SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1363

ESTIMACIJA POLOŽAJA MOBILNOG ROBOTA TEMELJENA NA KARTI PROSTORA

MARIJAN METIKOŠ

Zagreb, rujan 2003.

Zahvljujem se doc. dr. sc. Ivanu Petroviću i dipl. ing. Edouardu Ivanjku na pomoći tijekom izrade diplomskog rada.

Ovaj diplomski rad posvećujem pola svojim roditeljima koji su mi omogućili studiranje, a drugu polovicu ženici jer mi je pružala podršku tijekom čitavog studija.

SADRŽAJ

1. UVOD

Danas bi teško bilo zamisliti svijet bez autonomnih sustava. Oni su važan dio društva, a u budućnosti će njihova uloga sve više rasti. Sposobnost autonomnog rada, tj. bez intervencije čovjeka, omogućuje njihovu primjenu u velikom broju poslova, koje su do sada obavljali ljudi. Autonomni sustavi posjeduju sposobnost samoučenja te reagiranja na nepredviđene situacije u radu. Omogućuju kvalitetnu zamjenu čovjeka u obavljanju repetitivnih poslova (npr. na trakama), istraživanju opasnih i nepristupačnih područja, poslovima u opasnim okolinama (npr. nuklearnim elektranama) i dr. Postali su nezamjenjivi u suvremenoj proizvodnji zbog svoje točnosti i pouzdanosti.

Autonomna vozila i mobilni roboti predstavljaju vrlo važnu klasu autonomnih sustava. Autonomna vozila su jedan od najčešće korištenih autonomnih sustava. Koriste se za transport (materijala i osoba), istraživanje, pružanje raznih usluga (vodiči u muzejima) te u vojnim primjenama (izviđanje, čuvanje objekata).

Istraživanja iz područja autonomnih sustava vrlo su intenzivna. Obavljaju ih interdisciplinarni timovi znanstvenika specijalista iz područja automatike, mjerne tehnike, strojarstva i računarstva te komunikacija. Zahtijevaju integraciju osjetila (mjernih članova), algoritama za planiranje putanje i algoritama djelovanja (reakcije na promjene u radnoj okolini) u jedinstveni sustav. Jedino ako je integracija nabrojanih komponenata kvalitetno napravljena, autonomni je sustav sposoban snalaziti se u radnoj okolini, izvršavati radne zadatke i na odgovarajući način djelovati na nepredviđene situacije. Autonomni sustav koristi osjetila (mjerne članove) za prikupljanje informacija o radnoj okolini. Na temelju ugrađenih algoritama navigacije (planiranje putanje), dobivenih podataka o stanju u okolini i naredbi autonomni sustav izvršava svoj radni zadatak.

Za autonomnu navigaciju mobilnog robota kroz prostor navigacijski sustav robota mora u svakom trenutku znati položaj robota u prostoru s velikom točnošću. Jedan od načina koji to mobilnom robotu omogućuje je Markovljeva lokalizacija koja estimira položaj robota na osnovi karte prostora robota.

Da bi se robot mogao gibati s minimalnim troškovima potrebno je implementirati u njegov operacijski sustav jednu od metoda koje izračunavaju optimalnu stazu za njegovo kretanje. U tom smislu, kao najbolja, pokazala se gradijentna metoda koja omogućuje postizanje optimalne brzine u stvarnom vremenu čak i kad robot istražuje nova područja ili nailazi na neočekivane prepreke.

U ovom radu su detaljno razrađene obje metode (Markovljeva i gradijentna) te je dan opis dobivanja karte prostora korištenjem aplikacija instaliranih u *Saphiri* i informacija dobivenih senzorima robota.

2. POTREBNE APLIKACIJE

2.1. Uvod

Za upravljanje mobilnim robotom koristi se Saphira 8.x koja je bazirana na Arii, upravljačkom sustavu robota niže razine (proizvod tvrtke Activ Media Robotics). Dobro poznavanje Arie je neophodno za rad sa Saphirom jer Saphira koristi puno njezinih klasa. Aria pruža fleksibilan pristup niže razine robotskom okruženju, dok Saphira pojednostavljuje zadatak programera jer pruža sučelje tim mogućnostima. Saphira je robotska okolina razvoja aplikacija i radi u klijent/server okolini. Ona sadrži velik broj rutina za izgrađivanje klijenata, a u njih su integrirane funkcije Arie za slanje naredaba serveru, prikupljanje informacija s robotskih senzora te njihovo pakiranje za prikaz za korisničko sučelje. Saphira dodatno pruža i funkcije više razine za upravljanje robotom i interpretaciju senzora, primjenu Colberta za pisanje upravljačkih programa te lokalizaciju korištenjem karte prostora i navigaciju.

Saphira klijent spaja se na server (robot ili simulator robota) s osnovnim komponentama za robotska osjetila i navigaciju: motorima, kotačima, enkoderima i senzorima. Server upravlja pokretanjem i senzorima funkcija niže razine šaljući im informacije te odgovaranjem Saphiri kroz Aria robotsko sučelje.

U programskom jeziku Colbert korisnici mogu brzo napisati ili ispravljati složene procedure upravljanja zvane Activities.

Saphira sadrži i sofisticirane algoritme za odrađivanje težih zadataka kao što su lokalizacija ili navigacija. Lokalizacija je zadatak praćenja položaja robota u okolini pomoću lasera ili sonara, a navigacija je zadatak određivanja dobre putanje do određenog cilja držeći robota podalje od nevolja i zapreka. Za lokalizaciju i navigaciju Saphira koristi linijski crtane karte prostora. Za jednostavne okoline, te karte se mogu napraviti jednostavnim unosom koordinata linija u tekstualnu datoteku, a za složenije okoline postoje programi kojima se automatski grade karte na osnovi prikupljanja informacija s laserskih senzora koje se obrađuju određenim algoritmima (npr. *ScanStudio*).

Saphira dolazi sa simulatorom robota u svojoj okolini što omogućava korisnicima da ne moraju čitavo vrijeme koristiti stvarni robot za ispitivanje napravljenih aplikacija. Simulator ima modelirane pogreške sonara, lasera i enkodera. Čak je i komunikacijsko sučelje isto kao i za fizički robot tako da korisnici ne moraju preuređivati aplikacije kad se sa simulatora prebacuju na stvarni robot. Simulator također omogućuje izgradnju 2D zamišljenog modela prostora (World). U takvim modelima segmenti linija predstavljaju okomite površine (zidove, ormare i sl.).

2.1.1. Dokumentacija i izvori

Razni aspekti Saphire i Arie obrađeni su u raznim priručnicima koji se mogu pronaći u mapi*/docs* .

Programi za vježbanje s kompletnim izvornim kodom nalaze se u podmapama mape */tutorial*. U sljedećoj tablici možete vidjeti kratki popis dokumenata i programa za vježbu s izvornim kodom.

Dokument	**Sadržaj**	**Kod za vježbu**
`actions.pdf`	Opisuje različite modove gibanja robota u Saphiri/Arii.	`tutor/movit` `colbert/direct.act`
`loadable.pdf`	Priručnik o prijevodu i učitavanju C++ programa u Saphiri.	`tutor/loadable`
`colbert.pdf`	Priručnik za programiranje u Cobertu.	

`colbert-user.pdf`	Priručnik (manual) za robotski programski jezik Colbert.	`tutor/movit`
		`tutor/loadable`
		`colbert/direct.act`
		`colbert/bump.act`

`motion.pdf`	Tehnički opis probabilističkog gibanja robota i lokalizacije.	`tutor/sample-move`
		`tutor/sample-update`
		`tutor/sonardist`

	Programi za vježbu za sinhrone i asinhrone procese u Saphiri.	`tutor/task`
	Programi za vježbu (behaviors) za izbjegavanje prepreka i praćenje zidova.	`tutor/crawl`
	Primjeri aplikacija pomoću kojih možete praviti vlastite Saphira programe.	`apps/`

Tablica 2.1. Popis dokumenata i programa za vježbu

Web stranice na kojima možete pronaći potrebne izvore Saphira Softwarea su: http://robots.activmedia.com.

Saphira se nalazi pod stalnim razvojem u SRI International. Njihove informacije , vježbe, projekte, kao i linkove na ostale web stranice koje koriste Saphiru i Pioneer, mogu se pronaći na adresi: http://www.ai.sri.com/~konolige/saphira.

2.2. Instalacija Saphire i dodatnih modula

Posljednje informacije o instalaciji i pokretanju Saphire mogu se pronaći u datoteci *readme* u instalacijskom paketu. Informacije o promjenama u odnosu na prethodne verzije Saphire mogu se pronaći u datoteci *update* i ona bi trebala poslužiti kao vodič pri nadogradnji s neke starije verzije.

Instalacijski paket Saphira koji uključuje demonstracijski program *saphira.exe*, Colbert, simulator robota i datoteke s izvornim kodom dolaze kao zapakirana arhiva datoteka i mapa na CD-ROM-u ili Activ Media Robotics web stranicama. Svaka arhiva je konfigurirana i prevedena za određeni operacijski sustav. Odaberite verziju koja odgovara vašem operacijskom sustavu i možete započeti s instalacijom.

Verzije za Windows OS dolaze zapakirane programom PKZIP, a za UNIX OS zapakirane programima TAR i GZIP. Da bi se instalacijski paket raspakirao u upotrebljive datoteke potrebno je upotrijebiti odgovarajuće programe.

2.2.1. Instalacija Saphire za Windows OS

Potrebno je raspakirati ZIP arhivu u po volji odabranu mapu. Prilikom raspakiravanja stvaraju se dvije nove mape *Saphira* i *Aria* koje moraju biti djeca iste mape. U *tablici 1.2.* može se vidjeti struktura mapa Saphire i Arie.

Saphira/	
bin/	
saphira(.exe)	Saphira/Colbert izvršna datoteka
pioneer(.exe)	Simulator robota
*.dll	Win32 dinamičke datoteke
lib/	Razvojne datoteke
*.lib, *.so	Linux/UNIX dijeljene datoteke
colbert/	Colbert primjeri
ohandler/	
include/	Saphira header datoteke
tutor/	Programi za vježbu
movit/	Razni primjeri
.....	
worlds/	WLD datoteke za simulator
readme	Tekstualna datoteka u kojoj su objašnjene osnove
update	Datoteka usporedbi verzija
license	Licenca
Aria/	
lib/	Ključne datoteke Arie
docs	Dokumentacija Arie
include	Aria header datoteke
src/	Aria datoteke s izvornim kodom

Tablica 2.2. Instalirane mape u Saphiri i Arii (verzija 8.0)

Sve operacije Saphire i Arie zahtijevaju da budu podešene sustavske varijable. Ako se one ispravno ne podese tada Saphira klijent i simulator ili neće raditi uopće ili neće raditi ispravno. Zbog toga ih je dobro podesiti odmah nakon instalacije. Ako je bilo potrebno samo nadograditi Saphiru ili Ariu tada će sustavske varijable ostati podešene od prije.

U `Windowsima 95/98/ME`, pod pretpostavkom da je Saphira instalirana u mapu `C:\Saphira`, potrebno je u datoteku `C:\AUTOEXEC.BAT` dodati sljedeći redak: `SET SAPHIRA=C:\Saphira`. Ako pak imate `Windows NT/2000/XP` tada je potrebno otići na `Start/Settings/System` i kliknuti karticu `Environment` i dodati varijable `SAPHIRA` i `ARIA` s pripadajućim stazama.

Dinamički dijeljene datoteke su `C:\Saphira\bin\sf.dll` i `C:\Aria\bin\aria.dll`. Njih je potrebno kopirati u vašu vlastitu mapu u kojoj se nalaze i izvršne datoteke za pokretanje vaših aplikacija Saphire ili Arie. Da bi ih se moglo koristiti po cijelom sustavu mogu se kopirati u mapu `C:\Windows\system` kod `95/98/ME` ili `C:\Winnt\System32` kod `NT/2000/XP`.

Da biste isprobali je li instalacija uspješno završena potrebno je pokrenuti izvršnu datoteku `saphira.exe iz` mape `C:\Saphira\bin\`. Saphira klijent bi se trebao pojaviti s grafičkim i interaktivnim prozorom. U interaktivnom prozoru može se upisati naredba `help` da bi se prikazala lista naredaba koje se mogu koristiti.

Nakon toga potrebno je spojiti se sa simulatorom ili robotom. Za simulator je potrebno pokrenuti datoteku `C:\Saphira\bin\pioneer.exe` na istom računalu te u interaktivnom prozoru upisati naredbu `connect local`, a za spajanje s robotom potrebno je logirati se na operacijski sustav koji je instaliran u njemu i tamo pokrenuti Saphiru te se spojiti s robotom pomoću naredbe `connect serial` ili preko padajućeg izbornika.

2.2.2. Instalacija Saphire za Linux OS

Prilikom instalacije Saphire pod `UNIX` ili `Linux` OS-om preporuka je da se instalacijski paket raspakira u neku standardnu mapu, npr. `/usr/local` ili neku drugu svim korisnicima dostupnu. Zatim se postave dozvole za pristup toj mapi korisnicima koji će raditi na robotu. Kopirajte Saphira arhivu u tu mapu, a zatim raspakirajte Linux naredbom `tar -zxvf linux80a.tgz`. Prilikom raspakiravanja stvaraju se dvije nove mape *Saphira* i *Aria* koje moraju biti djeca iste mape. U *tablici 1.2.* može se vidjeti struktura mapa Saphire i Arie.

Sve operacije Saphire i Arie zahtijevaju da budu podešene sustavske varijable. Ako se one ispravno ne podese tada Saphira klijent i simulator neće raditi uopće ili neće raditi ispravno. Zbog toga ih je dobro podesiti odmah nakon instalacije. Ako je bilo potrebno samo nadograditi Saphiru ili Ariu tada će sustavske varijable ostati podešene od prije.

Pod `UNIX` ili `Linux OS`-om potrebno je u datoteku `.cshrc` ili drugu default shell skriptu zapisati retke:

```
export SAPHIRA=/usr/local/Saphira          (bash shell)

setenv SAPHIRA /usr/local/Saphira          (csh shell)
```

Dinamički dijeljene datoteke su `Saphira/lib/libsf.so` i `Aria/lib/libAria.so`. Njih je potrebno učiniti dostupnima za sve napravljene programe u Saphiri ili Arii na jedan od sljedeća tri načina:

1. Eksportirajte datoteke sljedećom shell naredbom:

```
export LD_LIBRARY_PATH=${SAPHIRA}/lib:${ARIA}/lib.
```

2. Kopirajte te dvije datoteke u mapu `/usr/lib` ili `/usr/local/lib`.
3. Dodajte odgovarajuće staze tih dviju datoteka u vašu OS sustavsku datoteku `/etc/ld.so.config` ili pokrenite `/sbin/ldconfig` za što su vam potrebne root privilegije.

Da biste isprobali je li instalacija uspješno završena potrebno je prvo pokrenuti X-Windows pomoću naredbe `startx`. Nakon toga odite u mapu `usr/local/Saphira/bin` i pokrenite izvršnu datoteku `saphira`. Saphira klijent bi se trebao pojaviti s grafičkim i interakcijskim prozorom. U interakcijskom prozoru može se upisati `help` za listu naredaba koje se mogu koristiti.

Nakon toga potrebno je spojiti se sa simulatorom ili robotom. Za simulator je potrebno pokrenuti datoteku `usr/local/Saphira/bin/pioneer` na istom računalu te u interakcijskom prozoru upisati naredbu `connect local`, a za spajanje s robotom potrebno je logirati se na operacijski sustav koji je instaliran u njemu i tamo pokrenuti Saphiru te se spojiti na robota naredbom `connect serial` ili preko padajućeg izbornika.

2.2.3. Podešavanje sustavskih varijabli

U prethodnom odjeljku opisani su načini za podešavanje sustavskih varijabli. Ovisno o operacijskom sustavu potrebno je editirati `AUTOEXEC.BAT` (`Win 95/98/ME`) ili `Start/Settings/System` (`Win NT/2000/XP`) ili ubaciti u `shell` skriptu naredbe `setenv` ili `export` (`UNIX ili Linux`). U ovom odjeljku opisat ćemo samo koje je sustavske varijable moguće podesiti za rad sa Saphirom.

- `SAPHIRA` – označava radnu mapu Saphira distribucije. Mora biti postavljena da bi ispravno radili Saphira klijent i simulator. U `UNIX` ili `Linux OS`-u ne bi na kraju staze trebalo stajati završni znak *slash*, npr. `/usr/local/Saphira`.
- `SAPHIRA_LOAD` – označava početnu mapu za pokretanje koju koristi programski jezik Colbert. Ova mapa se potražuje kad se učitava datoteka `startup.act` pri pokretanju Colberta. Ako nije podešena tada se postavlja uvijek na stazu iz koje se pokreće Saphira klijent.
- `SAPHIRA_COMSERIAL` – serijski port za spajanje s robotom. Unaprijed podešena vrijednost je `COM1` (`Windows`) ili `/dev/ttyS0` (`Linux`).
- `SAPHIRA_SERIALBAUD` – *baud rate* za serijski spoj. Unaprijed podešena vrijednost je 9600.
- `SAPHIRA_COMPORT` – lokalni TCP/IP port za spajanje na simulator robota. Može biti postavljen tako da se više kopija istog simulatora može pokretati na istom računalu, a klijenti se mogu spajati na njih. U tom slučaju potrebno je za ovu sustavsku varijablu koristiti brojeve veće od 8101. Ova varijabla utječe i na klijent i na simulator. Unaprijed podešena vrijednost ovisi o OS-u.
- `SAPHIRA_COMSERVER` – ime računala ili IP adresa TCP/IP spoja. Unaprijed podešena vrijednost je NULL.

2.2.4. Instalacija dodatnih modula

Potrebno je instalirati tri dodatna modula: *LaserMapping*, *LaserNavigation* i *Gradient* modul. Sva tri modula postoje u verzijama za Windows operacijski sustav i za Linux Red Hat.

LaserMapping isto kao i Saphira dolazi kao već gotova aplikacija. Postoje verzije za Windows operacijski sustav i za Linux Red Hat. Instalacija je jednostavna jer je potrebno samo raspakirati instalacijske pakete u istu mapu u kojoj se nalazi i mapa Saphira, npr. `/usr/local` (`Linux`) ili `C:\` (`Windows`) i to je to.

Mapa *LaserMapping* sadrži tri programa: jedan za kreiranje *log* datoteke (*sickLogger*), drugi za kreiranje karte (*ScanStudio*) i treći za uređivanje karte (*MapperProEditor*). Također se u toj mapi nalaze i test karte koje se mogu učitati, pogledati ili uređivati.

Saphira sadrži ekspeditivnu verziju Markovljeve Monte Carlo lokalizacije kojom se robot lokalizira u gobalnoj karti, a bazirana je na laserskom senzoru ili sonarima. Ona je sadržana u modulu *LaserNavigation*.

Za uspješno gibanje bazirano na lokalnim preprekama i kartama prostora, Saphira sadrži Gradijentni modul za planiranje trajektorije u stvarnom vremenu koji se zasniva na gradijentnoj metodi. Za planiranje staze i gibanje po karti gradijentni se postupak koristi zajedno s Markovljevom lokalizacijom da bi robot mogao znati gdje se na karti nalazi dok se giba.

Ova dva modula instaliraju se pod Windows OS-om preko *user friendly* instalacijskog sučelja, a pod Linux OS-om potrebno je raspakirati instalacijske pakete u istu mapu u kojoj je instalirana i Saphira, npr. `/usr/local`.

Na kraju je samo potrebno napomenuti da verzije svih ovih modula moraju biti identične kao i instalirani paketi Saphire i Arie jer u protivnom neće raditi.

2.3. Pregled Saphire

Sustav Arie/Saphire možemo promatrati kao sustav s dvije arhitekture: sustavskom i upravljačkom. Sustavska arhitektura je u potpunosti implementirana u Arii. Nju sačinjava cijeli skup rutina za komuniciranje i upravljanje robotom preko Host računala. Dizajnirana je tako da jednostavno definira aplikacije robota povezivanjem klijentovih programa. Zbog toga je sustavska arhitektura *Open Source* tipa. Korisnici koji žele napisati vlastite aplikacije za upravljanje robotom, a ne žele se zamarati složenošću upravljanja i komuniciranja s Hardwareom, mogu iskoristiti prednosti korištenja *procesa* i *stanja* sustavske arhitekture za samopodizanje aplikacija.

Na vrhu sustavskih rutina nalazi se upravljačka arhitektura robota koja predstavlja dio za upravljanje robotom koji rješava mnoge probleme navigacije, od najniže razine kontroliranja motora i senzora do procesa visoke razine kao što su planiranje staze ili prepoznavanje prepreka. Saphira i Aria dijele zaduženja upravljačke arhitekture, a sama Aria omogućava izvršavanje osnovnih elemenata gibanja i interpretacije senzora. Doprinos Saphire upravljačkoj arhitekturi sadrži čitav skup rutina za procesiranje senzorskih podataka, izgradnju modela okoline i upravljanje akcijama robota. Upravljačka arhitektura je dovoljno fleksibilna da bi korisnici mogli birati koje će od različitih metoda izabrati za postizanje određenog cilja. Ona je također *Open Source* tipa tako da korisnici mogu ubaciti vlastite metode za mnoge rutine koje su unaprijed definirane.

2.3.1. Arhitektura

Robot koji se koristi na zavodu na 9. katu FER-a je Pioneer 2 DX (P2 DX) i u njega je ugrađen Siemens C166 mikrokontroler koji upravlja njegovim operacijskim sustavom (Pioneer 2 Operating System). On također prikuplja informacije sa enkodera i sonara, održava brzinu svakog kotača posebno, služi za pozicioniranje... Sve to radi na osnovi uputa koje mu dolaze od Saphire. Drugim riječima robot (Server) je samo sirova snaga, a Saphira (klijent) je njegov mozak.

Saphiru možemo pokretati pod različitim operacijskim sustavima (Windows, Unix, Linux RedHat). Ako želimo razvijati svoje aplikacije za rad s robotom (npr. voziti robota po pravokutnoj ili kružnoj trajektoriji) onda ćemo to učiniti pomoću C ili C++ programskog jezika (kao što su npr. Visual C++ ili K Developer) koji će tada preko Saphire komunicirati s robotom, a to znači da uvijek moramo u *kod* prvo uključiti pokretanje Saphire. Nakon što nam naša C++ aplikacija pokrene Saphiru tada se jednostavno priključimo na robot tako da u padajućem izborniku Saphire odaberemo opciju "File -> Connect -> Serial Port -> Com A". Robot u sebi ima instaliran Linux Red Hat operacijski sustav, a s Host računalom (preko kojeg je upravljan) spojen je bežičnim ethernetom. Ako se želimo spojiti na simulator robota onda to činimo pomoću "File -> Connect -> Local".

2.3.1.1. Sustavska arhitektura

Sustavsku arhitekturu možemo zamisliti kao osnovni operacijski sustav za upravljanje robotom. Sve rutine Saphire i Arie su ustvari procesi koji se pozivaju svakih 100 ms s operacijskim sustavom ugrađenim u Ariu. Ti procesi podržavaju komunikaciju paketima s robotom, izgrađuju unutarnju sliku stanja robota i obavljaju složenije zadatke kao što su navigacija i interpretacija senzora.

Procesi se sastoje od stanja koja se registriraju operacijskim sustavom. Svakih 100 ms operacijski sustav prolazi kroz sva registrirana stanja i obavlja jedan korak u svakome od njih. Sve to se obavlja sinhrono jer su koraci fiksnog intervala. Nije potrebno brinuti se za promjenu vrijednosti stanja dok se procesi izvršavaju jer stanja ovise o čitavom sustavu koji se osvježava i mora biti stabilan prije nego se stanja pozovu. Procesi također mogu iskoristiti prednosti fiksnog vremenskog ciklusa da bi osigurali precizna vremenska kašnjenja koja su često korisna za upravljanje robotom. Ovakvo upravljanje robotom pomoću procesa ima i svojih ograničenja, a to je da se svak zadatak procesa mora obaviti unutar tih 100 ms. No, to danas i ne predstavlja neki problem jer su brzine procesora koji obavljaju te zadatke dovoljno velike.

