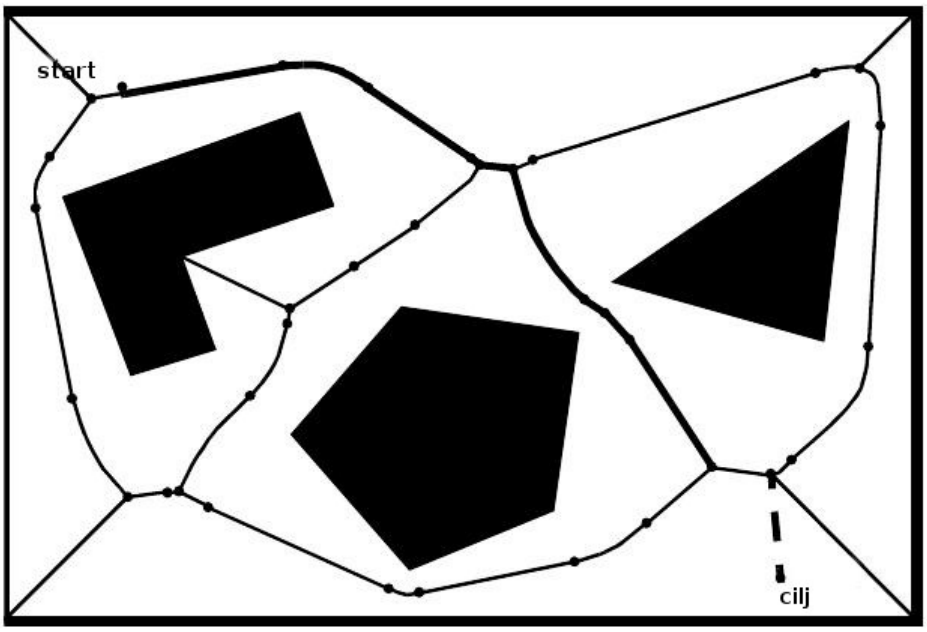
Voronoi dijagram ima znatnu slabost u slučaju lokalizacije senzora ograničenog dometa. Pošto ovaj algoritam planiranja putanje povećava rastojanje između robota i objekata u okruženju, bilo koji senzor kratkog dometa na robotu će biti u opasnosti da ne uspe da oseti svoju okolinu. Ako se koriste takvi senzori kratkog dometa za lokalizaciju, onda će izabrani put biti prilično slab sa tačke gledišta lokalizacije. S druge strane, metod grafa vidljivosti se može dizajniranti da zadrži robota što bliže objektima na mapi [1].

Postoji, međutim, važna suptilna prednost koju metod Voronoi dijagram ima u odnosu na većinu drugih tehnika za izbegavanje prepreka: mogućnost izvršavanja. Za putanju dobijenu planiranjem Voronoi dijagramom, robot sa nizom senzora, kao što su laserski daljinomeri ili ultrasonični, može pratiti Voronoi ivicu u fizičkom svetu koristeći jednostavna kontrolna pravila koja odgovaraju onima koja se koriste za kreiranje Voronoi dijagrama: robot maksimizuje čitanja lokalnog minimuma u vrednostima svojih senzora. Ovaj sistem kontrole će prirodno zadržati robota na Voronoi ivicama, tako da kretanje zasnovane na Voronoi-u može ublažiti enkoder netačnosti. Ova zanimljiva fizička osobina Voronoi dijagrama je korišćena da se sprovede automatsko mapiranje okruženja pronalaženjem i kretanjem na nepoznatim Voronoi ivicama, zatim gradeći konzistentnu Voronoi mapu okruženja [1].

3.2 Metod ćelijske dekompozicije

Ideja iza ćelijske dekompozicije je da se napravi razlika između geometrijskih oblasti ili ćelija koje su slobodne i oblasti koje su okupirane objektima. Osnovno planiranje putanje ćelijskom dekompozicijom se može opisati sledećim algoritmom:

- Podeliti F u jednostavne, povezane regione koji se nazivaju "ćelije".
- Utvrditi koje su otvorene ćelije susedne i izgraditi "graf povezanosti".



Slika 3.2: **Voronoi dijagram.** Voronoi dijagram se sastoji od linija konstruisanih od svih tačaka koje su ekvidistantne od dve ili više prepreka. Inicijalno q_{init} i cilj q_{goal} konfiguracije su mapirane u Voronoi dijagram u $q_{init}^{'}$ i cilj $q_{goal}^{'}$, svaka crtanjem linije čije rastojanje do prepreke najbrže raste. Pravac kretanja u Voronoi dijagramu se izabira takođe tako da rastojanje do prepreke najbrže raste. Tačke u Voronoi dijagramu predstavljaju tranzicije od duži (minimalnog rastojanja između dve linije) do paraboličkih segmenata (minimalno rastojanje između tačke i linije [1]
).

- Pronaći ćelije u kojima se nalaze početna i ciljna konfiguracija i pronaći putanju u grafu povezanosti koja spaja početnu i ciljnu ćeliju.
- Iz sekvence ćelija nađene odgovarajućim algoritmom pretrage, izračunati putanju unutar svake ćelije, na primer, prolazeći kroz središta granica ćelije ili sekvencom kretanja praćenjem ivica i kretanja duž pravih linija.

Važan aspekt metoda ćelijske dekompozicije je postavljanje granica između ćelija. Ako su granice postavljene kao funkcija strukture okruženja, tako da se dekompozicija vrši bez gubitka informacija, onda se metod naziva egzaktna ćelijska dekompozicija (*exact cell decomposition*). Ako dekompozicija rezultira aproksimaciji stvarne mape, sistem se naziva aproksimativna ćelijska dekompozicija (*approximate cell decomposition*) [1].

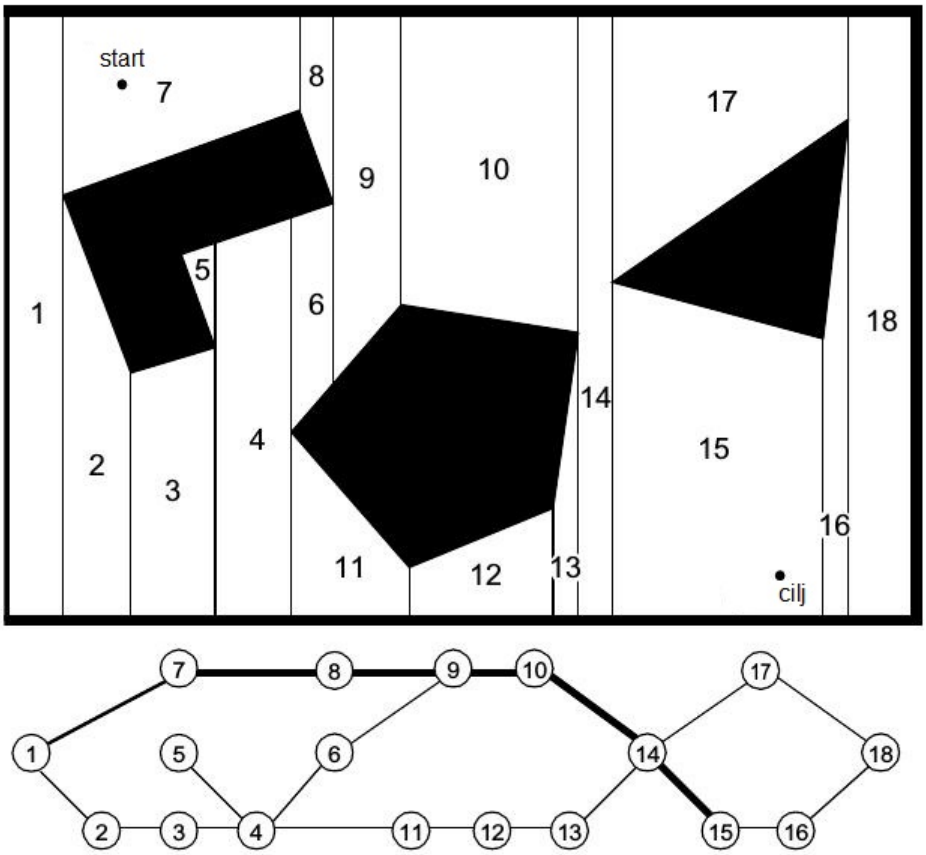
3.2.1 Egzaktna ćelijska dekompozicija

Slika 3.3 prikazuje egzaktnu ćelijsku dekompoziciju, pri čemu se granice ćelija zasnivaju na geometrijskoj kritičnosti. Dobijene ćelije su sve ili potpuno slobodne ili potpuno okupirane, i time je planiranje putanje u mreži završeno, kao i u metodama zasnovanim na mapi puteva. Osnovna apstrakcija iza ovakve dekompozicije jeste da je određena pozicija robota u svakoj ćeliji slobodnog prostora nebitna; ono što je bitno jeste zapravo mogućnost robota da za svaku slobodnu ćeliju pređe u njenu susednu slobodnu ćeliju [1].

Ključni nedostatak egzaktne ćelijske dekompozicije je broj ćelija i, stoga, ukupna efikasnost izračunavanja putanje, kao i na sistemima koji se zasnivaju na mapama puteva, zavisi od gustine i složenosti objekata iz okruženja. Ključna prednost je rezultat ove iste korelacije. U sredinama koje su izuzetno retke, broj ćelija će biti mali, čak i ako je geometrijska veličina okruženja veoma velika. Na taj način će reprezentacija biti efikasna u slučaju velikih retkih okruženja. Praktično govoreći, zbog kompleksnosti u implementaciji, tehnika egzaktne dekompozicije ćelija se koristi relativno retko u mobilnim aplikacijama robota, iako je i dalje dobar izbor kada je reprezentacija bez gubitka veoma poželjna, na primer, da se u potpunosti sačuva celovitost [1].

3.2.2 Aproksimativna ćelijska dekompozicija

Nasuprot egzaktnoj, aproksimativna ćelijska dekompozicija je jedna od najpopularnijih tehnika za planiranje putanje mobilnog robota. To je delom zbog popularnosti mrežno-zasnovanih reprezentacija



Slika 3.3: Primer egzaktne ćelijske dekompozicije [1]

okruženja. Ove reprezentacije okruženja su same po sebi dekompozicije mrežne veličine i kao takve su identične aproksimativnoj ćelijskoj dekompoziciji [1].

Najpopularniji oblik ovoga, prikazan na 3.4, je ćelijska dekompozicija korišćenjem fiksnih ćelija. Veličina ćelije ne zavisi od posebnih objekata u okruženju, pa uski prolazi mogu biti izgubljeni zbog neprecizne prirode mozaika. Praktično govoreći, ovo je retko problem zbog korišćenja ćelija veoma malih dimenzija (na primer, 5 cm sa svake strane). Velika korist od dekompozicije ćelijama fiksne veličine je niska složenost izračunavanja rezultujuće putanje [1].

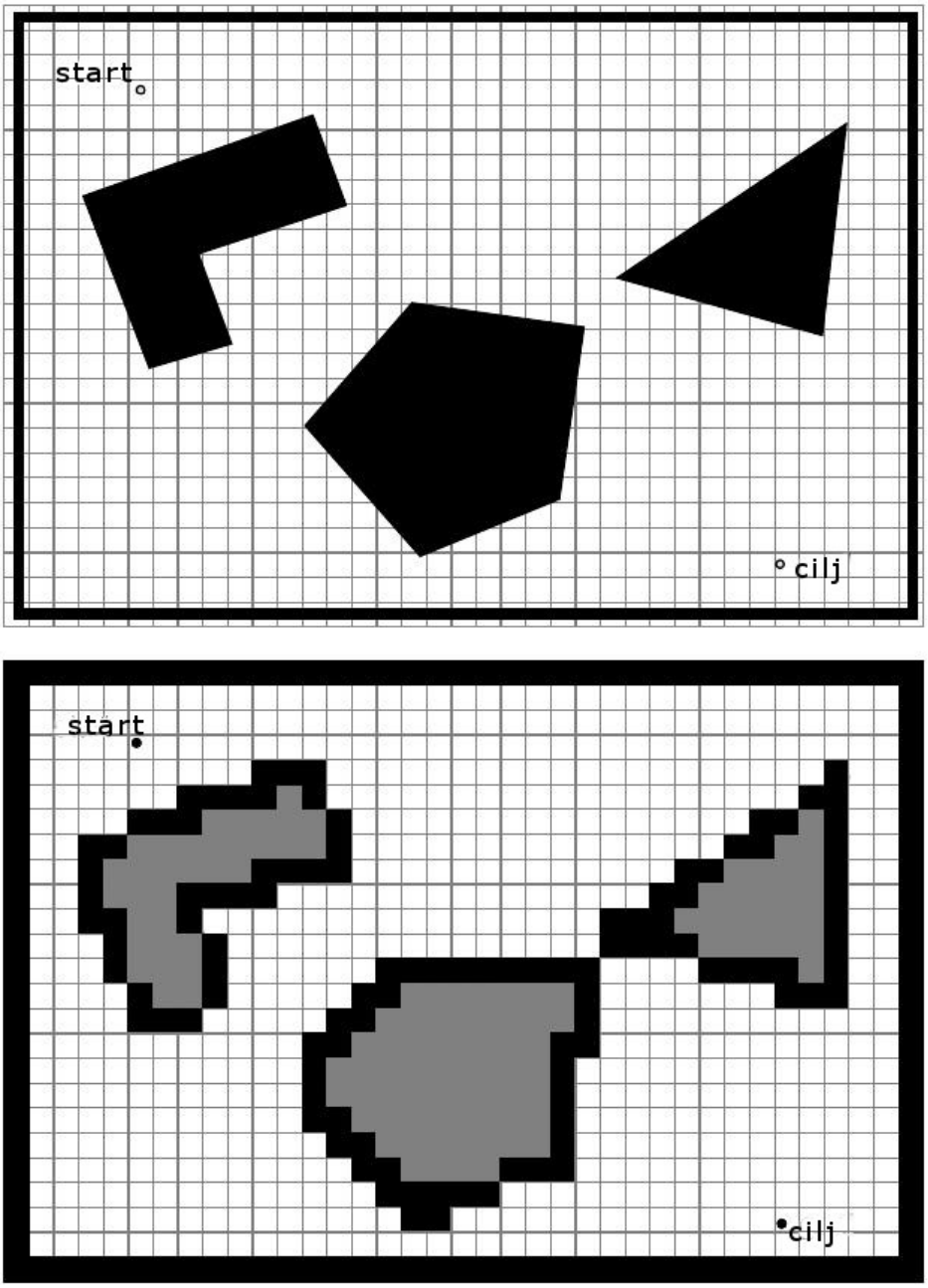
Na primer, $NF1$, je efikasna i jednostavna tehnika za implementaciju pronalaženja putanje u ćelijskim nizovima fiksne veličine. Algoritam jednostavno upošljava talasnu ekspanziju od ciljnog položaja ka spolja, obeležavajući svaku ćeliju sa svojom udaljenošću do cilja. Ovaj proces se nastavlja dok se ne dostigne ćelija koja odgovara početnoj poziciji robota. U ovom trenutku, planer putanje može proceniti rastojanje robota do ciljnog položaja, kao i da osmisli i specifično rešenje putanje jednostavno povezujući ćelije koje su susedne i uvek bliže cilju [1].

S obzirom da ceo niz može biti u memoriji, svaka ćelija je posećena samo jednom za vreme pretrage najkraće diskretne putanje iz inicijalne pozicije do ciljanog položaja. Dakle, pretraga je linearna samo po broju ćelija. Stoga kompleksnost ne zavisi od raštrkanosti i gustine okruženja, niti od složenosti oblika objekata u okruženju. Formalno, ova grasfajer transformacija je jednostavno pretraga u širinu implementirana u ograničenom prostoru niza susedstva [1].

Osnovna cena dekompozicije fiksnim ćelijama jeste memorija. Za velika okruženja, čak i raštrkana, ova mreža mora biti predstavljena u celini. Zbog pada cene memorije računara, ovaj nedostatak je ublažen poslednjih godina [1].

