SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. **1439**

PLANIRANJE GIBANJA AUTONOMNOG MOBILNOG ROBOTA U UNUTARNJIM PROSTORIMA

Marija Seder

Zagreb, svibanj 2004

Zahvaljujem se prof.dr.sc. Nedjeljku Periću, doc.dr.sc. Ivanu Petroviću,
dipl.ing. Kristijanu Mačeku i mr.sc. Jadranku Matušku
na pomoći tijekom izrade diplomskog rada.

Sadržaj

Poglavlje 1

Uvod

Jedan od ključnih problema u navigaciji mobilnih robota je pretraživanje prostora radi planiranja putanja gibanja mobilnog robota. Robot koji je orijentiran prema postizanju zadanog cilja određuje slijed akcija koje ga dovode u željeno stanje. Da bi odredio te akcije, bitno je formulirati problem koji treba riješiti. Formulacija problema ovisi o svim znanjima iz prostora koja su robotu dostupna. Primjer problema svodi se na poznavanje stanja u kojem se robot nalazi i u kojem će se stanju nalaziti ako izvrši neku akciju. Precizna definicija problema postaje temelj od kojeg započinje proces pretraživanja. Postoje brojni različiti procesi pretraživanja koji pristaju uz određene tipove problema. Formulacija cilja je prvi korak u rješavanju problema i određuje skup stanja u kojima robot ostvaruje zadani cilj. Formulaciju cilja prati bitan korak određivanja svih čimbenika koji utječu na ocjenu različitih načina ostarivanja cilja. Prijelazi između dva stanja u kojima se robot može nalaziti nazivaju se akcijama. Zadatak robota je određivanje akcija koje bi ga mogle dovesti u stanje cilja. Da bi uopće odredio te akcije, robot treba odrediti koji skup akcija uzeti u razmatranje. Formulacija problema je proces odabira skupa akcija i stanja i nadovezuje se na formulaciju cilja. Svaka akcija može imati pridruženu vrijednost koja predstavlja ocjenu pri ostvarivanju cilja. Ako se robot nalazi u stanju u kojem može izvoditi više od jedne akcije za koje nije određeno koliko one pridonose ostvarivanju cilja, razmatra se izvođenje različitih mogućih slijedova akcija koji vode do stanja s poznatom ocjenom i tada odabire najbolji slijed. Proces određivanja slijeda akcija koji najbolje vodi do cilja naziva se procesom pretraživanja.

U poglavlju 2 opisane su strategije pretraživanja prostora mobilnog robota. Pobliže upoznajemo tri algoritma za pretraživanje statičkih i dinamičkih prostora. Algoritam A* [5] optimalan je u pretraživanju statičkih prostora po kriteriju najkraćeg puta, ali se u slučaju velikih prostora povećava vrijeme pronalaženja cilja. Za dinamičke prostore može se također koristiti A* algoritam, ali se, zbog nepoznavanja konfiguracije prepreka u svakom trenutku u prostoru, mora svaki put računati globalna putanja od trenutne pozicije do cilja. Ovaj problem nalaženja globalne putanje u stvarnom vremenu u dinamičkim prostorima rješava D* algoritam [1]. D* algoritam pretražuje cijeli prostor samo jednom, na temelju statičkih

informacija o prostoru mobilnog robota. Daljnje pretraživanje se vrši tijekom gibanja mobilnog robota po dinamičkom prostoru na temelju novih senzorskih informacija tako da se globalna putanja dinamički prilagodi konfiguraciji prostora u omeđenom području dosega senzora. Fokusirani D* algoritam zbog dodatnih poboljšanja pretražuje uži dio prostora i ima brže vrijeme izvođenja inicijalnog proračuna [2]. Definirane su mrežne karte popunjenosti koje koriste algoritmi pretraživanja, kao i karta cijena prijelaza u pojedino polje koju koriste D* algoritmi. U ovom poglavlju se također opisuje problem navigacije mobilnog robota. Gibanje robota po globalnoj, geometrijskoj, putanji nije optimalno za kinematička svojstva robota. Stoga se koristi algoritam lokalnog izbjegavanja prepreka, nazivan dinamičkim prozorom, koji omogućava sigurno gibanje robota i u slučaju kada globalna putanja, odnosno cilj, nije dostupan. Dopuštene trajektorije koje se generiraju tim algoritmom, koriste se u integraciji sa proračunatom globalnom geometrijskom putanjom u svrhu dobivanja glatke globalne trajektorije robota.

U poglavlju 3 upoznajemo se sa stvarnim mobilnim robotom Pioneer 2DX i problemima vezanim za realizaciju algoritama u stvarnom vremenu. Opisani su implementacijski aspekti algoritma pretraživanja prostora. Algoritmi su implementirani u programskom jeziku c++.

U poglavlju 4 izloženi su simulacijski i eksperimentalni rezultati Simulacijska istraživanja funkcionalnosti provođena su u Saphirinom programskom okruženju na simulatoru mobilnog robota Pioneer 2DX. Sakupljeni su rezultati primjene A* i D* algoritma. Eksperiment se izvodio na mobilnom robotu na Zavodu za automatiku. Uspoređeni su eksperimentalni i simulacijski rezultati.

Poglavlje 2

Algoritmi planiranja putanje gibanja mobilnog robota

2.1 Optimalno pretraživanje prostora

2.1.1 Definicija procesa pretraživanja

Algoritam pretraživanja koristi problem kao ulazni podatak, a vraća rješenje u obliku slijeda akcija koje robot treba izvršiti da bi došao do cilja.

Postoji nekoliko tipova problema ovisno o tome koliko robot ima dostupnih informacija o prostoru u kojem se nalazi. Ako robot poznaje prostor toliko da točno može odrediti svoje trenutno stanje u njemu i stanje u kojem bi se nalazio kad bi izvršio neku akciju, tada je riječ o problemu **jednog stanja**. Ako prostor robotu nije potpuno poznat, mora razmatrati više od jednog stanja u kojima bi se mogao nalaziti kada bi izvršio neku akciju. Taj tip problema naziva se problemom **više stanja**. **Nepredviđeni tip** problema je onaj u kojem robot ne može predvidjeti koje stanje će nastati ako izvrši neku akciju jer se može dogoditi slučajni događaj. Ako robot nema nikakvo znanje o rezultatima njegovih akcija i prostoru u kojem se nalazi, tada se radi o **problemu istraživanja**. Robot mora istraživanjem skupljati informacije o tom prostoru i na taj način graditi mapu.

Razmotrimo prvi tip problema pri definiciji procesa pretraživanja. Problem jednog stanja definiraju sljedeće komponente:

- **početno stanje** u kojem se robot nalazi;
- **operator** određuje skup mogućih akcija koje su robotu na raspolaganju. Operator je funkcija koja svakom stanju X pridružuje skup stanja koja robot može ostvariti nekom akcijom primijenjenom u tom stanju. Ta funkcija se još naziva i funkcijom sljedbenika.
- **prostor stanja** je skup svih stanja dohvatljivih iz početnog stanja slijedom izvršenih akcija.

- **put** u prostoru stanja definiran je kao bilo koji niz akcija provedenih od jednog stanja do drugog.
- **ciljni test** je funkcija prepoznavanja cilja i ona se primjenuje u svakom stanju da bi se detektiralo ostvarenje cilja.
- **cijena puta** je funkcija koja pridružuje brojčanu vrijednost putu. U najčešćim slučajevima je cijena puta zbroj svih pojedinačnih vrijednosti plaćenih za izvršavanje akcija duž puta. Cijena puta predstavlja kriterij po kojem bi rješenje problema bilo optimalno. Kriterij primjerice može biti proteklo vrijeme, prevaljena udaljenost ili utrošena energija. Označava se funkcijom g.

2.1.2 Strategije pretraživanja

Rješenje procesa pretraživanja je put od početnog stanja do stanja koje zadovoljava ciljni test. Do rješenja se dolazi pretraživanjem prostora stanja. Primjenjivanjem operatora na trenutno stanje dolazimo do novog skupa stanja. Taj se proces naziva proširivanjem stanja. Bit pretraživanja je izdvajanje jedne mogućnosti iznad ostalih koje se ostavljaju za kasnije razmatranje u slučaju da prva nije dovela do rješenja. Taj je odabir određen strategijom pretraživanja. Proces pretraživanja najbolje je prikazati stablom čvorova. Korijen stabla je čvor koji odgovara početnom stanju. Operator određuje rubne čvorove. U svakom koraku se, ovisno o strategiji, bira jedan rubni čvor koji će biti proširen novim čvorovima sljedbenicima. Ako čvor sadrži stanje cilja, dobiva se rješenje pretraživanja. Čvor se može predstaviti sljedećom strukturom podataka:

- **stanje** u prostoru stanja koje predstavlja čvor;
- **predhodnik** iz kojeg je taj čvor nastao;
- **operator** koji se primijenio da stvori čvor;
- **broj predhodnika**
- **cijena puta** od početnog stanja do čvora;

Važno je napomenuti da dva čvora mogu sadržavati podatak o istom stanju ako su generirana slijedom različitih akcija. Na stablu se nalazi skup čvorova spremnih za proširivanje. Strategija pretraživanja je funkcija koja odabire sljedeći čvor koji će biti proširen iz tog skupa. Određena strategija pretraživanja se ocjenjuje kroz 4 kriterija:

- **kompletnost** - osigurava li strategija pronalazak rješenja ako ono postoji;
- **vremenska složenost** - koliko traje pronalaženje rješenja;
- **memorijska složenost** - koliko memorije je potrebno za izvršavanje pretraživanja;

- **optimalnost** - pronalazi li strategija najbolje rješenje po kriteriju cijene puta ako postoji više rješenja.

Strategije pretraživanja mogu se podijeliti na **neinformiranu** i **informiranu strategiju**. **Neinformirana strategija** nema nikakvu informaciju o broju koraka ili cijeni puta od trenutnog stanja do cilja. **Informirana strategija** sadrži te dodatne informacije i dolazi do rješenja mnogo učinkovitije. U nastavku se opisuje samo jedan bitniji primjer neinformiranog pretraživanja i nekoliko primjera informiranog pretraživanja.

Jedan bitan primjer **neinformirane strategije** je **uniformno pretraživanje**. U ovom se pretraživanju uvijek proširuje onaj čvor koji ima najmanju vrijednost cijene puta g. Svojstvo ovog pretraživanja je optimalnost i kompletnost. Nađeno rješenje bit će najjeftinije, a pri tome neće biti potrebno pretražiti cijelo stablo čvorova.

Informacije koje koristi **informirana strategija** pretraživanja koristi se u **funkciji ocjenjivanja** koja pridružuje ocjenu svakom čvoru koji se može proširiti čvorovima sljedbenicima. Čvor koji ima najbolju ocjenu bit će proširen prvi. Ova strategija se naziva **best-first-search**. **Greedy-search** je strategija koja minimizira estimiranu cijenu dohvaćanja cilja. Čvor čije je stanje proglašeno najbliže cilju uvijek se proširuje prvi. Cijena dohvaćanja cilja ne može biti točno procijenjena. Funkcija koja računa takvu estimiranu cijenu naziva se heurističnom funkcijom:

$h(n)$ – estimirana cijena najjeftinijeg puta od n-tog čvora do cilja.

Heuristična funkcija h može biti bilo koja funkcija sa zahtjevom da je $h(G) = 0$, gdje je G cilj. Heuristična funkcija vezana je za tip problema. Ova strategija nalazi rješenje vrlo brzo, ali ne mora biti optimalna. Također može biti nekompletna ako ne pazimo na sprečavanje ponovljenih stanja prilikom pretraživanja.

2.1.3 A* algoritam pretraživanja

Strategija koja kombinira dvije funkcije ocjene, heuristične funkcije i cijene puta, jednostavnim zbrajanjem, postaje optimalna i kompletna. Riječ je o A* pretraživanju. Funkcija ocjene koju ova strategija koristi je

$$f(n) = g(n) + h(n) \tag{2-1}$$

Funkcija f naziva se ukupnom cijenom puta i prema izrazu 2.1.3 određuje estimiranu cijenu najjeftinijeg puta koji prolazi kroz n-ti čvor.

Čvor koji ima najmanju vrijednost funkcije f biti će proširen prvi. Za ovo pretraživanje može se dokazati da je optimalno i kompletno. Uvjet koji se postavlja na funkciji h je da estimacija nikad ne prekorači cijenu najjeftinijeg puta do cilja. Takva heuristika naziva se **dopuštenom heuristikom** i ima svojstvo optimističnosti jer je estimirana cijena uvijek manja od stvarne cijene najjeftinijeg puta do cilja. Svojstvo optimističnosti prenosi se i na funkciju f pa tako vrijedi: ako je h dopuštena, tada $f(n)$ ne prekoračuje stvarnu vrijednost najjeftinijeg puta kroz stanje n.

Svojstva A* pretraživanja Slijedeći bilo koji put na stablu čvorova od korijena čvora, vrijednost funkcije f nikada se ne smanjuje. To svojstvo je ispunjeno samo ako je heuristika monotona. Lako se može pokazati da je heuristika monotona ako i samo ako posjeduje svojstvo nejednakosti trokuta. Ako se primjerice radi o pretraživanju prostora, pravocrtna udaljenost kao heuristika zadovoljava monotonost. Ako monotonost heuristike nije ispunjena, mogu se učiniti male izmjene u proračunu funkcije f da bi ona zadržala svojstvo monotonosti: svaki put kada se generira novi čvor, može se ispitati je li funkcija f, pridružena tom čvoru, manja od funkcije f pridružene čvoru čiji je on sljedbenik. Ako je to slučaj, sljedbeniku se pridružuje vrijednost f čvora iz kojeg je generiran. Za čvor Y koji je nastao iz čvora X primjenjuje se sljedeća relacija:

$$f(y) = max(f(x), g(y) + h(y)) \tag{2-2}$$

Svrha stvaranja ovog svojstva funkcije f je da se stvori slika o tome na koji način A* pretraživanje djeluje. Ako bismo sve jednake vrijednosti funkcije f nacrtali u prostoru stanja, dobili bismo konturne linije oko početnog čvora. Budući da A* proširuje čvorove s najmanjom vrijednosti f, možemo uočiti da se put nalazi unutar konturnih linija rastućih vrijednosti funkcije f.