Iako se različiti procesi mogu pisati u programskom jeziku C++, ipak je bolje pisati akcije u jeziku *Colbert*. On sadrži bogat skup programa *user-friendly* sintakse, što omogućava puno lakše pisanje aplikacija za upravljanje robotom. Akcije su specijalni tip procesa koje se obavljaju unutar istih 100 ms kao i ostali procesi. Akcije prevodi sam *Colbert* pa ih korisnik može pregledavati, ulaziti u njih i mijenjati ih bez napuštanja aplikacije. Na taj se način programeri mogu koncentrirati na usavršavanje algoritama.

Aria podržava i protokole komunikacije paketima za slanje naredaba serveru robota i primanje informacija s robota. Tipično klijent pošalje od jedne do četiri naredbe u sekundi iako server podržava i do 100 naredaba u sekundi što ovisi o brzini serijske komunikacije i prosječnoj veličini paketa. Svi klijenti već u sljedećoj sekundi primaju 10 ili više paketa informacija s robota. Ti paketi sadrže očitanja senzora i među ostalim informacije enkodera.

2.3.1.2. Upravljačka arhitektura

Mobilni roboti operiraju u geometrijskom prostoru pa je reprezentacija tog prostora kritična za njihov nastup. U Saphiri postoje dvije geometrijske reprezentacije. Lokalni percepcijski prostor (LPS) predstavlja egocentrični koordinatni sustav radijusa od nekoliko metara centriranog oko robota. Za veći pregled Saphira koristi globalnu kartu prostora (GMS) da bi predstavila objekte koji su dio okoline robota u globalnim (apsolutnim) koordinatama. LPS je koristan za praćenje gibanja robota kroz kratke intervale gibanja objedinjavajući očitanja senzora i registrirajući prepreke koje je potrebno zaobići. LPS daje robotu osjećaj lokalnog okruženja. Glavni prozor u Saphiri prikazuje kako robot u LPS-u ostaje centriran na ekranu gibajući se prema naprijed, a okolina se okreće oko njega pri njegovu gibanju. Održavanje robota na fiksnom mjestu omogućava lagano opisivanje strategija za izbjegavanje prepreka gibajući se prema ciljnom položaju.

Za složenije upravljanje gibanjem robota, Aria osigurava mogućnost implementacije *Behaviorsa* kao skupa upravljačkih pravila. *Behaviorsi* imaju razinu prioriteta i razinu aktivnosti, kao i druge, dobro definirane, varijable stanja koje posreduju u interakciji s ostalim *Behaviorsima* i rutinama koje ih pozivaju. Npr. rutina može provjeravati je li određeni *Behaviors* postigao određeni cilj ispitivanjem određenih varijabli stanja *Behaviorsa*.
Verzija 8.x uključuje nekoliko bitnih izmjena u upravljanju *Behaviorsima*. Aria implementira općenitu arhitekturu *Behaviorsa* u kojoj su oni C++ objekti. Interakcija među *Behaviorsima* implementirana je klasom *resolver*. Oni se mogu uključiti ili isključiti tako da im se pošalju signali iz interakcijskog prozora u Saphiri ili iz Colbert prozora.

Da bi mogli raditi sa složenim akcijama traženja cilja, Saphira omogućuje metodu u kojoj se akcije robota raspoređuju korištenjem upravljačkog jezika *Colbert*. S njim se mogu izrađivati datoteke akcija koje ih obavljaju određenim redoslijedom da bi se dobio odziv robota u njegovoj okolini. Npr. tipična akcija bi mogla gibati robota kroz hodnik dok izbjegava predmete i provjerava gdje se nalaze prepreke.

Rutine interpretacije senzora su procesi koji izlučuju podatke sa senzora ili lokalnog percepcijskog prostora (LPS) i vraćaju ih u LPS. Saphira aktivira različite procese ovisno o zadatku koji se obavlja. Detekcija prepreka i rekonstrukcija karte prostora su samo neke od rutina koje trenutno postoje. Sve rade s podacima dobivenim sa sonara, lasera ili enkodera.

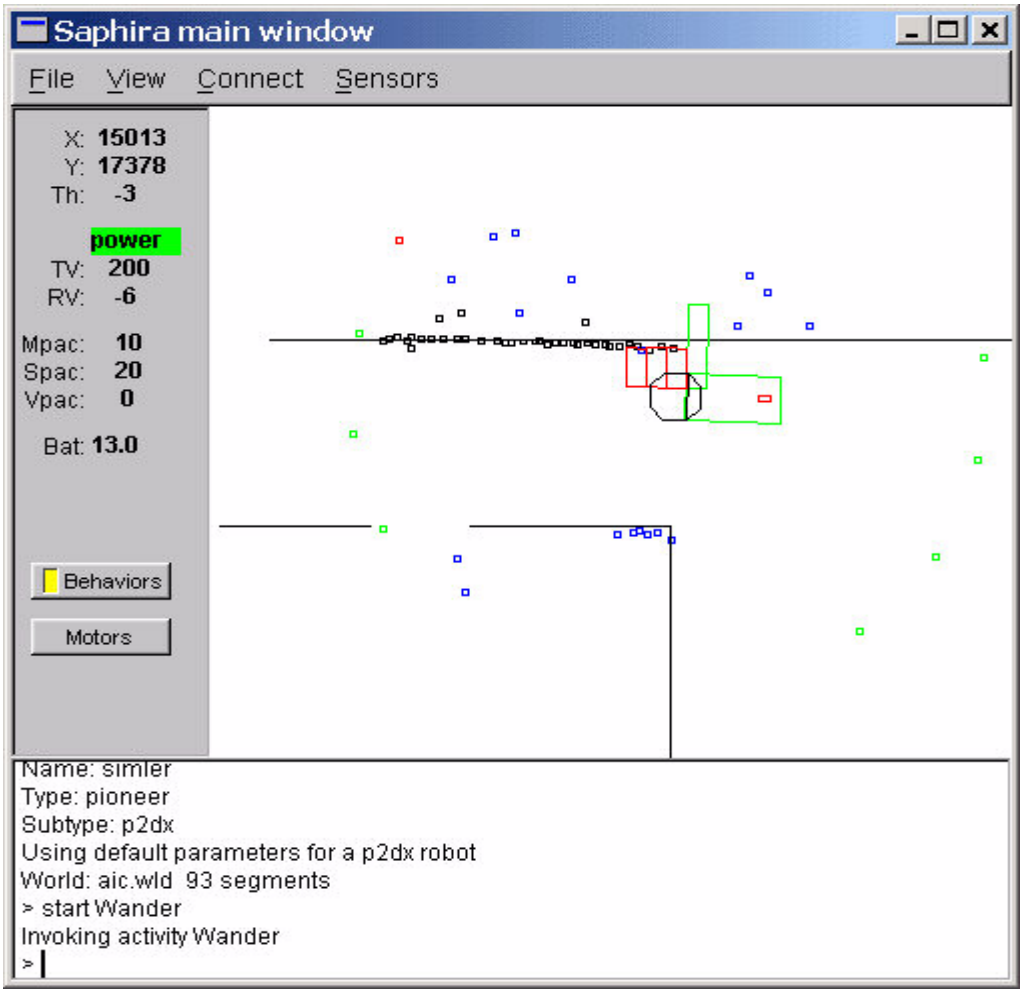
U globalnoj karti prostora, Saphira održava skup internih struktura podataka (artifakata) koji predstavljaju radnu okolinu robota. Artifakti uključuju hodnike, vrata, zidove i sobe. Ove karte se mogu uključiti datotekom u kojoj je sadržana karta prostora ili gibanjem robota u okolini i puštanjem Saphiri samoj da izluči bitne informacije o prostoru.

2.3.3. Pokretanje klijenta

Ovaj odjeljak prikazuje neke mogućnosti Saphire kroz primjer klijenta. Također je ilustrirano grafičko sučelje za interakciju s klijentima.

Da bi se pokrenula neka aplikacija potrebno je pokrenuti izvršnu datoteku "saphira(.exe)" u mapi "Saphira/bin/". Jedino što ova datoteka zahtijeva za svoje pokretanje su ključne datoteke "sf.dll/aria.dll" ili "libsf.so/sibAria.so". Njih smo instalirali još pri instalaciji Saphire kad smo postavljali varijable okoline.

Klijent Saphire će inicijalizirati prozor za prikaz prikazujući LPS kao što je prikazano na *slici 1.1.* Robot se nalazi u centru prikaza, koji prikazuje informacije sonara koje su vraćene u program klijenta. Informacijski prozor nalazi se na lijevoj strani prozora, padajući izbornici na vrhu, a interaktivni tekstualni prozor u donjem dijelu.



Slika 2.1. Prikaz lokalnog percepcijskog prostora u klijentu Saphire

Mali kvadrati na slici prikazuju očitanja sonara: zeleni prikazuju trenutne vrijednosti sonara, plavi i crni prikazuju prošle vrijednosti. Mali crveni kvadrat odmah ispred robota prikazuje kutni položaj prema kojem se robot giba. Veći zeleni kvadrati označuju područja osjetljivosti za Behaviorse koji upravljaju robotom. Linije predstavljaju artifakte iz učitane karte. Informacije o položaju robota, brzini i unutarnjim stanjima prikazane su na lijevom dijelu prozora.

Klijent Saphire inicijalno učitava samo minimalan skup procesa. Mogućnosti klijenta se povećavaju korištenjem Colbert naredaba i datoteka. Primjer jedne akcije napisane u Colbertu je "Saphira/colbert/demo.act" koja se koristi kao primjer u daljnjem dijelu teksta. Kada se Saphira klijent pokreće on pretražuje datoteku "Saphira/colbert/sysStartup.act". Ta inicijalizacijska datoteka učitava sustav grafičkog prikaza i tada demonstrira datoteku *demo.act*. Ove datoteke se ne bi smjele mijenjati osim ako ne želite prikaz grafičkog sučelja i demo programa. Inicijalizacija sustava može se premostiti na način da se postavi datoteka istog imena u mapu iz koje se pokreće Saphira pa se ta datoteka učitava umjesto one iz mape "Saphira/colbert".

Datoteka *demo.act* definira akciju pretraživanja, zatim je poziva kao i nekoliko predefiniranih Behaviorsa za izbjegavanje prepreka.

Nakon učitavanja inicijalizacijske datoteke Saphira traži korisničku inicijalizacijsku datoteku *myStartup.act*. Ako je ta datoteka pronađena u lokalnoj mapi ili u mapi danoj pomoću varijable okoline SAPHIRA_LOAD, učitati će se u sustav. U Windowsi OS-u korisnici mogu Saphiru pokretati iz lokalne mape postavljajući u nju Shortcut do *saphira.exe* izvršne datoteke.

2.3.4. Objašnjenje funkcija glavnog prozora Saphire

Kao što smo već ranije napomenuli spajanje Saphire na robot ili simulator je slično. Prvo, kad se koristi simulator, potrebno je uvjeriti se da su u simulator učitani ispravni parametri robota. Inače Saphira sama prepoznaje tip servera robota i učitava njegove parametre pri njegovu prvom spajanju tako da nije potrebno učitavati datoteku parametara u Saphiru osim ako ne koristite neku konfiguraciju parametara koju ste sami podesili.

Ako se je na robot potrebno spojiti preko serijskog porta tada se u interaktivnom prozoru može upisati naredba `connect serial` da bi se spojili na standardni serijski port. Ako je vaš radio modem spojen na neki drugi serijski port, koristite naredbu `connect serial <port>`, gdje `<port>` označava ime serijskog porta, npr. "`/dev/ttyS1`" ili "`COM 2`".

Za spajanje na simulator potrebno je prvo pokrenuti simulator koji potom osluškuje TCP/IP port. Zatim se u interaktivnom prozoru može upisati naredba `connect` koja otvara lokalni port do simulatora i pokreće sve što je potrebno.

Ako postoji problem u spajanju na simulator ili server robota tada se komunikacijska veza neće uspostaviti i prikazati će se poruka s opisom problema u Saphirinom informacijskom prozoru. Tipični uzroci pogrešaka pri spajanju na simulator i njihova rješenja su:

- Potrebno je pobrinuti se da je pokrenut simulator i niti jedan drugi Saphira klijent ili server robota na istom stroju.
- U rijetkim slučajevima komunikacijski kanal može biti blokiran. To se može dogoditi ako se server ili simulator prethodno nepravilno isključe. U tom slučaju potrebno je obrisati *Pipe* (cijevnu) datoteku i pokušati ponovno. Ako ni to ne pomogne tada je jedini lijek resetiranje računala.

- Potrebno je uvjeriti se da su komunikacijski kabeli dobro spojeni.
- Radio modem klijenta treba se nalaziti u radnom području frekvencija i raditi na ispravnom kanalu te mora imati dovoljno jak signal.
- Treba provjeriti je li možda neka druga aplikacija koristi serijski port.

Kada se jednom Saphira spoji na server ili simulator prikazat će informacije o stanju robota i omogućiti upravljanje robotom pomoću naredaba ili tipaka na tipkovnici.

2.3.4.1. Lokalni prikaz prostora (LPS)

Prikaz lokalnog percepcijskog prostora zapravo je pogled na robota i njegovu okolinu iz ptičje perspektive. LPS se može prebacivati između roboto-centričnog prikaza i globalnih koordinata koristeći "Display->Local" padajući izbornik.

Dijelovi glavnog prozora Saphire prikazuju:

- Ikonu robota koja se nalazi u centru prozora i prikazuje robota relativno u odnosu prema svojoj okolini. Ako je namješten lokalni prikaz tada se LPS prikazuje u roboto-centričnim koordinatama što znači da robot ostaje u centru ekrana, a okolina se giba oko njega. U globalnom prikazu (GMS) okolina je fiksirana, a ikona robota luta po ekranu. Veličina ikone robota kontrolira se vrijednostima *RobotRadius* i *RobotDiagonal* koje se nalaze u datoteci parametara robota.
- Akumulirana očitanja sonara javljaju se na ekranu kao mali prazni kvadrati plave i crne boje. Trenutna očitanja sonara prikazuju se zelenom bojom. Broj akumuliranih očitanja sonara može biti postavljen od strane korisnika.
- Upravljačka točka koja se javlja u obliku malog praznog kvadrata crvene boje odmah ispred robota predstavlja upravljačku točku smjera prema kojoj se robot giba. Tu točku postavlja server u roboto-centričnim koordinatama. Kontroler robota podešava svoj smjer prema toj točki pokušavajući stalno održavati njezin smjer.
- Područja osjetljivosti prepreka nacrtana su pomoću nekoliko Behaviorsa za izbjegavanje prepreka. Ti kvadrati mijenjaju boju kada se uoči neka prepreka.

Robotom se može upravljati pomoću tipkovnice klijenta. Tipke prikazane *tablicom 1.3.* rade samo kada je aktivan glavni prozor Saphire što znači da ga je potrebno prvo aktivirati s klikom lijeve tipke miša u grafičkom prozoru.

Key	**Action**
i, ↑	Povećanje brzine prema naprijed
m, ↓	Smanjenje brzine prema naprijed
j, ←	Inkrementalno povećanje zakreta ulijevo
l, →	Inkrementalno povećanje zakreta udesno
k, space	Sve se zaustavlja

Tablica 2.3. Naredbe tipkovnice Saphira klijenta

2.3.4.2. Informacijski prozor

Informacijski prostor nalazi se na lijevoj strani ekrana. On sadrži podatke vraćene od servera robota.

Status (St) prikazuje status servera robota kao *no con*, *power* ili *no servo* kada su zaglavljeni motori.

Velocity (Tr, Rot) označava translacijsku (Tr) brzinu robota u [mm/s] i rotacijsku (Rot) brzinu u [°/s].

Position (X, Y, Th) označava apsolutni položaj u [mm/°]. Bitno je uočiti da to nije brzina dobivena od servera u kojoj su akumulirane pogreške, nego globalni položaj robota baziran na karti u Saphiri u kombinaciji s integracijom položaja vraćenog sa servera.

Communication (MPac, SPac, VPac) su komunikacijske vrijednosti koje prikazuju broj paketa danog tipa primljenog u posljednjoj sekundi. One su korisne za provjeru komunikacijske veze sa serverom. U normalnoj situaciji klijent prima 10 paketa s motora (MPac) te oko 25 paketa podataka sa sonara (SPac) u sekundi. Paketi vizije (VPac) trenutno nisu podržani.

Battery prikazuje razinu napunjenosti baterije robota (Bat).

Behaviors tipka je upaljena ako su u Saphiri omogućene akcije Behaviorsa. One su nadjačane s direktnim akcijama, npr. tipkama palice za igru ili tipkovnice. U tom slučaju akcije Behaviorsa mogu se opet uključiti pomoću ove tipke.

Motors tipka se koristi da bi se omogućio rad motora na fizičkom robotu (motori na simulatoru su uvijek uključeni bez obzira na ovu tipku).

2.3.4.3. Tekstualni (interaktivni) prostor

Interaktivni tekstualni prostor nalazi se na dnu prozora. U njemu Saphira ispisuje informacije o sustavu, a također i korisnici mogu upisivati naredbe za prevodioca Colberta.

U interaktivnom prostoru mogu se obavljati sljedeći zadaci:

- učitati datoteke s akcijama i promijeniti radnu mapu,
- spojiti se ili odspojiti sa servera ili simulatora,
- definirati, pokrenuti ili zaustaviti akcije,
- pratiti stanja akcija,
- potražiti pomoć o API-u i o funkcijama Colberta,
- pregledati i postaviti interne varijable Saphire.

Prevodioc omogućuje korisnicima da pišu i ispravljaju programe za vrijeme odvijanja aplikacija u Saphiri. Obično će se korisnički kod nalaziti u tekstualnoj datoteci koja se u sustavu učitava naredbom `load`. Ta datoteka može sadržavati mješavinu akcija, definicija i poziva datoteka i funkcija. Korisnik može akcije pozivati iz interaktivnog prozora naredbom `start`. Tijekom izvršavanja korisnik može pregledavati stanja varijabli Saphire te pokrenuti ili zaustaviti druge akcije. Ako dođe do pogreške, tada će se akcija u kojoj se pogreška dogodila suspendirati i ispisat će se poruka o pogrešci. Korisnik tada može promijeniti

tekstualnu datoteku s kodom, ponovno je učitati i pokrenuti promijenjene akcije. Nema potrebe za izlaskom iz aplikacije i ponovnim prevođenjem koda. Čak i nove C++ funkcije mogu se dinamički povezati u sustav tako da se učitaju u dinamičku . *dll* datoteku.

Najvažnije Colbert naredbe opisane su u *dodatku 1*.

2.3.4.4. Padajući izbornici

Glavni prozor klijenta sadrži nekoliko padajućih izbornika. Oni omogućuju upravljanje prikazom informacija u lokalnom percepcijskom prostoru (LPS), upravljaju komunikacijom sa serverom i učitavaju te zatvaraju datoteke parametara i karte prostora.

- **Izbornik Connect:**

Ovaj izbornik omogućuje otvaranje i zatvaranje veze sa simulatorom ili serverom robota. Sadrži tri opcije: standardni serijski port, lokalni port i TCP vezu. Ako se odabere bilo koja od ove tri, klijent će se spojiti s robotom ili simulatorom. Parametri kao što su *baud rate* ili imena portova mogu se zadati u interaktivnom Colbert prozoru ili preko *library* datoteka.

Opcija *disconnect* zatvara otvorenu vezu prema robotu ili simulatoru.

- **Izbornik File:**

Izbornik *File* sadrži naredbe za učitavanje raznih tipova datoteka i za izlazak iz Saphire.

Load World File otvara dijalog za učitavanje *wld* datoteke u Saphiru. Također se može koristiti i Colbert naredba *loadworld*.

Load Activity File otvara dijalog za učitavanje datoteke akcija Colberta u Saphiru. Također se može koristiti i Colbert naredba *load*.

Load Library File otvara dijalog za učitavanje prevedene *library* datoteke (*dll* ili *shared object file*) u Saphiru. Također se može koristiti i Colbert naredba *loadlib*.

Exit uzrokuje da se program klijenta završi s prethodnim zatvaranjem svih otvorenih veza.

- **Izbornik View:**

Ovaj izbornik upravlja raznim aspektima prikaza.

Ako se klikne na opciju *Grow* ili *Shrink* tada će se prikaz lokalnog percepcijskog prostora u Saphiri povećati (približiti) odnosno smanjiti (udaljiti).

Pomoću opcije *Rate* može se upravljati korakom osvježavanja prikaza. Na nekim računalima, veliki brojevi za korak osvježavanja troše dosta procesorskog vremena. Smanjenjem ovog broja postižu se bolje performanse rada.

Ako je uključena opcija *Robot Onscreen* tada je prikaz robota na ekranu uvijek vidljiv.

Opcija *Robocentric* mijenja prikaz tako da se sve gleda iz perspektive robota koji je centriran na ekranu i usmjeren prema gore.

Global mode prikazuje gibanje robota u globalnom koordinatnom sustavu.

Robot Visible i *Artifacts Visible* uključuju ili isključuju u prikazu crtanje robota ili artifakata (zidova, prepreka i sl.).

Activity Window, ako je selektiran, prikazat će prozor akcija za pregled stanja akcija Colberta ili Bahaviorsa.

- **Izbornik Sensors:**

Padajući izbornik *Sensors* sadrži ulazne podatke za senzore (sonare i laser). Ako se odabere neka od opcija, prikazat će se dijalog za unos parametara za odgovarajući senzor. Pomoću tih dijaloga mogu se kontrolirati veličina i prikaz prošlih stanja senzora tako da druge rutine mogu dobiti bolji pregled okoline, nego u slučaju da se koristi jedno mjerenje.

Postoje dva tipa spremnika za prošla mjerenja senzora. Trenutni spremnik (*current buffer*) sadrži prošlih N mjerenja senzora i ona se zamjenjuju novima kako ova pristižu. po principu FILO (*first in – last out*). Npr. u jednom tipičnom primjeru spremnik sonara će spremati posljednjih 30 očitanja sonara. Pošto se ta očitanja dinamički zamjenjuju, objekti koji dolaze u vidokrug i zatim iz njega izlaze, neće ostaviti trajni utjecaj na podatke u spremniku. Trenutna očitanja sonara u LPS prozoru označena su plavom bojom.

Akumulacijski spremnik (*accumulation buffer*) dulje čuva ona očitanja koja imaju najmanju mjernu nesigurnost. Kada dolaze nova očitanja, stara se brišu – tipični predmeti u spremniku ostaju dulje vrijeme. Očitanja akumulacijskog spremnika pojavljuju se u crnoj boji (za sonare) u LPS prozoru.

Klizne trake u dijalogu spremnika sonara služe za promjenu broja mjesta u spremniku. *Clear/Reset* tipka briše sva trenutna mjesta u spremniku. Prikaz spremnika može se omogućiti ili onemogućiti u istom dijalogu pomoću opcije *Display*.

3. SNIMANJE KARTE ZAVODA ZA APR

3.1. Uvod

Za početak je potrebno imati instaliranu Saphiru, Saphira-laser i Saphira-navigation. Sve troje mora biti instalirano u istoj verziji. Saphira dolazi kao gotovi (već prevedeni) proizvod i zajedno sa simulatorom Pioneer nalazi se u mapi $SAPHIRA/bin, pri čemu je $SAPHIRA makro naredba koja označava mapu u kojoj je instalirana Saphira.

LaserMapping isto kao i Saphira dolazi kao već gotova izvršna aplikacija. Postoje verzije za Windows operacijski sustav kao i za Linux Red Hat.