Slika 3.5 ilustruje aproksimativnu dekompoziciju ćelijama promenljive veličine. Okruženje je ograničeno spolja pravouganikom i iznutra sa četiri indentična poligona. Pravougaonik se rekurzivno deli na manje pravougaonike. Svaka dekompozicija generiše četiri identična nova pravougaonika. Na svakom nivou rezolucije samo ćelije čija unutrašnjost leže u potpunosti u slobodnom prostoru se koriste za izgradnju grafa povezanosti. Planiranje putanje u takvim adaptivnim reprezentacijama se može nastaviti u hijerarhijskom maniru [1].

Počevši sa grubom rezolucijom, rezolucija se redukuje sve dok planer putanje ne identifikuje rešenje ili dok se ne dostigne ograničenje rezolucije (npr., $k\bullet$ veličine robota). Za razliku od egzaktnog metoda ćelijske dekompozicije, aproksimativan pristup može žrtvovati kompletnost, ali je matematički manje zahtevan i na taj način je lakši za implementaciju. Aproksimativna ćelijska dekompozicija sa ćelijama različite veličine će

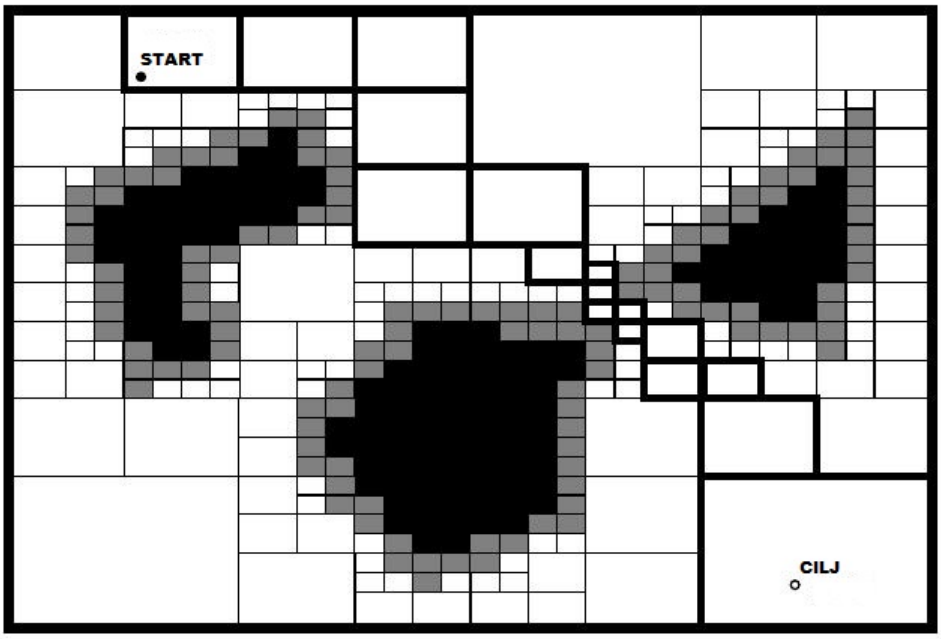


Slika 3.4: Fiksna ćelijska dekompozicija [1]

se, za razliku od aproksimativne dekompozicije sa fiksnim ćelijama, prilagoditi kompleksnosti okruženja, i samim tim će za proređena okruženja sadržati manji broj ćelija, konzumirajući drastično manje memorije [1].

3.3 Metod potencijalnog polja

Metod potecijalnog polja za planiranje putanje stvara polje ili gradijent preko mape robota koji usmerava robota do ciljane pozicije iz više prethodnih pozicija. Ovaj pristup je prvobitno izmišljen za planiranje putanje manipulator robota i često se koristi i u mnogim varijantama u zajednici mobilne robotike. Metod potencijalnog polja tretira robota kao tačku pod uticajem veštačkog potencijalnog polja $U(q)$. Robot se kreće tako što prati polje, baš kao što bi se lopta kotrljala nizbrdo. Cilj (minimum u ovom prostoru) deluje kao atraktivna sila na robota dok prepreke deluju kao nagibi, ili odbojne sile. Superpozicija svih sila je primenjena na robota, koji se, u većini slučajeva, posmatra kao tačka u prostoru konfiguracije (Slika 3.6). Takvo veštačko potencijalno polje glatkom putanjom vodi robota pravo ka cilju dok izbegava sve poznate prepreke [1].



Slika 3.5: **Primer aproksimativne ćelijske dekompozicije.** Pravougonik koji ograničava slobodan prostor je podeljen u četiri jednaka pravougaonika. Ako ceo pravougaonik leži u slobodnom prostoru ili u prepreci konfiguracionog prostora, ne deli se dalje. Inače, rekurzivno se deli u četiri pravougaonika sve dok se ne dobije neka unapred predefinisana rezolucija. Bele ćelije se nalaze van prepreka, crne iznutra, i sive kao deo oba regiona [1].

Važno je napomenuti, da je ovo više nego samo planiranje putanje. Rezultujuće polje je takođe zakon upravljanja robota. Pod pretpostavkom da robot može da lokalizuje svoju poziciju u odnosu na mapu i potencijalno polje, on uvek može da odredi svoju sledeću potrebnu akciju baziranu na polju [1].

Osnovna ideja koja stoji iza svih pristupa potencijalnih polja je da se robot privlači ka cilju, dok ga odbijaju prepreke koje su poznate unapred. Ako se pojave nove prepreke prilikom kretanja robota, može se ažurirati potencijalno polje u cilju integrisanja ove nove informacije. U najjednostavnijem slučaju, za robota se predstavlja da je tačka, zbog čega je orijentacija robota θ zanemarena i rezultujuće potencijalno polje dvodimenzionalno (x, y). Ako se pretpostavi diferencijabilna potencijalna funkcija polja $U(q)$, može se pronaći srodna veštačka sila $F(q)$ koja deluje na poziciju $q = (x, y)$.

$$F(q) = -\nabla U(q), \tag{3.1}$$

gde $-\nabla U(q)$ označava vektor gradijenta U na poziciji q.

$$\nabla U(q) = \begin{bmatrix} \frac{\partial U}{\partial x} \\ \\ \frac{\partial U}{\partial y} \end{bmatrix} \tag{3.2}$$

Potencijalno polje koje deluje na robota se onda računa kao suma privlačnog polja cilja i odbojnih polja prepreka:

$$U(q) = U_{att}(q) + U_{rep}(q) \tag{3.3}$$

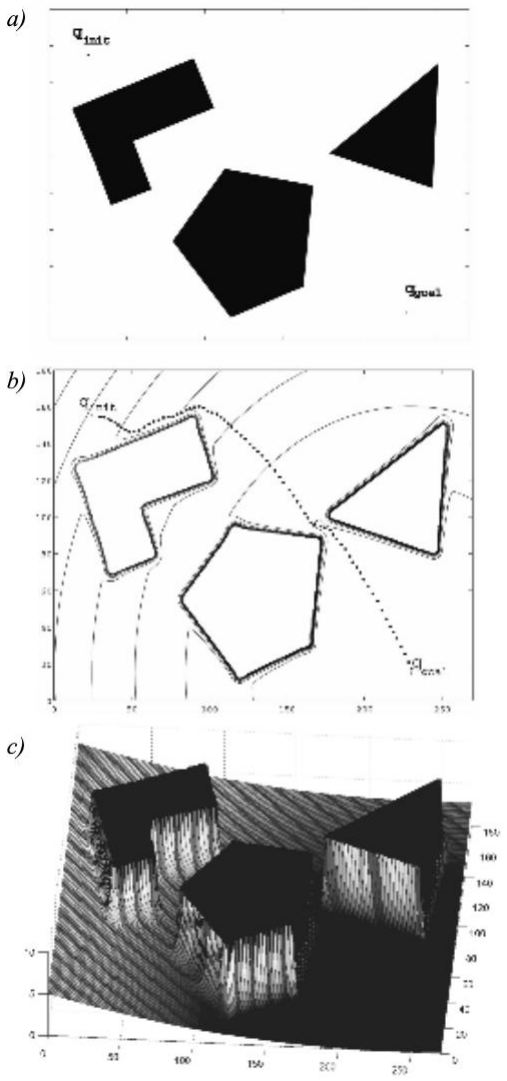
Slično, sile se mogu takođe odvojiti u privlačni i odbojni deo:

$$\begin{aligned} F(q) &= F_{att}(q) - F_{rep}(q) \\ &= -U_{att}(q) - U_{rep}(q) \text{ [1].} \end{aligned} \tag{3.4}$$

3.3.1 Privlačni potencijal

Privlačni potencijal može, na primer, biti definisan kao parabolička funkcija:

$$U_{att}(q) = \frac{1}{2} k_{att} \cdot \rho^2_{goal}(q), \tag{3.5}$$



Slika 3.6: **Tipično potencijalno polje generisano privlačenjem cilja i dve prepreke.** (a) Konfiguracija prepreka, start (gore levo) i cilj (dole desno). (b) Ekvipotencijalni grafik i putanja koja je generisala polje. (c) Rezultujuće potencijalno polje koje je generisano privlačenjem cilja i dve prepreke [1].

gde je k_{att} pozitivan faktor skaliranja i $\rho_{goal}(q)$ označava Euklidsko rastojanje $||q - q_{goal}||$. Ovaj Privlačni potencijal je diferencijabilan, što dovodi do privlačne sile F_{att}:

$$\begin{aligned} F_{att}(q) &= -\nabla U_{att}(q) \\ &= -k_{att} \cdot \rho_{goal}(q) \nabla \rho_{goal}(q) \\ &= -k_{att} \cdot (q - q_{goal}), \end{aligned} \tag{3.6}$$

koja konvergira ka 0, kako se robot približava cilju [1].

3.3.2 Odbojni potencijal

Ideja koja stoji iza odbojnog potencijala je da generiše silu koja odvlači od svih poznatih prepreka. Ovaj odbojni potencijal bi trebalo da bude veoma snažan kada je robot blizu objekta, ali ne bi trebalo da utiče na njegovo kretanje, dok je robot daleko od objekta. Jedan primer takvog odbojnog polja je:

$$U_{rep}(q) = \begin{cases} \frac{1}{2} k_{rep} (\frac{1}{\rho(q)} - \frac{1}{\rho_0})^2 & \text{ako } \rho(q) \leq \rho_0, \\ 0 & \text{ako } \rho(q) \geq \rho_0 \end{cases}, \tag{3.7}$$

gde je k_{rep} faktor skaliranja, $\rho(q)$ minimalno rastojanje od q do objekta i ρ_0 distanca uticaja objekta.

Funkcija odbijajućeg potencijala U_{rep} je pozitivna ili nula i teži ka beskonačnosti kako se q približava objektu.

Ako je granica objekta konveksna i diferencijabilna u delovima, $\rho(q)$ je diferencijabilna svuda u slobodnom prostoru konfiguracije. Ovo dovodi do odbojne sile F_{rep}:

$$F_{rep}(q) = -\nabla U_{rep}(q)$$
$$= \begin{cases} k_{rep}(\frac{1}{\rho(q)} - \frac{1}{\rho_0})\frac{1}{\rho^2(q)}\frac{q-q_{obstacle}}{\rho(q)} & \text{ako } \rho(q) \leq \rho_0, \\ 0 & \text{ako } \rho(q) \geq \rho_0 \end{cases} \tag{3.8}$$

Dobijena sila $F(q) = F_{att}(q) + F_{rep}(q)$ deluje na tačku koju je robot izložio atraktivnim i odbojnim silama koje pomeraju robota daleko od prepreka i prema cilju (Slika 3.6). Pod idealnim uslovima, postavljanjem vektora brzine robota proporcionalno vektoru sile polja, robot se može nesmetano navoditi ka cilju, slično kotrljanju lopte oko prepreka i nizbrdo [1].

Međutim, postoje neka ograničenja sa ovim pristupom. Jedno je lokalni minimum koji se pojavljuje u zavisnosti od oblika i veličine prepreka. Drugi problem se može pojaviti ako su objekti konkavni. To bi moglo da dovede do situacije u kojoj postoji više minimalnih rastojanja $\rho(q)$, rezultirajući u oscilaciji između dve najbliže tačke do objekata, što bi očigledno moglo da žrtvuje kompletnost [1].

3.3.3 Prošireni metod potencijalnog polja

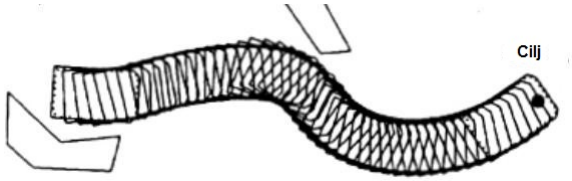
Khatib i Chatila su predložili prošireni metod potencijalnog polja. Kao i sve metode potencijalnog polja ovaj pristup omogućava korišćenje privlačnih i odbojnih sila koje potiču od veštačkog potencijalnog polja. Međutim, dve stvari su dodate osnovnom metodu potencijalnog polja: rotaciono potencijalno polje i potencijalno polje sa zadatkom [1].

Rotaciono potencijalno polje pretpostavlja da je odbojna sila funkcija rastojanja od prepreke i orijentacije robota u odnosu na prepreku. To se radi koristeći faktor pojačanja koji smanjuje odbojnu silu kada je prepreka paralelna sa pravcem kretanja robota, jer takav objekat ne predstavlja neposrednu pretnju putanji robota [1].

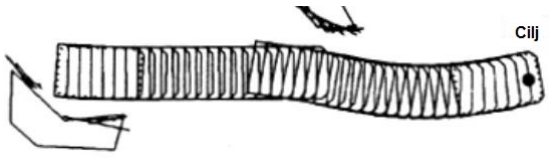
Rezultat je poboljšano praćenje ivica prepreka, što je bilo problematično za ranije implementacije metoda potencijalnih polja. Potencijalno polje sa zadatkom razmatra sadašnju brzinu robota i iz nje filtrira one prepreke koje ne bi trebalo da utiču na kratkoročni potencijal zasnovan na osnovu brzine robota. Ponovno skaliranje je napravljeno, ovaj put svih potencijalnih prepreka kada ne postoje prepreke u sektoru Z ispred robota. Z sektor se definiše kao prostor koji će robot "zagaziti" tokom narednog kretanja. Ovo daje glatke putanje kroz prostor. Primer koji upoređuje klasično potencijalno polje i prošireno potencijalno polje je prikazan na slici 3.7 [1].

Veliki broj varijacija i poboljšanja metoda potencijalnih polja je predloženo i implementirano od strane mobilnih robotičara. U većini slučajeva, ove varijacije imaju za cilj da se poboljšaju ponašanja potencijalnih polja u lokalnom minimumu istovremeno umanjujući šanse oscilacija i nestabilnosti, prilikom kretanja robota kroz uzak prostor [1].

Potencijalna polja su izuzetno laka za impementaciju, pa su postala uobičajna alatka u aplikacijama mobilnih robota uprkos svojim teorijskim ograničenjima [1].



(a) Klasični potencijal



(b) Rotacioni potencijal sa parametrom β

Slika 3.7: Poređenje klasičnog i proširenog poteciljalnog polja. [24]

4. Inteligentni kontrolni sistemi

Kod prethodno pomenutih "klasičnih" (konvencionalnih) metoda postoji mnogo propusta, kao što su veliko vreme izvršavanja u višim dimenzijama, zaglavljivanje u lokalnom minimumu, velika složenost izračunavanja, nemogućnost da se izbore za neizvesnostima okruženja stvarnog sveta (nepouzdanost mape, nepreciznost senzora [6, 27]). Neki od ovih problema se u gore pomenutim metodama rešavaju putem dobro definisane funkcije cene. Ovu funkciju obično podešavaju istraživači sve dok se robotovo ponašanje ne smatra prihvatljivim [10].