U uniformnoj strategiji pretraživanja, koja je zapravo jednaka A* uz funkciju $h \equiv 0$, konturne linije bi bile koncentrične kružnice sa središtem u početnom čvoru. Što je heuristika točnija, to će se konturne linije funkcije f sve više rastezati prema cilju i biti fokusirane oko optimalnog puta. Završno svojstvo ovog algoritma je optimalna učinkovitost za bilo koju heurističnu funkciju. Nijedan algoritam ne proširuje manje čvorova od A* algoritma.

Dokaz optimalnosti A* algoritma Neka je G optimalni ciljni čvor s vrijednošću cijene puta f^*. Zamislimo slučaj da je A* odabrao G_2 kao ciljni čvor koji je suboptimalan i vrijedi $f(G_2) > f^*$. Neka postoji neki čvor n koji se nalazi na optimalnom putu do stanja G. Budući da je h dopuštena heuristika, mora vrijediti relacija $f^* \geq f(n)$. Ako n ne bude odabran za daljnje proširenje čvorova prema čvoru G2, znači da je $f(n) \geq f(G2)$. Iz ove dvije relacije slijedi da je $f^* \geq f(n)$ što je u kontradikciji s početnom pretpostavkom.

Dokaz potpunosti A* algoritma Budući da algoritam proširuje čvorove prema rastućim vrijednostima funkcije f, proširenje nekog čvora na ciljni čvor se mora dogoditi. To neće biti slučaj ako ima beskonačno mnogo čvorova s vrijednosću funkcije $f(n) < f*$. To se može dogoditi u 2 slučaja: ako postoji čvor s beskonačnim faktorom granjanja ili postoji put s konačnom cijenom, ali s beskonačno mnogo čvorova duž puta. Egzaktna tvrdnja bi bila da je A* kompletan na lokalno konačnim grafovima.

Složenost A* algoritma kao nedostatak Broj čvorova unutar konturne krivulje koja zahvaća cilj raste eksponencijalno s duljinom optimalnog puta. Pokazano je da se eks-

ponencijalni porast javlja u slučaju da pogreška heurističke funkcije raste brže od logaritma stvarne cijene puta. Matematički izraženo, subeksponencijalni porast je ispunjen ako vrijedi

$$abs(h(n) - h^*(n)) < O(logh^*(n)) \tag{2-3}$$

gdje je $h^*(n)$ stvarna cijena dostizanja cilja čvora n. Za gotovo sve heuristike u praktičnoj upotrebi, pogreška je barem proporcionalna cijeni puta. Međutim, upotreba dobre heuristike još uvijek uveliko štedi memoriju u usporedbi s uniformnim pretraživanjem. Vremenska složenost algoritma nije toliko upitan nedostatak budući da čuva sve generirane čvorove u memoriji pa prije ostane bez memorije, nego vremena.

Pretraživanje statičkih prostora Prostor u kojemu se ništa ne mijenja u vremenu dok robot prolazi kroz njega zovemo statičkim. Akcije koje robot izvršava u tom prostoru nisu povezane s vremenskim trenutkom u kojem ih izvršava na nekom dijelu prostora.

A* algoritam je primijenjen u pretraživanju statičkog prostora u kojem se nalazi mobilni robot.

Mrežne karte popunjenosti Da bi se A* mogao primijeniti, prostor je mrežasto podijeljen na konačan broj polja jednakih površina oblika kvadrata. Svako polje prostora treba imati definirane susjede kako bi se moglo konstruirati stablo pretraživanja. Također se definiraju koja polja predstavljaju start i cilj. Takav podijeljeni prostor predstavlja **mrežnu kartu popunjenosti**. Veličina polja prostora određuje **rezoluciju** mrežne karte popunjenosti. Ona polja koja ne sadrže prepreku smatraju se praznima i pridružuje im se vrijednost 0, dok se punim poljima, tj. onima koji sadrže prepreku, pridružuje vrijednost 1.

Opis A* algoritma pretraživanja mrežne karte popunjenosti Robot mora poznavati cijelu mrežnu kartu popunjenosti da bi izračunao optimalni put prema A* algoritmu. Algoritam pretražuje samo ona polja koja ne sadrže prepreku. Čvor je strukturiran na sljedeći način:

- **stanje** predstavlja jedno polje na mrežnoj karti popunjenosti;
- **pokazivač** na predhodni čvor;
- **broj predhodnika**
- **g**(n) - ukupna prijeđena udaljenost od startnog do n-tog čvora;
- **h**(n) - estimirana udaljenost cilja od trenutne pozicije. Postoje brojne heuristike koje možemo koristiti na mrežnoj karti popunjenosti. Jedan primjer heuristike funkcija čija je vrijednost određena udaljenošću trenutnog stanja n od cilja po onoj koordinatnoj osi koja daje veći iznos; drugi primjer je euklidska udaljenost trenutnog stanja i cilja;

Heuristiku možemo odrediti kombinirajući više dopuštenih heuristika određujući je po maksimalnoj vrijednosti iz skupa dopuštenih heuristika kako bi se sačuvalo svojstvo monotonosti.

- $f(n)= g(n) + h(n)$ - estimirana ukupna cijena najjeftinijeg puta koji prolazi kroz n-to stanje;

A* algoritam koristi OPEN listu na koju sprema one čvorove koji se mogu proširiti na svoje susjede. S OPEN liste skida čvor koji ima najmanju vrijednost f i proširuje njegovim susjedima koji dolaze na OPEN listu. Na listu se ubacuju samo oni čvorovi koji ne sadrže prepreku. Prilikom ubacivanja susjeda na OPEN listu, određuju mu se svi parametri koji čine strukturu čvora. Algoritam počinje s praznom listom na koju se prvotno ubacuje startni čvor sa vrijednošću funkcije g postavljenom na 0. Svi čvorovi početno nemaju definirane predhodnike, a vrijednosti funkcija g, h i f su postavljene na neki veliki broj. Startni čvor se skida s OPEN liste i proširuje na susjede koji se ubacuju na OPEN listu. Budući da se sa liste uvijek skida onaj čvor s najmanjom vrijednosti f, pretraživanje će biti fokusirano što bržem pronalaženju ciljnog čvora. Svaki novi čvor koji dolazi na OPEN listu, pokazuje na čvor iz kojeg je nastao. Algoritam završava kada ciljni čvor dođe na listu. Optimalni put se odredi slijeđenjem pokazivača na predhodni čvor krenuvši od cilja sve dok se ne dođe do starta.

Pretraživanje dinamičkih prostora Prije opisan A* algoritam bio je primjenjivan u statičkom prostoru i uz uvjet da robot ima točnu mrežnu kartu popunjenosti koja opisuje taj prostor. Javlja se problem pretraživanja prostora koji se kroz vrijeme mijenjaju, na primjer, promijeni se raspored prepreka u prostoru pa mrežna karta popunjenosti više ne odgovara stvarnom prostoru. Isti problem se javlja ako ne poznajemo točan raspored prepreka u prostoru. Razvili su se razni pristupi koji pokušavaju riješiti taj problem. Svim pristupima je zajedničko da se inicijalno isplanira globalna putanja na temelju informacija koje robot sadrži o prostoru. Robot je opremljen senzorima pa može skupljati podatke iz prostora prilikom kretanja. Kada otkrije novu prepreku, o kojoj prethodno nije imao informaciju, tada lokalno modificira prije izračunatu putanju ili izračuna cijeli put ponovo ako prethodni put sadržava prepreku. Postoje različiti načini modificiranja putanje tijekom gibanja. Jedan od načina je zaobilaženje prepreke koja se nalazi na putu, sve dok se ne pronađe točka na prepreci koja je najbliža cilju. Ovaj algoritam gubi na optimalnosti izračunatog puta. Drugi je pristup da se ponovo izračuna optimalni put od pozicije u kojoj se nalazi robot, kada ustanovi odstupanje između inicijalne karte i stvarnog okruženja primjenom A* algoritma. Ovakvo je planiranje putanje optimalno po kriteriju najkraćeg puta, ali je neučinkovito sa strane utroška memorije i vremena, naročito za velike karte prostora. Algoritam koji rješava problem neoptimalnosti i neučinkovitosti izračunatog puta jest D* algoritam. Ime mu dolazi od riječi dinamički, a nastavlja se na A* algoritam.

2.1.4 D* algoritam pretraživanja

Svojstva D* algoritma Ovaj algoritam proširuje svojstva A* algoritma i rješava problem njegove neučinkovitosti u dinamičkim prostorima. Svojstovo je ovog algoritma da može naći optimalnu putanju u nepoznatim, nepotpuno poznatim i promjenljivim prostorima. Poput A* algoritma koristi mrežnu kartu popunjenosti statičkog prostora koju dodatno popunjuje novim vrijednostima iz dinamičkog prostora. Koristi katru cijena prijelaza koja je će biti opisana kasnije. Bitno svojstvo D* algoritma, po čemu se razlikuje od A*, jest postavljanje pokazivača na sljedeći čvor. Pretraživanje počinje od ciljnog čvora, a završava proširenjem čvorova do starta. Osnovni D* algoritam ne koristi heurističnu funkciju kao estimiranu vrijednost cijene puta od nekog čvora X do ciljnog čvora, već se koristi stvarna vrijednost cijene puta od čvora X do cilja. Razlog tomu je način širenja čvorova od ciljnog čvora prema startu pa se vrijednost $h(X)$ može točno odrediti. Ovaj algoritam naziva se osnovnim D* algoritmom. Postoji još fokusirani D* algoritam koji koristi heurističnu funkciju kao estimiranu cijenu puta od startnog čvora do čvora X koja se označava s g i taj algoritam najbrže dolazi do optimalne putanje, što će biti detaljnije obrazloženo u odsječku 2.1.5.

Osnovne formulacije algoritma Problem se formulira kao skup stanja u prostoru koja predstavljaju lokaciju robota, a jedno stanje predstavlja cilj. Svako stanje odgovara jednom polju mrežne karte popunjenosti. Problem nalaženja puta postavlja se kao pronalaženje sekvence prijelaza stanja kroz mrežnu kartu popunjenosti od nekog početnog stanja do cilja ili određivanje da takva sekvenca ne postoji. Susjedna stanja su povezana dvosmjerenim dužinama. Svaka dužina ima pridruženu određenu vrijednost koja predstavlja cijenu prijelaza iz jednog stanja u drugo. Dužina koja spaja dva stanja X i Y imat će pridruženu vrijednost $c(X, Y)$ kao cijenu prijelaza iz stanja Y u X ili vrijednost $c(Y, X)$ kao cijenu prijelaza iz X u Y, ovisno o tome u kojem smjeru se vrši prijelaz. Cijena dužine predstavlja cijenu prijelaza u pojedino stanje iz bilo kojeg susjednog stanja pa se u daljnjem tekstu koristi i tim pojmom. Umjesto $c(X, Y)$ koristi se skraćeni izraz $c(X)$ za vrijednost cijene prijelaza u stanje X. Robot polazi od početnog stanja i prati dužine izlagajući se tako cijeni puta, sve dok ne dostigne ciljno stanje, označimo ga s G. Stanja u prostoru predstavljena su čvorovima koji onda čine stablo pretraživanja. Svaki čvor koji sadrži stanje X na putu sadrži pokazivač na sljedeće stanje Y, i to se opisuje izrazom $n(X) = Y$. D* koristi pokazivače da bi odredio puteve do cilja iz bilo kojeg čvora na grafu. Dva čvora na grafu su susjedi ako između stanja koja oni predstavljaju ne postoji niti jedno drugo stanje. Povezuje ih jedna dužina.

Problem određivanja cijena prijelaza u pojedino stanje Cijenu prijelaza mogu odrediti različiti kriteriji, kao što su prevaljen put, potrošena energija, proteklo vrijeme, itd. Usredotočimo se na cijenu koja predstavlja prevaljen put. Putanja će se odrediti ovisno o tome kako odredimo cijenu prijelaza u pojedino stanje. Cijenu dužine određuje ono stanje prema kome je dužina usmjerena. Ako nam nije bitno koliko blizu prepreci optimalni put prolazi,

tada postavljamo da je prijelaz u zauzeto stanje X velik broj definiran sa $c(X) = \text{OBSTACLE}$, a prijelaz u slobodno stanje Y, mali broj definiran sa $c(Y) = \text{EMPTY}$. Međutim, ako želimo putanju odgurnuti prema slobodnom dijelu prostora, definiramo varijablu COSTMASK. Stanja koja se nalaze u blizini prepreke dobivaju nešto veće vrijednosti cijena dužina koje su usmjerene prema njima, ovisno o tome koliko se nalaze blizu prepreke. Blizina je definirana brojem stanja zapisanom u varijabli COSTMASK. Ako je, npr. varijabla COSTMASK postavljena na 3, tek će četvrto stanje od prepreke imati vrijednost prijelaza EMPTY, a stanja prema prepreci se vrijednosti prijelaza inkrementiraju što je njihova udaljenost od prepreke manja. Na taj se način dobivaju fini prijelazi cijena dužina od prepreke prema slobodnom prostoru. To je od posebne važnosti pri planiranju gibanja robota po hodnicima gdje bi putanja trebala prolaziti sredinom hodnika.

Karta cijena prijelaza Kao što je definirana mrežna karta popunjenosti, tako je prikladno definirati **kartu cijena prijelaza** u pojedino polje. Karta cijene prijelaza određuje se prema zapisanim vrijednostima prepreka u mrežnoj karti popunjenosti. Rezolucije obiju karata su jednakih vrijednosti. Svakom polju se pridružuje vrijednost cijene prijelaza iz bilo kojeg susjednog polja u to polje prema prije opisanom postupku.

Nova informacija iz okruženja Ako robot dobije novu informaciju da je neko stanje zauzeto o kojem je predhodno imao informaciju da je ono slobodno, promijenit će se vrijednosti cijena dužina koje su usmjerene prema tom stanju. Vrijednost prijelaza u to stanje, označimo ga s X, postavit će se na $c(X) = \text{OBSTACLE}$. Stanja koja se nalaze u okolini od novo promijenjenog stanja možemo promijeniti prema unaprijed opisanom postupku.