Pod Windowsima mapa *LaserMapping* sadrži tri programa: jedan za pravljenje *log* datoteke (*sickLogger*), jedan za kreiranje karte (*ScanStudio*) i jedan za uređivanje karte (*MapperProEditor*). Također se u toj mapi nalaze i test karte koje se mogu učitati, pogledati ili uređivati.

U verziji istog programa za Linux u mapi *LaserMapping* ne nalazi se jedino program *sickLogger* pa ga je trebalo napraviti uz pomoć već gotovog koda. Nije bilo moguće koristiti gotovi program za Windows OS jer je na robotu instaliran Linux OS.

Da bi se mogao napraviti program *sickLogger* potrebno je imati instaliranu Ariu. Datoteka *sickLogger.cpp* koju je potrebno prevesti da bi se dobio program *sickLogger* nalazi se u mapi $ARIA/examples.

SickLogger je program koji je potrebno pozvati s jednim argumentom, a to je staza do datoteke u koju se upisuju očitanja senzora robota što će biti kasnije detaljnije objašnjeno. Ako se argument ne navede tada će se za zapisivanje koristiti datoteka *1scans.2d* koja će biti pohranjena u mapu iz koje se pokreće program*sickLogger* .

Staze do instaliranih programa na robotu su:

/usr/local/Saphira

/usr/local/Aria

/usr/local/*LaserMapping*

Demonstracijski koraci procesa mapiranja bili bi sljedeći:

1. napraviti program *sickLogger* koji će očitane koordinate spremati u datoteku
2. podesiti sve što je potrebno da bi proradila palica za igru za vožnju robota
3. napraviti *log* datoteku tako da provozamo robota programom*sickLogger*
4. pretvoriti *log* datoteku u kartu u *ScanStudiu*
5. preurediti kartu u *MapperProEditoru* ako je potrebno
6. iskoristiti mapu da bi se robot mogao lokalizirati i upravljati Saphirom

3.1.1. Uspoređivanje karte (Map Matching)

Map Matching, poznato još i kao pozicioniranje na osnovi karte (*Map-based positioning*) je tehnika u kojoj robot koristi senzore za stvaranje karte okoline u kojoj se nalazi. Nakon toga se ta lokalna karta uspoređuje s globalnom koja je prethodno pohranjena u memoriji. Ako se pronađe poklapanje na dvjema kartama tada robot može izračunati svoj stvarni položaj u okolini.

Glavne prednostipozicioniranja na osnovi karte jesu:

- Može biti iskorišteno da bi se napravilo osvježavanje karte okoline. To je bitno jer se ta karta može iskoristiti za globalno planiranje putanje robota kao i za izbjegavanje prepreka.
- Omogućuje robotu učenje nove okoline i kroz to učenje poboljšava točnost pozicioniranja.

Nedostaci pozicioniranja na osnovi karte vezane su uz specifične zahtjeve navigacije. Npr. pozicioniranje zahtjeva sljedeće:

- da ima dovoljno stacionarnih objekata koji se ne mijenjaju i lako se mogu međusobno razlikovati tako da se mogu iskoristiti za uspoređivanje.
- da karta okoline bude dovoljno točna da bi se mogla iskoristiti (ovisno o zadatku).
- da se na raspolaganju imama moćno računalo za prikupljanje i obradu informacija senzora.

Treba napomenuti da je glavnina radova u pozicioniranju na osnovi karte okoline ograničena na upotrebe u labaratorijima i u jednostavnim okolinama.

Uspoređivanje karte je ostvarivanje povezanosti između lokalne i unaprijed pohranjene globalne karte okoline. Rad na principima vezanim uz Map Matching obično se fokusira na opći problem, a to je usporedba slike svoje vlastite pozicije i orijentacije relativno u odnosu na model, ne bi li se dobilo preklapanje. Takvi principi usporedbe mogu se podijeliti na dva pristupa:

1. *Iconic-based* estimacija pozicije uspoređuje točke očitanja senzora, s oblicima na karti, zasnovano na minimalnoj udaljenosti. Pozicija robota određuje se na način da se minimizira pogreška udaljenosti između očitanja senzora i odgovarajućih točaka na karti na koje se ta očitanja odnose. Tada se na osnovi nove pozicije robota, veze između očitanja senzora i karte preračunavaju i proces se ponavlja dok razlika u nagomilanoj pogrešci udaljenosti između točaka očitanja senzora i linija na karti ne padne ispod određenog praga. Ovaj algoritam uspoređuje svako očitanje senzora prema karti po čemu se i razlikuje od sljedećeg pristupa.
2. *Feature-based* estimacija pozicije na osnovi informacija sa senzora stvara značajke koje uspoređuje s pohranjenom kartom. Ovaj estimator pozicije brži je od prethodnog i ne zahtijeva da robot bude početno dobro postavljen, ali mu je zbog toga potrebno više točaka očitanja za usporedbu pa mu je stoga smanjena točnost.

Jedan od problema koji se javlja u pozicioniranju na osnovi karte leži u tome da očitanja senzora i model okoline mogu biti u različitim formatima. Tipično rješenje ovog problema jest da se aproksimacija položaja bazirana na odometriji iskorištava da bi se iz pohranjenog globalnog modela okoline generirala scena koju će vidjeti robot. Tada se ta scena uspoređuje

sa stvarnom koja se dobije iz informacija sa senzora. Jednom kada se poronađu preklapanja tih dviju slika (očekivane i stvarne), položaj robota estimira se s reduciranom nesigurnošću.

Da bi se trenutna očitanja senzora mogla pouzdano podudarati s pohranjenim globalnim modelom potrebno je omogućiti nekoliko algoritama provjere istovremeno. Realističan model za odometriju i njezinu nesigurnost jest osnova za pravilno funkcioniranje pzicioniranja na osnovi karte jer se detekcija oblika kao i osvježavanje položaja zasnivaju na proračunu estimacije iz odometrije.

3.2. Program SickLogger

Program *sickLogger* nalazi se u računalu robota i služi za kreiranje tekstualne *log* datoteke s ekstenzijom *.2d*. Ta datoteka će kasnije biti korištena za izgradnju karte prostora. *SickLogger* kao argument prihvaća stazu datoteke u koju će spremati podatke dobivene laserom. Ako, prilikom pozivanja programa ne zadamo stazu *log* datoteke, tada će se podaci zapisati u datoteku *1scans.2d* u mapi u kojoj se nalazi program *sickLogger*. Format datoteke *1scans.2d* dan je u *dodatku 4*.

Program se automatski spaja s robotom i pokreće laser. U trenutku kad se pokreće laser, *sickLogger* onemogućuje sonare i prema tome možemo zaključiti kad je robot spreman za pokretanje pomoću palice za igru (kad sonari prestanu raditi prestaje se čuti njihov zvučni signal). Ako se sickLogger ne može spojiti na robot zbog bilo koje pogreške, Aria će to javiti i program će se isključiti.

Pošto se u instaliranoj verziji *Saphira 8.x* ne nalazi gotova izvršna aplikacija *sickLogger*, bilo ju je potrebno napraviti tako da se prevede datoteka *sickLogger.cpp* koja se može pronaći u Arii.

Prijevod koda pod Linuxomprovodi se naredbom:

```
make –f Makefile
```

pri čemu je datoteka *Makefile* datoteka u kojoj se nalazi izvorni kod za prijevod programa. U našem slučaju prevodi se datoteka *sickLogger.cpp*, a to je kao jedan od argumenata navedeno u samoj datoteci *Makefile*. Izvorni kod datoteke *Makefile* dan je u *dodatku 2*, a datoteke *sickLogger.cpp* u *dodatku 3*.

Datoteke *sickLogger.cpp* i *Makefile* potrebno je staviti u istu mapu tako da bi se prijevod programa mogao lakše obaviti. Prijevod datoteke *sickLogger.cpp* obavlja se upravo pomoću gore navedene naredbe nakon čega se u istoj mapi stvore dvije datoteke: *sickLogger.o* i izvršna *sickLogger*. Kad je napravljen program prelazi se na sljedeći korak.

3.2.1. Prednosti korištenja lasera

Točnost skeniranja prostora pomoću lasera, u okviru jedne ravnine, je za oko 25 puta bolja od ostalih radijacijskih senzora: 180 očitanja u rasponu od 180° prema 1 očitanju svakih 8° kod sonara. Laserska zraka je jako fokusirana i teško se izobličuje ili apsorbira u reflektirajućem mediju, tako da dolazi do svega nekoliko loših očitanja, što čini točnost lasera još superiornijom u odnosu na sonare.

Laser može osjetiti objekte na puno većoj udaljenosti: 10-50 m dok su sonari ograničeni na 3-6 m. Pošto su sobe i komercijalni prostori obično duljina stranica od 8 m, robot zasnovan na sonarima lako se može naći na brisanom prostoru jer su objekti izvan njegova dometa.

Jedini nedostatak lasera (u odnosu na sonare) na robotu P2DX na zavodu jest što skenira u ravnini s nagibom manjim od 1°. Sonari operiraju u obliku stošca te tako imaju puno širi vidokrug na udaljenostima do 6 m. Stoga se korištenjem sonara može implementirati akutno izbjegavanje prepreka, a to je jedan od glavnih razloga da roboti budu opremljeni i s laserom i sa sonarima jer njihova kombinacija daje najbolje rezultate.

3.3. Podešavanje palice za igru (Joysticka)

Da bi se robot mogao upravljati pomoću palice za igru potrebna je jedna najobičnija PC palica za igranje igrica. Potrebno ju je samo priključiti na 15 pinski konektor na robotu. Nakon što je palica za igru priključena robot se može uključiti ako već nije bio uključen. Za provjeru rada palice za igru na robotu se pritisne bijela tipka MOTORS jedan put, nakon čega bi se trebao čuti ritmični nisko tonski zvučni signal robota koji nam ukazuje da je pokrenut mod palice za igru.

Palica za igru je samokalibrirajuća, tako da kad prvi put robota postavimo u mod palice za igru, njegov operacijski sustav (P2OS) detektira inicijalni položaj palice i pohranjuje te vrijednosti kao centrirani stop položaj. Prema tome da bi se robot ispravno pokretao potrebno je prethodno ručno kalibrirati palicu za igru i tada ju, prije pokretanja moda palice za igru, držati u srednjem položaju zbog ispravne samokalibracije.

Tipka za pucanje (*Fire Button*) na palici služi za pokretanje robota. Kad se pritisne, robot se počinje gibati; a kad se otpusti, robot se zaustavlja. Robot bi se trebao moći gibati: naprijed, natrag, lijevo i desno brzinom relativno pomaku palice.

Granične ili maksimalne translacijske i rotacijske brzine upravljanja palicom mogu se podesiti preko P2OS konfiguracijskih parametara JoyVelMax i JoyRVelMax. Unaprijed definirane vrijednosti su 1200 mm/s i 125 °/s što su za naš pokus prevelike vrijednosti te su postavljene na 400 mm/s i 75 °/s.

Za mijenjanje konfiguracijskih parametara potrebno je zapisati te promjene u FLASH ROM memoriju mikrokontrolera na robotu.

Da bi se parametri palice za igru mogli promijeniti i zapisati u FLASH memoriju prvo je potrebno omogućiti serijski spoj između računala i palice jer je to najpouzdaniji način komunikacije. Nakon toga odvijačem je potrebno pomaknuti FLASH prekidač na robotu prema prednjem dijelu robota čime je omogućeno pisanje u FLASH ROM. Nakon što je to obavljeno može se uključiti robot ili se resetirati ako je bio uključen. Nakon što je završio s inicijalizacijom potrebno ga je postaviti u BOOT mod tako da se drži pritisnuta tipka MOTORS, a zatim se pritisne i otpusti tipka RESET. Tipku MOTORS je još nakon toga potrebno držimati pritisnutu 3 ili 4 sekunde i potom je otpustiti (znači tipka MOTORS se uopće ne otpušta, nego se drži pritisnuta cijelo vrijeme). Nakon toga bi na LCD ekranu titrajuća zvjezdica, koja označuje robotovo "srce", trebala prestati titrati. Ako se robot resetirao tada se nije dovoljno dugo držala pritisnuta tipka MOTORS. U tom slučaju potrebno je pokušati opet.

Kad se Pioneer nalazi u BOOT modu potrebno je pokrenuti *p2oscf* program iz mape *p2os*. Da bi se on mogao pokrenuti potrebno je logirati se kao root korisnik. *P2oscf* prihvaća jedan opcijski parametar: ime porta za serijsku komunikaciju:

```
% p2oscf <com-port>   (npr.  % p2oscf /dev/ttyS0)
```

Naredba p2oscf povlači iz ROM-a robotovog mikrokontrolera trenutne operacijske parametre koje koristi robot. Njome se također može uređivati privremena kopija liste parametara. Konstante robota, za razliku od varijabli, nije moguće promijeniti.

Promijenjene vrijednosti ne zapisuju se u FLASH ROM dok se eksplicitno ne pohrane naredbom `save`. Čak i tada će *p2oscf* naredba parametre upisati u FLASH ROM samo tada

kada je neki od parametara promijenjen. Pisanje u FLASH ROM je zagarantirano samo unutar 100 ciklusa tako da je preporuka proizvođača da se parametri mijenjaju samo kad je nužno potrebno.

Da bi se pogledala lista promjenljivih parametara potrebno je upisati znak 'v' iza čega se pritisne tipka ENTER. Može se utipkati '?' da bi se ispisala lista naredaba koje je moguće koristiti. Da bi se vidjela samo trenutna vrijednost jednog parametra, potrebno je upisati samo njegov naziv, a za promjenu vrijednosti potrebno je utipkati njegov naziv i iza toga željenu vrijednost parametra. Ta vrijednost može biti znakovni niz (koji se tada upisuje bez navodnika) te decimalni ili heksadecimalni broj.

U našem slučaju potrebno je promijeniti dvije vrijednosti na sljedeći način:

```
>JoyVelMax 100          (unaprijed podešena vrijednost je 1200)

>JoyRVelMax 50          (unaprijed podešena vrijednost je 125)
```

Tako promijenjene parametre P2OS još uvijek ne koristi, sve dok nisu pohranjeni u FLASH ROM upravljačkom naredbom `save`.

Inače, većina parametara može se unaprijed definirati tako da se koriste umjesto onih iz FLASH ROM-a. To se radi pomoću određenih naredaba sa strane klijenta.

P2oscf naredbe su sljedeće:

Naredba	Opis
`keyword <value>`	Sama naredba `keyword` prikazuje trenutnu editiranu vrijednost. Ako želimo primijeniti njezinu vrijednost tada moramo iza naredbe dodati i željenu vrijednost.
`c` ili `constants`	Prikazuje P2OS vrijednosti konstanti. Ove korisnik ne može mijenjati.
`v` ili `variables`	Prikazuje trenutne, promijenjene vrijednosti varijabli (mogu biti drukčije od onih koje se nalaze u ROM-u).
`r` ili `restore <pathname>`	Vraća editirane *p2oscf* varijable na vrijednosti trenutno pohranjene u FLASH ROM-u ili u datoteci ako je uključen argument.
`save <pathname>`	Pohranjuje trenutno editirane vrijednosti u FLASH ROM i izlazi iz programa ili pohranjuje trenutno editirane vrijednosti na disk za kasniju uporabu i ostaje u editoru.
`q` ili `quit`	Izlazi iz *p2oscf* bez pohranjivanja promjena u FLASH.
`?` ili `help`	Prikazuje naredbe s njihovim opisom.

Tablica 3.1. P2oscf naredbe za uređivanje parametara u FLASH ROM-u

3.4. Kreiranje *log* datoteke

Robotom upravljamo sa host računala, a na njega se spajamo preko bežičnog etherneta (Access Pointa koji se nalazi u hodniku zavoda) naredbama:

```
xhost 161.53.68.21

ssh -X -l student 161.53.68.21
```

gdje je `161.53.68.21` IP adresa robota, a student korisničko ime na računalu unutar robota.

Da bismo dobili podatke za našu mapu prvo priključimo palicu za igru na 15 pinski port na robotu i postavimo robot na početnu točku koja će biti početna položaj (HOME position). S tog mjesta ćemo uvijek pokretati robota bilo za kreiranje karte, bilo za eksperimente lokalizacije s Ariom ili Saphirom

Nakon toga pokrenemo izvršnu datoteku *sickLogger* iz mape */home/student/metikos/* unutar robota. Taj program kao argument prihvaća stazu datoteke u koju će spremati podatke dobivene laserom. Za ekstenziju datoteke možemo staviti nastavak *.2d*. Ako, pri pozivanju programa *sickLogger* ne zadamo stazu *log* datoteke, tada će se podaci zapisati u datoteku *1scans.2d* u mapi u kojoj se nalazi program *sickLogger*. Format datoteke *1scans.2d* dan je u *dodatku 4*.

Program se automatski spaja s robotom i pokreće laser. U trenutku kad se pokreće laser, *sickLogger* onemogućuje sonare i prema tome možemo zaključiti kad je robot spreman za pokretanje pomoću palice za igru (kad sonari prestanu raditi prestaje se čuti njihov zvučni signal). Ako se *sickLogger* ne može spojiti na robot zbog bilo koje pogreške, Aria će to javiti i program će se isključiti.

Nakon toga ako već to nismo ranije obavili, potrebno je kalibrirati palicu za igru na sljedeći način: ostavimo držač u centru i pritisnemo tipku za pokretanje (robot se ne bi smio gibati); tada otpustimo tipku. Nakon toga potrebno je rotirati držač oko graničnih položaja dva ili tri puta držeći ga u svakom kutu sekundu ili dvije. Nakon toga s smo premni za kretanje.

U osnovi, potrebno je gibati se iza robota cijelim putem njegove vožnje. Program *sickLogger* očitane vrijednosti lasera pohranjuje u tekstualnu datoteku svakih nekoliko stupnjeva i na zadanoj udaljenosti što je definirano u datoteci *sickLogger.cpp* (nju smo koristili za izradu izvršne aplikacije *sickLogger*).

3.4.1. Strategija gibanja

Prije nego smo pokrenuli robot dobro je imati razvijenu strategiju gibanja robota koja će ovisiti o tome kako izgleda okolina. Strategije razlikujemo prema tome imaju li petlje ili ne.

Ako naša okolina ne sadržava nikakve petlje (npr. uredska okolina s jednim hodnikom i povezanim sobama, tada je strategija gibanja robota jednostavna. Samo robota treba pomoću palice za igru provesti kroz sobe širom hodnika. Međutim ako je robot opremljen laserom koji ima kut gledanja od 180° tada algoritam mapiranja još uvijek može detektirati petlje zbog različitih kuteva gledanja kada se giba u nasuprotnim smjerovima kroz sobe ili hodnik. Zbog toga je potrebno prijeći na algoritam gibanja koji uključuje okoline s petljama.

Ako okolina sadrži petlje, kao što je to slučaj u većini okolina, tada algoritam uključuje nekoliko pretpostavki i ograničenja.

3.4.1.1. Petlje se zatvaraju samo jednom

Jedna pretpostavka je da se nakon zatvaranja petlje, rezultirajuća karta ne mora više mijenjati tijekom petlje i pozicija svih očitanja lasera koja pripadaju petlji ostaje fiksna. To znači da ako se nastavljamo ponovno gibati po petlji više ništa se ne mijenja.

3.4.1.2. Petlje trebaju biti zatvorene kad jesu zatvorene

Postoji ograničenje koje nam govori da ako se petlja nije uspjela zatvoriti prvi put, tada se vjerojatno neće uspjeti zatvoriti niti sljedeći put kad robot stigne na isto mjesto. To znači da ako vozimo robota kroz petlju natrag na već posjećeni dio morali bismo biti sigurni da robot prepoznaje tu petlju. U tom slučaju treba nastaviti vožnju robota u već posjećenom području dok on sam ne detektira petlju. Tek nakon što je detektirao i zatvorio petlju, može se nastaviti gibati u novom, neistraženom području.

3.4.1.3. Prvo male petlje

Obično je bolje mapirati okolinu zatvarajući male petlje prije gibanja prema većim petljama jer to općenito smanjuje prostor pretraživanja za detekciju petlji. Međutim to ovisi o okolini jer u nekim slučajevima gdje su male petlje unutar jedne velike petlje, može biti bolje da se prvo zatvori velika petlja jer se može narušiti neka od sljedećih stavki.

3.4.1.4. Nema petlje nekoliko metara prije sljedeće petlje

Nakon zatvaranja petlje ili nakon gibanja u već mapiranom prostoru robot bi trebao ići nekoliko metara u neistraženo područje prije nego zatvori sljedeću petlju. Razlog tome je da nakon zatvaranja petlje ili gibanja u već mapiranom prostoru, resetira se prostor pretraživanja detekcije petlje i sva očitanja koja pripadaju prethodnoj petlji se fiksiraju tako da ne mogu više promijeniti svoj položaj.

3.4.1.5. Nema petlji tijekom istraživanja novog područja

Ovo je slično prethodnoj točki. Nakon što je petlja zatvorena, prostor pretraživanja detekcije petlje se resetira i položaj očitanja lasera se fiksira. To može biti negativnost u slučaju kada postoji nekoliko malih petlji unutar jedne velike.

Gornji naputci trebali bi dati nekakvi osjećaj za odluku o tome kako napraviti strategiju za gibanje u bilo kakvom prostoru. Ako postoji napravljeni tlocrt okoline tada on može dosta pomoći u razvijanju strategije gibanja koja uključuje gornje kriterije.

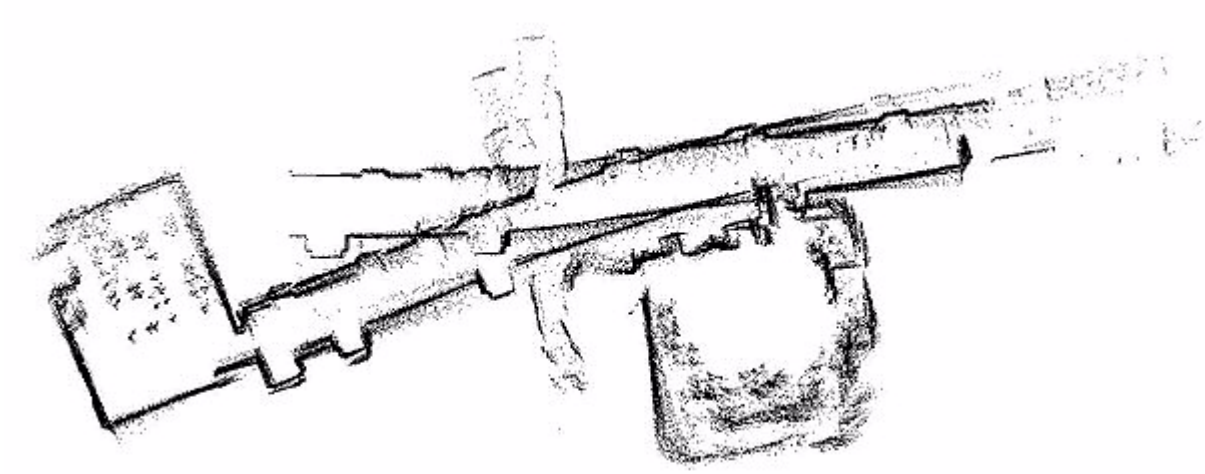
Drugi pristup zasnivao bi se na principu pokušaja i pogreške tako da se iz iskustva za određenu okolinu napravi novi plan gibanja.

U skladu s navedenim, robot je provezen par puta po zavodu te su dobivene pripadajuće *log* datoteke koje je bilo potrebno obraditi u programu *ScanStudio*Format *log* datoteke dan je u *dodatku 4*.

3.5. Pretvaranje *log* datoteke u kartu

ScanStudio je program koji služi za pretvaranje tekstualne *log* datoteke (u koju su spremljeni podaci očitanja s lasera) u kartu prostora. Opis programa *ScanStudio* dan je u *dodatku 5*.