Inteligentni kontrolini sistemi su nova klasa tehnika koja dosta obećava u oblasti planiranja putanje. Ove tehnike su zasnovane na mogućnosti čovekovog mozga da obavlja kompleksne zadatke rezonovanjem, adaptiranjem i reagovanjem na promene u okruženju. Ovo metode zasnovane na učenju ponašanjem (*behaviour learning*) mogu se koristiti da reše kompleksne probleme kontrole sa kojima se autonomni roboti suočavaju u nepoznatim okruženjima stvarnog sveta [6]. Ovakvi prustupi zavise od trenutne situacije (stanja) i skupa ponašanja dizajniranih za to stanje, pa ih to čini pouzdanim u dinamičkim okruženjima stvarnog sveta [27].

U kontrastu sa klasičnim metodama, algoritmi zasnovani na inteligentnim kontrolnim sistemima ne garantuju postizanje rešenja, ali je sigurno da ako pronađu određeno rešenje, to bi bilo u kraćem vremenu u poređenju sa klasičnim metodama. Treba navesti da ove metode mogu otkazati u pronalaženju rešenja ili mogu naći loše rešenje (lokalni optimum). Neke od tehnika za koje se može reći da su tehnike inteligentne kontrole jesu neuronske mreže, fazi logika, genetski algoritmi i optimizacija kolonijom mrava [27].

4.1 Neuronske mreže

Interes za neurnoske mreže potiče od želje za razumevanjem principa koji vode u nekom maniru ka razumevanju osnovnih funkcija ljudskog mozga i izgradnje mašine koja može da izvršava kompleksne zadatake. U suštini, neuronska mreža se bavi kognitivnim zadacima kao sto su učenje, adaptacija, generalizacija i optimizacija. Zaista, prepoznavanje, učenje, donošenje odluka i izvršavanje akcija konstruišu glavne probleme navigacije. One poboljšavaju mogućnosti učenja i adaptacije vezane za varijacije u okruženjima gde su informacije kvalitativne, netačne, nesigurne i nekompletne. Obrada nepreciznih i podataka sa šumom neuronskom mrežom je više efikasno, nego obrada klasičnim tehnikama, jer su neuronske mreže dosta tolerantne na šum [11].

Neuronska mreža je masivni sistem paralernih distribuiranih procesorskih elemenata (neurona) povezanih u topologiji grafa. Učenje u neuronskoj mreži može biti nadgledano ili nenadgledano. Nadgledano učenje koristi kalsifikovan uzorak informacija, dok nenadgledani koristi samo minimum informacija bez neke preklasifikacije. Algoritmi nenadgledanog učenja daju manju složenost izračunavanja i manju preciznost nego algoritmi nadgledanog učenja. Znači, nenadgledani algoritmi uče brže, u jednom prolazu kroz podatke sa šumom. Neuronska mreža može da izrazi znanje implicitno u težinama, posle učenja. Matematički izraz koji je široko prihvaćena aproksimacija Hebianovog pravila učenja:

$$w_{i,j}(t+1) = w_{i,j}(t) + \eta x_i(t) y_j(t), \tag{4.1}$$

gde su x_i i y_j izlazne vrednosti neurona i i j, koji su povezani sinapsom $w_{i,j}$ i η je stopa učenja. Može se primetiti da je x_i ulaz u sinapsu [11].

Koncept korišćenja neuronske mreže za planiranje robotovog kretanja je prvi put je predstavljen u radu [28]. Biološki inspirisan pristup neuronskih mreža, za planiranje kretenja robota bez kolizija u realnom vremenu u dinamičkom okruženju, predstavljen je u radu [29]. Ovaj generalni model se može primeniti na "tačka" mobilne robote, manipulator robote i multi-robotske sisteme. Prostor stanja neuronske mreže je konfiguracioni prostor robota, dok je dinamčki varirano okruženje predstavljeno dinamički aktivnim pejzažom neuronske mreže. Cilj globalno privlači robota u celom prostoru stanja, dok prepreke lokalno guraju robota dalje od njih. Kretanje robota u realnom vremenu je planirano kroz dinamički aktivan pejzaž neurnoske mreže bez eksplicitne pretrage slobodnog prostora ili putanja bez kolizije, bez eksplicitne optimizacije bilo kakve funkcije cene, bez bilo kakvog prethodnog znanja o dinamičkom okruženju, bez nekog procesa učenja i bez ikakvih lokalnih procedura za proveru kolizija. Stoga je model algoritma računski efikasan [30].

U radu [31] je prikazan pristup neuronskih mreža za planiranje putanje dvodimenzionog kretanja robota. Takođe, u radu [32] predstavljen je pristup neuronske mreže za lokalnu navigaciju mobilnog robota preko mapa percepcije (*Perception maps*). Za potrebe zaobilaženja prepreka rekurentan tip neuronskih mreža je korišćen sa tehnikom povratne propagacije gradijenta za trening mreže [33]. Korišćenje nadgledane neuronske mreže za navigaciju robota u parcijalno poznatom okruženju predstavljeno je u članku [34]. Interesantno rešenje korišćenjem Džordanove arhitekture neuronskih mreža opisano je u radu [35]. Ovde robot uči interni model okruženja rekurentnom neuronskom mrežom, predviđa sukcesiju ulaza senzora i na osnovu modela generiše korak navigacije kao komandu motora [11]. Keš-genetski zasnovana modularna

fazi neurnoska mreža je predstavljena u radu [36] za planiranje putanje robota. Korišćenjem neuronske mreže radialne bazne funkcije (*Neural Radial Basis Function networks*) i A* algoritma razvijen je fleksiblini metod za planiranje putanje u realnom vremenu [37]. Model neuronske mreže je razvijen za kretanje robota u realnom vremenu i kontrolu robotskih manipluatora [38]. Problem planiranja kretenja robota je rešen u radu [39] korišćenjem Hopfildovih neuronskih mreža u fazifiranom (*fuzzified*) okruženju. Takođe 2003. godine, ne učeći, pristup veštačkih neuronskih mreža za planiranje kretenja za Pionir robota, je proširen [40]. U radu [41] predstavljen je pristup neuronskih mreža za dodeljivanje dinamičkih zadataka multi robotima. Pristup planiranja putanje za mobilnog robota zasnovan na *RL-ART2* neuronskoj mreži razvijen je 2007. godine [42, 30].

4.2 Fazi logika

Fazi logika je forma više-vrednosne logike koja se bavi aproksimativnim rezonovanjem [43]. Ona obezbeđuje formalnu tehniku za predstavljanje i implementiranje ljudskog heurističkog znanja i akcija zasnovanih na percepciji [6]. U poređenju sa tradicionalnom binarnom logikom, promenljive fazi logike mogu da imaju istinitosne vrednost koje se prostiru u stepenu između 0 i 1. Fazi logika je proširivana da podrži koncept parcijalne istine, gde istinitosne vrenodsti mogu da se prostiru od potpune istine do potpune neistine [43].

Mogućnosti fazi logike da koristi lingvističke varijable i da vrši pouzdano donošenje odluka uprkos nepouzdanim i nepreciznim informacijama čini je korisnim alatom u kontrolnim sistemima. Fazi kontrolni sistemi su sistemi zasnovani na pravilima ili sistemi zasnovani na znanju koji sadrže kolekciju fazi *IF-THEN* pravila baziranih na domenskom znanju ili ljudskoj ekspertizi. Jednostavnost fazi sistema baziranih na pravilima, mogućnost rada na širokom spektru zadataka bez eksplicitnih izračunavanja i merenja čine ih dosta popularnim među naučnicima i istraživačima [44].

Dakle, na osnovu jednostavnog dizajna, lake implementacije i pouzdanosti fazi kontrolnih sistema, mnogi pristupi su razvijeni da reše problem navigacije mobilnog robota u praćenju cilja, praćenju putanje, obilaženju prepreka, modeliranju okruženja, itd. [44].

Različiti pristupi zaobilaženja prepreka su razvijeni tokom poslednjih decenija koji su dali efektivna rešenja za problem navigacije u nepoznatim i dinamičkim okruženjima. U radu [45] predstavljen je dvoslojni fazi sistem zaključivanja u kome prvi sloj uzima senzorska čitanja. Levi i desni prolazi robota su dobjeni kao izlaz fazi sistema prvog sloja. Izlazi prvog sloja zajedno sa pravcem cilja su korišćeni kao ulaz za drugi sloj. Finalni izlaz kontrolera su linearna brzina i stopa skretanja robota. Drugi nivo fazi sistema zaključivanja koristi ponašanja zaobilaženja kolizija, praćenja prepreka i traženja cilja da bi postigao robusnu navigaciju u nepoznatim okruženjima. Pristup fazi kontrole za kooperativne mikro fudbalere robote je predstavljen u radu [46]. Planer je generisao putanju do destinacije a kontrola fazi logike robotov ciljni pravac koji je zaobilazio prepreke i druge robote uzimajući u obzir dinamičke pozicije prepreka, robota i lopte. Reaktivni navigacioni metod za omnidirekcione robote koji koristi fazi logiku opisan je u radu [47]. Fazi pravila su generisala komandu aktiviranja da bi se dobilo kretanje bez kolizija u dinamičkom okruženju. Fazi logika je takođe transparentni sistem koji je mogao biti podešen skupom pravila učenja ili manuelno. U radu [48] opisan je navigacioni metod zasnovan na ponašanju koristeći fazi logiku čija se navigaciona strategija sastojala od tri vrste ponašanja. Lokalno ponašanje za zaobilaženje prepreka se sadržalo od skupa pravila fazi logike koja generišu robotovu brzinu na osnovu rastojanja od prepreka. Kontrolni sistem koji se sastojao od fazi logičkog kontrolera i Petri mreže za multi robotsku navigaciju opisan je u radu [49]. Fazi pravila su upravljala robotom na osnovu distribucije prepreka ili pozicije cilja. Pošto pozicije prepreka nisu bile precizno poznate, fazi logika se u ovom slučaju pokazala kao pogodna tehnika za zadatak zaobilaženja prepreka u gustom okruženju. Kombinacija kontrolera fazi logike i skupa pravila za prevenciju kolizija, implementiranih kao model Petri mreže ugrađene u kontroler mobilnog robota, omogućilo mu je da obilazi i prepreke koje uključuju druge mobilne robote. Godine 2007. dizajniran je fazi kontroler za zaobilaženje prepreka autonomnog vozila korišćenjem negativnih fazi pravila [50]. Negativna fazi pravila definišu set akcija koja treba izbegavati da bi se vozilo usmerilo ka cilju u prisustvu prepreka. U radu [51] predočen je fazi kontrolni sistem za praćenje cilja i zaobilaženje prepeka za mobilnog robota. Donošenje odluka je kontrolisano fazi kontrolnom strategijom zasnovanom na podacima o okruženju dobijenih stereo sistemom vizije. Fazi kontroler zasnovan na sistemu vizije za zaobilaženje prepreka je predložen za humanoidnog robota u radu [52]. Najbliža prepreka robotu koju je sistem vizije primetio i ugao razlike između pravca cilja i robotovog pravca izmerenih pomoću električnih kompasa predstavljaju ulaz u fazi sistem na osnovu čega je dobijena odgovarajuća odluka o kretanju robota u nepoznatom okruženju [44].

4.3 Genetski algoritmi

Genetski algoritmi su probabilistički algoritmi pretrage zasnovani na mehanizmima prirodne selekcije i prirodne genetike. Uveo ih je Džon Holand početkom sedamdesetih godina dvadesetog veka. Genetski algoritam otpočinje skupom rešenja zvanim populacija. Rešenje je predstavljeno hromozomima. Veličina populacije se održava kroz svaku generaciju. U svakoj generaciji se funkcijom prilagođenosti izračunava

cena svakog hromozoma, i na osnovu te vrednosti se probabilistički biraju hromozomi za narednu generaciju. Neki od odabranih hromozoma se nasumično uparuju i proizvode potomstvo. Prilikom stvaranja potomstva mutacije i ukrštanja se nasumično odigravaju. Pošto hromozomi sa većom cenom imaju veću verovatnoću da budu izabrani, hromozomi nove generacije mogu imati veću prosečnu cenu u odnosu na hromozome iz starije generacije. Proces evolucije se ponavlja sve dok uslov nije zadovoljen (proizveden je maksimalan broj generacija ili je postignut zadovoljavajući nivo vrednosti funkcije prilagođenosti za populaciju) [53].

Genetski algoritam je tehnika pretrage koja se koristi da nađe egzaktna ili aproksimativna rešenja optimizacije ili problem pretrage. Genetski algoritmi su kategorizovani kao globalne heuristike pretrage. Genetski algoritmi su posebna klasa evoluconih algoritama koji koriste tehnike inspirisane evolucionom biologijom kao što su selekcija, mutacija, ukrštanje i nasleđivanje [27].

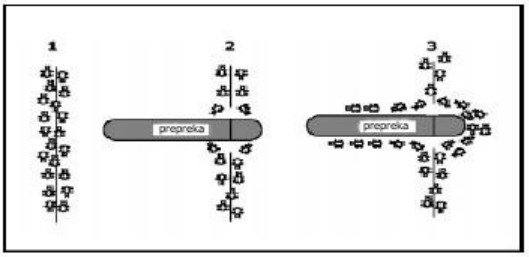
Selekcija je operacija koja rangira hromozome na osnovu njihove cene (ovde se cena odnosi na rastojanje do cilja) i vrši izbor visoko rangiranih hromozoma (oni koji predstavljaju kraću ili glađu putanju) za generisanje nove populacije. Ukrštanje je operacija koja koristi novo generisanu populaciju od hromozoma koje je izabrao operator selekcije. Tipično, operator ukrštanja deli izabrane hromozome u dva dela i razmenjuje te delove između njih u cilju stvaranja nove populacije. Mutacija je operacija koja nasumično, potpuno ili parcijalno menja sadržaj jednog ili više izabranih hromozoma. Operator mutacije se obično koristi kada populacija konvergira lokalnom optimumu, kad je zaglavljena između prepreka, ili kada se performanse nisu poboljšale za određeni broj generacija zbog nedostatka genetske različitosti [27].

Aplikacija genetskog algoritma za planiranje putanje mobilnog robota zahteva razvoj odgovarajućih hromozoma za reprezentaciju putanje, mehanizam za praćenje putanje, metod za zaobilaženje prepreka i prikladanu definiciju rešenja koja bi pružila mehanizme za minimiziranje rastojanja putanje, kao i glatke putanje [30].

Razne studije su urađene na osnovu korišćenja geneteskih algoritima u domenu planiranja putanje. U radu [54] predstavljen je genetski pristup za generaciju putanja bez kolizija. Prustup baziran na genetskom algoritmu za planiranje multi-putanja je prikazan u radu [55]. Još jedan pristup genetskog algoritma za rešavanje problema najkraće putanje opisan je u radu [56]. Genetski-fazi algoritam za navigaciju mobilnog robota kroz statičke prepreke predstavljen je u radu [57]. Višestruko planiranje putanja za grupu robota u 2D okruženju korišćenjem genetskog algoritma predstavljeno je u radu [58]. Impelemtacija višestrukog planiranja putanje za grupu robota u 3D okruženju korišćenjem genetskog algoritma predočena je u radu [59]. Optimalno planiranje putanje za mobilne robote bazirane na hibridnom genetskom algoritmu je opisano u radu [60]. U radu [61] predstavljeno je korišćenje modifikovanog genetskog algoritama za planiranje putanje neimenovanog vazdušnog vozila (*Unmanned aerial vehicle, UAV*) koji je za cilj imao generisanje putanja koje maksimiziraju sakupljene podatke iz regiona od interesa izbegavajući zabranjene regione. U sličnoj studiji, predstavljenoj u radu [62], istraživan je potencijal unapređene verzije genetskog algoritma u planiranju putanje leta za UAV ističući mogućnost genetskog algoritma da postigne globalni optimum kroz poboljšanje globalne i lokalne pretrage. Predložena modifikacija se fokusirala na diversifikaciji populacije genetskog algoritma korišćenjem koncepta dualne populacije [27].