Opis D* algoritma

OPEN lista D* koristi OPEN listu čvorova kao i A* algoritam. OPEN lista u D* algoritmu ima novu svrhu da prenosi informaciju o promjenama cijena dužina i da odredi cijene puta nekog čvora na mrežnoj karti popunjenosti. Svaki čvor X sadrži sljedeće informacije:

- **stanje** predstavlja jedno polje na mrežnoj karti popunjenosti;
- $c(X)$ je cijena prijelaza iz bilo kojeg susjednog stanja u stanje X. Ova je funkcija određena prema prije opisanom postupku.
- $t(X)$ daje informaciju o tome je li stanje na listi ($t(X) = \text{OPEN}$), ili stanje nikad nije bilo na listi ($t(X) = \text{NEW}$) te je li stanje bilo na listi i više se tamo ne nalazi ($t(X) = \text{CLOSED}$);
- $n(X)$ je pokazivač na sljedeće stanje;

- $h(X)$ je zbroj svih cijena dužina što se nalaze na optimalnom putu od stanja X do ciljnog stanja G. Ta se funkcija određuje tako da se zbrajaju cijene dužina krenuvši od ciljnog čvora G koje ima vrijednost $h(G) = 0$.
- $k(X)$ je tzv. **ključna** funkcija D* algoritma. Ta funkcija grupira stanja X ako se dogodila promjena okupiranosti nekog stanja u dvije grupe: tzv. RAISE i LOWER stanja. Računa se kao min(h(X) prije promjene, h(X) u trenutku kad je stanje došlo na listu).

Ako je $k(X) < h(X)$, znači da je vrijednost $h(X)$ porasla s obzirom na staru vrijednost i onda to stanje dobiva nadimak "RAISE". Ta situacija događa se u slučaju novo otkrivene prepreke. Na primjer ako pokazivač nekog stanja X pokazuje na stanje Y ($n(X) = Y$), a otkrijemo novu informaciju da je pozicija na mrežnoj karti popunjenosti koju opisuje stanje Y zauzeto onda će vrijednost $h(X)$ porasti za vrijednost $c(Y) =$ OBSTACLE jer optimalni put od stanja X do stanja G sadrži prepreku[1]. U algoritmu RAISE stanje X prenosi informaciju porasta $h(X)$ vrijednosti na sva susjedna stanja koja imaju pokazivač usmjeren prema njima, odnosno za koje je $n(Y) = X$. Drugi nadimak koji stanje X može dobiti je LOWER i to ako je $k(X) == h(X)$. Tom se stanju ili smanjila vrijednost $h(X)$ ili je ostala jednaka. LOWER stanje X na isti način prenosi informaciju smanjenja vrijednosti $h(X)$ kao što RAISE stanje prenosi informaciju porasta. Ova LOWER stanja još i preusmjeravaju pokazivače susjednih stanja ako mogu smanjiti njihovu h vrijednost. Svaki put kada se stanje skida s OPEN liste, prošireno je svojim susjedima da bi im prenijelo promjenu cijene puta. Ti su susjedi ubačeni na OPEN listu i isti se proces nastavlja. Skida se ono stanje koje ima najmanju vrijednost funkcije k. To stanje se naziva najboljim stanjem na OPEN listi.

Inicijalni proračun optimalnog puta OPEN lista je inicijalno prazna. Sva stanja grafa inicijalno imaju sljedeće informacije:

- $t(X)=$ NEW
- $c(X)$ je određen početnim informacijama o okruženju
- $n(X)= -1$ kao oznaka da pokazivač na sljedećeg nije postavljen
- $h(X)=$ LARGE kao oznaka nekog velikog broja
- $k(X)=$ LARGE

Prvo stanje koje dolazi na listu je cilj G što je u suprotnosti s A* algoritmom gdje se počinjalo od starta. Stanju G se postavljaju sljedeće vrijednosti:

- $t(G)=$ OPEN

[1]Napomena: iako je Y na putu bliže cilju od X, ne smijemo zaboraviti da se vrijednost $h(X)$ računa od cilja.

- $n(G) = -1$
- $h(G) = 0$
- $k(G) = 0$

Stanje G je najbolje stanje na listi (i jedino), označimo ga s N. Skida se najbolje stanje s liste i postavlja se $t(N) =$ CLOSED, a na listu dolaze svi njegovi susjedi. Najbolje stanje N svakom susjedu Y postavlja pokazivač $n(Y) = N$ i određuje novi $h(Y)$ kao zbroj $h(N)$ i cijene dužine od susjeda prema stanju N: $h(Y) = h(N) + c(N)$.

Bitno je ovdje primijetiti da najbolje stanje može sadržavati prepreku. To se ustanovljuje tek pri ubacivanju nekog susjeda Y, koji onda dobiva veliku vrijednost $h(Y)$ jer je sljedeće stanje susjeda Y prepreka. Algoritam onda tako zaobilazi to stanje Y s velikom cijenom puta $h(Y)$, da nijedan njegov susjed neće postaviti pokazivač na njega nakon daljnjeg proračuna. Funkcija INSERT(x, h_{NEW}) ubacuje stanje na listu, računa joj $k(X)$ i $h(X)$, postavlja $t(X) =$ OPEN. Traži se najbolje stanje N, tj. ono s najmanjom vrijednošću $k(N)$, skida se s liste, ubacuju se novi susjedi, određuju pokazivači, opet traži najbolji i tako sve dok stanje Start ne postane najbolje stanje na listi, s nadimkom "LOWER". Do tog trenutka je algoritam prošao kroz sva stanja koja se nalaze unutar kružnog područja sa središtem u ciljnoj poziciji, a start se nalazi blizu ruba tog područja. Unutar tog područja sva stanja imaju postavljene pokazivače na sljedeće stanje koje se nalazi na optimalnom putu od bilo kojeg stanja do cilja. Da se robot nađe na bilo kojoj poziciji unutar tog kruga, znao bi doći do cilja slijedeći pokazivače od svakog stanja u kojem se nalazi. To nije bio slučaj kod A* algoritma. Tamo se iz startnog čvora moglo doći optimalnim putem u bilo koje stanje unutar područja zatvorenog konturnom linijom određenom vrijednošću $f(G)$. Nakon što je inicijalni proračun optimalnog puta završen, robot slijedi pokazivače iz svakog stanja u kojem se nalazi i napreduje prema cilju.

Dinamički proračun optimalnog puta Algoritam se ponovo izvršava samo u slučaju promjene okupiranosti nekog stanja. Čim se dogodi prva promjena nekog stanja X, ono mora imati oznaku CLOSED kao uvjet da se nalazi u tom kružnom području u kojem se robot giba, ubacuju se na OPEN listu sva stanja kojima se promijenila okupiranost i vrijednost c. Bitno je ovdje primijetiti da OPEN lista nije prazna zbog inicijalnog proračuna. Traži se najbolje stanje N na listi, a to će biti ono stanje koje je najbliže optimalnoj putanji robota, ili se na njoj čak i nalazi. Ako stanje N sadrži prepreku, svim njegovim susjedima Y će porasti $h(Y)$, i oni će ponovo doći na listu da bi se proširila informacija porasta cijene puta $h(N)$. Ti susjedi postaju RAISE stanja i šire informaciju porasta cijene puta svim susjedima koji imaju postavljene pokazivače na njih. Tako se širi porast cijene puta sve do trenutne pozicije. Dalje preuzimaju ulogu LOWER stanja koja će nastati prilikom traženja nekog drugog susjeda pomoću kojeg bi se smanjila cijena puta. LOWER stanja šire smanjenje cijene puta prema cilju, preusmjeravaju pokazivače i stvaraju novi optimalni put. Algoritam se

izvršava sve dok vrijednost najboljeg stanja na listi $k(N)$ ne prekorači vrijednost cijene puta stanja u kojem se robot trenutno nalazi $h(R)$, s R smo označili trenutno stanje robota. U nastavku je dan pseudokod naprijed opisanog D* algoritma.

Pseudokod D* algoritma

Funkcija INSERT(X, h_{new})

if $t(X) = NEW$ then $k(X) = h_{new}$
else
  if $t(X) = OPEN$ then
    $k(X) = \text{MIN}(k(X), h_{new})$; DELETE$(X)$
  else $k(X) = \text{MIN}(h(X), h_{new})$
$h(X) = h_{new}$
PUT–STATE(X)

Funkcija PROCESS–STATE()

$X =$ MIN–STATE()
if $X = NULL$ then return -1
$k_{old} =$ GET–KMIN(); DELETE(X)
if $k_{old} < h(X)$ then
  for each neighbour Y of X:
    if $h(Y) \leq k_{old}$ and $h(X) > h(Y) + c(Y, X)$ then
      $b(X) = Y; h(X) = h(Y) + c(Y, X)$
if $k_{old} = h(X)$ then
  for each neighbour Y of X:
    if $t(Y) = NEW$ or
      $(b(Y) = X$ and $h(Y) \neq h(X) + c(X, Y))$ or
      $(b(Y) \leq X$ and $h(Y) > h(X) + c(X, Y))$ then
      $b(Y) = X$; INSERT$(Y, h(X) + c(X, Y))$
else
  for each neighbor Y of X:
    if $t(Y) = NEW$ or
      $(b(Y) = X$ and $h(Y) \neq h(X) + c(X, Y))$ then
      $b(Y) = X$; INSERT$(Y, h(X) + c(X, Y))$
    else
      $if b(Y) \neq X$ and $h(Y) > h(X) + c(X, Y)$
        INSERT$(X, h(X))$
      else
        if $b(Y) \neq X$ and $h(X) > h(Y) + c(Y, X)$ and
          $t(Y) = CLOSED$ and $h(Y) > k_{old}$ then
          INSERT$(Y, h(Y))$
return GET–KMIN()

Funkcija MODIFY–COST$(X, Y, cval)$
$c(X, Y) = cval$
if $t(X) = CLOSED$ then INSERT$(X, h(X))$
return GET–KMIN()

Funkcija SEARCH–PATH(R)
while $t(S) \neq CLOSED$ and $k_{old} \neq -1$
k_{old} = PROCESS-STATE()
if $t(s) = NEW$ then return $NO - PATH$
if $s(X, Y) \neq c(X, Y)$ for some (X, Y) then
for each (X, Y) such that $s(X, Y) \neq c(X, Y)$:
k_{old} = MODIFY–COST$(X, Y, s(X, Y))$
while $k_{old} < h(R)$ and $k_{old} \neq -1$
k_{old} = PROCESS-STATE()
if $k_{old} = -1$ then return $NO - PATH$
return $PATH$

2.1.5 Fokusirani D* algoritam pretraživanja

Osnovni D* algoritam ne naslijeđuje svojstvo najbržeg pronalaženja cilja A* algoritma jer ne koristi heurističnu funkciju. Zbog toga što D* algoritam ne koristi heurističnu funkciju, bit će pretraženo cijelo kružno područje sa središtem u cilju. U slučaju velikih prostora dolazi do izražaja problem predugog trajanja inicijalnog izvođenja algoritma. Algoritam koji koristi heurističnu funkciju kao estimiranu cijenu reverznog puta od čvora n do startnog čvora, označavanu slovom g naziva se **fokusiranim D* algoritmom**. Svaki čvor koji predstavlja stanje X sadrži još i dodatne informacije uz sve one preuzete iz osnovnog D* algoritma:

- $g(X)$ je estimirana cijena puta od stanja X do ciljnog stanja. Ovdje se mogu koristiti brojne heuristike koje su već prije nabrojane u opisu A* algoritma.
- $f(X)= g(X) + h(X)$ - estimirana cijena najjeftinijeg puta koji prolazi kroz stanje X kao zbroj heurističke funkcije g i ključne funkcije k koja čuva staru vrijednost funkcije h kao zbroja svih cijena dužina što se nalaze na predhodno izračunatom optimalnom putu od stanja X do ciljnog stanja G.

U fokusiranom D* algoritmu je najbolje stanje na OPEN listi ono s najmanjom vrijednosti funkcije f, a među stanjima s jednakim vrijednostima funkcije f je najbolje ono s najmanjom vrijednosti ključne funkcije k. Na taj način je pretraživanje fokusirano unutar konturne krivulje koja usko zahvaća startni i ciljni čvor kao kod A* algoritma i time je inicijalni proračun brži. Nakon inicijalnog proračuna OPEN lista sadrži znatno manji broj stanja nego u slučaju osnovnog D* algoritma što je bitna prednost za daljnji dinamički proračun koji nastupa samo u slučaju promjene okupiranosti nekog stanja. Stanje X kojem se dogodila

promjena mora imati oznaku CLOSED, kao pokazatelj da se nalazi unutar uskog područja zatvorenog konturnom krivuljom oko starta i cilja, da bi se ubacio na OPEN listu. Budući da je pretraženo područje manje, bit će manje promijenjenih stanja s oznakom CLOSED pa će traženje najboljeg stanja na listi biti manje vremenski zahtjevno.

2.2 Integracija algoritma pretraživanja s algoritmom za izbjegavanje prepreka

2.2.1 Uvodna razmatranja

Globalna putanja dobivena algoritmom pretraživanja prostora je geometrijska krivulja koja nije glatka, jer se u slučaju karte popunjenosti prostora putanja između dva susjedna stanja sastoji od ravnih odsječaka čiji se smjer može promijeniti samo kao višekratnik kuta 45°. U slučaju pronalaženja najkraćeg puta moguća promjena kuta je -45°, 0° i +45°. Direktno zadavanje takve krivulje kao referentne trajektorije robota (tj. geometrijske putanje parametrizirane u vremenu), nije zadovoljavajuće zbog dva osnovna razloga:

1. neholonomska ograničenja robota ne dozvoljavaju praćenje po dijelovima prekinute putanje;

2. praćenje odsječaka od stanja do stanja gdje se smjer naglo mijenja mora rezultirati znatnim usporenjem robota, koje je u većini slučajeva nepotrebno, ako je nastavak putanje gladak.

Posljedica toga je smanjenje prosječne brzine robota i veći utrošak energije za usporavanje i ubrzanje. Osnovni je cilj neposredno ili posredno pretvoriti po dijelovima prekinutu krivulju u neki oblik glatke krivulje, koja u vremenu predstavlja trajektoriju mobilnog robota. U neposrednom pristupu se kod određivanja trajektorije moraju uzeti u obzir dinamička i kinematička ograničenja kod vremenskog parametriziranja same krivulje (primjerice kubni splineovi). Pri tome će dobivena krivulja prostorno gledajući odstupati od prvotno zadane po dijelovima prekinute linije, što znači da se mora dodatno provjeriti da li novo nastala krivulja ne prolazi u neku prepreku u prostoru. Kada u prostoru postoje dinamički objekti, dakle karta prostora se mijenja s vremenom, to može, uz ponovo planiranje putanje pomoću nekog od algoritama pretraživanja, značiti i značajni proračun za reparametriziranje globalne trajektorije, od starta do cilja. Alternativno, ako se implementira modul lokalnog izbjegavanja prepreka, može se koristiti skup mogućih trajektorija robota proračunatih u tom modulu u koordinaciji s globalnom geometrijskom putanjom. Modul lokalnog izbjegavanja prepreka je, hijerarhijski gledano, osnovni modul koji djeluje neovisno od globalnog planiranja putanje i omogućava sigurno gibanje robota i u slučaju kada globalna putanja, odnosno cilj, nije dostupan.