Nakon što se *ScanStudio* pokrene, sa "File" izbornika odabere se opcija "Load raw log file" nakon čega se otvori pretraživač pomoću kojeg na disku pronađemo jednu od prethodno stvorenih *log* datoteka. Nakon što datoteku učitamo možemo odmah vidjeti kako se očitanja lasera ne poklapaju jedna s drugim. Razlog tomu su odometrijske i druge pogreške. Rezultat izvođenja prethodne naredbe može se vidjeti na *slici 2.1*.

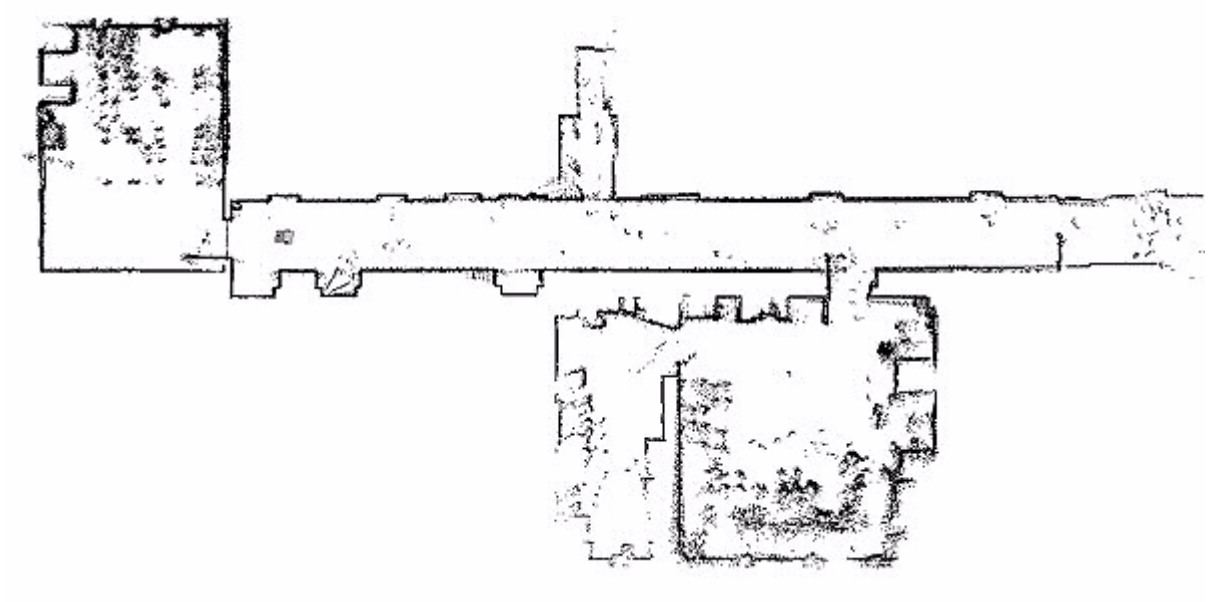


Slika 3.1. Odometrijske pogreške robota uslijed gibanja pri mapiranju

Međutim tako učitanu *log* datoteku ne možemo pretvoriti u kartu, nego je moramo učitati na drugi način.

Idemo pnovno na izbornik "File->Input log file". Opet se otvara pretraživač pomoću kojeg opet učitamo istu *log* datoteku. Što se dalje događa ovisi o opcijama i parametrima podešenim u samom programu *ScanStudio*. Program će ili automatski ili pritiskanjem tipke '.' (ako je podešen interaktivni način rada) prolaziti kroz očitanja lasera u *log* datoteci. Za to vrijeme će se na ekranu stvarati karta prostora, korak po korak.

Svako očitanje ispravlja se tako što *ScanStudio* provjerava poklapanje slika i rješava se odometrijskih pogrešaka. Nakon što je mapiranje gotovo, možemo kartu pohraniti opcijom "File->Save corrected scans as log file" u novu *log* datoteku za kasnije procesiranje. Rezultat mapiranja prikazan je na *slici 2.2*.

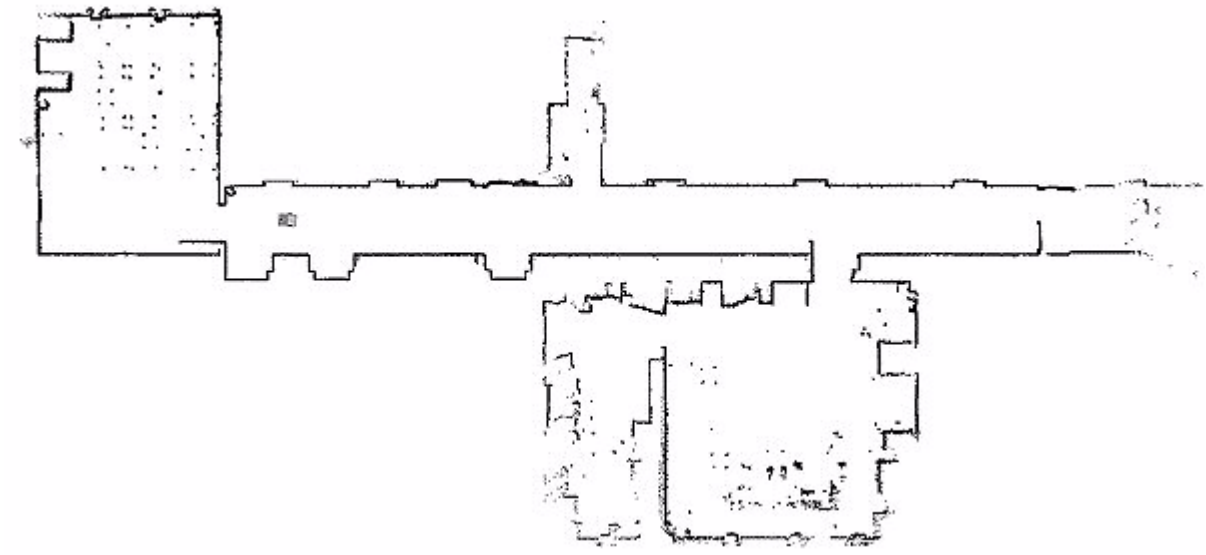


Slika 3.2. Karta prostora nakon što su ispravljene odometrijske pogreške

Sljedeće je da dobivenu kartu registriramo na način da se očitanja bolje podudaraju pomoću opcije "Mapping->Register all". Ovaj proces zahtjeva dosta vremena, dvostruko dulje nego što je trajalo početno procesiranje. Da bi znali kad je postupak gotov, možemo pratiti opterećenje procesora jer ne postoji drugi način označavanja završetka postupka registriranja.

Korak iza ovog je da se pročiste očitanja dodatno pomoću opcije "Mapping->Clean scans". Ovaj proces uklanja s karte sve predmete koji su se za vrijeme mapiranja gibali. Prije početka potrebno je upisati vrijednost za "Grid resolution". Unaprijed definirana vrijednost je 50 mm što je dobra vrijednost za početak, no mogu se isprobati i druge vrijednosti (prijedlog je da budu između 50 mm i 100 mm). Rezultat pročišćavanja očitanja može se otkazati opcijom "Mapping->Unclean scans".

Rezultat prethodna dva koraka prikazan je na *slici 2.3*.



Slika 3.3. Karta prostora nakon pročišćavanja i uklanjanja gibajućih predmeta

Nakon toga karta se može pohraniti opcijom "File->Save map" za daljnje uređivanje u programu *MapperProEditor* ili korištenje u Arii ili Saphiri. U datoteci s ekstenzijom MAP zapisane su koordinate svake registrirane točke, a njezin format dan je u *dodatku 6*.

Karta se još dodatno može obraditi u programu *ScanStudio*opcijom "Advanced ->Line model from scans" koja po određenom algoritmu dodaje ravne linije na kartu i nakon toga se može pohraniti kao *world* datoteka koju će koristiti simulator robota Pioneer, naredbom "Advanced->Save line scans as world file". Ta WLD datoteka po svojoj svrsi je drukčija od karte koja se pohranjuje kao MAP datoteka jer se učitava kao prostor (world) u simulator robota te služi za simuliranje okoline. U WLD datoteci zapisane su koordinate početnih i završnih točaka svih linija koje čine prostor robota. Njezin format može se pogledati u *dodatku 7*.

Radi boljeg prikaza detalja slika u ovom odjeljku, one su samo djelomično prikazane. Konačan izgled karte zavoda obrađene pomoću *MaperProEditora* dan je u *dodatku 13*.

3.6. Preuređivanje karte

Iako je mapa u ovom obliku već spremna za korištenje, moguće je izbaciti neke pokretne dijelove (ljude koji su stajali i gledali eksperiment). Da bi se to učinilo potrebno je otvoriti program *MapperProEditor* i u njega, pomoću opcije "File->Open" učitati kartu.

Dotični program može se vidjeti na sljedećoj *slici 2.4*.

Slika 3.4. Izgled prozora programa MapperProEditor

Potom se mogu iskoristiti alati s lijeve strane da bi se dijelovi karte obrisali ili preuredili. Može se također dodati dio zida koji se zaboravilo mapirati ili neki dio koji je dodan u prostor nakon što je mapiranje već bilo napravljeno.

Na kraju je uređenu kartu potrebno pohraniti opcijom "File->Save as"

4. MARKOVLJEVA LOKALIZACIJA (ML)

Lokalizacija je estimacija položaja robota iz informacija dobivenih sa senzora.. Ključna ideja Markovljeve lokalizacije jest da se održi vjerojatnost gustoće preko cijelog prostora svih mogućih položaja robota u njegovoj okolini. Ovaj pristup predstavlja to područje metrički koristeći fino raspodijeljenu mrežu koja aproksimira gustoće, a također uključuje i tehniku filtriranja koja mobilnom robotu omogućuje pouzdanu estimaciju položaja čak i u gusto naseljenim područjima u kojima gomile ljudi blokiraju senzore robota kroz dulje vrijeme. Pomoću ove metode robot se može globalno lokalizirati kad se izgubi u prostoru.

4.1. Opis ML principa

Cilj lokalizacije jest estimacija položaja robota u njegovoj okolini. Pri tome robot mora imati na raspolaganju kartu okoline i informacije sa senzora. Tehnike lokalizacije razvijene do sada mogu se podijeliti u dvije grupe.

1. Lokalne tehnike (*tracking*) teže tome da kompenziraju odometrijske pogreške koje se javljajutijekom navigacije robota. Među utim, one zahtijevaju da početni položaj robota bude približno poznat i one se tipično ne mogu oporaviti ako izgube korak s položajem robota unutar određenih granica.

2. Globalne tehnike napravljene su tako da mogu estimirati položaj robota čak i prilikom globalne nesigurnosti. Tehnike ovog tipa rješavaju tzv. *wake-up-robot* problem na taj način da se robot može lokalizirati bez unaprijed poznatog položaja. Tehnike globalne lokalizacije su moćnije od lokalnih jer se mogu nositi sa situacijama u kojima se robot nalazi pod utjecajem većih pogrešaka položaja.

Ovdje ćemo predstaviti ovu drugu metodu koja globalno estimira položaj robota u okolini. Markovljeva lokalzacija koristi vjerojatnosnu okosnicu da bi održala gustoću vjerojatnosti položaja preko čitavog područja mogućih položaja robota. U uobičajenom slučaju u kojem je robot siguran za svoj položaj, metoda se sastoji od unimodalne distribucije centrirane oko stvarnog položaja robota. Na osnovi vjerojatnosne prirode pristupa Markovljeva lokalizacija može globalno estimirati položaj robota te se može suočavati s kritičnim situacijama u kojima se robot može ponovno lokalizirati ako se izgubio. Ove stavke su osnovni preduvjet za potpuno autonomnog robota dizajniranog za rad kroz dulje vremensko razdoblje.

Ova metoda koristi finu metričku diskretizaciju prostora stanja tako da ovaj pristup ima nekoliko prednosti u odnosu na ostale koji su koristili Gausovu raspodijelu s grubom diskretizacijom ili pak topografskom reprezentacijom za aproksimaciju najvjerojatnijeg položaja robota. Prvo, pruža točniju estimaciju položaja koja je potrebna za mnoge zadatke robota, i drugo, ujedinjuje podatke jednog senzorskog ulaza kao što je npr. jedna zraka ultrazvučnog senzora. Većina prijašnjih pristupa Markovljevoj lokalizaciji je pomoću prikaza podataka sa senzora označavala prisutnost ili odsutnost objekata karte te su se raspali ako se okolina nije dobro podudarala (poravnavala) s njihovimpretpostavkama.

Možda je najbitnije za napomenuti da su prijašnje izvedbe Markovljeve lokalizacije pretpostavljale da je okolina statička. Zbog toga se nisu uspješno lokalizirale u dinamičkim okolinama. Da bi se nosila s takvim situacijama, ova metoda primjenjuje tehniku filtriranja koja osvježava gustoću vjerojatnosti položaja koristeći samo ona mjerenja koja su poznati objekti sadržani na karti proizveli s velikom sličnošću.

4.1.1. ML za statičke okoline

Ograničenje pretpostavke da je okolina statička učinjeno je zbog jasnoće prikaza. Ta pretpostavka naziva se još i Markovljevom pretpostavkom. Njezin postulat je da je lokacija robota jedino stanje u okolini koje sustavno utječe na očitanja senzora. Ona se narušava ako roboti dijele istu okolinu s ljudima

4.1.1.1. Uvod u ML priračun

Označimo položaj mobilnog robota trodimenzionalnom varijablom $l = (x, y, Th)$ koja se sastoji od x–y kartezijskog koordinatnog sustava i smjera direkcije Th. Sa l_t ćemo označiti stvarni položaj robota u trenutku t, a s L_t odgovarajuću slučajnu varijablu.

Logično je da robot ne zna svoj stvarni položaj. Umjesto toga, on sa sobom nosi uvjerenje o tome gdje bi se mogao nalaziti. Neka nam $Bel(L_t)$ označava uvjerenje o položaju robota u trenutku t. $Bel(L_t)$ je distribucija vjerojatnosti kroz čitavi prostor mogućih položaja. Npr. $Bel(L_t)$ je gustoća vjerojatnosti da se robotu pridružuje mogućnost da je njegova lokacija u trenutku t jednaka l. Uvjerenje se osvježava putem odgovora dvaju različitih tipova događaja: izmjerenim informacijama preko senzora okoline (npr. kamera, sonari, laser) i odometrijskim očitanjem (npr. broj okretaja kotača). Mjerenja dobivena senzorima okoline označimo sa s, odometrijska mjerenja sa a, a odgovarajuće slučajne varijable sa S i sa A.

Robot prima protok informacija o mjerenjima senzora s i očitanjima odometrije a, pa označimo sa:

$$d = \{d_0, d_1, ..., d_T\} \qquad (4\text{-}1)$$

protok informacija mjerenja, gdje svaki d_t (gdje je $0 \leq t \leq T$) označuje ili mjerenje osjetila okoline ili odometrijsko mjerenje. Varijabla t označava indeks podatka (a ne vremena), a varijabla T je indeks najsvježije prikupljenog podatka. Skup d čine svi raspoloživi podaci senzora i njega ćemo u daljnjem tekstu nazivati riječju *podaci*.

4.1.1.2. Rekurzivna lokalizacija

Markovljeva lokalizacija estimira posteriori distribuciju preko L_T na svim raspoloživim podacima koju ćemo označiti sa:

$$P(L_T = l \mid d) = P(L_T = l \mid d_0, d_1, ..., d_T) \qquad (4\text{-}2)$$

Prije izvoda ovih inkrementalnih posteriori jednadžbi osvježavanja, izrazimo jasno ključnu prepostavku koja je osnova za izvod, a naziva se *Markovljeva pretpostavka*. Ona kaže da ako je poznata lokacija robota l_t, buduća mjerenja su neovisna o prošlima i obratno:

$$P(d_{t+1}, d_{t+2}, ... \mid L_t = l, d_0, ..., d_t) = P(d_{t+1}, d_{t+2}, ... \mid L_t = l) \ \forall\, t \qquad (4\text{-}3)$$

Drugim riječima pretpostavimo da je lokacija robota jedino stanje u okolini, i da je poznavanje njega jedino što je potrebno da bi se mogli predvidjeti budući podaci. Ta pretpostavka je prilično netočna za okoline koje sadrže i druge gibajuće objekte osim samog robota.

Prilikom izračunavanja $P(L_T = l \mid d)$, razlikujemo dva slučaja ovisno o tome je li najnoviji podatak d_T očitanje senzora okoline ili odometrijsko mjerenje.

1. Ako je najnoviji podatak očitanje senzora okoline $d_T = s_T$ onda slijedi:

$$P(L_T = l \mid d) = P(L_T = l \mid d_0, ..., d_{T-1}, s_T) \quad (4\text{-}4)$$

Bayesovo pravilo predlaže da ovaj izraz transformiramo u izraz:

$$\frac{P(s_T \mid d_0, ..., d_{T-1}, L_T = l) \cdot P(L_T = l \mid d_0, ..., d_{T-1})}{P(s_T \mid d_0, ..., d_{T-1})} \quad (4\text{-}5)$$

koji se zbog Markovljeve pretpostavke može pojednostavniti u:

$$\frac{P(s_T \mid L_T = l) \cdot P(L_T = l \mid d_0, ..., d_{T-1})}{P(s_T \mid d_0, ..., d_{T-1})} \quad (4\text{-}6)$$

Također možemo uočiti da se nazivnik može zamijeniti konstantom $_T$, jer ne ovisi o L_T, tako da imamo:

$$P(L_T = l \mid d) = \alpha_T \cdot P(s_T \mid L_T = l) \cdot P(L_T = l \mid d_0, ..., d_{T-1}) \quad (4\text{-}7)$$

Može se uočiti inkrementalna priroda prethodne *jednadžbe (3-7)* pa možemo napisati:

$$Bel(L_T = l) = P(L_T = l \mid d_0, ..., d_T) \quad (4\text{-}8)$$

što znači da uvjerenje robota prema *jednadžbi (3-7)* postaje:

$$Bel(L_T = l) = \alpha_T \cdot P(s_T \mid l) \cdot Bel(L_{T-1} = l) \quad (4\text{-}9)$$

U ovoj jednadžbi zamijenili smo izraz $P(s_T \mid L_T = l)$ s izrazom $P(s_T \mid l)$ koji se zasniva na pretpostavci da je nezavisan o vremenu.

2. Ako je najnoviji podatak očitanje odometrije $d_T = a_T$ onda slijedi

Ovdje računamo $P(L_T = l \mid d)$ koristeći teorem totalnog probabiliteta:

$$P(L_T = l \mid d) = \int P(L_T = l \mid d, L_{T-1} = l') \cdot P(L_{T-1} = l' \mid d) \cdot dl' \quad (4\text{-}10)$$

Razmotrimo prethodni izraz s desne strane znaka jednakosti. Markovljeva pretpostavka predlaže da učinimo sljedeće:

$$P(L_T = l \mid d, L_{T-1} = l') = P(L_T = l \mid d_0, ..., d_{T-1}, \alpha_T, L_{T-1} = l') \quad (4\text{-}11)$$

$$= P(L_T = l \mid \alpha_T, L_{T-1} = l') \quad (4\text{-}12)$$

Drugi dio izraza s desne strane u *jednadžbi (3-10)* može se isto tako pojednostavniti razmatrajući da $_T$ ne nosi sa sobom nikakve informacije o položaju L_{T-1}:

$$P(L_{T-1} = l' \mid d) = P(L_{T-1} = l' \mid d_0, ..., d_{T-1}, \alpha_T) \quad (4\text{-}13)$$

$$= P(L_{T-1} = l' \mid d_0, ..., d_{T-1}) \quad (4\text{-}14)$$

Ako ubacimo *jednadžbe (3-12) i (3-14)* natrag u *jednadžbu (3-10)* dobit ćemo očekivani rezultat:

$$P(L_T = l \mid d) = \int P(L_T = l \mid \alpha_T, L_{T-1} = l') \cdot P(L_{T-1} = l' \mid d_0, ..., d_{T-1}) \cdot dl' \qquad (4\text{-}15)$$

Primjetite da je *jednadžba (3-15)* također inkrementalnog oblika. Ako upotrijebimo gornji izraz za uvjerenje robota tada možemo napisati:

$$Bel(L_T = l) = \int P(l \mid \alpha_T, l') \cdot Bel(L_{T-1} = l') \cdot dl' \qquad (4\text{-}16)$$

Primjetite da smo koristili $P(l \mid \alpha_T, l')$ umjesto $P(L_T = l \mid {}_T, L_{T-1} = l')$ jer smo iskoristili pretpostavku da se ne mijenja s vremenom.

4.1.1.3. ML algoritam

Jednadžbe (3-9) i *(3-16)* koriste se za osvježavanje i one formiraju okosnicu algoritma Markovljeve lokalizacije. Cijeli algoritam prikazan je u *tablici 3.1.* Sa $P(l \mid a, l')$ označit ćemo model gibanja robota jer on opisuje kako gibanje robota utječe na njegov položaj. Uvjetna vjerojatnost $P(s \mid l)$ naziva se još i percepcijskim modelom jer modelira rezultate senzora robota.

U algoritmu Markovljeve lokalizacije izraz $P(L_0 = l)$, koji inicijalizira uvjerenje robota $Bel(L_0)$, odražava prethodno znanje o početnom položaju robota. Ova raspodjela može biti inicijalizirana svojevoljno, ali u praksi prevladavaju dva slučaja:

1. Ako je položaj robota, relativno u odnosu prema karti, potpuno nepoznat, $P(L_0)$ je obično ravnomjerno raspodijeljen.

2. Ako je početni položaj robota približno poznat, tada je $P(L_0)$ obično Gausova raspodjela centrirana oko položaja robota.

4.1.1.4. ML implementacija

Može seprimjetiti da princip Markovljeve lokalizacije ostavlja otvorenim:

1. na koji način se predstavlja uvjerenje robota *Bel(L)* i

2. kako se izračunavaju uvjetne vjerojatnosti $P(l \mid a, l')$ *i* $P(s \mid l)$.

Prema tome postojeći pristupi Markovljevoj lokalizaciji uglavnom se razlikuju u predstavljanju prostora stanja i izračunavanju percepcijskog modela. U ovom odjeljku ćemo se uglavnom koncentrirati na različite implementacije Markovljeve lokalizacije fokusirajući se na ove dvije teme.

1. Predstavljanje u prostoru stanja je vrlo uobičajen način predstavljanja uvjerenja robota *Bel(L)* zasnovano na Kalmanovom filtru koji je opet baziran na ograničenoj pretpostavci da se položaj robota može modelirati pomoću unimodalne Gausove distribucije. Postojeće implementacije su se pokazale robusnima i točnim u praćenju položaja robota. Zbog te pretpostavke ovim tehnikama nedostaje sposobnost da prikažu situacije u kojima se položaj robota održava preko više posebnih uvjerenja. Kao rezultat toga, lokalizacijski pristupi koji koriste Kalmanov filtar obično zahtijevaju da početni položaj robota bude poznat i ne mogu se ponovno lokalizirati u slučaju da se robot izgubi. Dodatno, Kalmanov filtar se zasniva na modelima senzora koji generiraju estimate s Gausovom nesigurnošću. Ta pretpostavka ne vrijedi za sve situacije.