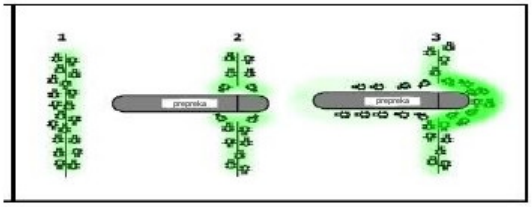
4.4 Optimizacija mravljom kolonijom

Algoritam optimizacije mravljom kolonijom (OMK) je pristup rešavanja problema optimizacije simuliranjem ponašanja pravih kolonija mrava. Optimizacija kolonijom mrava modelira ponašanje mrava, koji su poznati po tome da mogu da pronađu najkraću putanju od mravinjaka do izvora hrane. Mravi postižu ovo deponovajući supstancu zvanu feromon prilikom njihovog kretanja. Ovaj hemijski trag mogu detektovati drugi mravi, koji će verovatnije pratiti trag koji je bogat feromonima. Ova infomacija o tragu se može iskoristiti za prilogođavanje na nagle neočikavane promene terena, kao u slučaju kada prepreka blokira prethodno korišćen deo puta (Slika 4.1). Teskt u ovom odeljku se zasniva na radu [63].



Slika 4.1: Prepreke između mravinjaka i hrane [63].

Najkraća putanja oko takve prepreke bi probabilistički bila izabrana isto toliko često kao i duža putanja - međutim zbog toga što bi u jedinici vremena veći broj mrava putavao kraćom putanjom, ta putanja bi bila rekonstruisana mnogo brže (Slika 4.2).



Slika 4.2: Ostavljanje feromona omogućava mravima da rekonstruišu najkraću putanju [63].

Pošto su mravi više skloni da izaberu putanju sa višim nivoom feromona, mravi ubrzano konvengiraju ka jačem tragu feromona, i tako skreću sve više i više mrava putem kraćih putanja. Ovo pojedino ponašanje kolonija mrava je inspirisalo algoritam optimizacije kolonijom mrava, u kojem skup veštačkih mrava sarađuje u cilju pronalaženja rešenja za dati problem optimizacije deponovanjem tragova feromona kroz prostor pretrage. Postojeće implementacije algoritma se bave isključivo diskretnim prostorima pretrage i demonstrirali su pouzdanost i efikasnost da reše raznovrsne kombinatorne probleme optimizacije.

OMK algoritam je korišćen za rešavanje raznih problema navigacije i planiranja putanje mobilnog robota. Jedno od predloženih rešenja za planiranje putanje je koristilo OMK alagoritam za nalaženje najkraće optimalne putanje do cilja i bektrek (*backtrack*) algoritamsku strategiju za zaobilaženje prepreka. OMK algoritam je u jednoj studiji korišćen za navigaciju robota u dinamičkom okruženju. U ovoj studiji kretanja robota u dinamičkom okruženju, istraživači su uporedili dve različite sheme reinicijalizacije feromona i opisali najbolju od njih na osnovu rezultata simulacije. Algoritam OMK je takođe bio korišćen i za navigaciju robota u okruženjima sa statičkim preprekama, gde su za reprezentaciju okruženja bile korišćene mreže sa jednakim brojem kolona i redova.

5. Implementacija

Implementacija algoritama navigacije realizovana je programom koji je napisan korišćenjem programskog jezika C++ i aplikacionog okvira Qt. Algoritmi implementirani u programu su Bag1 (2.1.1), Lokalni dinamički prozor (2.5.1) i Aproksimativna ćelijska dekompozicija (3.2.2).

Pored implementacije algoritma za navigaciju, program se sastoji i od implementacije okruženja u kojem se odvija grafička simulacija algoritama, kao i od implementacije grafičkog korisničkog interfejsa koji omogućava kontrolu simulacije i ažuriranje okruženja.

5.1 Klasa NavigationAlgorithm

Apstraktna klasa `NavigationAlgorithm` predstavlja baznu klasu algoritama navigacije.

```
class NavigatonAlgorithm : public QObject
{
  Q_OBJECT

public:
    static const QStringList algorithmsList;

    NavigatonAlgorithm();
    virtual ~NavigatonAlgorithm() = default;

    NavigatonAlgorithm(const NavigatonAlgorithm&) = delete;
    NavigatonAlgorithm& operator=(const NavigatonAlgorithm&) = delete;
    NavigatonAlgorithm(const NavigatonAlgorithm&&) = delete;
    NavigatonAlgorithm& operator=(const NavigatonAlgorithm&&) = delete;


public slots:
    virtual void computeStep(QVector<QPolygonF> sensor, QPointF curr,
    QPointF ori, qreal robotAngle) = 0;
    virtual void resetAlgorithmState() = 0;
    virtual void setSimulationData(enviromentdefinitons::SimulationData algorithmData);

signals:
    void AlgorithamFinished(bool);
    void newStepCompputed(qreal, qreal);
    void simulationDataSet();

protected:
    qreal   _currTransVel;
    qreal   _currRotVel;

    QPointF _startPos;
    QPointF _goalPos;
    qreal   _goalThreshold;
    qreal   _robotThreshold;
    qreal   _sensorR;

    qreal   _maxRobotTransVel;
    qreal   _maxRobotRotVel;
    qreal   _maxRobotTransAcc;
    qreal   _maxRobotRotAcc;

    qreal   _enviromentWidth;
    qreal   _enviromentHeight;
    QList<QGraphicsItem*> _obstacles;
};
```

`NavigationAlgorithm` klasa sadrži atribute koji predstavljaju podatke o početnoj pozicji robota (`_startPos`), poziciji cilja (`_goalPos`), ograničenjima robotovih translacionih i rotacionih brzina i ubrzanja (`_maxRobotTransVel`, `_maxRobotRotVel`, `_maxRobotTransAcc`, `_maxRobotRotAcc`), poluprečniku opsega robotovog senzora (`_sensorR`), pragu udaljenosti od prepreka (`_robotThreshold`), pragu udaljenosti od cilja (`_goalThreshold`), dimenzijama i preprekama okruženja (`_enviromentWidth`, `_enviromentHeight`, `_eniviromentObstacles`) i trenutnoj translacionoj i rotacionoj brzini robota (`_currTransVel`, `_currRotVel`).

Metoda setSimulationData

Metoda `setSimulationData` je slot metoda u kojem se postavljaju vrednosti određenih atributa klase `NavigationAlgorithm` potrebnih za pokretanje algoritma. Vrednosti ovih atributa dobijene su putem parametra `simData`. Parametar `simData` je tipa strukture `SimulationData`.

Struktura `SimulationData` definisana je u domenu imena `enviromentdefinitons` i sastoji se od članova koji predstavljaju podatke o početnoj poziciji robota, poziciji cilja, ograničenjima robotovih translacionih i rotacionih brzina i ubrzanja, poluprečniku opsega robotovog senzora, pragu udaljenosti od prepreka, pragu udaljenosti od cilja i dimenzijama i preprekama okruženja.

Metoda je povezana sa signalom `startValuesReady` klase `Enviroment`(5.5).

Signal endSimulation

Signal `endSimulation` obaveštava okruženje da se algoritam završio. Parametar signala tipa `bool` predstavlja izlaz algoritma. Ako je vrednost parametra `false`, algoritam se završio neuspešno, u suprotom ako je vrednost paremetra `true`, algoritam se završio uspešno.

Signal newStepCompputed

Signalom `newStepCompputed` se obaveštava robot da je algoritam završio sa izračunavanjem rotacione i translacione brzine narednog korake navigacije. Signal prosleđuje dva parametra tipa `qreal` koji predstavljaju translacionu i rotacionu brzinu narednog koraka navigacije robota.

Signal simulationDataSet

Signal `simulationDataSet` obaveštava okruženje da je algoritam spreman za početak simulacije, što podrazumeva da su vrednosti atributa klase `NavigationAlgorithm` potrebnih za pokretanje algoritma postavljene.

5.2 Klasa Bug1

Klasa `Bug1` algoritam nasleđuje klasu `NavigationAlgorithm` i predstavlja implementaciju algoritma Bag1.

```
class Bug1Algorithm : public NavigatonAlgorithm
{

public:
    Bug1Algorithm();
    virtual ~Bug1Algorithm() = default;

    Bug1Algorithm(const Bug1Algorithm&) = delete;
    Bug1Algorithm& operator=(const Bug1Algorithm&) = delete;
    Bug1Algorithm(const Bug1Algorithm&&) = delete;
    Bug1Algorithm& operator=(const Bug1Algorithm&&) = delete;

public slots:
    virtual void computeStep(QVector<QPolygonF> sensor, QPointF curr,
                             QPointF ori, qreal robotAngle) override;
    virtual void resetAlgorithmState() override;

protected:
    enum class ALGORITHM_MODES {MOVE_TO_GOAL, FOLLOW_OBSTACLE, LEAVE_OBSTACLE};

    bool isObstacleInTheWay(qreal minDist);

    qreal minmumDistance(const QVector<QPolygonF>& sensor, const QPolygonF &spanPoly,
                         const QPointF &curr, const QPointF & innerPoint);

    qreal computeTransVelocity(const qreal &minDist, const qreal& threshold) const;
    qreal computeFolowGoalRotVelocity(const qreal& angle) const;
    qreal computeFollowObstacleRotVel(const QPointF &curr, const QPointF &ori,
                                      const QVector<QPolygonF>& sensor);

    void setFrontalSensorSpan(const QPointF &curr, const QPointF &ori);
    void setSideSensorSpan(const QPointF& curr, const QPointF& ori);

    bool isGoalReachable(const QPointF &hitPos, const QPointF &leavePos) const;

    QPolygonF _frontalSensorSpan;
    QPolygonF _leftSensorSpan;
```

```
    QPolygonF _rightlSensorSpan;

    ALGORITHM_MODES _mode;

    QPointF     _hitPos;
    QPointF     _hitPosErr;
    QPointF     _leavePos;
    bool        _hitPosLeft;
    bool        _hitErrSet;
    bool        _hitErrLeft;

    qreal       _minGoalDist;
};
```

Metoda computeStep

Metoda `computeStep` je implementacija čistog virtualnog metoda bazne klase `NavigationAlgorithm`. Metoda služi da se izračunaju translaciona i rotaciona brzina narednog koraka kretanja robota. `computeStep` je slot metoda i povezuje se sa signalom `computeNextStep klase Robot`(5.6), putem koga dobija podatke o trenutnom očitavanju senzora, trenutnoj poziciji i orijentaciji robota, i trenutnom uglu između robota i cilja u odnosu na x-osu okruženja.

U metodi se prvo proverava da li je robot stigao do cilja, i ukoliko jeste, putem signala `AlgorithmFinished` okruženje se obaveštava da je algoritam uspešno izvršen. Nakog prosleđivanja signala za završetak algoritma, izlazi se iz metode.

Ukoliko robot nije stigao do cilja proverava se trenutni režim rada algoritma. Režim rada algoritma određen je vrednošću klasne promenljive `_mode`, koja je nabrojanog tipa `ALGORITHM_MODES`.

Ukoliko je vrednost promenljive `_mode` jednaka `MOVE_TO_GOAL`, algoritam je u režimu rada kretanja ka cilju. U režimu kretanja ka cilju prvo se proverava da li postoje prepreke na putu robota. Postojanje prepreki se određuje pozivanjem metoda `minimumDistance`, koji računa minimalno rastojanje do prepreka i `isObstacleInTheWay`, koji na osnovu minimalnog rastojanja utvrđuje da li ima ili nema prepreka na putu robota. U slučaju da nema prepreka na putu, pomoću metoda `computeTransVelocity` i `computeFolowGoalRotVelocity` izračunavaju se nove brzine kretanja koje usmeravaju robota ka cilju. U slučaju da postoji prepreka na putu, trenutna pozicija robota se čuva kao udarna tačka sa preprekom i `_mode` dobija vrednost `FOLLOW_OBSTACLE`.

Ukoliko je vrednost promenljive `_mode` jednaka `FOLLOW_OBSTACLE`, algoritam je u režimu rada praćenja prepreke. U režimu rada praćenja prepreke, se pomoću metoda `computeTransVelocity` i `computeFollowObstacleRotVel`, izračunavaju translaciona i rotaciona brzina koje će voditi robota uz ivicu prepreke. Nakon izračunavanja brzina, na osnovu rastojanja do cilja utvrđuje se da li je trenutna pozicija bolji kandidat od trenutnog kandidata za tačku napuštanja prepreke. Ukoliko je na trenutnoj poziciji rastojanje do cilja manje, trenutna pozicija postaje novi kandidat tačke napuštanja prepreke. Nakon određivanja tačke napuštanja proverava se da li je robot ponovo naišao na tačku udara. Ukoliko je naišao na tačku udara, vrši se provera da li je tačka udara jednaka tački napuštanja, i ukoliko jeste metoda signalom `AlorithmFinished` obaveštava okruženje da se algoritam završio neuspešno. U suprotonom, ako tačka napuštanja nije jednaka tački udara, algoritam prelazi u režim rada napuštanja prepreke i vrednost proomenljive `_mode` se postavlja na `LEAVE_OBSTACLE`.

Ukoliko je vrednost promenljive `_mode` jednaka `LEAVE_OBSTACLE`, algoritam je u režimu rada napuštanja prepreke. U režimu reda napuštanja prepreke proverava se da li je robot stigao do tačke napuštanja. Ukoliko jeste prelazi se u režim rada kretanja ka cilju, a ukoliko nije uz pomoć metoda `computeTransVelocity` i `computeFollowObstacleRotVel` izračunava se nova translaciona i rotaciona brzina.

Posle izvršavanja instrukcija vezanih za određeni režim rada, signalom `NewStepComputed` robotu se prosleđuju rotaciona i translaciona brzina narednog koraka kretanja.

Metoda isObstacleInTheWay

Metoda `isObstacleInTheWay` ima parametar `minDist` koji predstavlja minimalno rastojanje između prepreka i robota. Upoređivanjem parametra **minDist** sa pragom udaljenosti od prepreka `_robotThreshold`, određuje se da li ima prepreka na putu. Prepreke postoje na putu, ako je **minDist** manji od `_robotThreshold`, u suprotnom nema prepreka na putu.

Metoda minmumDistance

Metoda `minimumDistance` služi da se izračuna minimalno rastojanje između delova prepreka koje se nalaze u opsegu senzora i robota. Metodi se prosleđuju delovi prepreka kao lista poligona putem parametra `sensor`. Minimalno rastojanje se određuje tako što se pronalazi minimalno rastojanje između robota i svake ivice svakog poligona iz `sensor`.

Metoda computeTransVelocity

Metoda `computeTransVelocity` služi za izračunavanje translacione brzine kretanja robota. Ukoliko se robot približava prepreci ili znatno približava cilju, rezultat je brzina čiji je intenzitet manji ili jednak od intenziteta translacione brzine trenutnog koraka kretanja. U slučajevima gde nema opasnosti od prepreka i cilj nije u blizini, rezultat je brzina čiji je intenzitet veći ili jednak od intenziteta brzine trenutnog koraka kretanja.

Metoda computeFolowGoalRotVelocity

U metodi `computeFolowGoalRotVelocity` izračunava se rotaciona brzina narednog koraka kretanja koje usmerava robota ka cilju.

Metoda computeFollowObstacleRotVel

Metoda `computeFollowObstacleRotVel` se koristi za izračunavanje rotacionih brzina u režimu rada praćenja prepreke. Rezultat metode je brzina koje će voditi robota duž ivice prepreke. U slučaju da se robot previše udaljava od prepreke, rezultat je rotaciona brzina suprotnog smera koja će dovoljno približiti robota prepreci. U slučaju da se robot previše približava prepreci, rezultat je rotaciona brzina koja će dovoljno udaljiti robota od prepreke.