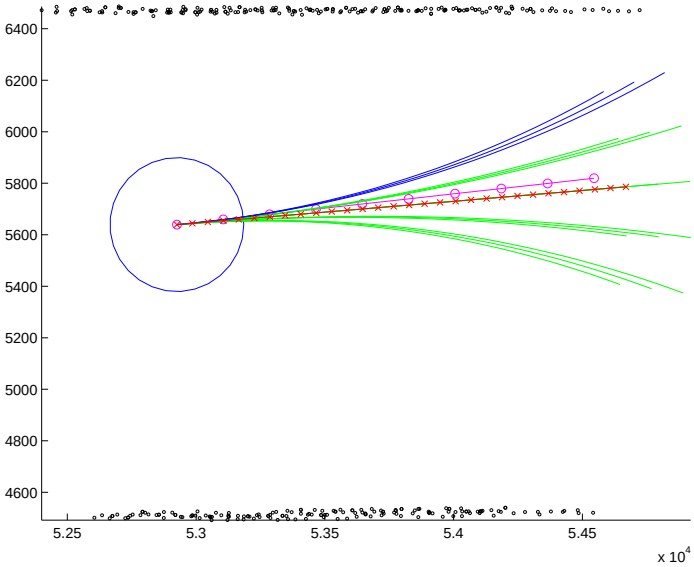
2.2.2 Algoritam dinamičkog prozora za izbjegavanje prepreka

Algoritam dinamičkog prozora [3], koji se koristi za lokalno izbjegavanje prepreka, temelji se na proračunu mogućih trajektorija robota neposredno u prostoru brzina, koji se sastoji od trenutne translacijske i rotacijske brzine (vektor brzine) i diskretiziranog skupa vektora brzina koji su dohvatljivi iz trenutnog stanja s obzirom na dinamička ograničenja robota

(translacijsko i rotacijsko ubrzanje). U [3] se pokazuje da se, uz pretpostavku konstantnog vektora brzine, trajektorija robota u N koraka unaprijed može aproksimirati kružnim lukom. Na taj se način određuje prostorna konfiguracija mogućih trajektorija robota prema lokalnoj okolini. U diskretnom skupu vektora brzina najprije se onemogućuju one vrijednosti koje vode prema sigurnom sudaru s određenom preprekom. Od preostalih vektora brzina dobiva maksimalnu ocjenu onaj, kod kojeg je vrijeme dolaska do najbliže prepreke najveće.

2.2.3 Algoritam integracije

Proračunate moguće trajektorije u modulu lokalnog izbjegavanja prepreka mogu se koristiti za određivanje globalne trajektorije, ako se uvede određeni kriterij usporedbe s geometrijskom putanjom dobivenom pomoću algoritma pretraživanja. U [4] je izveden kriterij prema kojem se skup mogućih trajektorija uspoređuje s tzv. učinkovitom putanjom. Ta putanja predstavlja linijski segment između trenutne pozicije robota i određene željene točke, tj. podcilja na globalnoj geometrijskoj putanji. Taj trenutni podcilj ovisi o lokalnoj konfiguraciji geometrijske putanje i trenutne brzine robota. S obzirom na trenutnu brzinu robota duljina efektivne putanje određuje se kao duljina trajektorije robota uz maksimalno moguće ubrzanje robota u jednom koraku i konstantnog slijeđenja te brzine sljedećih N koraka unaprijed (N je broj vezan za definirani efektivni domet senzora, u ovom slučaju laserskog senzora i maksimalne brzine robota). Na taj će se način izabrati veće brzine robota, ako to dopušta lokalna konfiguracija putanje. U slučaju veće zakrivljenosti geometrijske putanje, koja se određuje pomoću točaka pregiba (promjena smjera), duljina učinkovite putanje se smanjuje, što ima za posljedicu smanjenje brzine robota. Geometrijska putanja je prema algoritmu pretraživanja po rezoluciji jednaka rezoluciji mrežne karte popunjenosti (npr. 10cm u većini aplikacija). Ako se poveća rezolucija mrežne karte popunjenosti, algoritam pretraživanja će generirati glađu geometrijsku putanju, no time se povećava i prostor pretraživanja, što može biti ograničavajući faktor po vremenu izvođenja algoritma. U slučaju pomoću uzorkovanja dobivene geometrijske putanje između dva polja mrežne karte popunjenosti, može se postići da se trenutni podcilj nalazi na spojnici između dva polja. Na taj se način u svakom servo koraku može zadati novi podcilj koji direktno određuje željenu orijentaciju robota, a gornja granica orijentacije učinkovite putanje između trenutne pozicije robota i podcilja na geometrijskoj putanji je 45°. Učinkovita putanja mijenja smjer i duljinu virtualno putujući od trenutne pozicije robota po globalnoj geometrijskoj putanji. Ukupni rezultat je glatka globalna trajektorija robota gdje je vektor brzine određen s obzirom na lokalnu konfiguraciju geometrijske putanje. Pri tome nije potrebno dodatno provjeravati da li takva inkrementalna globalna glatka putanja dotiče neku prepreku, jer je taj uvjet automatski ispunjen s obzirom na to da se uzimaju u obzir samo dozvoljene trajektorije koje generira modul izbjegavanja prepreka. Prostorna konfiguracija mogućih trajektorija prikazana je na slici 2.1.



Sl. 2.1: Prikaz konfiguracije mogućih trajektorija u simulaciji vršenoj s kartom zavoda. Zelene trajektorije su one koje nemaju prepreku na putu; plave trajektorije sadrže prepreku; ljubičasti pravac predstavlja učinkovitu putanju; crvena trajektorija je izabrana optimalna trajektorija.

Poglavlje 3

Mobilni robot Pioneer 2DX

3.1 Osnovne značajke mobilnog robota

Model mobilnog robota koji je korišten u simulaciji i kao stvarni robot u eksperimentalnom dijelu ovog rada je *Pioneer 2DX* koji proizvodi *ActivMEDIA Robotics (robots.activmedia.com)*. To je inteligentna mobilna platforma sa opcijama za dodavanje kartica zasnovanih na proširenom PC104+ sa PCI sabirnicom i višestrukom PC104 standardu. S proširenim PC104+, robot dobiva mogućnost ugradnje Ethernet modema i lasera. Ima dva pogonska kotača i jedan stabilizacijski. Na njemu se nalazi osam prednjih sonara i osam stražnjih. Može postići maksimalnu brzinu od 1.6 m/s i nositi težinu do 23 kg. Da bi održali precizni proračun podataka pri toj brzini, koristi se 500 *tick* enkodera. Specifikacije mobilnog robota prikazane su u tablici 3.1.

Pioneer 2DX mobilni robot koristi mikrokontroler zasnovan na Siemensovu C166 čipu frekvencije 20 MHz, sa nezavisnim kontrolama motora i sonara. Mikrokontroler ima dva standardna RS232 komunikacijska porta i sabirnicu za proširenje koja prihvaća mnoge dodatke raspložive za robota. Na njemu se nalazi Pioneer 2 Operativni Sustav (P2OS).

P2OS je "*embedded*" operacijski sustav razvijen kod ActivMedia Robotics, koji je zadužen za operacije na razini mikrokontrolera Siemens C166 [6]. Funkcionalnost koju pruža s obzirom na pojedinu hardversku platformu je sljedeća:

Udaljenost među kotačima	321 mm
Promjer kotača	165 mm
Maksimalna brzina	1.6 m/s
Standardna pozicija enkodera	500 ticks
Omjer prijenosa	19.7:1
Broj odbojnika	5 naprijed, 5 otraga

Tablica 3.1: Specifikacije Pioneer 2DX mobilnog robota.

- kontrola i upravljanje motorima robota (PID upravljanje);
- prikupljanje podataka enkondera;
- propaljivanje sonara i prikupljanje njihovih podataka;
- kontrola odbijača robota;
- kontrola kompasa robota;
- kontrola hvataljke robota;
- kontrola dodatnih analognih i digitalnih ulaza/izlaza;
- upravljanje postolja dodatnog vizijskog sustava;
- serijska komunikacija;
- TCP/IP komunikacija;

Robot može raditi u jednom od ova tri moda :

- Samotestiranje
- Samostojeći mod
- Serverski mod

U robota je ograđeno i PC računalo sa instaliranim Red Hat Linux operativnim sustavom. Računalo je funkcionalno kao i svako drugo osobno računalo što znaći da mobilni robot posjeduje priključke za tipkovnicu, miš, monitor, mrežu te CD-ROM. Računalo posjeduje pristup mreži bežičnom vezom. Od ostalih značajki može se navesti ugrađena zvučna kartica (Aria podržava i prepoznavanje govora), framegrabber te dva PCMCIA utora (sve navedene kartice izvedene su korištenjem PC104 sabirnice). Računalo se može koristiti direktno spajanjem tipkovnice, miša te monitora na sam mobilni robot ili posredno preko drugog računala spajanjem preko mreže. Sprega klijentskog softvera koji se izvršava na PC računalu i P2OS-a daje mogućnost moderne klijent/server komunikacije za izvršavanje naprednih robotskih zadataka.

Kontrolni ciklus se izvodi svakih $100ms$, pri čemu mobilni robot kao P2OS server na kraju svakog ciklusa šalje sve relevantne podatke s robota pomoću tzv. SIP paketa (Standard P2OS Server Information Packet). Na početku svakog ciklusa očekuje se primanje paketa sa P2OS klijenta, a ako unutar određenog vremenskog intervala nije primljen niti jedan paket, veza sa klijentom postaje nevažeća i prekida se rad robota. Pri tome je osnovni oblik komunikacije pomoću serijske ili TCP/IP veze.

3.2 Saphira i Aria programsko okruženje

Klijent može biti bilo koji korisnički program koji je konforman na slanje zadanog kontrolnog paketa preko serijske ili TCP/IP veze. Kod SRI International Artificial Intelligence Center-a je razvijeno programsko okruženje Saphira [7][8], koje omogućava komunikaciju sa mobilnim robotom i interpretaciju senzorskih podataka sa robota (enkoderi, sonari i dodaci) u tzv. "*state reflectoru*", te prikupljanje podataka sa laserskog senzora (SICK) koji predstavlja odvojenu cjelinu.

Postoje dvije razine funkcija upravljanja mobilnim robotom:

1. niža razina funkcija upravljanja uključuje prikupljanje mjernih podataka s osjetila mobilnog robota i slanje naredbi mobilnom robotu;

2. primjeri funkcija upravljanja mobilnim robotom na višoj razini su praćenje položaja, snimanje karte radne okoline, planiranje putanje gibanja

Saphira predstavlja programsko okruženje za razvoj viših funkcija upravljanja mobilnim robotom. Programsko okruženje Saphira izgrađena je na temelju sučelja Aria koja predstavlja vezu između nižih i viših funkcija upravljanja mobilnim robotom. Oba okruženja predstavljaju skup gotovih klasa za obavljanje određenih zadaća kao što su izvršavanje sinkronih te asinkronih procesa.

Osnovna programska rutina je mikro-zadatak koji se izvodi u ciklusu $100ms$ te ima definiran skup procesnih stanja, preko kojih se u C/C++ programskom okruženju može kontrolirati mobilni robot.

1. "INIT" - je stanje u kojem dolazi inicijalizacija algoritma i operacija koje će mobilni robot izvršavati. Ovdje je potrebno učitati potrebne podatke za inicijalizaciju algoritama (npr. početni položaj mobilnog robota, parametri karte prostora koja se mora kreirati). U ovom trenutku još ne postoji veza s mobilnim robotom.

2. "WAIT" - je stanje između završene inicijalizacije te uspostavljanja veze s mobilnim robotom.

3. "CONNECT_INIT" - je stanje uspostavljanja veze s mobilnim robotom i služi za obnavljanje inicijalizacije algoritma podacima dobivenim sa mobilnog robota.

4. "RUN" - je stanje u kojem su implementirani algoritmi koji se izvršavaju svaki ciklus sinkronog procesa. Na kraju se ovog stanja provjerava postojanje veze s mobilnim robotom. U slučaju prekinute veze s mobilnim robotom, treba se zaustaviti izvršavanje ovog stanja te spremiti sve rezultate.

5. "END" - je stanje u koje se dolazi prekidanjem "RUN" stanja.

6. "NOP" - je stanje između prekida veze s mobilnim robotom i zatvaranja aplikacije za upravljanje robotom.

Saphira na višem hijerarhijskom nivou pruža mogućnost definiranja "*threadova*" koji se izvode asinkrono s obzirom na osnovni ciklus od $100ms$ i predstavljaju zadatke čiji proračun nije čvrsto vezan za osnovni ciklus. Saphira grafičko okruženje omogućava on-line praćenje stanja robota i okoline, ovisno o korisničkoj reprezentaciji iste. U njemu prikazuje kartu unutar koje se mobilni robot nalazi te podatke o očitanju osjetila za mjerenje udaljenosti (plava boja odgovara sonaru, a zelena laseru). Također ispisuje trenutni položaj te translacijska i rotacijska brzina mobilnog robota. Za prikazani položaj je bitno naglasiti da je to estimirani položaj robota koji se razlikuje od stvarnog položaja mobilnog robota u simulatoru ili stvarnom prostoru. Aria okruženje se može koristiti zasebno bez Saphira dijela.

Od dodatnih komercijalnih modula razvijenih kod SRI International-a koji se mogu integrirati u Saphira programsko okruženje najvažniji su gradijentni modul planiranja putanje (Gradient Path Planning Module) [9] i Markovljev modul za lokalizaciju robota (Markov Lokalization Module)[10].

3.3 Daljinsko upravljanje mobilnim robotom

Prilikom eksperimentiranja s mobilnim robotom potrebno je razlikovati dva računala. Jedno računalo je lokalno i s njega se spajamo na računalo ugrađeno u mobilni robot (udaljeno računalo). Grafički prikaz s udaljenog računala može biti proslijeđen na lokalno računalo za prikaz grafičkog sučelja pokrenute aplikacije. Aplikacija se može izvršavati na udaljenom računalu i svi rezultati (datoteke sa sakupljenim podacima), koje aplikacija stvara, bit će spremljeni na udaljenom računalu.