Da bi se zaobišla ova ograničenja koriste se razni pristupi za prikaz nesigurnosti položaja robota. U jednom pristupu koristi se recimo Markovljeva lokalizacija za

navigaciju u okolini zasnovanu na orijentirima s karte, a prostor stanja organiziran je prema gruboj topološkoj strukturi okoline s četiri moguće orijentacije robota. Ovakvi pristupi, u principu, rješavaju problem globalne lokalizacije, ali zbog grube rezolucije prikaza prostora stanja točnost estimiranog položaja je ograničena. Pomoću topoloških pristupa može se tek nazrijeti položaj robota, a k tome još i zahtijevaju da okolina bude ortogonalna te da postoje određeni orijentiri ili apstraktna obličja koja se mogu izlučiti iz informacija očitanja senzora. Ove pretpostavke otežavaju korištenje topografskih pristupa u nestrukturiranim okolinama.

for each location *l* **do** // inicijalizira uvjerenje

$$Bel(L_0 = l) \longleftarrow\!\!\!- P(L_0 = l) \qquad (4\text{-}17)$$

end for

forever do

if new sensory input s_T is recieved **do**

$$\alpha_T \longleftarrow\!\!\!- 0 \qquad (4\text{-}18)$$

for each location *l* **do** // primjenjuje percepcijski model

$$Bel(L_T = l) \longleftarrow\!\!\!- P(s_T \mid l) \cdot Bel(L_{T-1} = l) \qquad (4\text{-}19)$$

$$\alpha_T \longleftarrow\!\!\!- \alpha_T + Bel(L_T = l) \qquad (4\text{-}20)$$

end for

for each location *l* **do**

$$Bel(L_T = l) \longleftarrow\!\!\!- \alpha_T^{-1} \cdot Bel(L_T = l) \qquad (4\text{-}21)$$

end for

end if

if an odometry reading $_T$ is recieved **do**

for each locatio *l* **do**

$$Bel(L_T = l) \longleftarrow\!\!\!- \int p(l \mid l', \alpha_T) \cdot Bel(L_{T-1} = l') \cdot dl' \qquad (4\text{-}22)$$

end for

end if

end forever

Tablica 4.1. Algoritam Markovljeve lokalizacije

2. Percepcijski modeli senzora su se razvijali za različite tipove senzora kao dodatak raznim reprezentacijama prostora stanja. Ti modeli se razlikuju u načinu na koji računaju vjerojatnost trenutnog mjerenja. Za razliku od topoloških pristupa koji prvo izlučuju obličja iz informacija senzora ovi modeli operiraju nad tim mjerenjima senzora. Tehnike za aproksimaciju podataka senzora uglavnom se razlikuju u efikasnosti i načinu na koji modeliraju karakteristike senzora i karte okoline.

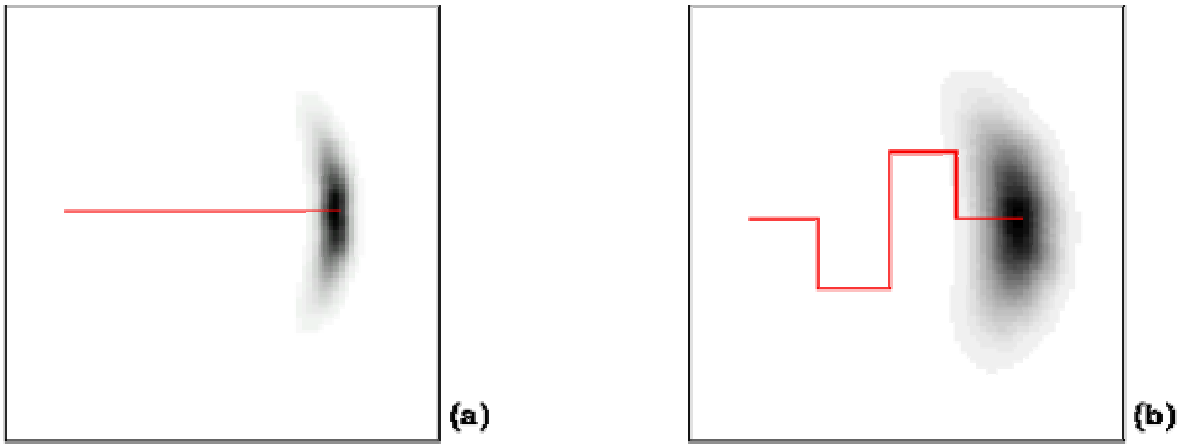
Da bi se mogle iskoristiti jake strane prethodnih metoda, naš pristup zasnovat ćemo na fino podijeljenom i manje ograničavajućem prikazu prostora stanja. Ovdje se uvjerenje robota aproksimira fino uzorkovanom i ravnomjerno raspoređenom mrežom, gdje je prostorna rezolucija između 10 i 40 *cm*, a kutna rezolucija između 2° i 5°. Prednost ovakvog pristupa u usporedbi s tehnikama zasnovanim na Kalmanovom filtru jest mogućnost da se prikaže više-modalna distribucija koja je preduvjet za globalnu lokalizaciju. Nasuprot topološkim pristupima Markovljevoj lokalizaciji, naš pristup omogućuje točnu estimaciju položaja u puno širem opsegu okolina, uključujući i one koje nemaju obličja koja bi se mogla identificirati. Pošto ne ovise o apstraktnim obličjima, one mogu objediniti podatke sa senzora u uvjerenje robota i obično daju rezultate koji su točniji za cijeli red veličine. Očito je da je kod ovakvog prikaza prostora stanja pomoću mreže usko grlo veličina samog prostora stanja koja se mora održavati. U odjeljku 3.1.2.4. bit će objašnjene tehnike pomoću kojih se jako velike mreže mogu osvježavati u stvarnom vremenu.

4.1.2. ML za dinamičke okoline

U ovom odjeljku opisana je metrički promjenljiva Markovljeva lokalizacija koja uključuje odgovarajuće modele senzora i gibanja. Također ćemo prikazati tehniku filtriranja koja je dizajnirana da zaobiđe pretpostavku modela statičkog prostora i omogući lokalizaciju mobilnog robota čak i u ljudima gusto naseljenim okolinama. Zatim ćemo opisati još i kako izgleda naš prostor stanja opisan fino podijeljenom mrežom, i na kraju tehnike koje efikasno osvježavaju velike prostore stanja.

4.1.2.1. Model gibanja

Da bi se uvjerenje robota pri njegovu gibanju osvježavalo, moramo specificirati model gibanja $P(l \mid l', a_t)$. Zasnovano na pretpostavci normalno raspoređenih pogrešaka translacije i rotacije, koristit ćemo miješano dvije nezavisne, centrirane u nulu, Gausove razdiobe čiji repovi su odrezani. Varijance tih razdioba proporcionalne su duljini mjerenog gibanja.



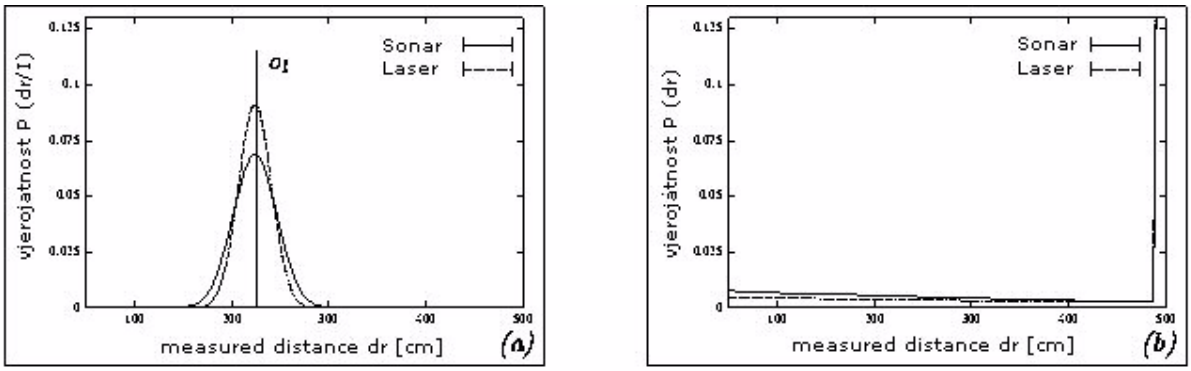
Slika 4.1. Distribucije (u obliku banane) različitih gibanja robota

Slika 3.1 prikazuje rezultirajuće gustoće za dva primjera staze u slučaju da uvjerenje robota započinje dirakovom razdiobom. Obje raspodjele su trodimenzionalne *(x, y, Th)*, a *slika 3.1* prikazuje njihove 2D projekcije u *x, y* prostor.

4.1.2.2. Percepcijski model za blizinske senzore

Vjerojatnost $P(s \mid l)$ da je očitanje senzora s izmjereno na položaju l treba biti proračunata za svaki položaj l u svakom koraku osvježavanja algoritma Markovljeve lokalizacije. Zbog toga je od ključne važnosti za on-line estimaciju položaja da se kvantitet može efikasno izračunati. Moravec je 1988 godine predstavio metodu za općenito izračunavanje ne Gausove funkcije gustoće vjerojatnosti $P(s \mid l)$ preko diskretnog skupa svih mogućih udaljenosti izmjerenih ultrazvučnim senzorom na lokaciji l. U prvoj implementaciji ovog pristupa pokušalo se koristiti sličnu metodu, ali je na kraju ispalo da je preskupa za lokalizaciju u stvarnom vremenu.

Da bi se zaobišla ova nepogodnost, razvijen je model senzora koji omogućuje izračunavanje $P(s \mid l)$ bazirano samo na udaljenosti o_l najbližoj prepreci na karti duž smjera usmjerenja senzora. Ova udaljenost može se izračunati s ponovnim precrtavanjem na mreži karte ili pomoću CAD modela okoline. Mi ćemo razmotriti diskretizaciju $d_1,..., d_n$ mogućih udaljenosti izmjerenih aproksimacijskim senzorom. Veličina koraka $_d = d_{i+1} - d_i$ u našoj diskretizaciji jednaka je za svaki i, a d_n odgovara maksimalnom koraku aproksimacijskog senzora. Tipične vrijednosti za n su između 64 i 256, a za maksimalni korak d_n između 500 i 1000 cm. Označimo sa $P(d_i \mid l)$ vjerojatnost mjerenja udaljenosti d_i pri lokaciji robota l. Da bismo mogli izračunati ovu vjerojatnost razmotrit ćemo sljedeća dva slučaja:



Slika 4.2. Vjerojatnost mjerenja d_i ako je prepreka a) na udaljenosti o_l , ili b) nepoznata

a) Poznate prepreke: Ako senzor registrira prepreku, rezultirajuća distribucija modelira se Gausovom razdiobom sa srednjom vrijednosti jednakom udaljenosti do te prepreke. Neka nam $P_m(d \mid l)$ označava vjerojatnost mjerenja udaljenosti d pri lokaciji robota l, uz pretpostavku da se zraka senzora odbija od najbliže prepreke na karti. Označimo udaljenost do te prepreke sa o_l. U tom slučaju, vjerojatnost $P_m(d \mid l)$ dana je Gausovom razdiobom sa srednjom vrijednosti o_l:

$$P_m(d \mid l) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(d-o_l)^2}{2\sigma^2}} \qquad (4\text{-}23)$$

Standardna devijacija ove razdiobe modelira nesigurnost izmjerene udaljenosti bazirane na:

- granularnosti diskretizacije *L*, koja predstavlja položaj robota,
- točnosti modela prostora i
- točnosti senzora.

Slika 3.2.a daje primjere takvih Gausovih razdioba za ultrazvučne i laserske senzore. Ovdje je udaljenost do najbliže prepreke 230 cm. Razmotrimo ovdje da laserski senzor ima veću točnost od ultrazvučnog senzora, što je prikazano manjom varijancom.

b) Nepoznate prepreke: U Markovljevoj lokalizaciji pretpostavlja se da je model prostora statičan i potpun. Međutim, okoline mobilnog robota su obično popunjene objektima koji se ne nalaze na karti. Zbog toga postoji vjerojatnost (različita od nule) da je zraka senzora reflektirana od prepreke koja se ne nalazi u modelu okoline. Uzimajući u obzir da su ti objekti ravnomjerno raspoređeni u okolini, vjerojatnost $P_u(d_i)$ detekcije nepoznate prepreke na udaljenosti d_i neovisna je o lokaciji robota i može se modelirati geometrijskom razdiobom koja proizlazi iz sljedećeg razmatranja. Udaljenost d_i izmjerena je ako se zraka ne reflektira od prepreke na bližoj udaljenosti $d_{j<i}$ a reflektira se baš na udaljenosti d_i. U tom slučaju rezultirajuća vjerojatnost iznosi:

$$P_u(d_i) = \begin{cases} 0 & i = 0 \\ c_r(1 - \sum_{j<i} P_u(d_j)) & i \neq 0 \end{cases} \qquad (4\text{-}24)$$

U ovoj jednadžbi konstanta c_r označava vjerojatnost da je zraka senzora reflektirana od nepoznate prepreke u bilo kojem rasponu danom diskretizacijom.

Tipična razdioba za mjerenja sonara i lasera prikazana je na *slici 3.2.b*. U ovom primjeru, relativno velika vjerojatnost mjerenja 500 cm, je zbog toga što je maksimalni raspon aproksimacijskog senzora postavljen na 500 cm. Tako, ta udaljenost predstavlja vjerojatnost mjerenja od barem 500 cm.

Očito je da se samo jedan od ova dva slučaja može dogoditi u jednom trenutku: zraka senzora ili je reflektirana od poznatog ili nepoznatog objekta. Tako je $P(d_i \mid l)$ mješavina dviju razdioba P_m i P_u. Da bismo odredili kombiniranu vjerojatnost $P(d_i \mid l)$ mjerenja udaljenosti d_i pri lokaciji robota l razmotrimo sljedeće dvije situacije: Udaljenost d_i izmjerena je ako zraka senzora:

1. a) nije reflektirana od poznate prepreke prije dostizanja položaja d_i

$$a_1 = 1 - \sum_{j<i} P_u(d_j) \qquad (4\text{-}25)$$

b) je reflektirana od poznate prepreke na udaljenosti d_i

$$a_2 = c_d P_m(d_i \mid l) \qquad (4\text{-}26)$$

2. a) nije reflektirana ni od poznate niti nepoznate prepreke prije dostizanja položaja d_i

$$b_1 = 1 - \sum_{j<i} P(d_j \mid l) \qquad (4\text{-}27)$$

b) je reflektirana od nepoznate prepreke na udaljenosti d_i

$$b_2 = c_r \qquad (3\text{-}28)$$

Parametar c_d u *jednadžbi (3-26)* označava vjerojatnost da će senzor detektirati najbližu prepreku na karti. Ova razmatranja za kombiniranu vjerojatnost mogu se sumirati u *jednadžbu (3-29)*. S dvostrukom negacijom i ubacivanjem *jednadžbi (3-25)* i *(3-28)* konačno dobivamo *jednadžbu (3-32)*:

$$P(d_i \mid l) = p((a_1 \wedge a_2) \vee (b_1 \wedge b_2)) \quad (4\text{-}29)$$

$$= \neg p(\neg(a_1 \wedge a_2) \vee \neg(b_1 \wedge b_2)) \quad (4\text{-}30)$$

$$= 1 - \left(\left[1 - P(a_1 a_2)\right] \cdot \left[1 - P(b_1 b_2)\right]\right) \quad (4\text{-}31)$$

$$= 1 - \left(1 - \left(1 - \sum_{j<i} P_u(d_j)\right) \cdot c_d P_m(d_i \mid l)\right) \cdot \left(1 - \left(1 - \sum_{j<i} P(d_j)\right) \cdot c_r\right) \quad (4\text{-}32)$$

Da bi se održala vjerojatnost mjerenja d_n, iskoristit ćemo sljedeću jednakost: Vjerojatnost mjerenja udaljenosti veće ili jednake maksimalnom dometu senzora, jednaka je vjerojatnosti neizmjerene udaljenosti kraće od d_n. U našoj inkrementalnoj metodi ova vjerojatnosti se može lako odrediti:

$$P(d_n \mid l) = 1 - \sum_{j<n} P(d \mid l) \quad (4\text{-}33)$$

Kad sve sumiramo, vjerojatnost mjerenje senzora izračunava se inkrementalno za različite udaljenosti s početkom na udaljenosti $d_1 = 0$ cm. Za svaku udaljenost razmatra se vjerojatnost da je zraka senzora stigla do odgovarajuće udaljenosti i reflektirala se ili od najbliže prepreke na karti (u smjeru senzora) ili od nepoznatog objekta.

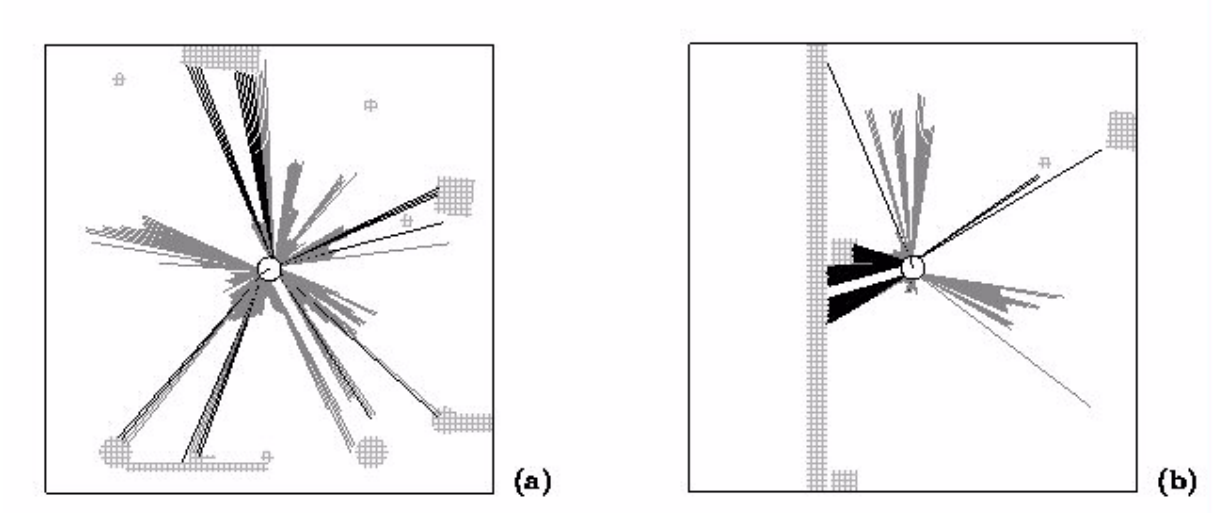
4.1.2.3. Tehnike filtriranja za dinamičke okoline

Markovljeva lokalizacija pokazala se robusnom za uobičajene promjene okoline kao što su otvorena/zatvorena vrata ili ljudi koji prolaze. Nažalost, lokalizacija ne uspijeva u slučajevima kad puno aspekata okoline nije pokriveno s modelom prostora. To se događa kada je okolina napunjena ljudima koji pokrivaju senzore robota i tako dolazi do puno nepredviđenih mjerenja.

Razlog zbog kojeg Markovljeva lokalizacija pada na testu u takvim situacijama jest kršenje Markovljeve pretpostavke na kojoj se zasnivaju gotovo sve lokalizacijske tehnike. Kao što smo već rekli, ta pretpostavka tvrdi da su mjerenja senzora razmatrana u trenutku t neovisna o svim drugim mjerenjima uz poznato trenutno stanje prostora L_t. U slučajevima lokalizacije u ljudima naseljenima područjima ovo pravilo je jasno prekršeno ako se pri tom koristi statički model prostora.

Da bismo ilustrirali ovu točku, na *slici 3.3* prikazana su tipična očitanja lasera tijekom jednog pokusa u muzeju (očitanja maksimalnog dometa su izostavljena). Slika također prikazuje prepreke sadržane u karti. Očito je da su očitanja uvelike iskvarena jer ljudi u muzeju nisu predstavljeni u statičkom modelu prostora. Različite boje zraka ukazuju na dvije klase kojima pripadaju: crne linije pripadaju statičkim objektima na karti i neovisne su jedna o drugoj ako je položaj robota poznat; sive linije su zrake reflektirane od ljudi i one se ne mogu predvidjeti pomoću modela okoline i zbog toga nisu neovisne jedna o drugoj. Pošto okoliš pun ljudi povećava vjerojatnost robota da se nalazi u blizini modeliranih prepreka, robot brzo gubi pojam o svom položaju kad objedinjuje sva mjerenja senzora. Da bi se neovisnost senzorskih mjerenja ponovno uspostavila mogli bismo uključiti položaj robota i poziciju ljudi

u varijable stanja *L*. Nažalost, to je nepraktično jer se tada složenost proračuna povećava eksponencijalno u broju neovisnih varijabli stanja koje se moraju estimirati.



Slika 4.3. Tipični prikazi očitanja lasera kada je robot okružen ljudima

Jedna od solucija bliskih rješavanju ovog problema mogla bi biti da se karta prilagođava promjenama u okolini. Problem je u tome što i tehnike za izgradnju karti prostora isto tako pretpostavljaju da je okolina gotovo statična. Jedan drugi pristup bio bi da se percepcijski model adaptira tako da točno reflektira takve situacije. Povodom toga potrebno je napomenuti da naš percepcijski model već pridružuje određenu vjerojatnost događajima kod kojih je senzorska zraka reflektirana od nepoznatih prepreka. Ovakvi pristupi ne rade pouzdano s uobičajenom blokadom senzora i nikako nisu dostatni u slučajevima gdje je više od 50% mjerenja iskvareno. Naša lokalizacijska metoda zbog toga uključuje filtre koji su dizajnirani da detektiraju iskvarena očitanja. U usporedbi s modifikacijom statičkog modela senzora opisanog gore, ovi filtri imaju prednost da ne usrednjavaju mjerenja svih mogućih situacija i da je njihova odluka bazirana na trenutnom uvjerenju robota.

Ovi filtri su dizajnirani da odaberu očitanja senzora koja ne dolaze od objekata sadržanih u karti. U ovom odjeljku predstavit ćemo dva različita tipa filtra. Prvi se naziva *entropijski filtar* pošto filtrira očitanje bazirano jedino na njegovu utjecaju na uvjerenje *Bel(L)*. Drugi filtar je *filtar udaljenosti* koji odabire očitanja prema tome koliko su kraća od očekivane vrijednosti.

1. **Entropijskifiltri :**

Entropiju uvjerenja robota preko *L* označit ćemo sa *H(L)* i definirati je kao:

$$H(L) = -\sum_{l} Bel(L = l) \log Bel(L = l) \qquad (4\text{-}34)$$

Entropija je ovdje mjera nesigurnosti rezultata slučajne varijable *L*. Što je veća entropija to je veća i nesigurnost robota o svom položaju. Entropijski filtar mjeri relativnu promjenu entropije pri objedinjavanju očitanja senzora u uvjerenje *Bel(L)*. Zbog točnosti označit ćemo sa *s* mjerenje senzora (u našem slučaju samo jedno mjerenje). Promjena entropije uvjerenja *Bel(L)* uz dano mjerenje *s* definira se kao:

$$\Delta H(L \mid s) := H(L \mid s) - H(L) \qquad (4\text{-}35)$$

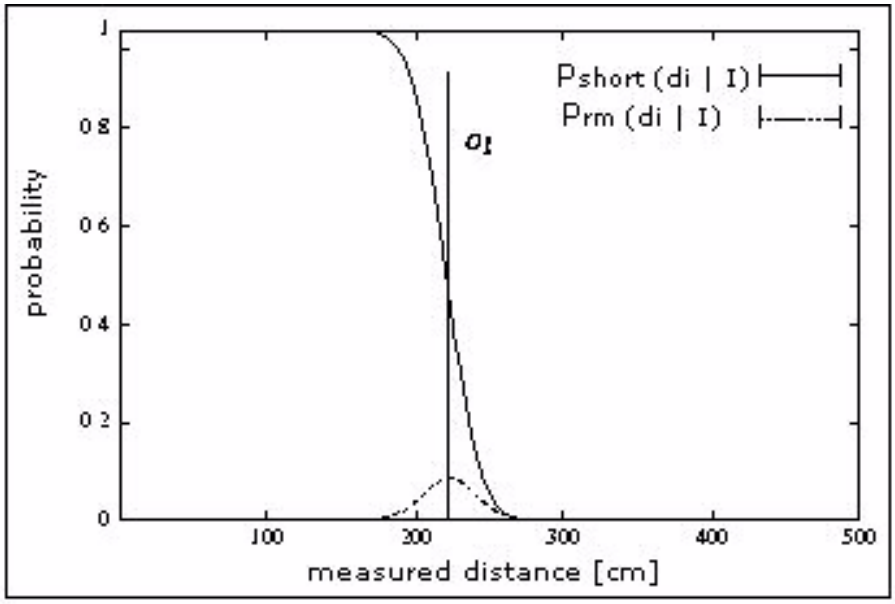
Izraz *H(L | s)* označava entropiju uvjerenja *Bel(L)* nakon objedinjavanja mjerenja senzora *s (jednadžbe (3-19) do (3-21))*. Pozitivna promjena entropije govori da nakon objedinjavanja mjerenja *s*, robot je manje siguran za svoj položaj, a negativna entropija označava porast sigurnosti robota za svoj položaj. Stoga je zadatak entropijskog filtra da izluči sva mjerenja senzora *s* kod kojih je $\Delta H(L \mid s) < 0$.