5.3 Klasa DynamicWindow

Klasa `DynamicWindow` algoritam nasleđuje klasu `NavigationAlgorithm` i predstavlja implementaciju algoritma Lokalni dinamički prozor.

```
class DynamicWindow : public NavigatonAlgorithm
{

public:
    using VelocityTuple = QPair<qreal, qreal>;

    DynamicWindow();
    virtual ~DynamicWindow() = default;

    DynamicWindow(const DynamicWindow&) = delete;
    DynamicWindow& operator=(const DynamicWindow&) = delete;
    DynamicWindow(const DynamicWindow&&) = delete;
    DynamicWindow& operator=(const DynamicWindow&&) = delete;

public slots:
    virtual void computeStep(QVector<QPolygonF> sensor, QPointF curr,
                             QPointF ori, qreal robotAngle) override;
    virtual void resetAlgorithmState() override;
    virtual void setSimulationData(enviromentdefinitons::
                                   SimulationData algorithamData) override;
protected:
    void createPossibleVelocitiesSet(const qreal &maxRobotTransVel,
                                     const qreal &maxRobotRotVel);

    void createDynamicWindow();

    void createAdmissibleVelocitiesSet(const QVector<QPolygonF> &sensor,
                                       const QPointF &curr, const QPointF &ori,
                                       const qreal &robotAngle);

    void chooseVelocityCandidate(const qreal &goalDistance, const qreal& goalAngle);


    qreal obsDistanace(const VelocityTuple &velTuple,
                       const QVector<QPolygonF> &sensor,
                       const QPointF &curr, const qreal& robotAngle) const;
    qreal obsDistanace(const QVector<QPolygonF> &sensor,
                        const QPointF &curr, bool& onceFlag) const;
    qreal obsDistanace(const QVector<QPolygonF> &sensor,
                        const QPointF &curr,const QPointF &ori) const;

    QPolygonF setFrontalSensorSpan(const QPointF &curr, const QPointF &ori) const;
    qreal     setThrersholdAngle(const QPointF &curr, const QPointF & ori) const;

    qreal speedCost(const qreal& transVel, const qreal &goalDistance) const;
    qreal goalCost(const qreal& rotVel, const qreal &goalAngle) const;
```

```
    qreal costFunction(const VelocityTuple &velTuple, const qreal& obsDistance,
                       const qreal& goalDistance, const qreal &goalAngle) const;


    bool isVelTupleAdmisible(const VelocityTuple& velTuple, const QPointF& curr,
                             const QPointF& ori, const qreal &minDistance,
                             const qreal &robotAngle);

    qreal _maxDistanceValue;

    qreal   _loopTime;

    QVector<VelocityTuple> _possibleVelocties;
    QVector<VelocityTuple> _dynamicWindow;
    QVector<QPair<VelocityTuple, qreal> > _admisibleVelocities;
};
```

Metoda computeStep

Metoda `computeStep` je implementacija čistog virtualnog metoda bazne klase `NavigationAlgorithm`. Metoda služi da se izračunaju translaciona i rotaciona brzina narednog koraka kretanja robota. Metoda `computeStep` je slot metoda i povezuje se sa signalom `computeNextStep` klase `Robot` (5.6), putem koga dobija podatke o trenutnom očitavanju senzora, trenutnoj poziciji i orijentaciji robota, i trenutnom uglu između robota i cilja u odnosu na x-osu okruženja.

U metodi se prvo proverava da li je robot stigao do cilja, i ukoliko jeste, putem signala `AlgorithmFinished` se obaveštava okruženje da je algoritam uspešno izvršen. Nakon ovoga metoda se završava.

Ukoliko robot nije stigao do cilja pozivanjem metode `createDynamicWindow` se kreira dinamički prozor. Nakon toga se od dinamičkog prozora pozivom metode `createAdmissibleVelocitiesSet` kreira skup prihvatljivih brzina.

Ukoliko skup prihvatljivih brzina nije prazan, iz njega se putem metode `chooseVelocityCandidate` biraju naredna translaciona i rotaciona brzina robota. Novo odabrane brzine se prosleđuju robotu putem signala `NewStepComputed`.

Ukoliko je skup prihvatljivih brzina prazan, signalom `AlorithmFinished` se šalje informacija okruženju da se algoritam završio neuspešno.

Metoda createPossibleVelocitiesSet

Metoda `createPossibleVelocitiesSet` se koristi za kreiranje skupa mogućih translacionih i rotacionih brzina. Skup mogućih brzina predstavljen je vektorom torki `_possibleVelocties`. Prvi element torke je translaciona, a drugi element rotaciona brzina. Radi poboljšanja performansi algoritma, broj translacionih i rotacionih brzina skupa mogućih brzina je ograničen konstantama.

Metoda createDynamicWindow

Metoda `createDynamicWindow` služi za formiranje dinamičkog prozora. Dinamički prozor je predstavljen vektorom torki translacionih i rotacionih brzina `_dynamicWindow`. Formiranje dimičkog prozora se izvršava primenom 2.29 na translacione i rotacione brzine torki iz skupa mogućih brzina.

Metoda createAdmissibleVelocitiesSet

Metoda`createAdmissibleVelocitiesSet` kreira skup prihvatljivih brzina primenjujući 2.28 na translacione i rotacione brzine torki iz dinamičkog prozora. Skup prihvatljivih brzina je predstavljen kao vektor parova torki translacionih i rotacionih brzina i njihovih rastojanja do najbliže prepreke `_admisibleVelocities`. Rastojanja se pored torki brzina čuvaju radi daljih koraka izračunavanja.

Metoda chooseVelocityCandidate

Metoda `chooseVelocityCandidate` odabira par iz skupa prihvatljivih brzina sa torkom brzina koji ima najveću vrednost funkcije cene. Vrednost funkcije cene torke brzina se izračunava pozivom metode `costFunction`.

Metoda costFunction

Metoda `costFunction` izračunava funkciju cene 2.31. Parametar *Speed* funkcije cene se dobija pozivanjem metode `speedCost`, koja izračunava parametar primenom 2.32. Parametar *Goal* funkcije cene se dobija pozivanjem metode `goalCost`, koja izračunava parametar primernom 2.33.

5.4 Klasa AproximateCellDecompostion

Klasa `AproximateCellDecompostion` algoritam nasleđuje klasu `NavigationAlgorithm` i predstavlja implementaciju algoritma Aproksimativna ćelijska dekompozicija.

```
class AproximateCellDecomposition : public NavigatonAlgorithm
{
public:
    AproximateCellDecomposition();
    virtual ~AproximateCellDecomposition() = default;

    AproximateCellDecomposition(const AproximateCellDecomposition&) = delete;
    AproximateCellDecomposition& operator=(const AproximateCellDecomposition&) = delete;
    AproximateCellDecomposition(const AproximateCellDecomposition&&) = delete;
    AproximateCellDecomposition& operator=(const AproximateCellDecomposition&&) = delete;

public slots:
    virtual void computeStep(QVector<QPolygonF> sensor, QPointF curr,
                             QPointF ori, qreal robotAngle) override;
    virtual void resetAlgorithmState() override;

protected:
    enum class ALGORITHM_MODE {CONSTRUCT_PATH, FOLLOW_PATH};

    qreal computeTransVelocity(const qreal &currGoalDistance) const;
    qreal computeRotVelocity(const qreal &currGoalAngle) const;

    void constructPath();
    bool isCellOccupied(const QRectF &cellRect) const;

    ALGORITHM_MODE _mode;
    QPointF        _currGoal;

    bool _rotate;
    bool _translate;

    QVector<QPointF> _path;
};
```

Metoda computeStep

Metoda `computeStep` je implementacija čistog virtualnog metoda bazne klase `NavigationAlgorithm`. Metoda se koristi za konstruisanje putanje do cilja i izračunavanje translacione i rotacione brzine narednog koraka kretanja robota. `computeStep` je slot metoda i povezuje se sa signalom `computeNextStep` klase `Robot`(5.6), putem koga dobija podatke o trenutnom očitavanju senzora, trenutnoj poziciji i orijentaciji robota, i trenutnom uglu između robota i cilja u odnosu na x-osu okruženja.

U metodi se prvo proverava da li je robot stigao do cilja, i ukoliko jeste, putem signala `AlgorithmFinished` obaveštava se okruženje da je algoritam uspešno izvršen. Nakon ovoga metoda se završava.

Ukoliko robot nije stigao do cilja proverava se trenutni režim rada robota. Režim rada robota određen je vrednošću klasne promenljive `_mode`, koja je nabrojanog tipa `ALGORITHM_MODES`.

Ukoliko je vrednost promenljive `_mode` jednaka `CONSTRUCT_PATH`, algoritam je u režimu rada konstruisanja putanje. Algoritam je u režimu rada konstrukcije putanje samo jednom na početku izvršavanja. Konstrukcija putanje vrši se pozivom metode `constructPath`. Ukoliko `constructPath` ne uspe da konstruiše putanju, signalom `AlorithmFinished` šalje se informacija okruženju da se algoritam žavršio neuspešno. Ukoliko `constructPath` uspe da konstruiše putanju, `_mode` dobija vrednost `FOLLOW_PATH` i algoritam prelazi u režim rada kretanja po putanji.

Ukoliko je vrednost promenljive `_mode` jednaka `FOLLOW_PATH`, algoritam je u režimu rada kretanja po putanji. U režimu rada kretanja po putanji pozivanjem metoda `computeTransVelocity` i `computeRotVelocity` računaju se translaciona i rotaciona brzina narednog koraka kretanja robota. Metode `computeTransVelocity` i `computeRotVelocity` daju brzine koji usmeravaju robota da se kreće po konstruisanoj putanju. Novoizračunate brzine prosleđuju se robotu putem signala `NewStepComputed`

Metoda constructPath

Metoda `constructPath` se koristi za konstrukciju putanje od početne pozicije robota do cilja. U metodi se prvo vrši diskretizacija okruženja na mrežu, čije su ćelije pravougaonici istih dimenzija. Nakon diskretizacije, ćelije mreže se obeležavaju na okupirane i slobodne. Ćelija je okupirana ako se u njoj nalazi prepreka, u suprotnom ćelija je slobodna. Rezultat obeležavanja ćelija se čuva u celobrojnoj matrici `grid`, gde element matrice `grid[i][j]` ima vrednost 1, ukoliko je ćelija na poziciji i, j okupirana, a ukoliko je ćelija slobodna vrednost 0.

Matrica `grid` se dalje, zajedno sa početnom pozicijom robota i pozicijom cilja, prosleđuje metodi `pathFind` definisanom u domenu imena `aStar`. Metoda `pathFind` pronalazi putanju do cilja koristeći algoritam A*. Implementacija algoritma A* preuzeta je sa [20]. Ukoliko putanju nije moguće pronaći `pathFind` vraća praznu putanju.

Nakon poziva metode `pathFind` rezultujuća putanja se smešta u atribut klase `_path`

5.5 Klasa Enviroment

Klasa `Enviroment` predstavlja okruženje u kome se odvija grafička simulacija algoritma. `Enviroment` nasleđuje klasu Qt okvira `QGraphicsScene`. `QGraphicsScene` klasa se koristi za vizualizaciju velikog broja dvodimenzionalnih grafičkih objekta. Klasa `Enviroment` vizualizuje objekte koji predstavljaju robota, cilj i prepreke.

Pored vizualizacije, klasa `Enviroment` takođe ima ulogu i u kontroli toka simulacije. Klasa sadrži metode `startSimulation`, `pauseSimulation` i `stopSimulation` koji pokreću, pauziraju i zaustavljaju simulaciju [64].

```
class Enviroment : public QGraphicsScene
{
    Q_OBJECT
public:
    Enviroment();
    virtual ~Enviroment();

    Enviroment(const Enviroment&) = delete;
    Enviroment& operator=(const Enviroment&) = delete;
    Enviroment(const Enviroment&&) = delete;
    Enviroment& operator=(const Enviroment&&) = delete;

    Robot* robot();

    bool containsItem(const QGraphicsItem*) const;

    void addMainItem(enviromentdefinitons::ENV_MAIN_ITEM_TYPE type, QPointF pos);

    QPointF goalPosition() const;

public slots:
    void prepareForSimulation();
    void startSimulation();
    void pauseSimulation();
    void stopSimulation();

    void preformGraphicsSceneMousePress(QGraphicsSceneMouseEvent*);
    void preformGraphicsSceneMouseMove(QGraphicsSceneMouseEvent*);
    void preformGraphicsSceneMouseRealese(QGraphicsSceneMouseEvent*);

    void clear();
    void deleteSelecetedItem();

protected slots:
   void endSimulation(bool);

signals:
    void simulationStarted();
    void simulationStopped();
    void simulationMessage(QString);
    void simulationEnded();

    void actionChanged();

    void mousePressed(QGraphicsSceneMouseEvent*);
    void mouseMoved(QGraphicsSceneMouseEvent*);
    void mouseRealesed(QGraphicsSceneMouseEvent*);

    void startValuesReady(enviromentdefinitons::SimulationData);

protected:
    void mousePressEvent(QGraphicsSceneMouseEvent*) override;
    void mouseMoveEvent(QGraphicsSceneMouseEvent*) override;
    void mouseReleaseEvent(QGraphicsSceneMouseEvent*) override;

private:
    QTimer _timer;
```

```
    Robot* _robot;
    Goal _goal;

    bool _paused;
```

Metoda prepareForSimulation

Metoda `prepareForSimulation` se poziva pre započinjanja simulacije. Prvo se proverava da li su i robot i cilj u okruženju. Ukoliko samo robot, ili samo cilj, nije u okruženju, signalom `simulationMessage` se grafičkom korisničkom interfejsu prosleđuje statusna poruka o tome da simulaciju nije moguće započeti. Ukoliko su i robot i cilj u okruženju, signalom `startValuesReady` se algoritmu prosleđuju podaci potrebni za njegovo pokretanje.

Metoda endSimulation

Slot metoda `prepareForSimulation` je povezana sa signalom `AlgorithamFinished` klase `NavigationAlgoritham` (5.1), preko koga dobija izlaz iz algoritma. U metodi najpre se zaustavlja simulacija, a nakon toga na osnovu dobijenog izlaza šalje se odgovarajuća statusna poruka grafičkom korisničkom interfejsu.

Signal simulationMessage

Signalom `simulationMessage` se grafičkom korisničkom interfejsu prosleđuje poruka o statusu simulacije. Dobijena poruka se na grafičkom korisničkom interfejsu prikazuje u vidu dijalog prozora.

5.6 Klasa Robot

Klasa `Robot` je implementacija elementa okruženja koji predstavlja robota. Klasa `Robot` nasleđuje klasu `QGraphicsObject` Qt programskog okvira. `QGraphicsObject` je implementacija grafičkog objekta klase `QGraphicsScene`, koji ima mogućnost korišćenja mehanizma signala i slotova [65, 66].

```
class Robot : public QGraphicsObject
{
    Q_OBJECT

public:

    Robot();
    virtual ~Robot();

    Robot(const Robot&) = delete;
    Robot& operator=(const Robot&) = delete;
    Robot(const Robot&&) = delete;
    Robot& operator=(const Robot&&) = delete;

    QRectF boundingRect() const override;
    QPainterPath shape() const override;

    virtual int type() const override;

    void paint(QPainter *painter, const QStyleOptionGraphicsItem *option,
               QWidget *widget) override;

    QPointF startPosition() const;
    void setStartPosition(const QPointF &startPosition);

    QPointF goalPosition() const;
    void setGoalPosition(const QPointF &goalPosition);

    QPointF currentPos() const;

    QVector<QPolygonF> getSensorData();

    void resetPosition();

    qreal angle() const;
    QPointF orientation() const;

    qreal threshold() const;

    qreal sensorRadius() const;
```

```
    virtual bool collidesWithItem(const QGraphicsItem *other,
                                  Qt::ItemSelectionMode mode) const override;

    void prepareForSimulation(const QPointF& goalPos);
    void resetAfterSimulation();

    void setSimulationRunning(bool);

    qreal maxTransVelocity() const;

    qreal maxRotVelocity() const;

    qreal maxTransAcc() const;

    qreal maxRotAcc() const;

signals:
    void computeNextStep(QVector<QPolygonF> collisions,QPointF, QPointF, qreal);
    void robotCrashed(bool);
    void newPathLength(qreal);

public slots:
    void setNewVelocities(qreal transVel, qreal rotVel);

protected:
    void setUpRobotRobotThreshold();
    void setUpRobotMovmentParameters();

    void initPath();

    virtual void advance(int step) override;

    QRectF detectiomRect() const;

    void colidesWithEnviromentEdges(QVector<QPolygonF>& polygons);

    void drawPath();

    qreal _sensorRadius;
    qreal _signalSpan;

    QPointF _startPosition;
    QPointF _goalPosition;

    qreal _height;
    qreal _width;

    QColor _color;

    qreal _transVelocity;
    qreal _rotVelocity;
    qreal _rotAngle;
    bool  _newVelComputed;

    bool _simulationRunning;
    bool _robotCrashed;

    qreal _threshold;

    qreal _maxTransVelocity;
    qreal _maxRotVelocity;
    qreal _maxTransAcc;
    qreal _maxRotAcc;

    QGraphicsPathItem* _robotPath;
};
```

Klasa `Robot` sadrži atribute koji predstavljaju trenutnu rotacionu i translacionu brzinu (`_rotVelocity`, `_transVelocity`), kao i atribute koji predstavljaju maksimalne vrednosti translacionih i rotacionih brzina i ubrzanja (`_maxTransVelocity`, `_maxRotVelocity`, `_maxTransAcc`, `_maxRotAcc`). Poluprečnik senzora i prag udaljenosti od prepreka predstavljeni su atributima `_sensorRadius` i `_threshold`.