Zbog veće fleksibilnosti komunikacije, a prvenstveno zbog mogućnosti izvođenja drugih složenijih algoritama, implementirana je i komunikacija sa off-line računalom na kojem se mogu algoritmi dodatno i brže izvesti. U ovom slučaju to je cijelokupni modul navigacije i izbjegavanja prepreka mobilnog robota. Od komercijalnih modula na korištenoj eksperimentalnoj robotskoj platformi primjenjuje se Markovljeva lokalizacija pomoću laserskog senzora, dok se za navigaciju koristi vlastiti implementirani modul. Algoritam lokalizacije mobilnog robota izvodi se na on-board računalu robota.

Pošto se algoritmi navigacije i lokalizacije mobilnog robota moraju izvesti unutar servo ciklusa od $100ms$, potrebno je osigurati da komunikacija bude što brža i učinkovitija. U tu svrhu koristi se UDP protokol (User Datagram Protocol), baziran na mrežnim soketima [11]. Mobilni robot je u tom slučaju UDP klijent koji šalje u svakom servo ciklusu informaciju o trenutnom stanju okoline i robota nakon izvršenog ažuriranja podataka senzora i lokalizacije u globalnoj mapi prostora. Off-board računalo predstavlja UDP server gdje se primljeni paket o stanju robota koristi za proračun navigacijskog algoritma, nakon kojeg se nazad na robota šalje komunikacijski paket sa željenim upravljačkim veličinama. Pri tome

između klijenta i servera nije uspostavljena čvrsta komunikacijska veza, već klijent šalje paket na određenu internet adresu i port bez prethodnog spajanja na server, što je jedna od osnovnih značajki UDP protokola. Za razliku od tog protokola češće korišteni TCP protokol (Transmission Control Protocol) uspostavlja čvrstu vezu između klijenta i servera i osigurava da poslani mrežni paketi sigurno stižu do odredišta, u redoslijedu u kojem su poslani. No, dodatna provjera svakog paketa znači dodatno usporenje u komunikaciji, a u određenim aplikacijama to je i nepotrebno, kao što je slučaj u upravljanju mobilnim robotom. Ako neki komunikacijski paket u određenom servo ciklusu ne stigne sa klijenta na server ili obrnuto, zbog same dinamike procesa navigacije mobilnog robota, paketi iz prošlog ciklusa nisu relevantni, jer se stanje okoline i robota već promijenilo. Dakle, potrebno je uzeti u obzir već novu informaciju o stanju sustava, pa stari paket nije valjan (smjer klijent-server). Nakon poslane informacije o stanju sustava klijent na robotskoj platformi čeka upravljački komunikacijski paket sa servera. U slučaju gubitka upravljačkog paketa robot se zaustavlja i čeka dolazak novih komunikacijskih paketa sa servera, uz redovito slanje informacijskih paketa svakih $100ms$. Ta funkcionalnost je neophodna za sigurno upravljanje mobilnih robota, a implementirana je pomoću watchdog timera. Korištenjem off-board računala povećan je ukupni računalni kapacitet u sustavu, no glavno ograničenje vezano je za korištenje lokalne ethernet komunikacijske mreže kao veze između robota i off-board računala zbog osiguravanja dovoljno velike brzine i pouzdanosti unutar same mreže. Ako se koristi neki oblik bežične wavelan komunikacije, to znači da je doseg kretanja robota ograničen samim dometom te mreže.

Poglavlje 4

Simulacijski i eksperimentalni rezultati

Simulacija se vrši u Saphira razvojnom okruženju na simulatoru Pioneer 2DX. Karta koja se koristi za simulaciju predstavlja jedan uredski unutrašnji prostor. Raspored prepreka u karti je takav da robot u inicijalnom proračunu izračuna jedan put, a prilikom slijeđenja tog puta laserskim senzorom otkriva da tim putem ne može proći i isplanira novu putanju koja vodi sigurno do cilja. Izvršene su simulacije za sva tri algoritma, dan je prikaz trajektorija te relevantnih parametara karakterističnih za algoritme te je napravljena usporedba vremena izvođenja pojedinih algoritama.

Eksperiment je izvršen na Zavodu za automatiku, na 9. katu C zgrade, Fakulteta elektrotehnike i računarstva. Karta koja se koristi za eksperiment nepotpuna je i robot prilikom gibanja sam otkriva prepreke. U prostoriji u kojoj mu je zadan cilj se nalaze brojne prepreke i on ih upisuje u kartu i preračunava putanju.

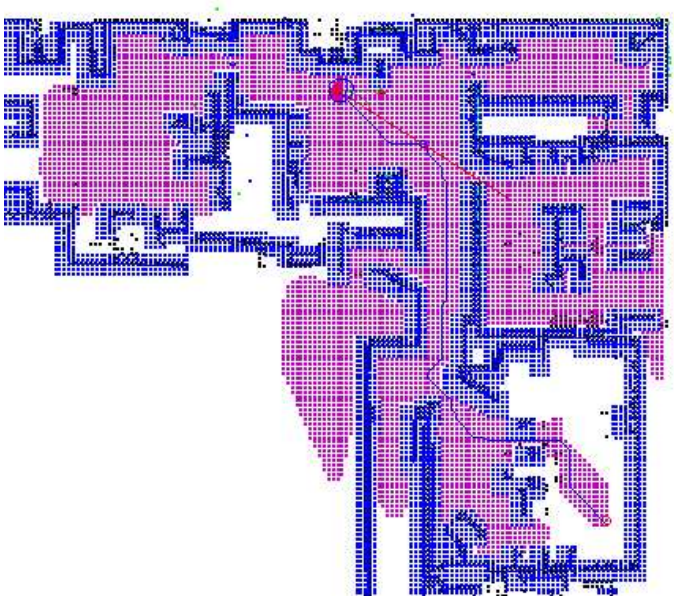
4.1 A* algoritam

Algoritam A* je implementiran u Saphira razvojnom okruženju. Mrežna karta popunjenosti dobivena je pretvaranjem statičkog prostora definiranog kartom koju Saphira koristi u obliku datoteke da bi u nju smjestila robota. Budući da dimenzije robota nisu uključene u algoritam pretraživanja, mrežna karta popunjenosti se dodatno treba popuniti maskom prepreka veličine robota oko već postojećih prepreka, da bi se izbjeglo uzimanje u proračun optimalne putanje onih polja na kojima se robot ne bi mogao nalaziti zbog svojih dimenzija. Mrežna karta popunjenosti se tijekom gibanja robota dodatno puni novim informacijama laserskih senzora i maskom robota. Kao što je već ranije opisano u 2.1.3 algoritam postavlja pokazivače na predhodne čvorove prilikom pretraživanja karte popunjenosti, a optimalni put se nalazi prateći pokazivače krenuvši od cilja pa sve do starta. Koristi se heuristična funkcija

dana izrazom 4.1 zbog koje se pretražuje dio prostora fokusiran oko optimalne putanje.

$$straight = max(abs(G.x - X.x), abs(G.y - X.y));$$
$$diagonal = min(abs(G.x - X.x), abs(G.y - X.y));$$
$$h(X, G) = (diagonal * \text{COSTDIAGONAL} + (straight - diagonal) * \text{COSTSTRAIGHT}); \quad (4\text{-}1)$$

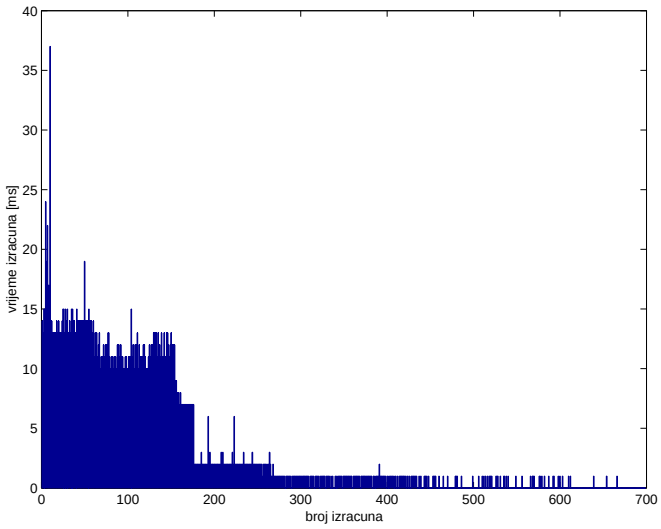
gdje su $G.x$, $G.y$, $X.x$, $X.y$ x i y koordinate na karti ciljnog i promatranog čvora, COSTDIAGONAL je duljina dijagonale polja, a COSTSTRAIGHT je duljina stranice polja. Na slici 4.1, koja prikazuje Saphirin prozor tijekom izvođenja simulacije, možemo vidjeti koja se sve polja pretražuju ovim algoritmom dok se robot nalazi u startu. Za A* algoritam je karakte-



Sl. 4.1: Prikaz Saphirinog prozora tijekom simulacije s kartom uredskog prostora, nakon inicijalnog izvršavanja A* algoritma. Ljubičasta polja su pretražena stanja koja su nekad bila na OPEN listi. Plava polja predstavljaju prepreku proširenu maskom robota.

ristično da je OPEN lista na kraju svakog izračuna prazna jer se cijeli proračun izvršava od trenutne pozicije do cilja s heuristikom koja osigurava proširivanje samo najboljih čvorova koji će zbog toga biti skinuti s liste. OPEN lista je realizirana kao cjelobrojna lista koja pamti samo pozicije stanja. Na taj se način ubrzava vrijeme izvođenja algoritma. Algoritam se računa u svakom ciklusu ponovo od trenutne pozicije do cilja da bi se mogla optimalna putanja povezati s algoritmom dinamičkih prozora. Vremena izračuna algoritma u simulaciji tijekom gibanja od startne do ciljne pozicije prikazana su grafom 4.2. Možemo primijetiti opadajuće iznose vremena izračuna što odgovara smanjenju prostora kojeg treba pretražiti u svakom sljedećem ciklusu.

Eksperiment za ovaj algoritam nije izveden zbog prevelikog vremena izračuna koji pre-

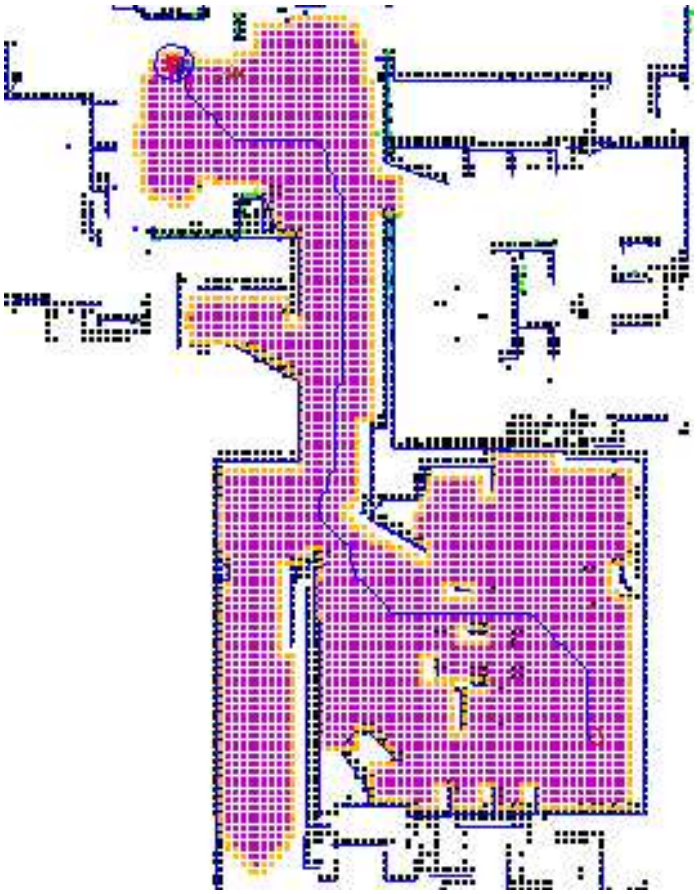


Sl. 4.2: Prikaz vremena izračuna A* algoritma u ovisnosti o broju ciklusa.

lazi trajanje ciklusa od $100ms$ što se u simulaciji tolerira, ali u eksperimentu nije izvedivo jer se prekida klijent/server veza.

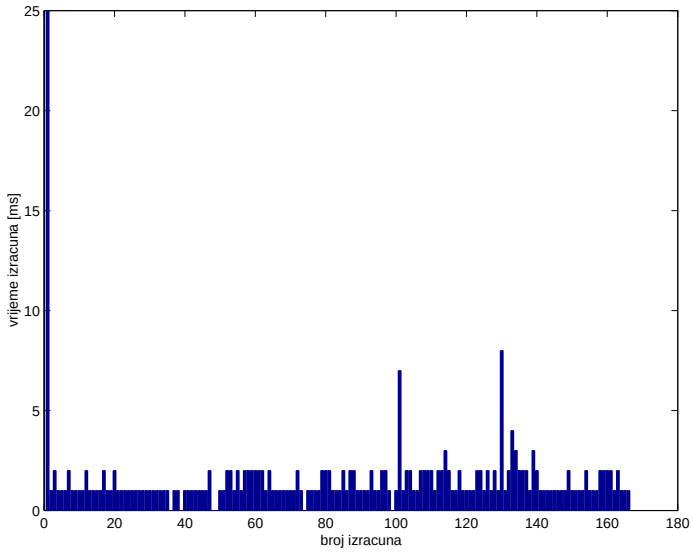
4.2 D* algoritam

Za implementaciju D* algoritma na mobilnom robotu koristi se ista mrežna karta popunjenosti statičkog prostora uz dodatna proširenja preprekama kao i za A* algoritam. Dodatno se koristi karta cijena prijelaza kreirana iz karte popunjenosti prema već opisanom postupku u poglavlju 2.1.4. Zbog toga što D* algoritam ne koristi heurističnu funkciju čvorovi će se proširivati od ciljnog čvora radijalno oko cilja sve dok se ne prošire do startnog čvora. Na slici 4.3, koja prikazuje Saphirin prozor tijekom izvođenja simulacije, možemo vidjeti koja se sve polja pretražuju ovim algoritmom dok se robot nalazi u startu. Ako usporedimo slike 4.1 i 4.3 vidimo da je ovim algoritmom pretražen znatno veći prostor što znači dulje vrijeme izvođenja. Međutim, to vrijeme je potrošeno samo jednom, u inicijalnom izračunu algoritma. Stanja na OPEN listi su rubna polja pretraženog područja zbog karakteristike pretraživanja D* algoritma. Na listi ostaju oni čvorovi koji imaju cijenu sličnu cijeni starta, a udaljenost starta od cilja određuje širinu područja oko cilja. Ti rubovi mogu sadržavati i prepreku jer D* algoritam uzima u proračun i ona polja koja sadržavaju prepreku. Optimalna putanja je izlomljena geometrijska linija. U integraciji s algoritmom dinamičkih prozora postaje glatka trajektorija, a robot postaje siguran od sudaranja s preprekom. Budući da glatka trajektorija odstupa od izračunate optimalne putanje, robot se više neće nalaziti na prije izračunatoj globalnoj putanji. Zbog svojstva D* algoritma da su svi pokazivači u kružnom



Sl. 4.3: Prikaz Saphirinog prozora tijekom simulacije s kartom uredskog prostora, nakon inicijalnog izvršavanja D* algoritma. Žuta polja predstavljaju ona stanja koja su na OPEN listi, a ljubičasta polja su pretražena stanja koja su nekad bila na OPEN listi. Plava polja predstavljaju prepreku proširenu maskom robota.

području oko cilja usmjereni prema optimalnoj putanji, robot će se ponovo nalaziti na optimalnoj putanji bez potrebe da se ponovo izvrši algoritam pretraživanja. To svojstvo A* algoritam ne posjeduje i zato se A* algoritam mora izvršavati u svakom ciklusu. Vremena izračuna algoritma u simulaciji tijekom gibanja od startne do ciljne pozicije prikazana su grafom 4.4. Postavljen je uvjet da se algoritam ne izvršava u svakom ciklusu nego samo pri promjeni pozicije polja u kojem se robot nalazi. Možemo primijetiti da jedini bitan iznos vremena izračuna određuje inicijalni proračun. Primjećuje se izračun u slučaju promjene na 100-tom i 140-om polju putanje. Ostali izračuni se odnose samo na ispitivanje da li je došlo do promjene i mogu se zanemariti.