Entropijski filtri rade dobro kada je uvjerenje robota fokusirano na ispravnim hipotezama. Međutim, mogu zakazati u situacijama u kojima je uvjerenje robota neispravno. Prednosti entropijskih filtara su u tome da ne prave nikakve pretpostavke o prirodi senzorskih podataka niti o vrsti smetnji koje se javljaju u dinamičkim okolinama.

2. Filtri udaljenosti:

Dizajnirani su posebno za blizinske senzore kao što su npr. laserski senzori. Senzori udaljenosti zasnivaju se na jednostavnoj tvrdnji: Kod blizinskih senzora nemodelirane prepreke tipično proizvode očitanja koja su kraća od udaljenosti očekivane na osnovi karte. Drugim riječima, koriste se samo ona mjerenja koja potvrđuju uvjerenje robota. U biti, filtri udaljenosti biraju ona očitanja senzora koja su zasnovana na njihovoj udaljenosti relativno u odnosu prema udaljenosti najbliže prepreke na karti.

Da budemo malo precizniji, ovaj filtar uklanja ona mjerenja senzora *s* koja su s vjerojatnošću većom od (ovaj prag se u svim eksperimentima postavlja na vrijednost 0.99) kraća od očekivanih i koja su zbog toga uzrokovana nemodeliranim objektima.



Slika 4.4. Vjerojatnosti $P_m(d_i \mid l)$ i $P_{short}(d_i \mid l)$

Neka $d_1, \ldots, d_n$ bude diskretni skup mogućih udaljenosti izmjerenih blizinskim senzorom. Sa $P_m(d_i \mid l)$ označimo vjerojatnost mjerene udaljenosti d_i ako se robot nalazi na lokaciji *l* i senzor detektira najbližu prepreku na karti u smjeru direkcije senzora. Razdioba P_m opisuje mjerenje senzora očekivano iz karte. Pretpostavka je da je to Gausova razdioba sa srednjom vrijednosti o_l do najbliže prepreke u smjeru direkcije senzora. Iscrtkana linija na *slici 3.4* predstavlja P_m za laserski senzor i udaljenost o_l od 230 cm. Sada možemo definirati još i vjerojatnost $P_{short}(d_i \mid l)$ da je izmjerena udaljenost d_i kraća od očekivane

prema kojoj se robot nalazi na lokaciji l. Ova vjerojatnost je očito ekvivalent vjerojatnosti da je očekivano mjerenje o_l veće od d_i uz položaj robota l pa se može izračunati na način:

$$P_{short}(d_i \mid l) = \sum_{j>i} P_m(d_j \mid l) \qquad (4\text{-}36)$$

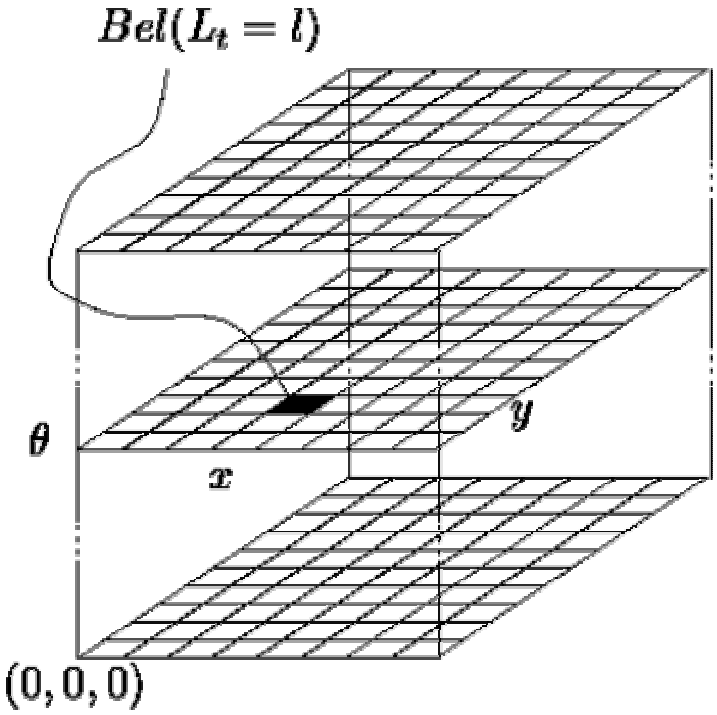
U praksi smo, međutim, zainteresirani za vjerojatnost $P_{short}(d_i)$ da je d_i kraće od očekivanog uz dano kompletno trenutno uvjerenje robota. Zbog toga moramo usrednjavati sve moguće položaje robota:

$$P_{short}(d_i) = \sum_{l} P_{short}(d_i \mid l) Bel(L = l) \qquad (4\text{-}37)$$

Uz danu razdiobu $P_{short}(d_i)$ možemo implementirati filtar udaljenosti tako da izlučimo sva mjerenja senzora d_i u kojima je $P_{short}(d_i) > \gamma$. Dok entropijski filtar filtrira mjerenja prema njihovom utjecaju na uvjerenje robota, filtar udaljenosti izlučuje jedino ona mjerenja koja su bazirana na njihovoj vrijednosti bez obzira na njihov utjecaj na uvjerenje robota.

4.1.2.4. Prostor stanja zasnovan na mreži

Vratimo se sada na temu: na koji način uspješno reprezentirati i proračunati razdiobu uvjerenja robota pomoću prostora stanja zasnovanog na mreži. Označimo sa L trodimenzionalnu ravnomjerno raspoređenu mrežu s prostornom rezolucijom između 10 cm i 40 cm i kutnom rezolucijom između 2° i 5°. Na sljedećoj slici prikazana je struktura mreže uvjerenja položaja. Svaki sloj te mreže se poklapa sa svim mogućim položajima robota istog smjera.



Slika 4.5. Prikaz prostora stanja robota pomoću prostorne mreže

Dok takva fino uzorkovana aproksimacija omogućava estimaciju položaja robota s visokom točnošću, loša strana te fine diskretizacije leži u prevelikom prostoru stanja koje se mora održavati. Za okolinu srednje veličine od oko 30x30 m^2 s kutnom rezolucijom 2° i

veličinom jedne ćelije mreže 15x15 cm^2 prostor stanja sastoji se od 7,200,000 stanja. Osnovni Markovljev algoritam osvježava svako od ovih stanja za svaki podatak senzora i za svaki djelić pokreta robota. Brzine današnjih računala omogućavaju osvježavanje tako velikih matrica u stvarnom vremenu.

Da bi se takva prostorna stanja mogla efikasno osvježiti, razvijene su dvije tehnike. Prva tehnika koristi unaprijedno izračunavanje modela senzora i omogućuje nam da odredimo vjerojatnost $P(s \mid l)$ senzorskih mjerenja operacijama pretraživanja (pregledavanja). Druga tehnika je selektivna strategija osvježavanja koja se fokusira na izračunavanje pomoću osvježavanja samo onog bitnog dijela prostora stanja. Bazirano na ove dvije tehnike, Markovljeva lokalizacija pomoću mreže prostora stanja, može se koristiti za on-line estimaciju položaja mobilnog robota tijekom njegova operiranja, koristeći jeftino PC računalo.

1. **Unaprijedno izračunavanje modela senzora:**

Kao što smo već opisali u *odjeljku 3.1.2.2*, percepcijski model $P(s \mid l)$ za blizinske senzore ovisi samo o udaljenosti o_l do najbliže prepreke u karti u smjeru zrake senzora. Zbog pretpostavke da je karta okoline statička, naša metoda unaprijed proračunava i pohranjuje ove udaljenosti o_l za svaku moguću lokaciju robota l u svojoj okolini. Slijedeći naš model senzora koristimo diskretizaciju $d_1, \ldots, d_n$ svih mogućih udaljenosti o_l. Ta diskretizacija je točno jednaka za očekivane i izmjerene udaljenosti. Tada pohranjujemo za svaku lokaciju l samo indeks očekivane udaljenosti o_l u trodimenzionalnu tablicu. Za tu tablicu potreban je samo jedan *byte* za jednu vrijednost ako se koristi 256 različitih vrijednosti za diskretizaciju o_l. Vjerojatnost $P(d_i \mid o_l)$ mjerenja udaljenosti d_i, uz najbližu prepreku na udaljenosti o_l, može se također unaprijed izračunati i pohraniti u dvodimenzionalnu tablicu za pretraživanje.

Kao rezultat, vjerojatnost $P(s \mid l)$ mjerenja s, uz danu lokaciju l, može se brzo izračunati s dva ugniježđena upita za pretragu. Prva pretraga izvlači udaljenost o_l do najbliže prepreke u smjeru senzora, uz danu lokaciju robota l. Druga pretraga koristi se za vjerojatnost $P(s \mid o_l)$. Uspješno proračunavanje zasnovano na ovakvom pretraživanju omogućuje našoj implementaciji brzo objedinjavanje čak i laserskih očitanja koja se sastoje od 180 vrijednosti u ukupnom stanju uvjerenja robota. U praksi, primjena tablica za pretraživanje dovodi do 10 puta bržeg proračunavanja u usporedbi s on-line proračunom udaljenosti do najbližeg objekta.

2. **Selektivno osvježavanje:**

Algoritam selektivnog osvježavanja zasnovan je na činjenici da se tijekom lokalizacije robota točnost estimacije položaja trajno povećava i gustoća se brzo koncentrira na one ćelije mreže koje predstavljaju stvarni položaj robota. Vjerojatnost ostalih ćelija mreže smanjuje se tijekom lokalizacije, a osnovna ideja ove metode optimizacije jest da se takve ćelije isključe iz daljnjeg osvježavanja.

Zbog tog razloga uključujemo prag (postavlja se na oko 1% vjerojatnosti položaja) i osvježavamo samo one ćelije mreže l s vjerojatnošću $Bel(L_t = l) > \varepsilon$. Da bi se omogućilo takvo selektivno osvježavanje pri održanju gustoće kroz čitav prostor stanja, aproksimirat ćemo $P(s_t \mid l)$ za ćelije sa $Bel(L_t = l) > \varepsilon$ pomoću apriori vjerojatnosti mjerenja s_t. Ovaj kvantitet označavamo sa $\widetilde{P}(s_t)$ i određujemo ga pomoću usrednjavanja svih mogućih lokacija robota:

$$\widetilde{P}(s_t) = \sum_l P(s_t \mid l) P(l) \qquad (4\text{-}38)$$

$\widetilde{P}(s_t)$ je neovisan o trenutnom stanju uvjerenja robota i može biti određen unaprijed. Inkrementalno pravilo osvježavanja za nove podatke senzora s_t mijenja se prema sljedećoj jednadžbi (usporedite s *jednadžbom (3-9)*):

$$Bel(L_t = l) \longleftarrow \begin{cases} \alpha_t \cdot P(s_t \mid l) \cdot Bel(L_{t-1} = l) & ako\ je\ Bel(L_{t-1} = l) > \varepsilon \\ \alpha_t \cdot \widetilde{P}(s_t) \cdot Bel(L_{t-1} = l) & ako\ je\ Bel(L_{t-1} = l) \le \varepsilon \end{cases} \qquad (4\text{-}39)$$

Množeći $\widetilde{P}(s_t)$ s normalizacijskim faktorom $_t$ ovu jednadžbu možemo napisati ponovno:

$$Bel(L_t = l) \longleftarrow \begin{cases} \widetilde{\alpha}_t \cdot \dfrac{P(s_t \mid l)}{\widetilde{P}(s_t)} \cdot Bel(L_{t-1} = l) & ako\ je\ Bel(L_{t-1} = l) > \varepsilon \\ \widetilde{\alpha}_t \cdot Bel(L_{t-1} = l) & ako\ je\ Bel(L_{t-1} = l) \le \varepsilon \end{cases} \qquad (4\text{-}40)$$

gdje je $\widetilde{\alpha}_t = \alpha_t \cdot \widetilde{P}(s_t)$.

Ključna prednost algoritma selektivnog osvježavanja danog *jednadžbom (3-40)* jest u tome da se sve ćelije sa *Bel(L$_{t-1}$ = l)* $\le$ osvježavaju s istom vrijednosti $\widetilde{\alpha}_t$. Zbog održanja glatkih prijelaza između globalne lokalizacije i praćenja položaja te da bi se fokusiralo na proračunavanje bitnih dijelova prostora stanja *L*, koristimo particioniranje (sjeckanje) prostora stanja. Pretpostavimo da je prostor stanja particioniran u *n* segmenata $_1$, ..., π_n. Segment $_i$ naziva se aktivnim u trenutku *t* ako sadrži lokacije s vjerojatnošću iznad praga ; inače ga nazivamo pasivnim jer su vjerojatnosti ćelija ispod tog praga. Očito je da možemo pratiti pojedine vjerojatnosti unutar pasivnog dijela $_i$ tako da akumuliramo normalizacijske faktore $\widetilde{\alpha}_t$ u vrijednost $_i$. Kad god segment $_i$ postane pasivan (npr. ako vjerojatnosti svih lokacija unutar $_i$ više ne prelaze prag), normalizator $_i(t)$ se inicijalitira na 1 i potom osvježava prema jednadžbi:

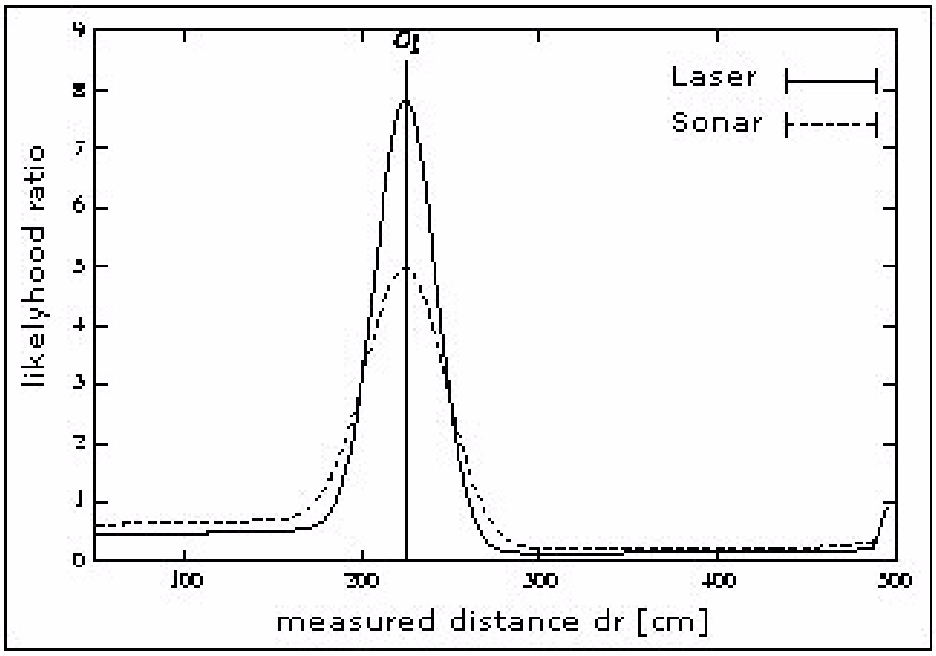
$$\beta_i(t+1) = \widetilde{\alpha}_t \cdot \beta_i(t) \qquad (4\text{-}41)$$

Čim neki segment opet postane aktivan možemo obnoviti vjerojatnosti pojedinih ćelija mreže tako da pomnožimo vjerojatnosti svake ćelije s akumuliranim normalizatorom $_i(t)$. Zbog praćenja položaja robota otkako je neki segment postao pasivan, dovoljno je samo inkorporirati akumulirano gibanje kad neki dio postane ponovno aktivan. Da bi se moglo uspješno detektirati kad je neki segment potrebno ponovno aktivirati, potrebno je pohraniti maksimalnu vjerojatnost P_i^{max} svih ćelija u dijelu vremena kada postaju pasivne. Kad god $P_i^{\max} \cdot \beta_i(t)$ prelazi prag , segment $_i$ aktivira se ponovno zbog toga jer sadrži barem jedan položaj s vjerojatnošću iznad praga . Trenutno se u našoj implementaciji particionira prostor stanja *L* tako da se svaki segment $_i$ sastoji od svih lokacija s jednakom orijentacijom relativno u odnosu prema početnom položaju robota.

Da bismo ilustrirali efekt ovakvog algoritma selektivnog osvježavanja, usporedimo osvježavanja aktivnih i pasivnih ćelija na nailazećim podacima senzora. Prema *jednadžbi (3-40)* razlika leži u omjeru $P(s_t \mid l) / \widetilde{P}(s_t)$. Primjer jednog takvog modela za blizinske senzore može se vidjeti na *slici 3.6* (na kojoj je s_t zamjenjen s blizinskim mjerenjem d_i). U početku lokalizacijskog procesa, sve ćelije su aktivne i osvježavaju se prema omjeru prikazanom na *slici 3.6.* Izmjerene i očekivane udaljenosti za ćelije koje ne predstavljaju stvarnu lokaciju robota obično značajno odstupaju tako da vjerojatnosti takvih ćelija brzo padaju ispod praga .

Sada efekt algoritma selektivnog osvježavanja postaje očit: Oni dijelovi prostora stanja koji se dobro ne poravnavaju s orijentacijom okoline, brzo postaju pasivni tijekom

lokalizacije robota. Potom se samo mali dio prostora stanja mora osvježiti čim robot točno odredi svoj položaj. Međutim, ako se robot izgubi, tada omjeri sličnosti za udaljenosti izmjerene na aktivnim lokacijama postaju manji od jednog prosječnog. Tako se vjerojatnosti aktivnih lokacija smanjuju dok se normalizator $_i$ pasivnih dijelova povećava dok ovi segmenti ponovno ne postanu aktivni. Dok se položaj robota jednom ne nađe među ovim aktivnim lokacijama, robot je sposoban ponovno uspostaviti ispravno uvjerenje.



Slika 4.6. Omjer $\frac{P(d_i \mid l)}{\widetilde{P}(d_i)}$ *za mjerenja sonara i lasera za očekivanu udaljenost* o_l *= 230 cm*

Prilikom eksperimentalnih proučavanja nije primjećeno da ovaj algoritam ima zamjetno negativan utjecaj na ponašanje robota. Čak nasuprot tome, ispalo je da je jako efikasan jer se samo mali dio prostora stanja (većinom manje od 5 %) mora osvježavati jednom kada je položaj robota ispravno određen, a vjerojatnosti aktivnih položaja općenito se sumiraju do najmanje 0.99. Tako algoritam selektivnog osvježavanja automatski adaptira vrijeme izračunavanja potrebnog za prebacivanje uvjerenja robota u njegovu sigurnost. Na taj način, ovaj sustav je sposoban efektivno pratiti položaj robota jednom kada je on određen. Dodatno, Markovljeva lokalizacija održava sposobnost otkrivanja lokalizacijskih pogrešaka i ponovne lokalizacije u slučaju neuspjeha. Jedina negativnost leži u fiksnoj reprezentaciji mreže koja ima neželjeni efekt da memorijske potrebe ostaju konstantne čak i ako je potrebno osvježavati samo mali dio prostora stanja.

4.2. Opis ML modula u Saphiri

Saphira sadrži ekspeditivnu verziju Markovljeve Monte Carlo lokalizacije kojom se robot lokalizira u gobalnoj karti, a bazirana je na laserskom senzoru ili sonarima. Markovljeva lokalizacija je proces estimiranja stanja robota, tj. njegova položaja. Osnovna ideja je da se gibanje robota podijeli u skup diskretnih koraka (npr. po 1 metar udaljenosti i 20° okreta među koracima). Svakim novim korakom položaj robota se estimira iz proračuna informacija enkodera i senzora uspoređujući te rezultate s kartom.

Ta estimacija je proces koji se odvija u dva koraka: predikcije na temelju informacija iz enkodera i osvježavanja na temelju informacija iz senzora. Općenito, nesigurnost položaja robota raste s korakom predikcije i smanjuje se s korakom osvježavanja, gdje se položaj registrira na karti. Proces se naziva Markovljevim jer se pretpostavlja da na njega ne utječu prošla stanja, tj. nije bitno kako je robot stigao do određenog položaja jer se sve potrebne informacije nalaze u trenutnom položaju i nesigurnosti robota.

Postoji nekoliko načina predstavljanja nesigurnosti položaja robota. Dobar pregled toga može se pronaći u dokumentu [11] u *docs* mapi. Za implementaciju ovoga izabrana je Monte Carlo metoda pomoću koje je nesigurnost položaja robota predstavljena sa skupom točaka koje su koncentrirane oko područja najveće vjerojatnosti u kojem bi se robot trebao nalaziti.

Postoji dosta parametara koji su bitni za proces Markovljeve lokalizacije. Glavni među njima su: broj točaka za prikaz nesigurnosti položaja, pojačanje koje se koristi pri osvježavanju informacija sa senzora i udaljenost između osvježavanja. Svi ovi parametri su unaprijed postavljeni na razumne vrijednosti, ali se mogu mijenjati preko grafičkog sučelja u Saphiri.

4.2.1. Učitavanje ML modula

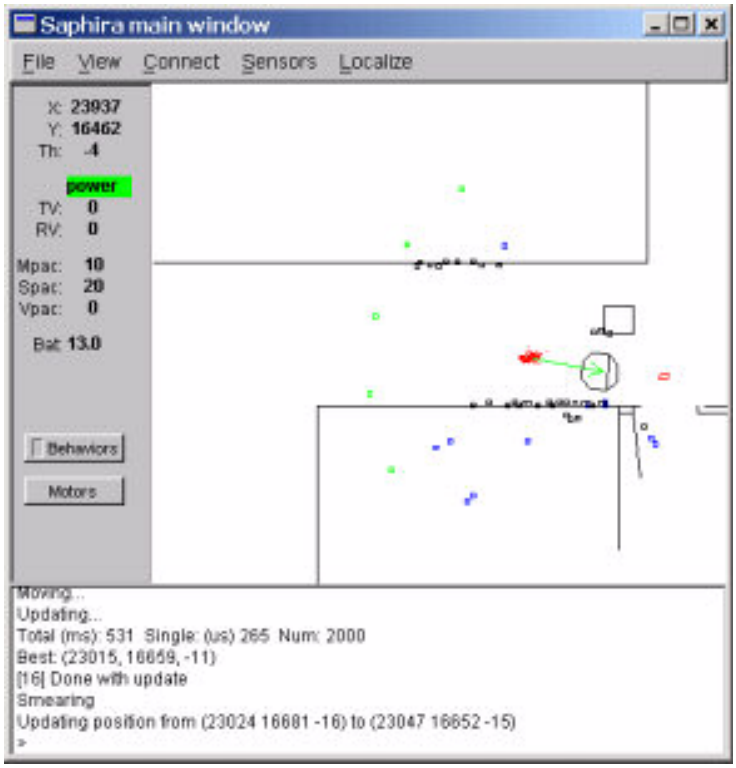
Ključne datoteke za algoritam Markovljeve lokalizacije prikazane su u sljedećoj tablici.