Metoda getSensorData

Metodom `getSensorData` se dobijaju trenutna očitavanja robotovog senzora. Očitavanja senzora se iz metode vraćaju u vidu vektora poligona. Prvo se proverava da li postoji kolizija senzora sa jednom ili više prepreka iz okruženja. Ukoliko kolizija postoji, formiraju se isečci od delova prepreka koji se nalaze u opsegu senzora. Isečci se dodaju u prazan vektor poligona. Vektor sa isečcima se dalje prosleđuje klasnoj metodi `colidesWithEnviromentEdges`. U metodi `colidesWithEnviromentEdges` se proverava kolizija senzora sa ivicama okruženja, i ukoliko kolizija postoji, u prosleđen vektor se dodaju isečci ivica okruženja koji se nalaze u opsegu senzora. Nakon poziva `colidesWithEnviromentEdges` vektor sa isečcima se vraća iz metode.

Metoda setNewVelocities

Metoda `setNewVelocities` je slot metoda koja se povezuje sa signalom `newStepCompputed` klase `NavigationAlgorithm` (5.1), putem kojeg dobija translacionu i rotacionu brzinu narednog koraka kretanja robota. Dobijene vrednosti brzina se dodeljuju atributima `_rotVelocity` i `_transVelocity`. Posle dodeljivanja vrednosti brzina, klasnom atributu `_newVelComputed` tipa `bool` se dodeljuje vrednost `true`, kako bi se naznačilo robotu da su vrednosti brzina za naredni korak kretanja dobijene.

Metoda advance

Metoda `advance` je reimplementacija metode natklase QGraphicsObject i poziva se u svakom koraku simulacije. Metoda izvršava korak kretanja navigacije robota. Ukoliko je vrednost atributa `_newVelComputed` jednaka `false`, vrednosti brzina za naredni korak kretanja još uvek nisu dobijene od algoritma i izlazi se iz metode. Ukoliko je vrednost atributa `_newVelComputed` jednaka `true`, korišćenjem translacione i rotacione brzine vrši se izračunavanje nove pozicije i orijentacije robota. Nakon dodele nove pozicije i orijentacije robotu, poziva se metod `getSensorData` kako bi se dobila trenutna očitavanja senzora. Očitavanja senzora, zajedno sa novom pozicijom, novom orijentacijom i uglom između robota i cilja se prosleđuju algoritmu za navigaciju putem signala `computeNextStep`.

Metoda paint

Metoda `paint` je reimplementacija metode natklase i definiše korake iscrtavanja robota u okruženju.

Signal computeNextStep

Signalom `SignalcomputeNextStep` se obaveštava algoritam da je potrebno izračunati translacionu i rotacionu brzinu za naredni korak kretanja robota. Signal putem parametra prosleđuje vrednosti trenutnog očitavanja senzora, trenutne pozicije i orijentacije robota, i trenutnog ugla između robota i cilja u odnosu na x-osu okruženja.

5.7 Korišćenje programa

Pokretanjem programa dobija se prozor kao na slici 5.1. Prozor se sastoji od bele pravougaone površine i palete alatki. Bela pravougona površina predstavlja okruženje u kojem se odvija grafička simulacija algoritma za navigaciju. Okruženju se mogu dodati objekti kao što su robot, cilj i prepreke. Simulacija se ne može pokrenuti ukoliko okruženju nisu dodati robot i cilj. Okruženje je ograničeno svojim ivicama, te se robot ne može kretati van okruženja, pa se ivice tokom simulacije posmatraju kao prepreke (zidovi). Robot je u okruženju predstavljen kao zeleni petougao, a njegov opseg senzora kao crvena isprekidana kružnica. Cilj je predstavljen kao crveni krug, dok su prepreke predstavljene kao plavi poligoni. Za vreme simulacije u okruženju se iscrtava putanja koju je robot prešao. Putanja je predstavljena kao isprekidana ljubičasta linija (Slika 5.2).

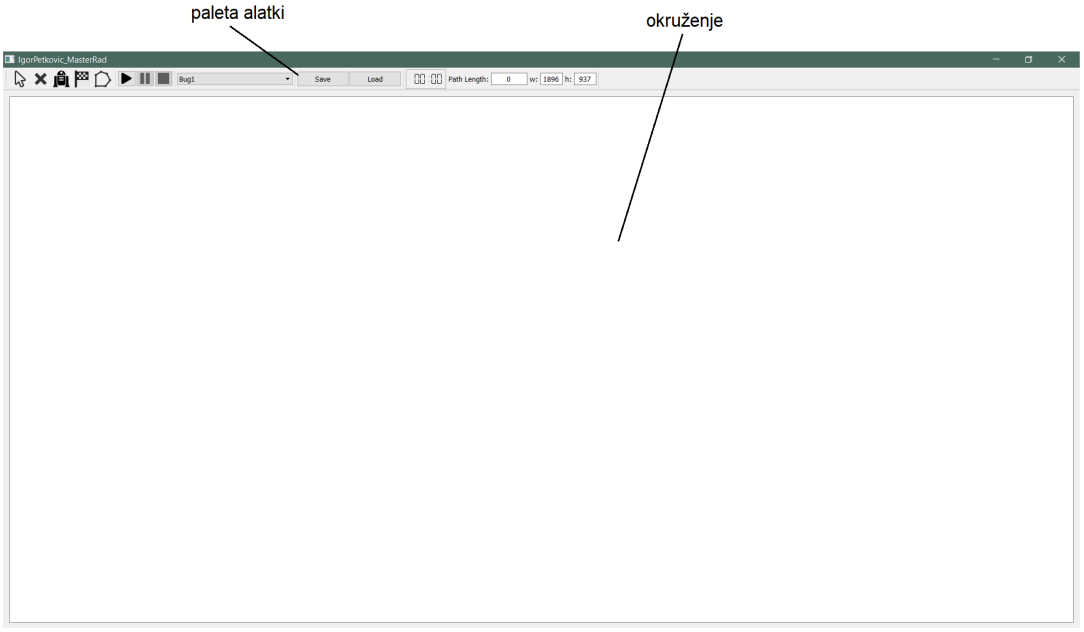
Iznad okruženja, na prozoru, nalazi se paleta alatki. Paleta alatki sadrži elemente koji omogućavaju kontrolu toka simulacije i ažuriranje okruženja, i elemente koji prikazuju informacije o simulaciji i okruženju.

Paleta alatki se sastoji od pet delova (Slika 5.3). U prvom delu (gledano sa leva na desno) nalaze se alatke koje služe za upravljanje objektima okruženja. Prva alatka u ovom delu služi za pomeranje objekata po okruženju, dok druga ima ulogu brisanja objekta. Trećom i četvrtom alatkom dodaje se robot, odnosno cilj okruženju, dok se poslednja koristi za crtanje prepreka.

Naredni deo palete sadrži tri dugmeta kojima se kontroliše simulacija. Prvim dugmetom se pokreće, drugim pauzira, a trećim zaustavlja simulacija. Zaustavljanjem simulacije robot se postavlja na početnu poziciju i uklanja se prethodno iscrtana putanja.

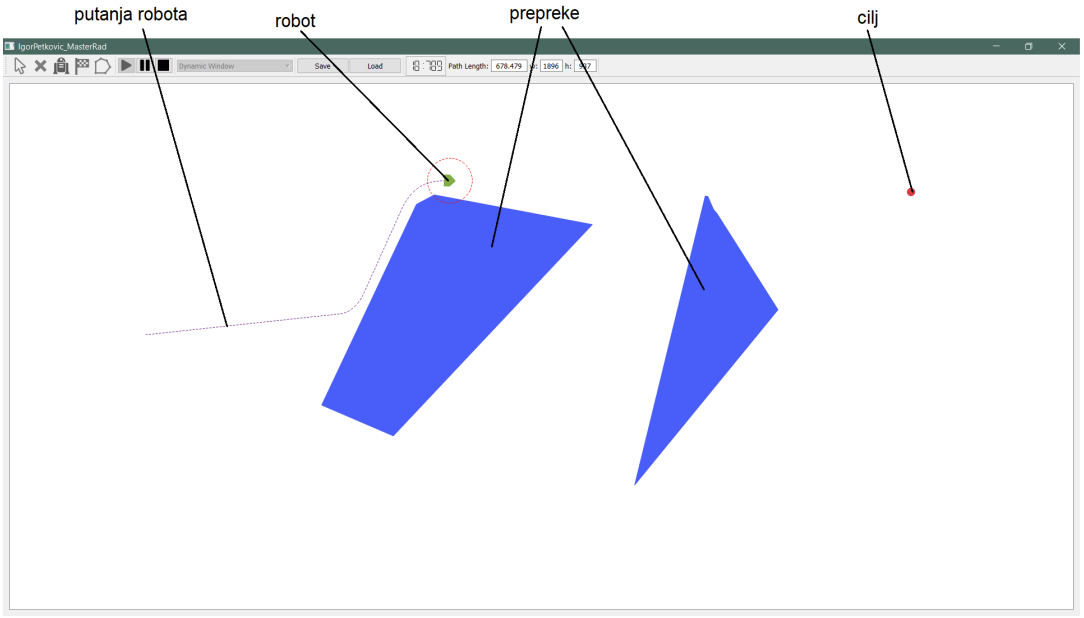
U trećem delu palete nalazi se padajuća lista koja kao izbor nudi imena svih implementiranih algoritama navigacije u programu. Lista se koristi za odabir algoritma koji će se izvršavati.

Četvrti deo palete sastoji se od dva dugmeta *save* i *load*. Dugme *save* upisuje trenutno okruženje u JSON datoteku. U JSON datoteci se čuvaju dimenzije prozora, pozicije cilja, prepreka i robota. Dugme *load* učitava sačuvano okruženje iz JSON datotake.



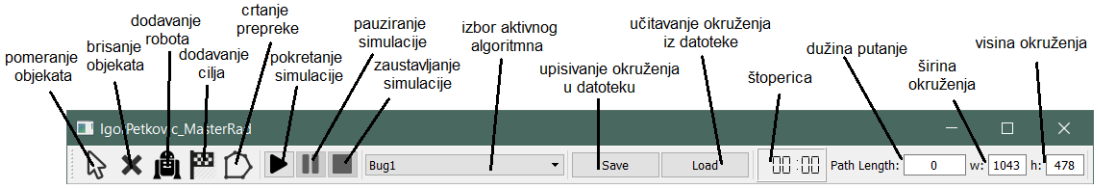
Slika 5.1: Početni prozor programa

.



Slika 5.2: Program u toku simulacije

.



Slika 5.3: Paleta alatki programa

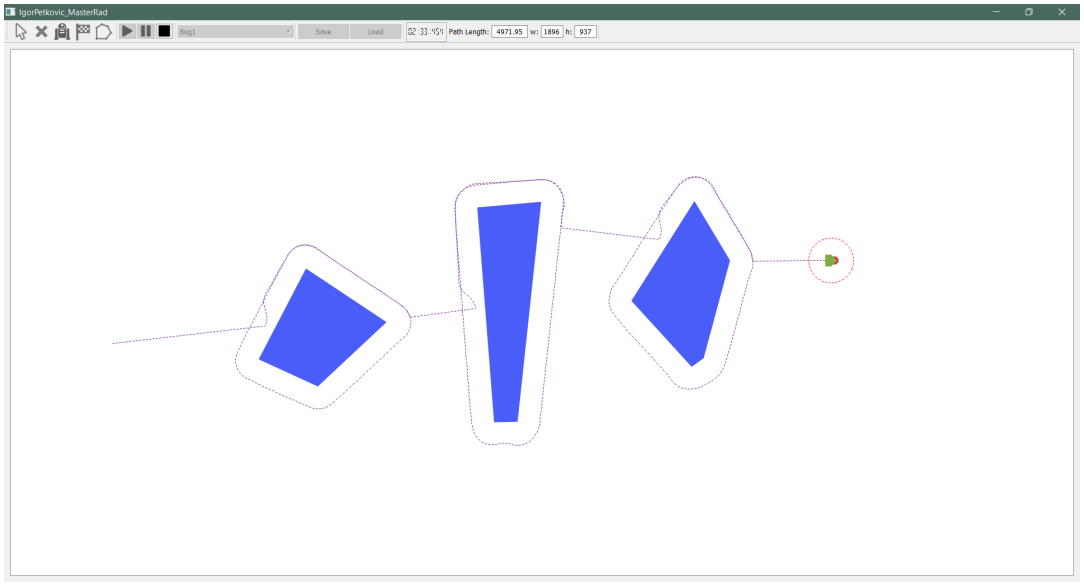
.

U poslednjem delu palete alatki nalaze se elementi koji prikazuju informacije o simulaciji i okruženju. Prvi element na poslednjem delu je štoperica koja prikazuje proteklo vreme simulacije. Pored štoperice nalazi se tekstualno polje označeno labelom *Path Length*, koje tokom simulacije prikazuje trenutnu dužinu putanje robota. Poslednja dva elementa čine dva tekstualna polja u kojima su prikazana trenutna visina i širina okruženja.

6. Rezultati Simulacije

Simulacija algoritama za navigaciju je izvršena u tri različita okruženja. Prvo okruženje je retko popunjeno i sadrži samo tri prepreke.

Robot vodjen algoritmom Bag1 u prvom okruženju je morao da obiđe sve tri prepreke kako bi stigao do cilja. Zbog kretanja oko čitavih prepreka algoritam Bag1 ima najlošije vreme izvršavanja i najdužu putanju u odnosu na ostale simulirane algoritme (Slika 6.1).



Slika 6.1: Rezultat algoritma Bag1 u retko popunjenom okruženju

Algoritam lokalnog dinamičkog prozora u prvom okruženju ima najkraće vreme izvršavanja, ali ne i najkraću putanju. Zbog svoje čiste lokalne prirode dinamčki prozor vodi robota efikasno oko prepreka, ali ne i najbližim putem do cilja (Slika 6.2).

Aproksimativna ćelijska dekompozicija u prvom okruženju ima najkraću putanju do cilja, ali ne i najkraće vreme izvršavanja. Zbog izvršavanja na diskretnoj mreži pravougaonika aproksimativna ćelijska dekompozicja daje kratku ali dosta oštru putanju. Oštar oblik putanje utiče na sporije kretanje robota tokom simulacije (Slika 6.3).

U drugom okruženju izvršena je simulacija koja se sastoji od velikog broja prereka, koje ispunjavaju veći deo površine okruženja.

Robot vođen algoritmom Bag1 u drugom okruženju mora da obiđe pet prepreka kako bi stigao da cilja. Obilazak oko tolikog broja prepreka ponovo daje najgori rezultat u poređenju sa ostala dva algoritma (Slika 6.4).