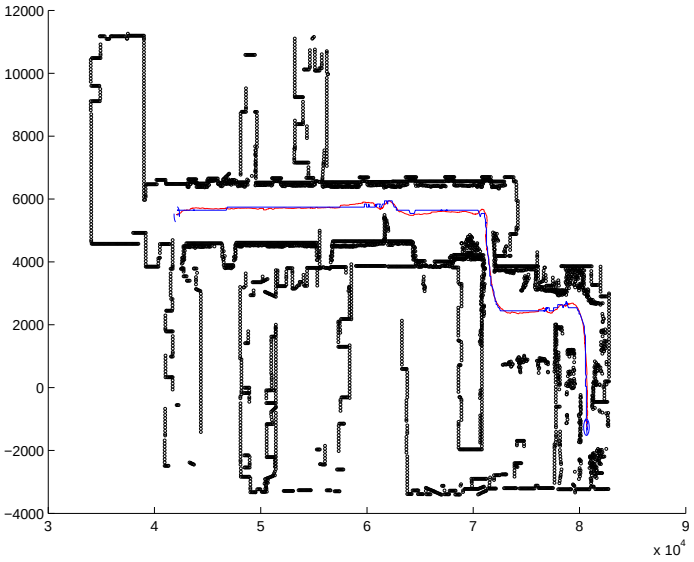
Sl. 4.4: Prikaz vremena izračuna D* algoritma u simulaciji kod svakog izvršavanja algoritma.

4.2.1 Eksperimentalni rezultati D* algoritma

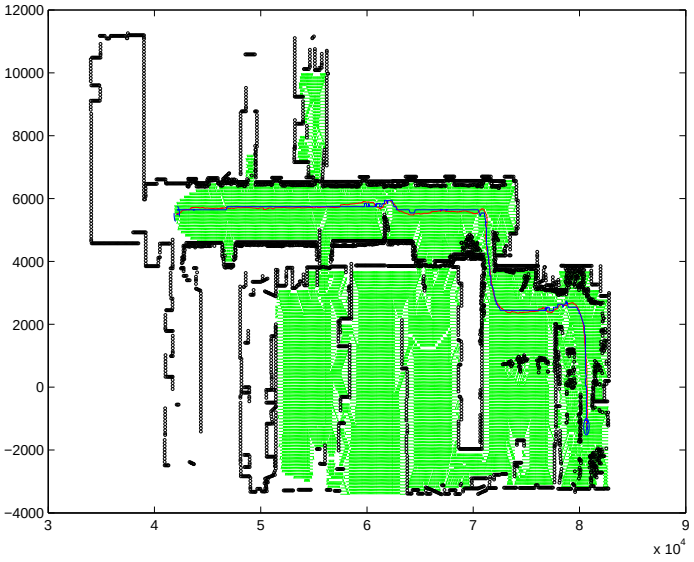
Konačno su prije teoretizirana razmatranja i problemi istestirani na stvarnom robotu. Slika 4.5 prikazuje gibanje robota po zavodu optimalnom putanjom izračunatom D* algoritmom i modulom izbjegavanja prepreka. Na slici 4.6 je zelenom bojom označeno područje koje je bilo pretraženo D* algoritmom. Na slici 4.7 je prikazan dio prostora u kojem je došlo do odudaranja informacija o prostoru iz laserskih očitanja od mrežne karte popunjenosti što se očituje preusmjeravanjem pokazivača na pojedinim poljima. Pokazivači su prikazani strelicama koje robot treba slijediti iz jednog polja u drugo da bi se gibao optimalnim putem. U opisu slike su navedena značenja pojedine boje strelice.

Na slici 4.8 je prikazan dio prostora s ucrtanim pokazivačima iz svakog polja kako bi se prikazalo usmjeravanje robota prema optimalnoj putanji.

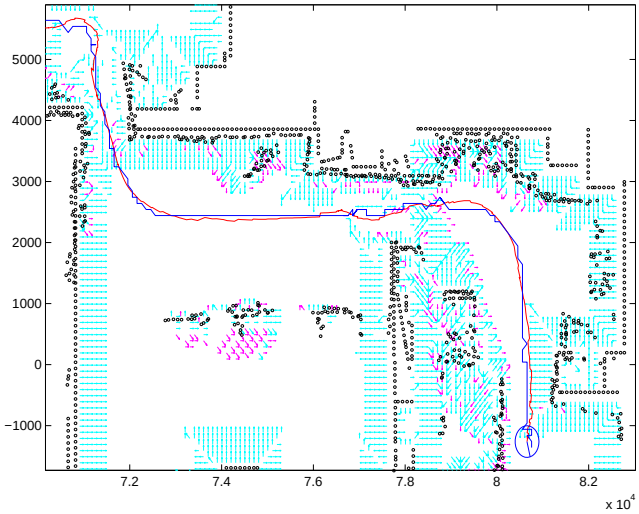
Vremena izračuna D* algoritma tijekom gibanja robota od startne do ciljne pozicije



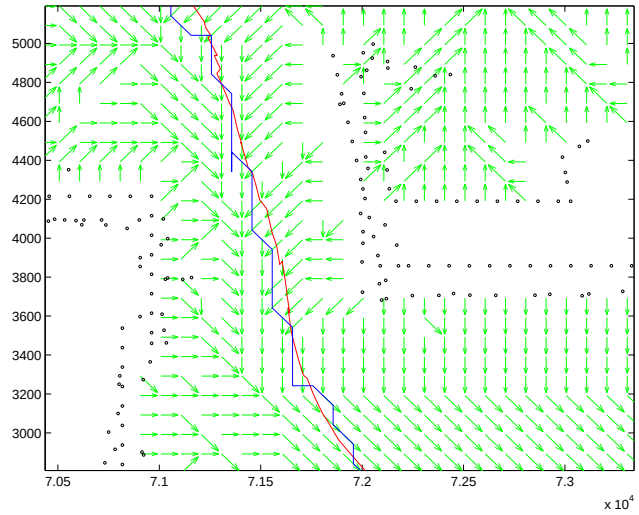
Sl. 4.5: Prikaz karte popunjenosti, globalne putanje koja je prikazana crvenom linijom i geometrijske planirane putanje koja je prikazana plavom linijom.



Sl. 4.6: Prostor pretražen D* algoritmom.

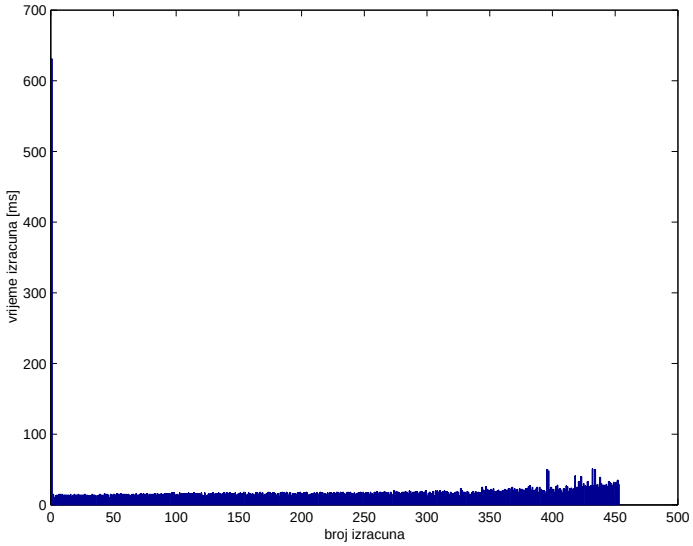


Sl. 4.7: Prikaz pokazivača koji su se preusmjerili zbog promjene popunjenosti prostora. Ljubičaste strelice predstavljaju RAISE stanja koja šire porast cijene puta na susjedna stanja, a svijetlo plave strelice predstavljaju LOWER stanja koja preusmjeravaju pokazivače u svrhu smanjenja cijene puta.



Sl. 4.8: Prikaz pokazivača na sljedeće polje te putanje koje ih slijede..

na zavodu prikazana su grafom 4.9. Bitan iznos vremena izračuna je u inicijalnom trenutku. Pred kraj putanje očituju se, u slučaju promjene, sve dulji proračuni jer se u zadnjoj prostoriji nalaze brojne prepreke koje nisu upisane u kartu.



Sl. 4.9: Prikaz vremena izračuna D* algoritma u eksperimentu kod svakog izvršavanja algoritma.

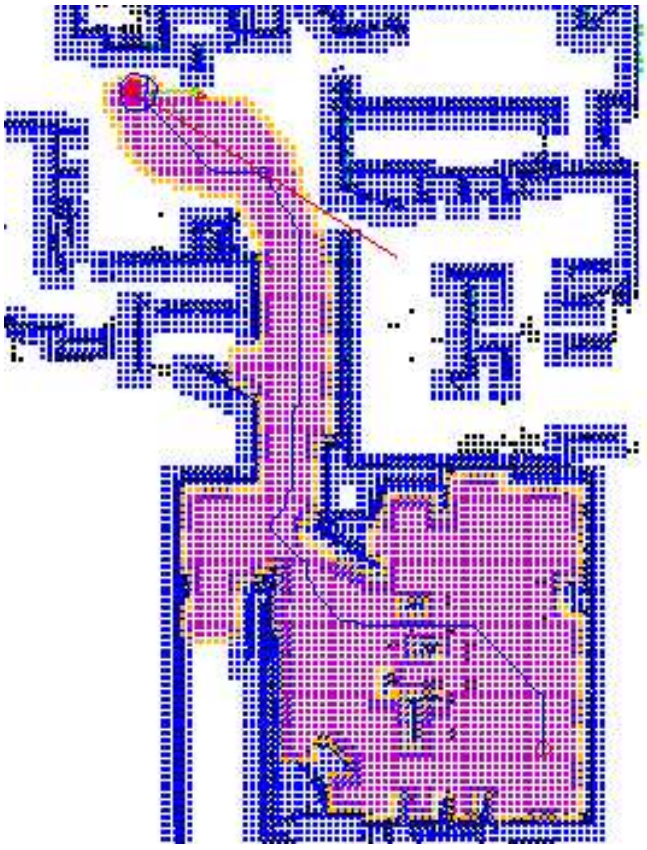
4.3 Fokusirani D* algoritam

Fokusirani D* algoritam ima brže vrijeme izvođenja od D* algoritma jer koristi heurističku funkciju $g(X)$ određenu izrazom 4.3 kao estimiranu vrijednost cijene od čvora X do startnog čvora, pa se pretražuje područje obuhvaćeno konturnom krivuljom oko starta i cilja, koju određuje funkcija $f(X) = g(X) + h(X)$ (što je slučaj kod već prije opisanog A* algoritma), a ne cijelo kružno područje kojem je cilj središte, a obuhvaća start.

$$\begin{aligned} straight &= max(abs(S.x - X.x), abs(S.y - X.y)); \\ diagonal &= min(abs(S.x - X.x), abs(S.y - X.y)); \\ g(X,S) &= (diagonal * \text{COSTDIAGONAL} + (straight - diagonal) * \text{COSTSTRAIGHT}); \end{aligned} \tag{4-2}$$

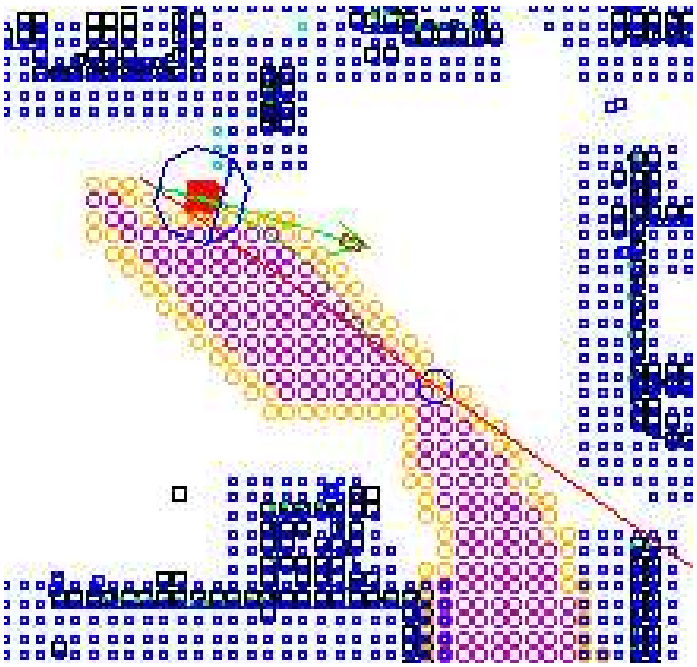
gdje su $S.x$, $S.y$, $X.x$, $X.y$ x i y koordinate na karti startnog i promatranog čvora, COSTDIAGONAL je duljina dijagonale polja, a COSTSTRAIGHT je duljina stranice polja. Nedostatak fokusiranog D* algoritma očituje se pri integraciji s algoritmom dinamičkog prozora. Naime, odstupanje trajektorije izračunate dinamičkim prozorom može biti takvo da se robot nađe na poziciji čiji čvor nije bio proširen algoritmom pretraživanja. Taj čvor neće imati pokazivač na sljedeći čvor i robot neće znati nastaviti putanju. Primjer ovakvog slučaja je ako se

cilj nalazi na suprotnoj strani nego što je robot orijentiran. Prilikom okretanja robot lako može izaći iz proračunatog područja. Proračunato područje ovisi o odabranoj heuristici. Što je heuristika bolja, to je proračunato područje uže, što je pozitivna osobina sa strane utrošenog vremena, a loša sa strane češćeg slučaja ispadanja s proračunatih putanja. Ovaj problem je međutim riješen malom izmjenom algoritma u inicijalnom proračunu optimalnog puta. Čvorovi se ne pretražuju dok se ne prošire do starta, već se granica pretraživanja pomiče za određeni broj polja dalje od starta u smjeru koji gleda suprotno od cilja. Ovu modifikaciju algoritma možemo uočiti na slici 4.10, koja prikazuje Saphirin prozor tijekom izvođenja simulacije. Za ovaj slučaj je granica pretraživanja pomaknuta za jedno dodatno polje od starta. Bez tog pomaka robot izađe iz izračunatog područja što je slučaj slike 4.11.