Ime datoteke	**Opis**
`lib/sfLoc.so` `bin\sfLoc.dll`	Ključne datoteke s rutinama za API sučelje za Markovljevu lokalizaciju s laserskim senzorom ili sonarima.
`lib/sfLocF1.so` `bin\sfLocF1.dll`	Datoteke za grafičko sučelje. Dodaju padajuće izbornike i prozore za mijenjanje parametara lokalizacije.
`colbert/flloc.act`	Ovom datotekom se inicijalizira ML sa sonarima.
`colbert/scan.act`	Koristi se za inicijalizaciju ML s laserskim senzorima, učitavanje karte napravljene u *ScanStudiu* i gradijentnog modula.

Tablica 4.2. Datoteke koje se koriste pri pokretanju ML algoritma

Najjednostavniji način da se učita *flloc.act*jest da se pokrene Colbert naredba *load flloc*. Ta naredba će učitati WLD datoteku *aic.wld*, ključne datoteke za Markovljevu lokalizaciju i grafičko sučelje te pokrenuti simulator sa istom WLD datotekom i spojiti se na njega. Nakon toga, crveni skup točaka bi trebao slijediti točnu stvarnu poziciju robota. Potrebno je odabrati

"Localize->Update Position" i Saphira klijent će pratiti položaj robota. Na *slici 3.7* možete vidjeti kako izgleda kretanje robota tijekom Markovljeve lokalizacije.



Slika 4.7. Kretanje robota tijekom Markovljeve lokalizacije

Na *slici 3.7* crveno područje označava skup točaka Markovljeve lokalizacije i ono se osvježava dok se robot kreće, tipično svakih 1 m ili 20°. Zelena strelica pokazuje najbolji estimat položaja robota u zadnjem koraku.

Učitavanjem datoteka zapravo se ne stvaraju lokalizacijski objekti. Da bi se to učinilo moraju se pozvati funkcije *mcSonarInit()*i *mcLrfInit()*. Funkcija *mcSonarInit()* poziva se iz datoteke *flloc.act*, a *mcLrfInit()* iz datoteke *lrfloc.act*. Inače, postoji cijela lista naredaba koje se mogu koristiti za upravljanje ML modulom, a njihov pregled je dan u *dodatku 12*.

ML parametrima može se manipulirati preko grafičkog sučelja. Potrebno je iz padajućeg izbornika odabrati stavku "Localize->Parameters" nakon čega se pojavljuje prozor u kojem se mogu mijenjati broj ML točaka, pojačanje i frekvencija osvježavanja. Ove promjene se odražavaju u programu istog momenta. Ako broj ML točaka postavimo na nulu tada će crveno područje skupa točaka nestati.

4.2.2. Opcije lokalizacijskog padajućeg izbornika

Koristeći padajući izbornik "Localize" može se utjecati na nekoliko aspekata prikaza ili performansi Markovljeve lokalizacije.

- "Localize->Display->Samples" – uključuje ili isključuje prikaz crvenog područja ML skupa točaka.
- "Localize->Display->Grid" uključuje ili isključuje prikaz mreže na karti.

- "Localize->Update Samples" koristi se za osvježavanje uzoraka Markovljeve lokalizacije. Ako je opcija uključena Markovljeva lokalizacija će raditi dok se robot kreće, a ako je isključena tada ML prestaje s radom.

- "Localize->Update Robot Pos" koristi se da bi se robot pri gibanju uvijek pomicao prema mjestu najboljeg estimata položaja dobivenog Markovljevom lokalizacijom. Ako je opcija isključena tada će skup ML točaka i stvarni položaj robota međusobno divergirati.

- "Localize->Sample Set" mijenja položaj skupa ML točaka. Opcija "Center on robot" uzrokuje da se pretpostavlja Gaussova raspodjela točaka oko centra robota, a opcija "Uniform" da se točke raspodijele ravnomjerno po prostoru karte.

- "Localize->Update Map" se koristi ako se u Saphiru učita nova WLD datoteka zbog usklađivanja struktura podataka koje koristi ML modul.

- "Localize->Parameters" pokreće prozor u kojem se mogu mijenjati ML parametri.

4.2.3. Inicijalizacijske funkcije

Postoji nekoliko funkcija za inicijalizaciju ML modula, a one se mogu pozvati iz Colberta ili C++ koda. Inicijalizacija postavlja objekt za ML proces te izvodi neka bitna procesiranja kao što je postavljanje strukture podataka za kartu.

U nastavku slijede 4 inicijalizacijske funkcije ML modula s njihovim objašnjenjem:

- *mcSonarInit()* – inicijalizira ML modul koristeći očitanja sonara za osvježavanje. WLD datoteka se treba učitati prije pozivanja ove naredbe jer ona također postavlja strukturu podataka za učitanu kartu.

- *mcLrfInit()* – dio je modula *Saphira Navigation* koji dolazi s kompletnim paketima mapiranja i navigiranja. Inicijalizira ML modul koristeći očitanja laserskih senzora za korake osvježavanja. Datoteka WLD treba biti učitana prije pokretanja ove funkcije jer ona također postavlja strukturu podataka za učitanu kartu.

- *mcLrfScanInit()* – dio je modula *Saphira Navigation* koji dolazi s kompletnim paketima mapiranja i navigiranja. Inicijalizira ML modul koristeći očitanja laserskih senzora za korake osvježavanja. Umjesto karte formata WLD ovaj oblik ML algoritma koristi MAP kartu napravljenu u *ScanStudiu*. MAP karta mora se učitati tek nakon poziva ove funkcije.

- *mcLoadScanMap(char *mfile)* – dio je modula *Saphira Navigation* koji dolazi s kompletnim paketima mapiranja i navigiranja. Učitava u sustav MAP kartu *mfile* kreiranu pomoću *ScanStudia*. Prije ove funkcije potrebno je pozvati funkciju *mcLrfScanInit()*.

4.3. Obavljanje eksperimenta za ML

Prije obavljanja samog eksperimenta potrebno je obaviti neke pripreme što podrazumijeva da je ispravno podešena sustavska varijabla SAPHIRA_LOAD koju smo postavili na vrijednost "C:\Program Files\Saphira" (u Windows * OS-u) ili "/usr/local/Saphira" (u Linux OS-u). To je obavljeno još pri samoj instalaciji Saphire.

Da bi rad na eksperimentu bio jednostavniji i fleksibilniji sve naredbe za učitavanje ML modula, lasera i podešavanje njihovih parametara, postavljene su u tekstualnu datoteku *laser.act* čiji je izvorni kod dan u *dodatku 8*.

Prije obavljanja eksperimenta na stvarnom robotu, dobro je sve korake isprobati na simulatoru robota.

4.3.1. Obavljanje eksperimenta na simulatoru

Uobičajeni koraci za provjeru rada Markovljeve lokalizacije na Pioneer simulatoru robota bili bi sljedeći:

1. Potrebno je napisati datoteku s izvornim kodom koja će se učitavati u interaktivnom prozoru Saphire. Prikaz datoteke s objašnjenjima dan je u *dodatku 8*.

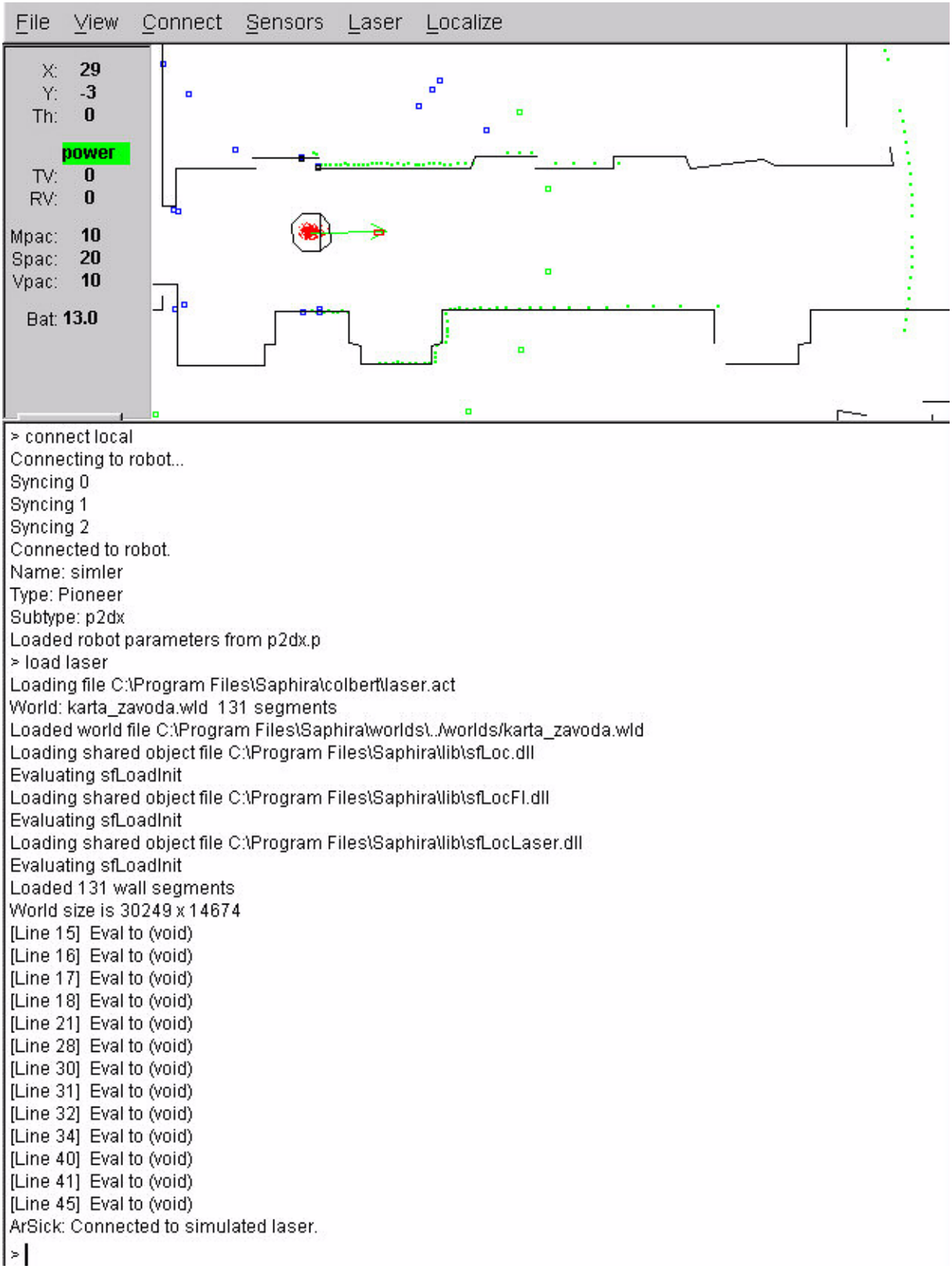
2. Pokrenuti simulator robota Pioneer i u njemu učitati parametre robota iz datoteke *p2dx.p*, te *wld* kartu robotskog okruženja koju smo u prethodnim eksperimentom mapiranja (opisano u *2. odjeljku*).

3. Pokrenuti Saphiru i spojiti se na simulator robota pomoću naredbe `connect local`.

4. Ako je spajanje na simulator uspješno tada se može pozvati datoteka *laser.act* pomoću Colbert naredbe `load laser`. Pomoću datoteke *laser.act* učitava se u Saphiru karta okoline robota, inicijaliziraju se laserski i ML modul, učitavaju se inicijalizacijske datoteke, inicijaliziraju se i pokreću sonari i laser te im se zadaju odgovarajući parametri kao i parametri Markovljeve lokalizacije. Ako je učitavanje uspješno obavljeno u interaktivnom prozoru u Saphiri trebale bi se pojaviti dojavne naredbe prikazane *slikom 3.8* kao i odgovarajuća slika robota s očitanjima lasera i sonara prikazana u LPS prozoru na istoj slici.

5. Kliknuti tipkom miša u LPS prozor Saphire da bi se omogućilo upravljanje robotom pomoću tipkovnice što je jednostavnije od zadavanja naredaba u interaktivnom prozoru. Tipke koje se mogu koristiti za upravljanje tipkovnicom dane su u *odjeljku 1.3.4.1.* Pri gibanju bi se robot trebao lokalizirati, tj. trebao bi se pomicati prema mjestu najboljeg estimata položaja dobivenog Markovljevom lokalizacijom. To je omogućeno u datoteci *laser.act* naredbom `mcUpdateRobotPose(1)`. Ta opcija može se kontrolirati i pomoću padajućeg izbornika pa ako se naknadno isključi, mogu se vidjeti odometrijske pogreške robota koje se povećavaju s prijeđenim putom.

6. Ako nije dobro podešen neki od parametara senzora ili Markovljeve lokalizacije tada se mogu dodatno podestiti u Saphiri pomoću padajućih izbornika.



Slika 4.8. Izgled interaktivnog prozora u Saphiri nakon učitavanja datoteke laser.act

4.3.2. Obavljanje eksperimenta na stvarnom robotu

Za obavljanje eksperimenta na stvarnom robotu potrebno je napraviti neke manje preinake u datoteci *laser.act*. To se odnosi na spajanje Saphire s robotom preko serijskog porta *ttyS1*, te lasera preko porta *ttyS2*. Spajanje na laser se može pokrenuti naredbom `sfStartLaser ("/dev/ttyS2")`. Tu naredbu je potrebno staviti na sam kraj datoteke *laser.act* tako da se poslije nje ne nalazi niti jedna druga naredba. Razlog tomu je da spajanje na laser traje nekih 45 do 60 sekundi i u tom periodu ne smije se ništa raditi niti u Saphiri niti na robotu.

Također je moguće zadati i ograničenja translacijske i kutne brzine te ubrzanja robota. To se radi pomoću naredaba `setMaxTransVel(int Vel)` i `setMaxRotVel(int Th)`. U našem slučaju ograničenja smo postavili na 200 mm/s (translacijska brzina) i 50 °/s (kutna brzina). Te naredbe upisuju se u datoteku *mySaphira.cpp* i prevode se u izvršnu aplikaciju kao što je opisano u *odjeljku 2.2.* Datoteka *mySaphira.cpp* dana je u *dodatku 11*.

U datoteci *laser.act*, umjesto naredbe za učitavanje *wld* datoteke `loadworld(char path)`, može se staviti naredba za učitavanje modela okoline u *map* formatu `mcLoadScanMap(char path)` koji je u svakom slučaju bolja aproksimacija okoline robota od modela danog *wld* formatom. Nakon toga potrebno je inicijalizirati objekt pomoću kojeg se karta predaje Saphiri što se čini pomoću naredbe `sfGradSetMap(mcGetObject())`.

Uobičajeni koraci za provjeru rada Markovljeve lokalizacije na stvarnom robotu bili bi sljedeći:

1. Potrebno je napraviti preinake u datoteci *laser.act* iz danoj u *dodatku 8* prema gornjim uputama. Preinake se pohranjene u novoj datoteci *laser_robot.act* danoj u *dodatku 9*.

2. Robot je potrebno postaviti na položaj iz kojeg smo ga pokretali dok smo radili mapiranje zavoda za APR (*Home Position*).

3. Pokrenuti Saphiru i spojiti se na stvarni robot pomoću padajućeg izbornika ili naredbe u interaktivnom prozoru. Potrebno se spojiti na serijski port *ttyS1*.

4. Ako je spajanje s robotom uspješno tada se može pozvati datoteka *laser.act* pomoću Colbert naredbe `load laser`. Pomoću datoteke *laser.act* učitava se u Saphiru karta okoline robota u *map* formatu, inicijaliziraju se laserski i ML modul, učitavaju se inicijalizacijske datoteke, inicijaliziraju se i pokreću sonari i laser te im se zadaju odgovarajući parametri kao i parametri Markovljeve lokalizacije. Ako je učitavanje uspješno obavljeno u interaktivnom prozoru u Saphiri bi se trebale pojaviti dojavne naredbe slične onima prikazanim *slikom 3.8* kao i odgovarajuća slika robota s očitanjima lasera i sonara prikazana u LPS prozoru na istoj slici. Treba naravno malo pričekati dok se laser spoji s robotom.

5. Kad je laser spojen tada je potrebno još samo omogućiti rad motora naredbom (tipkom) "Enable Motors" s lijeve strane prozora u Saphiri.

6. Kliknuti tipkom miša u LPS prozor Saphire da bi se omogućilo upravljanje robotom pomoću tipkovnice što je jednostavnije od zadavanja naredaba u interaktivnom prozoru. Tipke koje se mogu koristiti za upravljanje tipkovnicom danu su u *odjeljku 1.3.4.1.* Druga opcija je da spojimo palicu za igru (joystick) na robot i pomoću nje ga upravljamo. Pri gibanju bi se robot trebao lokalizirati, tj. trebao bi se pomicati prema mjestu najboljeg estimata položaja dobivenog Markovljevom lokalizacijom. To je omogućeno u datoteci *laser.act* naredbom `mcUpdateRobotPose(1)`. Ta opcija se može kontrolirati i pomoću

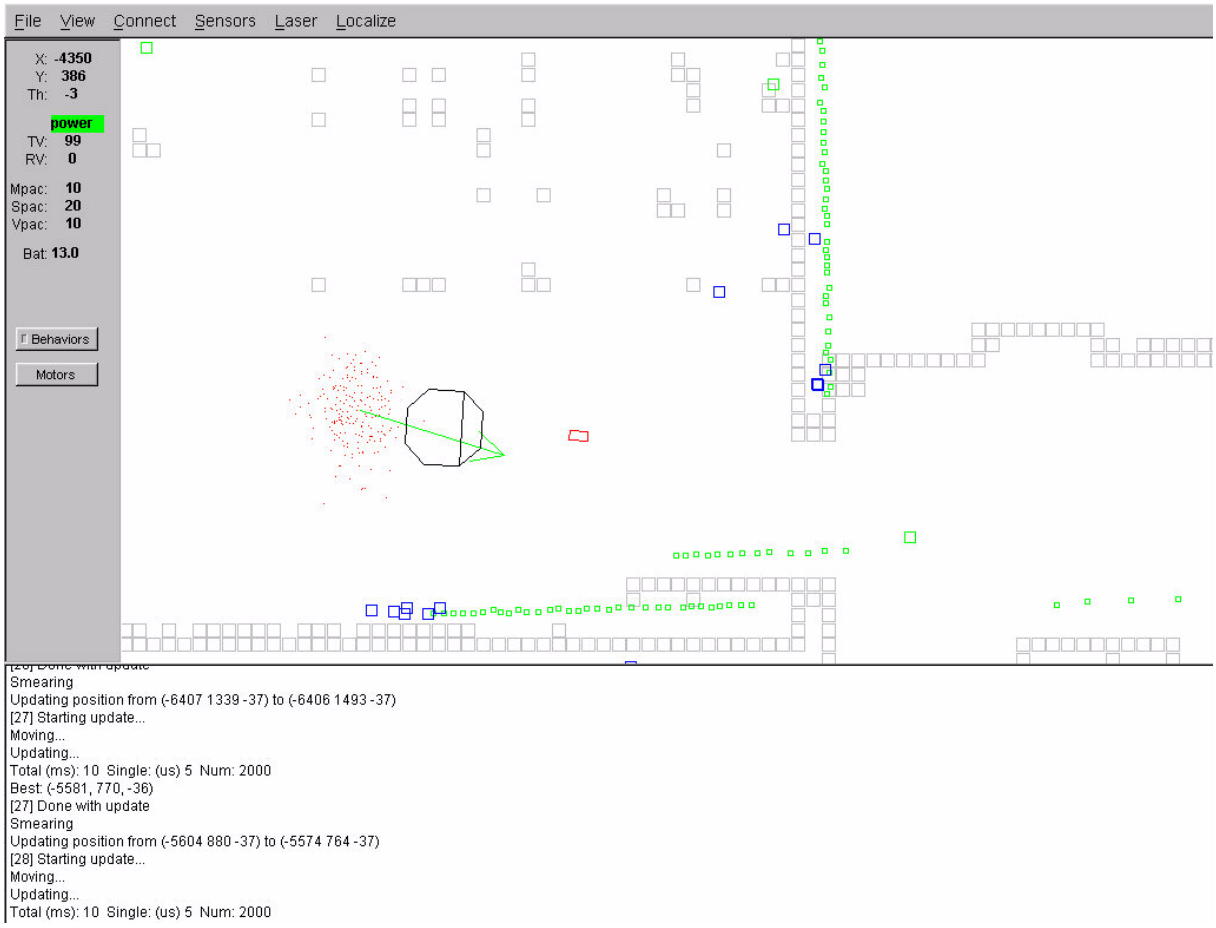
padajućeg ibornika pa ako se naknadno isključi, mogu se vidjeti odometrijske pogreške robota koje se povećavaju s prijeđenim putom.

7. Ako nije dobro podešen neki od parametara senzora ili Markovljeve lokalizacije tada se mogu dodatno podestiti u Saphiri pomoću padajućih izbornika.

Pri gibanju stvarnog robota ovim eksperimentom uočeno je vrlo dobro praćenje položaja robota u statičkim kao i dinamičkim okolinama (ljudi koji su šetali po zavodu). Javljali su se malo veći problemi dok je robot dolazio u područje u kojem je karta prostora imala relativno veliko odstupanje od stvarnog prostora pa se Markovljeva lokalizacija redovito rušila.

Markovljeva lokalizacija pokazala se robusnom za uobičajene promjene okoline kao što su otvorena/zatvorena vrata ili ljudi koji prolaze. Nažalost, lokalizacija ne uspijeva u slučajevima kad puno aspekata okoline nije pokriveno s modelom prostora. To se događa kada je okolina napunjena ljudima koji pokrivaju senzore robota i tako dolazi do puno nepredviđenih mjerenja.

Jedno od gibanja robota po zavodu upravljanog pomoću tipkovnice, prikazano je *slikom 3.9.* Parametri Markovljeve lokalizacije bili su sljedeći: pojačanje 50 %, broj ML točaka 2000, osvježavanje pozicije obavljalo se svakih 1 m ili 20°, veličina kvadrata u koji su se postavljali uzorci iznosila je 40 cm, a razlika u kutu unutar kojeg su se postavljali uzorci 20 °.



Slika 4.9. Kretanje robota po zavodu za APR tijekom Markovljeve lokalizacije

5. GRADIJENTNI POSTUPAK

Usprkos mnogim desetljećima istraživanja u mobilnoj robotici, pouzdano upravljanje uz zadovoljavajuću brzinu u nesigurnim i kompliciranim okolinama ostaje nepostignuti cilj. Ovdje ćemo predstaviti jedno rješenje za upravljanje robotom u stvarnom vremenu pomoću kojeg će robot postizati optimalnu brzinu čak i kada istražuje nova područja ili nailazi na neočekivane prepreke. Ova metoda koristi navigacijsku funkciju za generiranje gradijentnog polja koje predstavlja optimalnu stazu, stazu s najmanjim troškom, do cilja iz bilo koje točke u prostoru. Dodatno ćemo prikazati i integriranu fuziju senzora sustava koji omogućuje inkrementalnu izgradnju nepoznate okoline.

5.1. Uvod

Upravljanje gibanjem mobilnih robota općenito se može podijeliti u tri kategorije:

1. Planiranje gibanja. Generira se kompletna staza od robota do cilja i robot je slijedi.
2. Procjenjivanje lokalne staze. Proračunava se nekoliko staza u neposrednoj blizini robota da bi se provjerilo sadrže li prepreke.
3. Reakcija. Robot bira smjer u kojem će se gibati, a on se zasniva na trenutnim informacijama o preprekama i smjeru cilja.

Većina današnjih kontrolera su kombinacija ovih tehnika. Oni koriste planiranje gibanja za globalnu stazu i druge tehnike za provjeru nepoznatog prostora ili objekta. U metodama za lokalne staze, kontroler za planiranje gibanja generira točke putanje, a gibanje prema tim točkama određuje se provjeravanjem skupa putanja koje se nalaze u neposrednoj blizini robota da ne bi došlo do sudara.