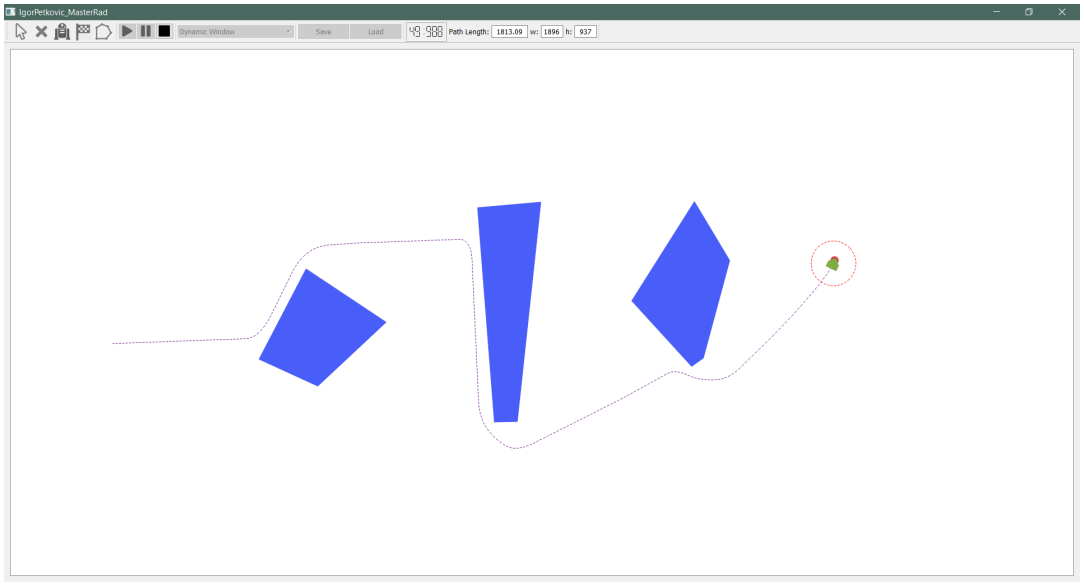
Lokalni dinamički prozor i u drugom okruženju daje najkraće vreme izvršavanja i putanju koja na određenim mestima vodi robota dalje od cilja (Slika 6.5).

Aproksimativna ćelijska dekompozicija i u drugom okruženju daje kratku oštru putanju i nešto duže vreme izvršavanja od algoritma lokalnog dinamičkog prozora (Slika 6.6).

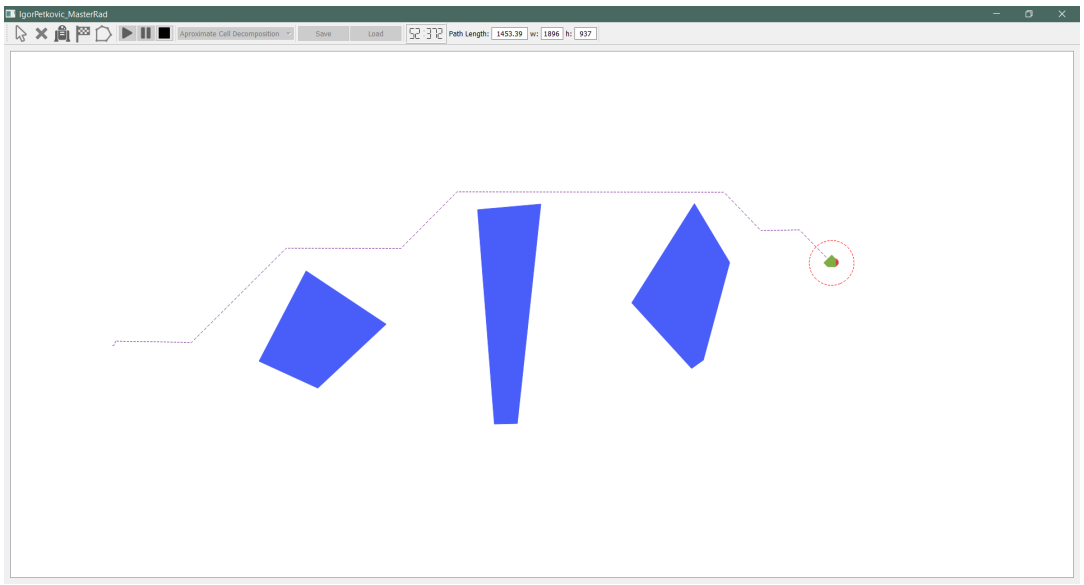
Treće okruženje deli prepreke tako da izgleda kao da se sastoji od praznih prostorija. U poređenju sa prvim i drugim okruženjem, treće okruženje bliže predstavlja realan problem navigacije robota.

Algoritam Bag1 zbog svoje lokalne prirode i čuvanja malog broja podataka o okruženju, čitavo treće okruženje posmatra kao jednu veliku prepreku. Zbog ovakve percepcije, robot u režimu rada praćenja prepreke algoritma Bag1 obiđe ivice svih prepreka u okruženju. Rezultat ovakvog ponašanja algoritma daje znatno duže vreme izvršavanja i znatno dužu putanju u poređenju sa ostalim algoritmima (Slika 6.4)

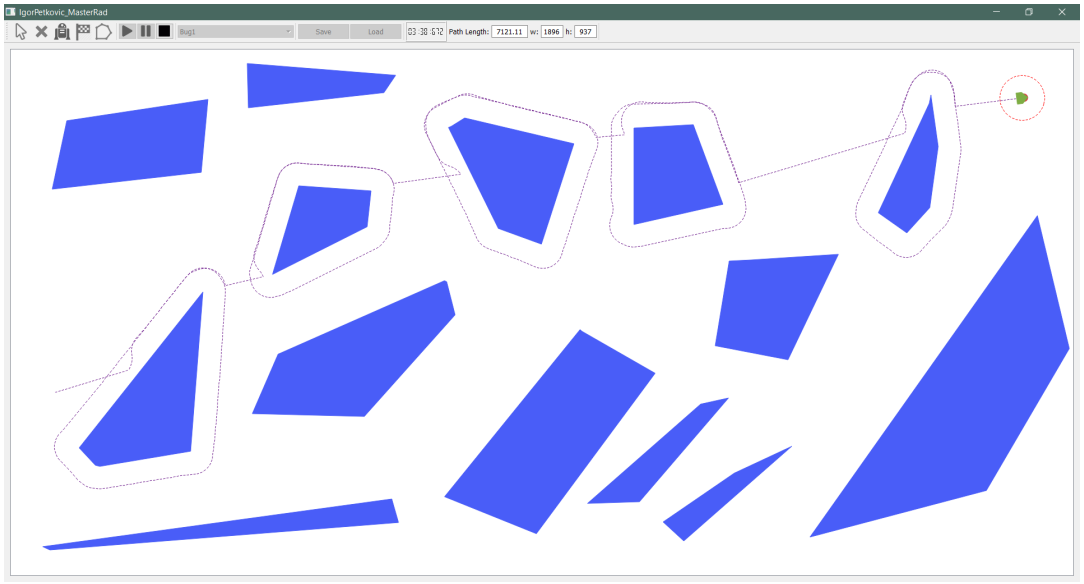
Algoritmi Lokalni dinamčki prozor i Aproksimativna ćelijska dekompozicija u trećem okruženju pokazuju slično ponašanje kao i u prethodna dva okruženja. Lokalni dinamički prozor ima najkraće vreme izvršavanja i putanju koja na nekim mestima vodi robota dalje od cilja (Slika 6.8). Aproksimativna ćelijska dekompozicija ima malo duže vreme izvršavanja od lokalnog dinamičkog prozora i kratku oštru putanju (Slika 6.6).



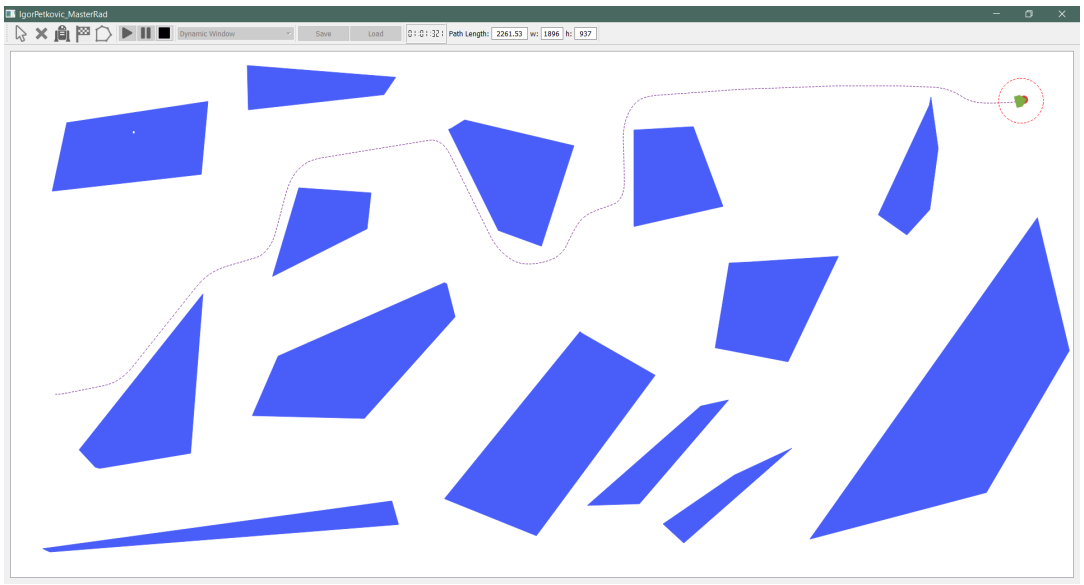
Slika 6.2: Rezultat algoritma Lokalni dinamički prozor u retko popunjenom okruženju



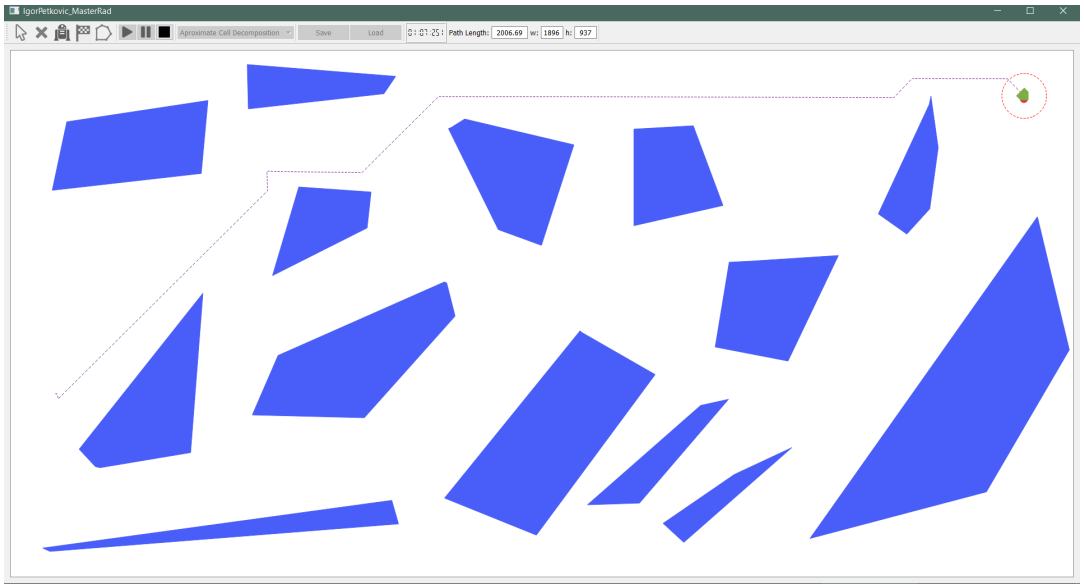
Slika 6.3: Rezultat algoritma Aprokismativna ćelijska dekompozicija u retko popunjenom okruženju



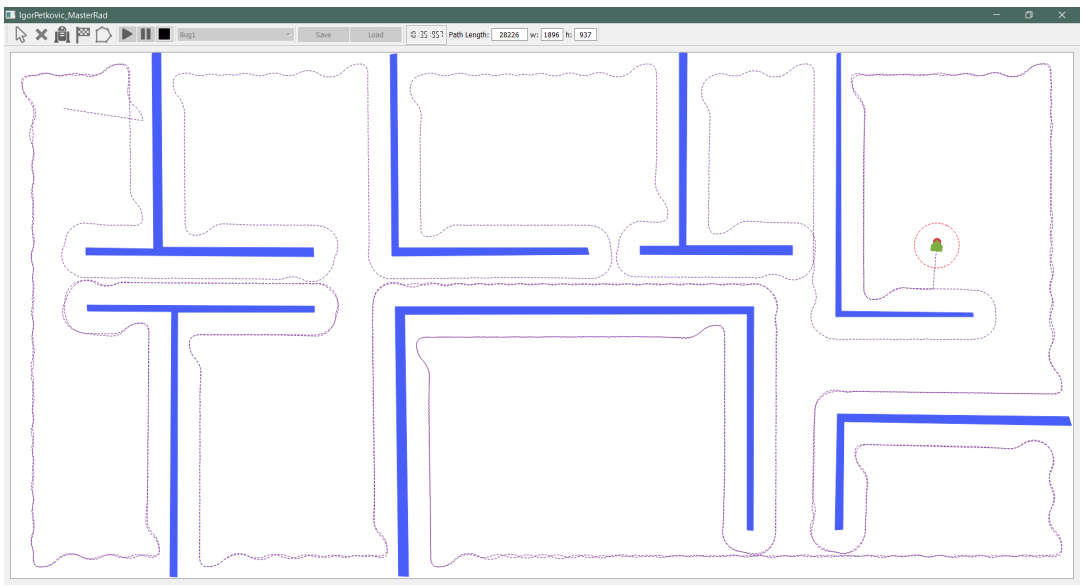
Slika 6.4: Rezultat algoritma Bag1 u gusto popunjenom okruženju



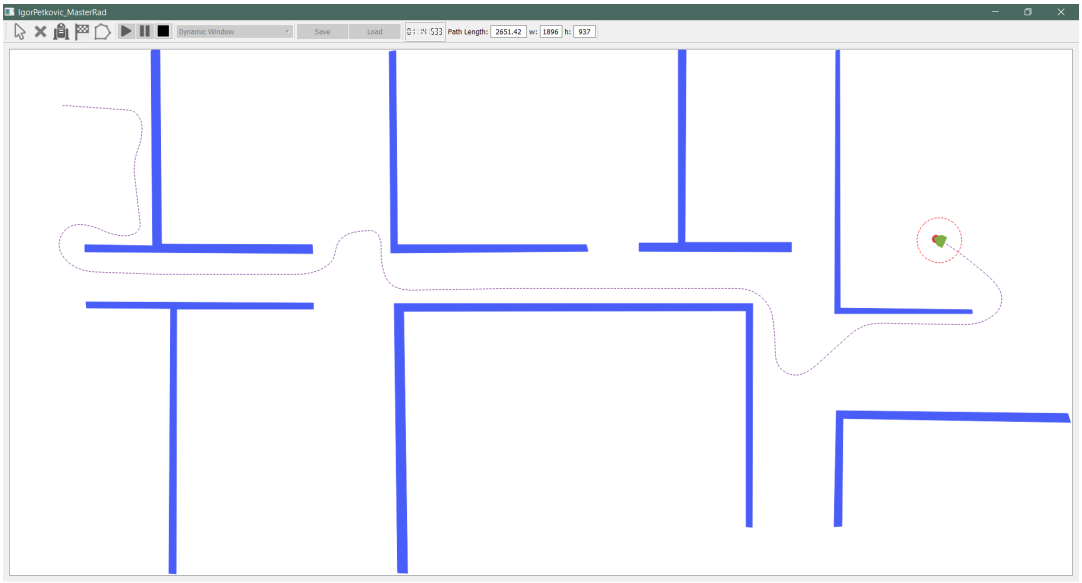
Slika 6.5: Rezultat algoritma Lokalni dinamički prozor u gusto popunjenom okruženju



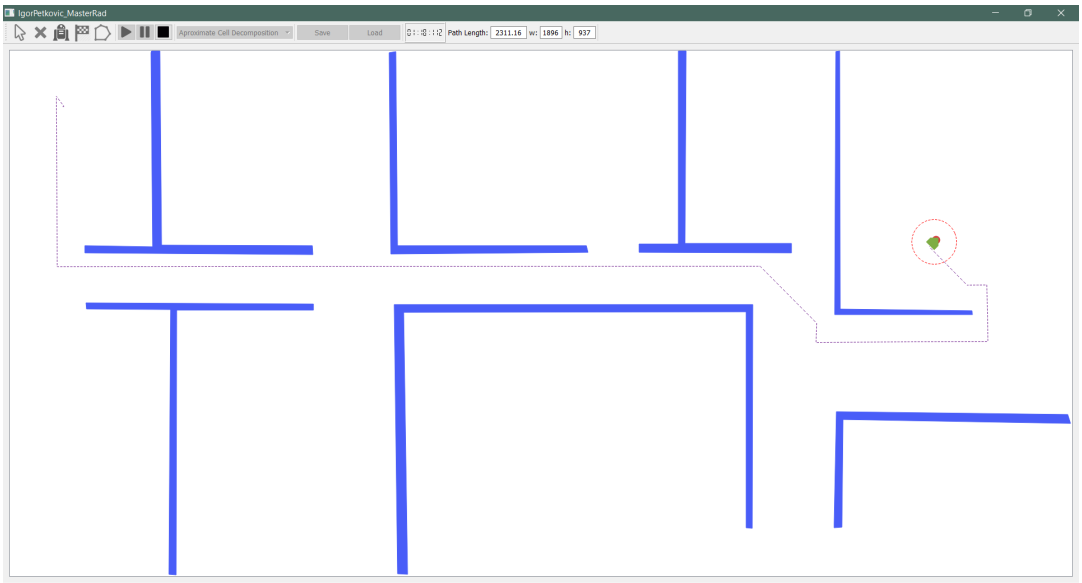
Slika 6.6: Rezultat algoritma Aproksimativna ćelijska dekompozicija u gusto popunjenom okruženju



Slika 6.7: Rezultat algoritma Bag1 u okruženju sa prostorijama



Slika 6.8: Rezultat algoritma Lokalni dinamički prozor u okruženju sa prostorijama



Slika 6.9: Rezultat algoritma Aprokismativna ćelijska dekompozicija u okruženju sa prostorijama

7. Zaključak

U ovom radu predstavljeni su algoritmi za navigaciju i planiranje putanje autonomnog mobilnog robota.