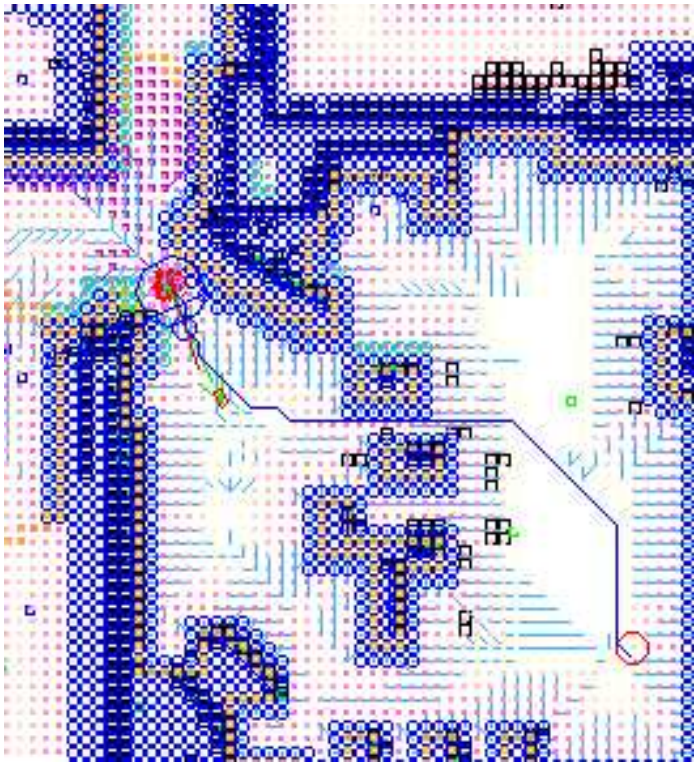
Sl. 4.10: Prikaz Saphirinog prozora tijekom simulacije s kartom uredskog prostora, nakon inicijalnog izvršavanja fokusiranog D* algoritma u kojem je granica pretraživanja pomaknuta za jedno dodatno polje. Žuta polja predstavljaju ona stanja koja su na OPEN listi, a ljubičasta polja su pretražena stanja koja su nekad bila na OPEN listi. Plava polja predstavljaju prepreku proširenu maskom robota.

Dodatna modifikacija uvedena za ovaj algoritam jest način proračuna u slučaju promjene okupiranosti. U D* algoritmu su se uzimale u proračun ona polja kojima se promje-

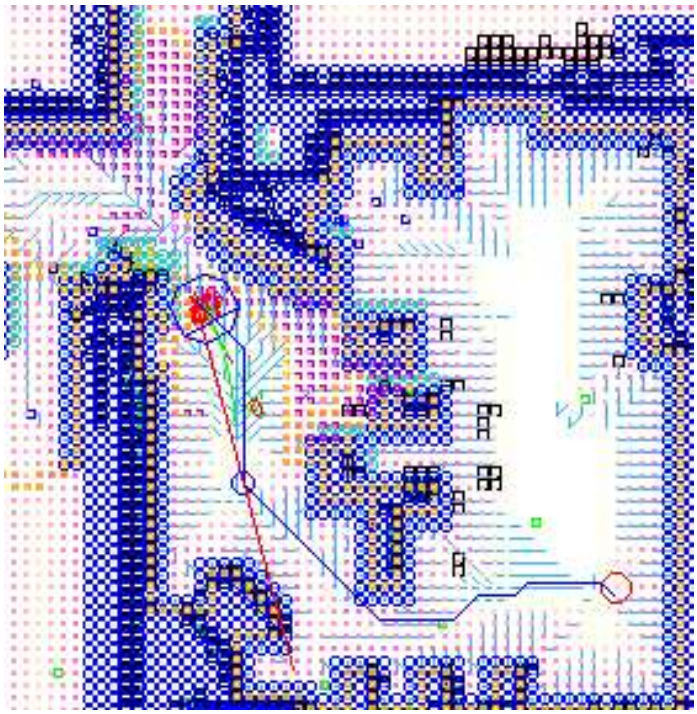


Sl. 4.11: Slučaj ispada robota iz izračunatog područja..

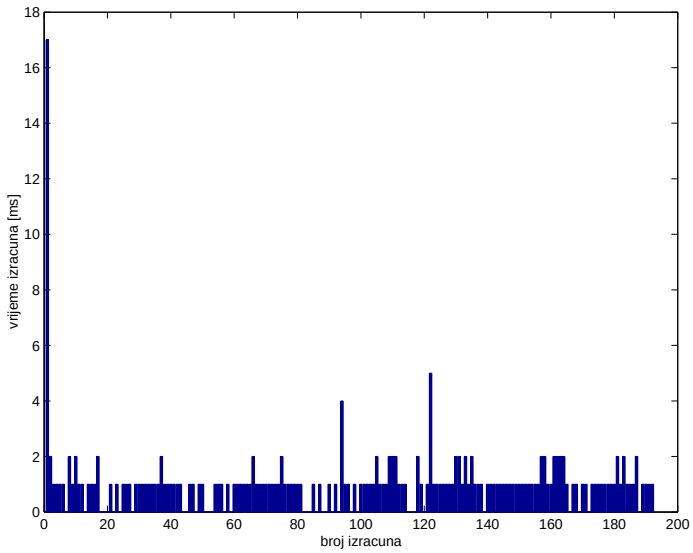
nila okupiranost, koje se nalaze u pretraženom području. Ovdje proračun polazi od onih promijenjenih polja koja se nalaze na optimalnoj putanji. Slika 4.12 pokazuje gibanje robota po optimalnoj putanji prije nego što otkrije da kroz uski prolaz ne može proći, odnosno, prije nego što u mrežnoj karti popunjenosti, u kojoj se sve prepreke proširuju s maskom robota, ne otkrije popunjenost polja koje se nalazi na optimalnoj putanji. Sljedeća slika 4.13 pokazuje izračunatu novu putanju do cilja izračunatu zbog promjene popunjenosti polja na prijašnjoj optimalnoj putanji. Na slikama su prikazani relevantni parametri algoritma pomoću kojih se objašnjava na koji način algoritam dolazi do nove putanje i navedeni su u opisu slike. Na slici 4.12 mogu se uočiti RAISE stanja kao ljubičasti kružići koji se na slici nalaze gore iza robota. Ta stanja su nastala zbog očitane promjene popunjenosti polja u blizini pozicije gdje se robot sada na slici nalazi. Područje koje je obrađeno zbog te promjene okruženo je žutim poljima kao stanjima na OPEN listi. Na slici 4.13 možemo primijetiti pojavu novih RAISE stanja te koje je područje pretraženo zbog promjene popunjenosti. Primijetimo još da su se pojavile plave crtice u okolini robota, koje predstavljaju preusmjerene pokazivače, koje preusmjeravaju robota na drugi put. Vremena izračuna algoritma u simulaciji tijekom gibanja od startne do ciljne pozicije prikazana su grafom 4.14. Kao za D* algoritam, postavljen je uvjet da se algoritam ne izvršava svaki ciklus nego po promjeni pozicije polja u kojem se robot nalazi. Najveći iznos vremena izračuna određuje inicijalni proračun. Grafovi kod D* i fokusiranog D* algoritma su sličnog oblika, samo su vremena fokusiranog D* algoritma znatno manja.



Sl. 4.12: Prikaz Saphirinog prozora tijekom simulacije s kartom uredskog prostora. Žuta polja predstavljaju ona stanja koja su na OPEN listi, a ljubičasta polja su RAISE stanja. Plava polja predstavljaju prepreku proširenu maskom robota. Zelena polja predstavljaju novo otkrivenu popunjenost. Iscrtana je optimalna putanja od trenutne pozicije do cilja. Svijetloplave crtice predstavljaju pokazivače koji su se preusmjerili zbog promjene. U svijetlim tonovima je točkasto prikazana karta cijena prijelaza..



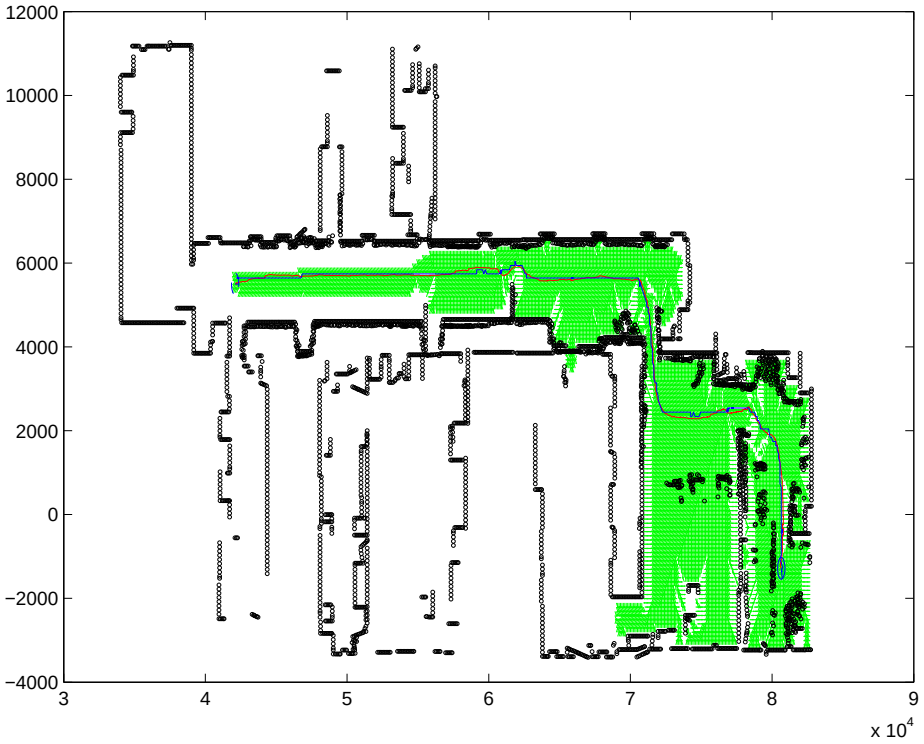
Sl. 4.13: Sljedeći trenutak prethodne slike.



Sl. 4.14: Prikaz vremena izračuna fokusiranog D* algoritma u simulaciji kod svakog izvršavanja algoritma.

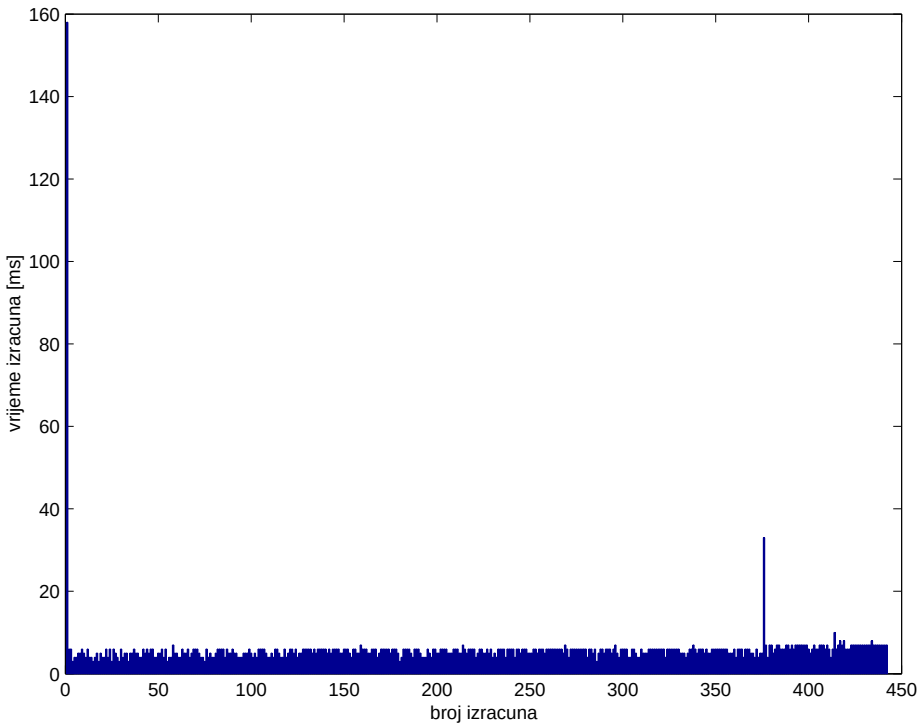
4.3.1 Eksperimentalni rezultati fokusiranog D* algoritma

Istestiran je fokusirani D* algoritam na stvarnom robotu. Slika 4.15 prikazuje gibanje robota po zavodu optimalnom putanjom izračunatom D* algoritmom i modulom izbjegavanja prepreka. Također je na slici označeno područje koje je bilo pretraženo algoritmom. Usporedbom slika 4.6 i 4.15 vidimo da je ovaj algoritam pretražio znatno manji prostor. Putanje su ostale iste budući da su oba algoritma optimalna. Vremena izračuna fokusiranog D*



Sl. 4.15: Prikaz karte popunjenosti, globalne putanje koja je prikazana crvenom linijom i geometrijske planirane putanje koja je prikazana plavom linijom.

algoritma tijekom gibanja robota od startne do ciljne pozicije na zavodu prikazana su grafom 4.16. Bitan iznos vremena izračuna je u inicijalnom trenutku, ali je on manji nego u slučaju D* algoritma. U slučaju promjene se očituju veći izračuni pred kraj gibanja robota jer se u zadnjoj prostoriji nalaze brojne prepreke koje nisu upisane u kartu.