Pored opisa algoritama, programski su realizovani algoritmi Bag1, Lokalni dinamički prozor i Aproksimativna ćelijska dekompozicija. Jedno od mogućih unapređenja programske realizacije moglo bi da obuhvati implementaciju algoritama navigacije i planiranja putanje koji su poboljšanja trenutno implementiranih, kao što su Bag2 i Globalni dinamički prozor i algoritama koji imaju drugačiji pristup navigaciji, kao što su Histogram vektorskog polja i Dijagram blizine.

Pored implementacije algoritama za navigaciju u programskoj realizaciji implementirano je i okruženje u kom se odvija grafička simulacija algoritma. Okruženje za simulaciju implementirano je tako da mu je moguće dodavati samo prepreke mnogougaonog oblika, pa bi moguće poboljšanje bilo u vidu implementiranja prepreka raznih oblika i prepreka koje su pokretne. Ovo poboljšanje bi proširilo mogućnost predstavljanja realnih problema navigacije i planiranja putanje u okruženju za simulaciju.

Literatura

[1] Illah R. Nourbakhsh Ronald Seigwart. *Introduction to Autonomous Mobile Robots.* A Bradford Book The MIT Press, 2004.

[2] Shivani Mehta Varun Chauhan, Chintu Rza. Design a tracking controller for single-link manipulator via dsc. *International Journal of Research in Advent Technology*, 2(6), Jun 2014.

[3] Ulrich Nehmzow. Mobile robotics: Research, applications and challenges. In *In Future Trends in Robotics. Institution of Mechanical Engineers*, 2001.

[4] Sebastian Thrun. When robots meet people: Research directions in mobile robotics. *IEEE Intelligent Systems*, 1998.

[5] irobot roomba vacuum cleaning robot. http://www.irobot.com/For-the-Home/Vacuum-Cleaning/Roomba.aspx.

[6] K.K.Saju Leena.N. A survey on path planning techniques for autonomous mobilerobots. *IOSR Journal of Mechanical and Civil Engineering*, pages 76–79, 2014.

[7] O.Hachour. Path planning of Autonomous Mobile robot. *INTERNATIONAL JOURNAL OF SYSTEMS APPLICATIONS, ENGINEERING AND DEVELOPMENT*, 2, 2008.

[8] Lim Chee Wang, Ser Yong Lim, and V.M.H. Ang. Hybrid of global path planning and local navigation implemented on a mobile robot in indoor environment. In *Intelligent Control, 2002. Proceedings of the 2002 IEEE International Symposium on*, pages 821–826, 2002.

[9] Larry T. Ross James W. Masterson, Stephen W. Fardo and Robert Towers. *Robotics: Theory and Industrial Applications* . Goodheart-Willcox Publisher, 2010.

[10] Bradley Hamner Joseph Djugash. Neural Networks for Obstacle Avoidance. 2005.

[11] Danica Janglova. Neural networks in mobile robot motion. *International Journal of Advanced Robotic Systems*, 1:15–22, 2004.

[12] J. Borenstein, Y. Koren, and Senior Member. The vector field histogram - fast obstacle avoidance for mobile robots. *IEEE Journal of Robotics and Automation*, 7:278–288, 1991.

[13] Ronald C. Arkin. *Inteligent Robotics and Autonomous Agents.* A Bradford Book The MIT Press, 2005.

[14] James Ng. *An Analysis of Mobile Robot Navigation Algorithms in Unknown Environments.* PhD thesis, University of Western Australia, School of Electrical, Electronic and Computer Engineering, 2010.

[15] I. Ulrich and J. Borenstein. Vfh+: reliable obstacle avoidance for fast mobile robots. In *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, volume 2, pages 1572–1577 vol.2, May 1998.

[16] I. Ulrich and J. Borenstein. Vfh*: local obstacle avoidance with look-ahead verification. In *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, volume 3, pages 2505–2511 vol.3, 2000.

[17] M. Isabel Ribeiro. Obstacle Avoidance. November 2005.

[18] Cody O. Deacon. Autonomous Collision Avoidance and Mapping for a Quadrotor Using Panning Sonar Sensors, 2014.

[19] Sofia Lindmark Per Eriksson, Felix Foborg. Collision avoidance with Vector Field Histogram+ and Nearness Diagram algorithms implemented on a LEGO Mindstorms NXT robot. February 2014.

[20] Robot locomotion. http://www.robotplatform.com/knowledge/Classification_of_Robots/Holonomic_and_Non-Holonomic_drive.html.

[21] Nak Yong Ko, Reid Simmons, Ko Reid, and G. Simmons. The lane-curvature method for local obstacle avoidance, 1998.

[22] H. Berti, A. D. Sappa, and O. E. Agamennoni. Improved dynamic window approach by using Lyapunov stability criteria. *Latin American applied research*, 38:289 – 298, 10 2008.

[23] A. Maroti, D. Szaloki, D. Kiss, and G. Tevesz. Investigation of dynamic window based navigation algorithms on a real robot. In *Applied Machine Intelligence and Informatics (SAMI), 2013 IEEE 11th International Symposium on*, pages 95–100, Jan 2013.

[24] Javier Minguez and Luis Montano. Nearness diagram navigation (nd): A new real time collision avoidance approach. In *In Proc. of the ieee/rsj international conference on intelligent robots and systems (iros'00*, pages 2094–2100, 2000.

[25] J.W. Durham and F. Bullo. Smooth nearness-diagram navigation. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 690–695, Sept 2008.

[26] Javier Minguez and Luis Montano. Nearness diagram (ND) navigation: collision avoidance in troublesome scenarios. *IEEE T. Robotics and Automation*, 20(1):45–59, 2004.

[27] David M.W. Powers Adham Atyabi. Review of classical and heuristic-based navigation and path planning approaches.

[28] R. J. P. Zacksenhouse, M.; DeFigueiredo and D. H. Johnson. A neural network architecture for cue-based motion planning. In *Proc. IEEE Int. Conf. on Decision and Control*, pages 324–327, 1988.

[29] Alexander Zelinsky. Using path transforms to guide the search for findpath in 2d. *International Journal of Robotics Research*, 13:315–325, 1994.

[30] Danial Nakhaeinia Tang Sai Hong and Babak Karasfi. Classic and heuristic approaches in robot motion planning – a chronological review. In *Proc. World Academy of Science, Engineering and Technology*, pages 101–106, 2007.

[31] C. Kozakiewicz and M. Ejiri. Neural network approach to path planning for two dimensional robot motion. In *Intelligent Robots and Systems '91. 'Intelligence for Mechanical Systems, Proceedings IROS '91. IEEE/RSJ International Workshop on*, pages 818–823 vol.2, Nov 1991.

[32] V. Santos, J.G.M. Goncalves, and F. Vaz. Perception maps for the local navigation of a mobile robot: a neural network approach. In *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, pages 2193–2198 vol.3, May 1994.

[33] Hemmen J.L. Schulten K. Domany, E. Models of neural networks. *Springer Verlag*, 1991.

[34] Sif F. Talaoubrid S. Chohra, A. Neural navigation approach of an autonomous mobile robot in a partially structured environment. In *Proceedings of IAV'95*, page 238–243, June 1995.

[35] J. Tani. Model-based learning for mobile robot navigation from the dynamical systems perspective. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 26(3):421–436, Jun 1996.

[36] Kun H. W.; Chin H. C. and Jiann D. L. A cache-genetic-based modular fuzzy neural network for robot path planning. In *Proc. IEEE Int. Conf. on Systems, Man, and Cybernetics*, volume 4, pages 3089–3094, 1996.

[37] T. Frontzek, N. Goerke, and R. Eckmiller. Flexible path planning for real-time applications using a*-method and neural rbf-networks. In *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, volume 2, pages 1417–1422 vol.2, May 1998.

[38] Xianyi Yang and M. Meng. A neural network approach to real-time motion planning and control of robot manipulators. In *Systems, Man, and Cybernetics, 1999. IEEE SMC '99 Conference Proceedings. 1999 IEEE International Conference on*, volume 4, pages 674–679 vol.4, 1999.

[39] N. Sadati and J. Taheri. Solving robot motion planning problem using hopfield neural network in a fuzzified environment. In *Fuzzy Systems, 2002. FUZZ-IEEE'02. Proceedings of the 2002 IEEE International Conference on*, volume 2, pages 1144–1149, 2002.

[40] D. Erickson. Non-learning artificial neural network approach to motion planning for the pioneer robot. In *Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 1, pages 112–117 vol.1, Oct 2003.

[41] Anmin Zhu and S.X. Yang. A neural network approach to dynamic task assignment of multirobots. *Neural Networks, IEEE Transactions on*, 17(5):1278–1287, Sept 2006.

[42] Fan Jian, Fei Minrui, and Ma Shiwei. Rl-art2 neural network based mobile robot path planning. In *Intelligent Systems Design and Applications, 2006. ISDA '06. Sixth International Conference on*, volume 2, pages 581–585, Oct 2006.

[43] Vilém Novák, Jirí Mockor, and Irina Perfilieva. *Mthematical principles of fuzzy logic*. Kluwer international series in engineering and computing science. Kluwer, Boston, MA, 1999.

[44] Danial Nakhaeinia Tang Sai Hong and Babak Karasf. Application of fuzzy logic in mobile robot navigation. In *Fuzzy Logic - Controls, Concepts, Theories and Applications*, 2012.

[45] Bing-Yung Chee, S. Y. T. Lang, and P. W. T. Tse. Fuzzy mobile robot navigation and sensor integration. In *Proceedings of IEEE 5th International Fuzzy Systems*, volume 1, pages 7–12 vol.1, Sep 1996.

[46] E. P. Dadios and O. A. Maravillas. Cooperative mobile robots with obstacle and collision avoidance using fuzzy logic. In *Proceedings of the IEEE Internatinal Symposium on Intelligent Control*, pages 75–80, 2002.

[47] Panagiotis G Zavlangas, Spyros Tzafestas, and Kaspar Althoefer. Fuzzy obstacle avoidance and navigation for omnidirectional mobile robots. 09 2000.

[48] H. Seraji and A. Howard. Behavior-based robot navigation on challenging terrain: A fuzzy logic approach. *IEEE Transactions on Robotics and Automation*, 18(3):308–321, Jun 2002.

[49] Dayal R. Parhi. Navigation of mobile robots using a fuzzy logic controller. *Journal of Intelligent and Robotic Systems*, 42(3):253–273, Mar 2005.

[50] J. H. Lilly. Evolution of a negative-rule fuzzy obstacle avoidance controller for an autonomous vehicle. *IEEE Transactions on Fuzzy Systems*, 15(4):718–728, Aug 2007.

[51] Chan-Hong Chao, Bo-Yan Hsueh, Ming-Ying Hsiao, Shun-Hung Tsai, and S Li. Fuzzy target tracking and obstacle avoidance of mobile robots with a stereo vision system. 11, 09 2009.

[52] Ching-Chang Wong, Chih-Lyang Hwang, Kai-Hsiang Huang, Yueh-Yang Hu, and Chi-Tai Cheng. Design and implementation of vision-based fuzzy obstacle avoidance method on humanoid robot. 13, 04 2011.

[53] Er. Sonal Dhar Er. Waghoo Parvez. Path planning optimization using genetic algorithm – a literature review. *International Journal of Computational Engineering Research*, 4.

[54] J. Solano and D.I. Jones. Generation of collision-free paths, a genetic approach. In *Genetic Algorithms for Control Systems Engineering, IEE Colloquium on*, pages 5/1–5/6, May 1993.

[55] C. Hocaoglu and A.C. Sanderson. Planning multi-paths using speciation in genetic algorithms. In *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on*, pages 378–383, May 1996.

[56] M. Gen, Runwei Cheng, and Dingwei Wang. Genetic algorithms for solving shortest path problems. In *Evolutionary Computation, 1997., IEEE International Conference on*, pages 401–406, April 1997.

[57] D. Kumar Pratihar, K. Deb, and A. Ghosh. Fuzzy-genetic algorithms and mobile robot navigation among static obstacles. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 1, page 334 Vol. 1, 1999.

[58] R. Ramakrishnan and S. Zein-Sabatto. Multiple path planning for a group of mobile robot in a 2-d environment using genetic algorithms. In *SoutheastCon 2001. Proceedings. IEEE*, pages 65–71, 2001.

[59] S. Zein-Sabatto and R. Ramakrishnan. Multiple path planning for a group of mobile robots in a 3d environment using genetic algorithms. In *SoutheastCon, 2002. Proceedings IEEE*, pages 359–363, 2002.

[60] Qing Li, Xinhai Tong, Sijiang Xie, and Yingchun Zhang. Optimum path planning for mobile robots based on a hybrid genetic algorithm. In *Hybrid Intelligent Systems, 2006. HIS '06. Sixth International Conference on*, pages 53–53, Dec 2006.

[61] H. Ergezer and K. Leblebicioglu. Path planning for uavs for maximum information collection. *IEEE Transactions on Aerospace and Electronic Systems*, 49(1):502–520, Jan 2013.

[62] J. Xiao-Ting, X. Hai-Bin, Z. Li, and J. Sheng-De. Flight path planning based on an improved genetic algorithm. In *2013 Third International Conference on Intelligent System Design and Engineering Applications*, pages 775–778, Jan 2013.

[63] Deepika P Vinchurkar Alpa Reshamwala. Robot path planning using an ant colony optimization approach: A survey. *(IJARAI) International Journal of Advanced Research in Artificial Intelligence*, 2, 2013.

[64] Qgraphicsscene class — qt widgets 5.9. http://doc.qt.io/qt-5/qgraphicsscene.html.

[65] Qgraphicsobject class — qt widgets 5.9. http://doc.qt.io/qt-5/qgraphicsobject.html.

[66] Qgraphicsitem class — qt widgets 5.9. http://doc.qt.io/qt-5/qgraphicsitem.htmll.