Sl. 4.16: Prikaz vremena izračuna fokusiranog D* algoritma u eksperimentu kod svakog izvršavanja algoritma.

4.4 Usporedba vremena izvođenja u simulaciji

	vrijeme inicijalnog izračuna	najveće vrijeme on-line izračuna
A* algoritam	$15ms$	$37ms$
D* algoritam	$25ms$	$8ms$
Fokusirani D* algoritam	$17ms$	$5ms$

Tablica 4.1: Usporedba vremena izvođenja algoritama.

4.5 Usporedba vremena izvođenja u eksperimentu

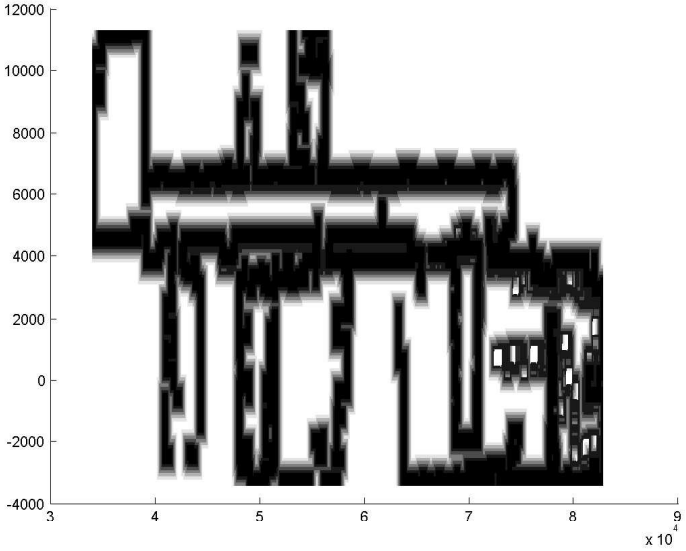
	vrijeme inicijalnog izračuna	najveće vrijeme on-line izračuna
A* (simulacija)	$152ms$	$192ms$
D* algoritam	$630ms$	$50ms$
Fokusirani D* algoritam	$160ms$	$37ms$

Tablica 4.2: Usporedba vremena izvođenja algoritama.

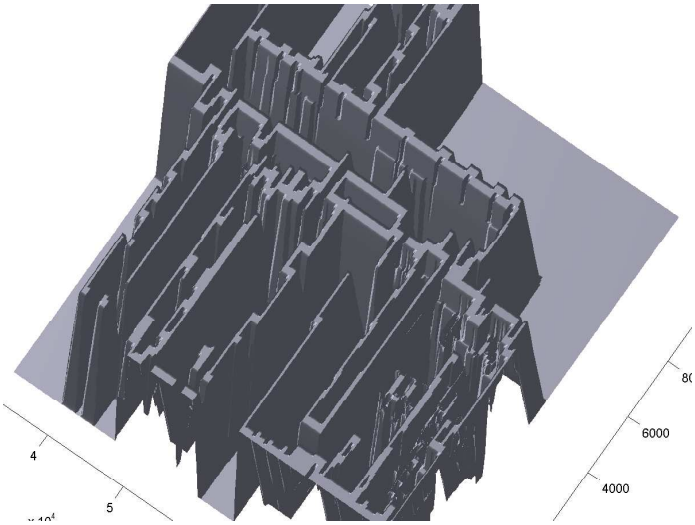
4.6 Karte zavoda

Za kraj ovog poglavlja je dan prikaz zavoda u obliku karta cijena prijelaza, slika 4.17. U opisu slike su pojašnjena značenja pojedinih nijansi.

Slika 4.18 predstavlja isto što i slika 4.17 ali u 3D prikazu.



Sl. 4.17: Karte zavoda prikazana u obliku karte cijene prijelaza. Crna boja polja označava područje u kojem se robot ne može nalaziti zbog popunjenosti prostora preprekom ili njegove širine. Bijela boja predstavlja slobodna polja u karti. Nijanse sive boje predstavljaju povećanje cijene prijelaza u pojedino polje u ovisnosti o blizini prepreke. Tamnija boja odgovara većoj cijeni. Varijabla COSTMASK postavljena je na 5 polja od prepreke.



Sl. 4.18: Mapa s vrijednostima cijene prelaska u pojedino stanje u 3D prikazu. Visoki zid predstavlja prepreku, ravan pod predstavlja slobodan prostor. Niže stepenice zida su nastale zbog povećane cijene prijelaza u pojedino stanje blizu zida.

Poglavlje 5

Zaključak

U pretraživanju prostora mobilnog robota za pronalaženje globalnog cilja korišten je tzv. A* algoritam, koji sadrži dvije funkcije ocjene: cijenu prijeđenog puta i heurističku funkciju procjene preostale cijene do cilja. Heuristika koja dobro procjenjuje stvarnu cijenu određenog puta može znatno ubrzati vrijeme izvođenja algoritma, s obzirom na uniformni tip pretraživanja gdje je heuristički doprinos jednak 0. No, mora biti ispunjen uvjet da heuristika ne precijeni cijenu puta, u protivnom potpunost algoritma (sigurno pronalaženje cilja, ako taj postoji) nije osigurana. Pretraživanje statičke okoline može se najbrže izvršiti upravo A* algoritmom. Međutim, u slučaju pretraživanja djelomično poznatih i dinamičkih prostora A* algoritam se mora izvršiti u svakom ciklusu u kojem dođe do promjene u okolini. Pri tome se pretraživanje ponovno izvodi od starta do cilja, bez korištenja informacija o konfiguraciji prepreka i putanje iz prijašnjih ciklusa, što u slučaju dinamičke okoline dovodi do neučinkovitog izvođenja. Pošto se vrijeme izvođenja između dvije točke uz statičku kartu prostora ne mijenja, to također znači da uz veliku kartu prostora može postojati ograničavajuća udaljenost za izvođenje algoritma. Izloženi D* algoritam rješava problem pretraživanja djelomično poznatih prostora ili dinamičkih prostora. Vrijeme izvođenja algoritma ovisi o broju već pretraženih čvorova i da li neki čvor sadrži novu informaciju o prostoru (mjerenje senzora). Ako se cijeli prostor inicijalno pretraži, potrebno je u kasnijem izvođenju obnoviti samo čvorove sa novim podacima i njihove susjedne čvorove, što može znatno ubrzati izvođenje algoritma u stvarnom vremenu. Pri tome je također zadovoljen kriterij potpunosti, dakle pronalaženja cilja ako taj postoji. D* algoritam se dodatno može ubrzati uvođenjem heurističke funkcije što vodi prema fokusiranom D* algoritmu. Na robotskoj platformi je implementiran modul lokalnog izbjegavanja prepreka koji se izvodi neovisno o globalnom pretraživanju prostora. Taj modul osigurava sigurno kretanje robota i u slučaju kada globalna putanja, odnosno cilj nije dostupan. Dozvoljene trajektorije koje se generiraju u tom modulu s obzirom na trenutni vektor brzina te kinematička i dinamička ograničenja robota, koriste se u integraciji sa proračunatom globalnom geometrijskom putanjom u svrhu dobivanja glatke globalne trajektorije robota. Pri tome se uvodi tzv. učinkovita putanja čija se

orijentacija i duljina u svakom servo koraku prilagođava s obzirom na trenutni vektor brzina robota i lokalnu konfiguraciju globalne geometrijske putanje. Inkrementalni odabir optimalne trajektorije prema geometrijskoj putanji temelji se na usporedbi udaljenosti točaka na obje krivulje. Izvedene su simulacije pretraživanja i gibanja robota u tipičnom unutarnjem uredskom prostoru te prikazan proces pretraživanja prostora kroz relevantne varijable karte prostora. Također je izveden eksperiment na realnoj robotskoj platformi i prikazana globalna geometrijska putanja te stvarna glatka trajektorija robota. Uspoređena su vremena izvođenja sva tri algoritma.

Poglavlje 6

Literatura

1. Anthony Stentz,"Optimal and Efficient Path Planning for Partially-Known Environments", Proc. IEEE International Conference on Robotics and Automation, May, 1994.

2. Anthony Stentz, "The Focussed D* Algorithm for Real-Time Replanning", Proc. International Joint Conference on Artificial Intelligence, August, 1995.

3. D. Fox, W. Burgard and S. Thrun, The Dynamic Window Approach to Collision Avoidance, IEEE Robotics & Automation Magazine, 4(1), 1997, pp.23-33.

4. Maček, K., Petrović, I., Ivanjko, E.: "An Approach to Motion Planning of Indoor Mobile Robots", IEEE International Conference on Industrial Technology - ICIT2003, pp. 969-973, Maribor, Slovenia, December, 2003.

5. Stuart J. Russell and Peter Norvig, "Artificial Intelligence - A Modern Approach", Prentice Hall, 1995.

6. "Pioneer & PeopleBot Mobile Robots: Computer & Systems Manual", ActivMedia Robotics, June, 2001.

7. "Saphira Software Manual", SRI International, 2001.

8. K. Konolige and K. Myers, "The saphira architecture for autonomous mobile robots. In Artificial Intelligent and Mobile Robots", chapter 9, pp. 211-242, 1996.

9. Kurt Konolige, "A gradient method for realtime robot control", In Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems, 2000.

10. Kurt Konolige, Ken Chou: "Markov Localization using Correlation ", International Joint Conference on Artificial Intelligence, pp. 1154-1159, 1999.

11. UNIX Network Programming, Volume 1, Second Edition: Networking APIs: Sockets and XTI, Prentice Hall, 1998, ISBN 0-13-490012-X.

Sažetak

U ovom radu se opisuju metode navigacije mobilnih robota u unutrašnjim prostorima. Opisane su strategije pretraživanja prostora mobilnog robota s naglaskom na pretraživanje grafova. Implementirani A* algoritam pretraživanja temelji se na funkciji cijene puta i heurističnoj funkciji kao estimiranoj cijeni ostatka dijela globalne putanje. Odabirom dobre heuristike se značajno ubrzava vrijeme izvođenja. A* algoritam je optimalan po kriteriju cijene puta, kompletan (nalazi globalnu putanju ako ona postoji) i najbrži u pronalaženju globalnog puta u statičkim prostorima. Međutim, u dinamičkim prostorima u kojima se raspored prepreka s vremeno mijenja, A* algoritam nije učinkovit budući da ne koristi nove informacije o promjenjivom prostoru. U tom slučaju se koristi D* algoritam koji obnavlja samo one čvorove na putu u kojima su se promijenila senzorska očitanja. D* algoritam je također optimalan i kompletan. Njegova se brzina izvođenja dodatno može poboljšati uvođenjem heurističke funkcije što vodi prema novoj inačici D* algoritma - fokusiranom D* algoritmu. Globalna geometrijska putanja nastala algoritmom pretraživanja mrežne karte popunjenosti treba biti pretvorena u glatku trajektoriju uzimajući u obzir dinamička i kinematička ograničenja robota. Dopuštene trajektorije generirane su modulom lokalnog izbjegavanja prepreka (algoritam dinamičkih prozora) koji omogućava siguran put robotu. Prihvaćene trajektorije su uspoređivane točku po točku s globalnom geometrijskom putanjom s ciljem nalaženja optimalne trajektorije koja zadovoljava lokalno izbjegavanje prepreka i praćenje globalne putanje prema cilju. Algoritmi su provjereni simulacijom i eksperimentom na Pioneer 2DX mobilnom robotu.

Ključne riječi: navigacija mobilnog robota, planiranje putanje, strategije pretraživanja, A* algoritam, D* algoritam, fokusirani D* algoritam, mrežne karte popunjenosti, izbjegavanje prepreka, praćenje puta

Abstract

This work presents a navigation method for mobile robots in indoor environments. A short survey of different techniques for acquiring the global path of mobile robot is presented with the main focus on graph based search methods of global geometrical path. The implemented A* graph search method is based on a path cost function and a heuristic function which estimates the cost along the remaining path to the global goal. Choosing a good heuristic function can significantly improve the computational cost. The A* algorithm is both optimal (the optimal path is found) and complete (it is guaranteed to find the global path if such exists) and performs fastest for the first global search in a static environment. However, in a dynamic environment where the environment changes at each servo tick the A* algorithm may perform poorly since no search information is used from previous iterations. Therefore, D* graph search algorithm is used, which allows updating of only those nodes along the path that actually change due to sensor measurements. The D* algorithm is also both optimal and complete. Its computation effort can be further improved by using a heuristic function for the remaining cost of the path which leads to the focused D* algorithm variant. The global geometric path generated by a graph search method on a grid map must be further transformed in a robot trajectory by taking into account the dynamic and kinematic robot constraints. In the presented work, the admissible robot trajectories are generated in a local obstacle avoider module (Dynamic Window approach) that ensures safe robot operation. Acquired trajectories are compared point to point to the global geometric path to find the optimal trajectory that satisfies both local obstacle avoidance and global path following towards a goal. The algorithms were verified both in simulation and on a Pioneer 2DX mobile robot where a good correlation was proven.

Keywords: mobile robot navigation, path planning, graph search methods, A* algorithm, D* algorithm, focussed D* algorithm, obstacle avoidance, path following

Životopis

Marija Seder rođena je 18.5.1981. godine u gradu Zagrebu, R. Hrvatskoj. Maturirala je 1999. godine u prirodoslovno matematičkoj V. Gimnaziji u Zagrebu.

Studentica je pete godine na Fakultetu elektrotehnike i računarstva. Prošle akademske godine (2002/2003) odobren joj je završetak studija s naglaskom na znanstveno istraživački rad pod mentorstvom Doc.dr.sc. Ivana Petrovića u području mobilne robotike. Dobitnica je priznanja Josip Lončar za primjeran uspjeh na drugoj godini studija (2000/2001) te četvrtoj godini studija (2002/2003).

Pohađala je 7 godina muzičku školu gdje je svirala cello. Rekreativno se bavi košarkom. Vlada engleskim jezikom, a nešto i njemačkim.