U ovom dijelu predstavit ćemo gradijentnu metodu lokalne navigacije koja kontinuirano proračunava optimalnu stazu do cilja prevladavajući ograničenja današnjih lokalnih metoda kao što je naprimjer zaglavljivanje u lokalnim minimumima. Ono što se proračunava je navigacijska funkcija u lokalnom prostoru robota, takva da gradijent navigacijske funkcije predstavlja smjer staze s najmanjim troškom iz bilo koje točke u prostoru. Metoda je dovoljno efikasna da bi se proračun mogao obavljati frekvencijom 10 Hz s najsromnijim računalnim resursima.

Sama po sebi, gradijentna metoda može generirati najoptimalniju stazu u statičkim i potpuno poznatim okolinama, ali u mnogim situacijama nalaze se komplicirani gibajući objekti (namještaj, vrata, ljudi) čiji položaj je nesiguran ili nepoznat. U drugom dijelu predstavit ćemo algoritam fuzije podataka senzora za gradnju lokalnog percepcijskog prostora (LPS) robota u koji se nove informacije stalno dodaju. Zajedno s gradijentnom metodom, omogućuje robotu efikasno istraživanje lokalnog područja i pronalaženje staze do cilja.

5.2. Navigacijska funkcija za optimalnu stazu

U ovom odjeljku razmatramo generaciju najoptimalnije staze do cilja u konfiguracijskom prostoru robota. Prikazujemo kako napraviti navigacijsku funkciju koja pridružuje potencijalno polje vrijednosti svakoj točki u prostoru. Putovanjem po gradijentu navigacijskog potencijala postiže se najoptimalnija staza do cilja.

Za nastavak, pretpostavit ćemo da je konfiguracijski prostor robota uniformno (jednolično) diskretiziran, ali nije nužno da bude tako. Vrijednosti navigacijske funkcije u proizvoljnim ne uzorkovanim trenucima mogu se proračunati interpolacijom uzorkovanih točaka. Cilj je skup uzorkovanih točaka.

5.2.1. Trošak staze

Željeli bismo pronaći stazu s minimalnim troškom do neke od skupa ciljnih točaka. Stoga predstavimo stazu s uređenim skupom točaka u uzorkovanom prostoru:

$$P = \{P_1, P_2, \ldots\} \tag{5-1}$$

gdje su to susjedne točke (dijagonalno ili pravocrtno) i niti jedna od njih se ne ponavlja. Posljednja točka na putanji mora biti ciljna točka i niti jedna druga ne smije biti ciljna. Skraćeno ćemo označavati stazu koja započinje s točkom k sa P_k.

U općem slučaju trošak staze $F(P)$ je proizvoljna funkcija diskretizirane putanje. S takvom funkcijom je teško raditi pa ćemo pretpostaviti da se trošak staze može podijeliti u sume *neophodnih troškova* i*okolnih troškova* gibanja od jedne točke do druge:

$$F(P) = \sum_i I(p_i) + \sum_i A(p_i, p_{i+1}) \tag{5-2}$$

I i A mogu biti proizvoljne funkcije. Za uobičajene situacije I će predstavljati trošak prelaska preko zadane točke i postavit će se prema karakteristikama područja, npr. visoki trošak pridružit će se blizini prepreke. Sljedeće pretpostavke bi trebale uključivati više troškove za nepoznata područja ili npr. klizava područja.

Duljina staze može se uzeti u obzir na način da uzmemo da A bude proporcionalno Euklidskoj udaljenosti puta robota između dvije točke. Nakon toga sve sume po A daju trošak proporcionalan duljini staze.

5.2.2. Navigacijska funkcija

Navigacijsku funkciju N možemo zamisliti kao pridruživanje potencijalne vrijednosti polja svakom elementu konfiguracijskog prostora na takav način da je ciljna točka uvijek negdje na "nizbrdici" bez obzira gdje se robot nalazio u prostoru. Za navigacijske funkcije, za razliku od metoda s potencijalnim poljima, karakteristično je da se ne mogu zaglaviti u lokalnom minimumu i nisu potrebna nikakva pretraživanja za određivanje smjera u kojem je potrebno ići da bi se stiglo do cilja.

Ključne ideja gradijentne metode jest da se navigacijskoj funkciji u točki pridruži staza s minimalnim troškom koja započinje u toj točki. Matematički,

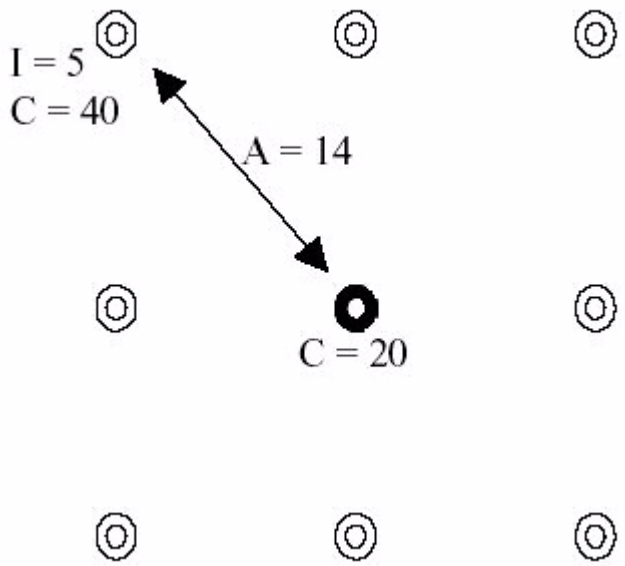
$$N_k = \min_{P_k} F(P_k) \tag{5-3}$$

gdje, kao i prije, P_k započinje u točki k.

Ako su *neophodni troškovi* jednaki nuli tada navigacijska funkcija predstavlja upravo udaljenost do najbliže ciljne točke. Putovanje u smjeru gradijenta N postiže najveću redukciju troška staze, tj. minimalnu udaljenost do ciljne točke.

5.2.3. Algoritam linearnog planiranja (LPN)

Izračunavanje vrijednosti *jednadžbe (4-3)* u svakoj točki je komplicirano jer veličina konfiguracijskog prostora može biti velika i pronalazak staze s najmanjim troškom uključuje iteraciju preko svih odlazećih staza od te točke. Ovdje se nećemo pozabaviti s rješavanjem problema veličine konfiguracijskog prostora, a i gradijentna metoda je pogodna samo za mali broj dimenzija. Pretpostavit ćemo da se robot giba na podu, da je okruglog oblika i da se može okrenuti unutar vlastitog radijusa tako da će jednostavni XY koordinatni sustav biti dovoljan. Proračun koji ćemo ovdje opisati može se obaviti za proizvoljni prostor, ali nije praktičan za dimenzije veće od 3 ili 4. Koristit ćemo algoritam linearnog programiranja zvan LPN.

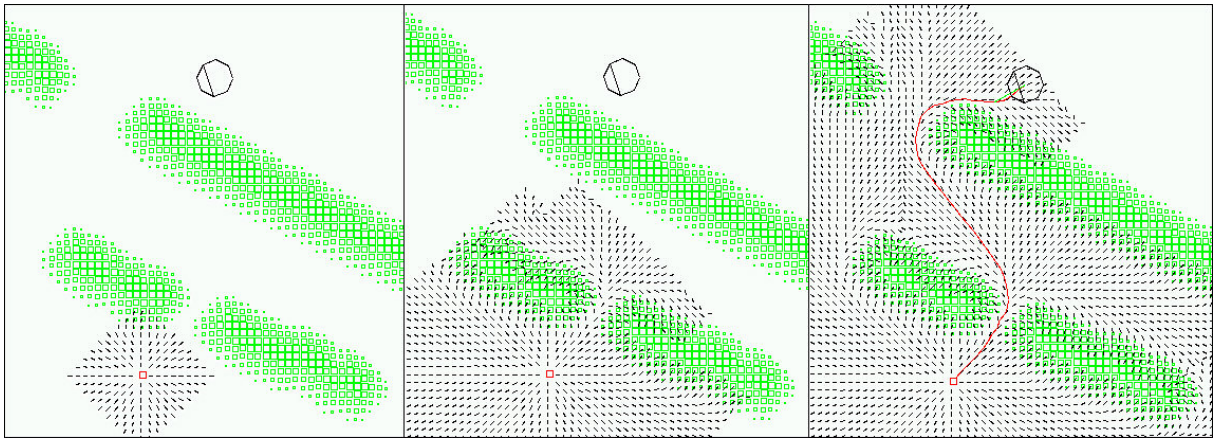


Slika 5.1. Osvježavanje troška točke u okolini aktivne točke

Inicijalno ćemo svim ciljnim točkama pridružiti vrijednost 0, a svim ostalim točkama vrijednost ∞. Zatim se sve ciljne točke stavljaju na *aktivnu listu* točaka. U svakoj iteraciji algoritma, operiramo na svakoj točki *aktivne liste*, zatim je mičemo s liste osvježavajući njezine susjedne točke. Razmotrimo proizvoljnu točku p kojoj smo upravo pridružili vrijednost. Ta točka je okružena s 8 susjednih točaka u pravilnoj mreži prikazanoj na *slici 4.1.* Za svaku od susjednih točaka q, također možemo izračunati trošak s ciljem da produžimo stazu od točke p do ove točke koristeći *jednadžbu (4-2)*. Ako je novi trošak do točke q manji od vlastitog troška, tada ga zamijenimo s novim i dodamo q na novu listu aktivnih točaka. Npr. na *slici 4.1* novi trošak dijagonalne susjedne točke iznosi 20+14+5 = 39, što je manje od vlastitog troška, stoga se ta nova točka dodaje na aktivnu listu. Proces se ponavlja dok aktivna lista ne postane prazna. Moguće je pokazati da se pomoću LPN algoritma proračunava staza s najmanjim troškom prema svakoj točki u prostoru.

Teorem 1: U svakoj točki prostora gradijent navigacijske funkcije proračunat iz LPN, ako postoji, usmjeren je u smjeru staze s najmanjim troškom prema cilju.

Slika 4.2 prikazuje LPN algoritam u nekoliko različitih faza počevši s jednom ciljnom točkom. Pravokutnici predstavljaju neophodni trošak jedne točke uzorkovanja gdje veliki pravokutnik predstavlja beskonačni trošak. Trošak se izračunava uz pomoć prepreka što će biti kasnije opisano. Smjer gradijenta u svakoj točki prikazan je crnom kratkom linijom. Tri slike prikazane su za slučajeve od 10, 30 i 70 iteracija. Interpolirana staza od robota do cilja prikazana je na posljednjoj slici.



Slika 5.2. Tri faze LPN algoritma s jednom ciljnom točkom za slučajeve 10, 30 i 70 iteracija

Navigacijska funkcija je implicitno opisana smjerom gradijenta u svakoj točki, na slici označenim s kratkim linijama. U slučajevima gdje je navigacijska funkcija uniformna, što je slučaj kod okolnih prepreka, gradijent nije definiran. Može se primijetiti kako se gradijentne točke jako odmiču od unutrašnjosti prepreka. Još jedna zanimljiva značajka navigacijske funkcije jest prisutnost *grebena*, točaka u kojima gradijent ima jednak iznos za različite smjerove. *Grebeni* predstavljaju točke izbora u pronalasku staze s minimalnim troškom. Bilo koji smjer može biti odabran jer svaki vodi do staze s istim minimalnim troškom. U trećem okviru na *slici 4.2* može se vidjeti *greben*, koji se nalazi odmah iznad ciljne točke, u kojem je svejedno ide li se s lijeve ili s desne strane prepreke.

Staza s minimalnim troškom prikazana je u trećem okviru *slike 4.2*. Tu stazu nije teško pronaći jednom kada je gradijentna funkcija poznata. Počevši od robota potrebno se pomaknuti za kratku udaljenost u smjeru gradijenta i u novoj točki ponaći gradijent interpolacijom, nakon čega se opet potrebno pomaknuti. Taj postupak se ponavlja. Može se uočiti da iako je na *slici 4.2* prikazan gradijent u svim točkama prostora, potrebno ga je samo računati širom staze kojom se giba robot.

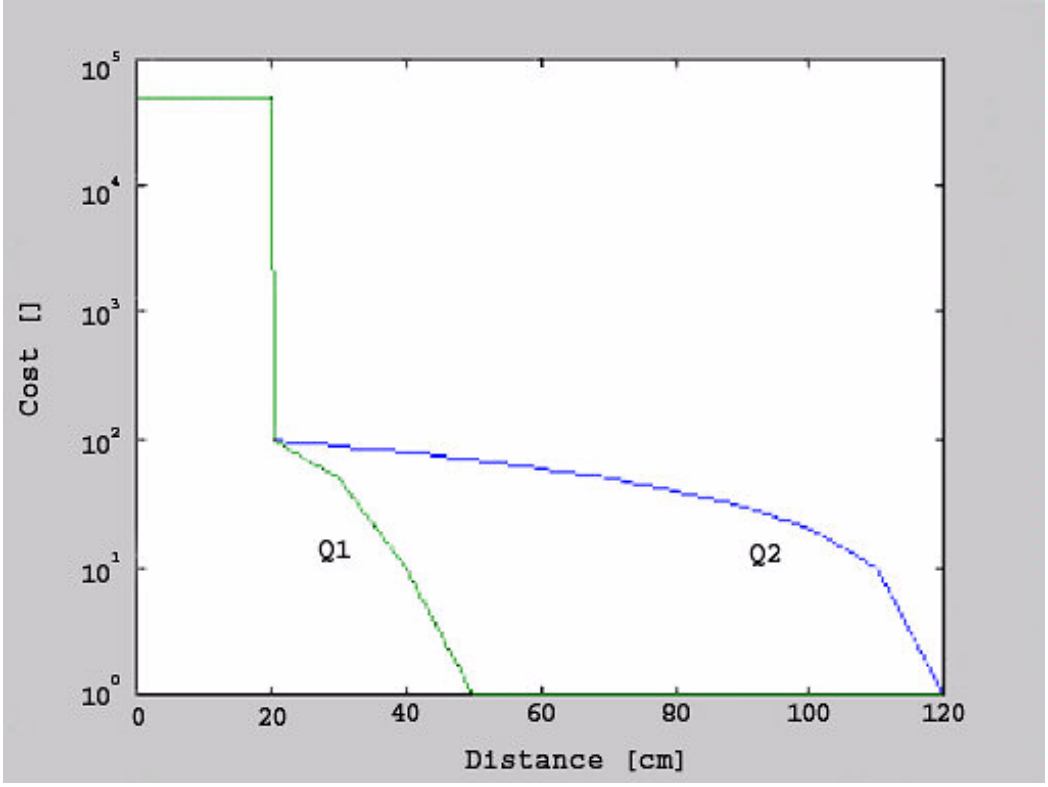
5.2.3. Troškovi prepreka

LPN algoritam izračunava općenitu formu navigacijske funkcije koja minimizira trošak staze i tako se ponašanje oko prepreka može kontrolirati pridruživanjem *neophodnih* troškova na sljedeći način.

Pretpostavimo da su prepreke u prostoru dane skupom točaka koje sačinjavaju prepreke. Neka *d(p)* označuje udaljenost od točke *p* do najbliže točke prepreke. Tada možemo napisati:

$$I(p) = Q(d(p)) \qquad (5\text{-}4)$$

gdje je *Q* funkcija koja se smanjuje. Na *slici 4.3* prikazane su dvije *Q* funkcije. Obje imaju jako velik trošak na udaljenosti od 20 cm od bilo koje prepreke, što označava radijus robota. U funkciji *Q1* nakon ove točke trošak naglo opada s udaljenosti, dok kod funkcije *Q2* opada znatno sporije. Tako će npr. *Q1* puštati robotu da se bliže približi prepreci. *Neophodni trošak* za *sliku 4.2.* izračunat je pomoću funkcije *Q1*.



Slika 5.3. Dva primjera funkcija neophodnog troška prepreka

Može se uočiti da funkcija *neophodnog troška Q2* ne dopušta prolaz robotu kroz uske dijelove, nego čini da dulja staza koja zaobilazi prepreke bude staza s najmanjim troškom.

U praksi se *neophodni troškovi* mogu pridružiti na sljedeći način. Neka je dan skup točaka prepreke i neka je svaka točka označena udaljenošću do druge najbliže točke prepreke koristeći LPN algoritam. Cilj predstavimo skupom točaka prepreke. Sve *neophodne troškove* postavimo na vrijednost 0 i sve *okolne troškove* na *euklidsku udaljenost*. U tom slučaju *Q* funkcija pretvara udaljenosti svake točke u *neophodne troškove*.

5.2.4. Vremensko usklađivanje

Vremenski zahtjevi LPN algoritma su proporcionalni broju točaka u prostoru. Moraju se obaviti dva LPN proračuna: jedan za *troškove prepreka* i drugi za navigacijsku funkciju. U većini slučajeva proračun troška prepreke može se zaustaviti nakon nekoliko iteracija, a navigacijska funkcija dominira potrošnjom vremena za proračun.

Na skromnom računalu PC 266 MHz Pentium, C implementacija zahtijeva prosječno 1 µs po jednoj točki prostora. Za prostor od 10 x 10 m s veličinom mreže od 10 cm, potrebno je 10 ms za proračun navigacijske funkcije. Tako je LPN algoritam pogodan za upravljanje u stvarnom vremenu. Tipično je da se odvija u koracima od 10 Hz što je dovoljno brzo da bi se robot mogao efikasno gibati brzinom od 1 m/s.

5.3. Modeliranje okoline

Da bi se mogao efikasno gibati, robot mora na raspolaganju imati dobar model okoline. U tom pogledu najinteresantnija su ona područja za koja robot ima djelomične informacije o položaju trajnih objekata (zidova, štokova od vrata), zatim ona u kojima postoje pokretni statički objekti (stolice, stolovi, vrata) te dinamički objekti (ljudi ili drugi roboti).

Radijacijski senzori s robota daju samo djelomične informacije o objektima u okolini s relativno velikom nesigurnošću. Npr. robot može znati da u prostoriji u koju je ušao postoje vrata s druge strane, ali ih možda ne može vidjeti jer su blokirana s ostalim objektima u prostoru. Robot može početi pretraživati položaj vrata, ali ima ograničen pogled tako da je samo manji dio sobe u svakom trenutku pod aktivnom pretragom. Ako robot zaboravi što je upravo vidio tada može misliti da se vrata nalaze u području koje je upravo pretražio.

5.3.1. Lokalni percepcijski prostor (LPS)

Jedna od solucija za rješavanje problema ograničenog pogleda robota jest da se objedine (fuzioniraju) informacije senzora dok se robot giba u povezanom geometrijskom prostoru. Takav bi LPS davao sveobuhvatan pogled trenutne okoline robota. Ako je okolina statička i robot može savršeno pratiti svoj položaj, tada će LPS savršeno izgraditi složenu sliku okoline u granicama pogrešaka senzora robota. U tipičnom slučaju narušene su obje pretpostavke tako da je LPS valjan samo u prostorno malom intervalu.

Gradijentna metoda i LPS modeliranje okoline napravljeni su da prevladaju probleme zaglavljivanja u lokalnim minimumima. Može se pokazati da će u statičkim okolinama ovim metodama robot uvijek stići do cilja.

Teorem 2: Ako senzori i odometrija robota nemaju pogrešku, okolina je statična i postoji staza do cilja unutar opsega koji obuhvaća LPS, tada će robot ako slijedi gradijentnu trajektoriju najzad stići na cilj.

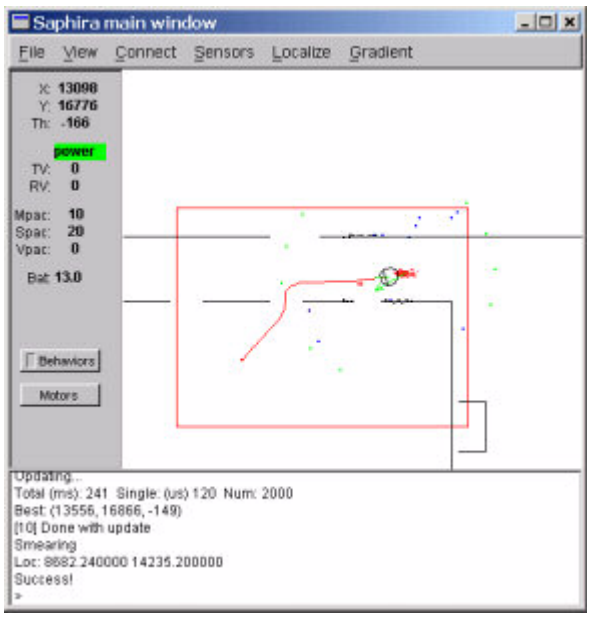
Očito je da niti jedan stvarni robot nije savršen i može se dogoditi da stazu do cilja nije moguće pronaći, npr. ako senzori pogrešno dojave da su vrata zatvorena. No, današnji senzori na robotima su već dosta precizni, a odometrijska točnost je jako dobra ako se pogreška zakreta dodatno ispravlja žiroskopima.

5.4. Opis gradijentnog modula

Za uspješno gibanje bazirano na lokalnim preprekama i kartama prostora, Saphira sadrži modul za planiranje trajektorije u stvarnom vremenu koji se zasniva na gradijentnoj metodi [9]. Za planiranje staze i gibanje po karti gradijentni postupak se koristi zajedno s Markovljevom lokalizacijom da bi robot mogao znati gdje se na karti nalazi dok se giba. Gradijentni modul je pakiran i kao aplikacija za demonstraciju sposobnosti robota i kao API za pozivanje gradijentnog postupka iz Saphira klijenta.

Planiranje staze gradijentnim postupkom je proces određivanja optimalne staze robota u stvarnom vremenu. Ove staze mogu uzeti u obzir i lokalne prepreke, registrirane sonarima ili laserom, i informacije s globalne karte kao što je lokacija zidova i ostalih strukturnih prepreka.

U svakom ciklusu sinkronizacije od *100 ms* gradijentni modul računa najoptimalniju stazu od robota do ciljne točke. Algoritam započinje pregledavajući najbližu okolinu povezujući robota s ciljnom točkom, a nakon toga priširuje područje pretraživanja ako nije pronađena niti jedna staza. Granice okoline mogu biti postavljene određenom naredbom od strane korisnika.



Slika 5.4. Kretanje robota s gradijentnim modulom

Crveno područje označava skup uzoraka Markovljeve lokalizacije koja je također učitana. Crvena linija koja povezuje robota s ciljnom točkom je optimalna staza izračunata gradijentnim algoritmom tijekom trenutnog vremenskog koraka. Crveni pravokutnik označava okolinu za gradijentnu metodu. Primjetite da staza ide oko zidova iako je robot ne može vidjeti.

Gradijentni postupak za određivanje optimalne staze koristi kvadratnu mrežu. Rezolucija mreže se može postaviti na tipično *10 mm*, a može se postaviti i maksimalna veličina mreže.

Staza se proračunava iz skupa prepreka koje se nalaze na karti ili su dobivene iz očitanja senzora. Evo nekoliko tipičnih izvora prepreka:

1. Linijski objekti na WLD karti. Potrebno je učitati kartu i pokrenuti naredbu *sfGradUseArtifacts(true)*.

2. Objekti na MAP karti stvorenoj pomoću laserskog mapiranja. Ova karta se učitava u lokalizacijski modul naredbom *mcLoadScanMap()*, a da bi joj se pristupilo iz gradijentnog modula mora se pokrenuti naredba *mcGetObject*. Na kraju je potrebno omogućiti korištenje karte naredbom *sfGradUseMap(true)*

   MAP karta i WLD karta mogu se koristiti istovremeno.

3. Očitanja s lasera ili sonara koja se mogu uključiti ili isključiti naredbama *sfGradUseLaser()* i *sfGradUseSonar*.

Za izbjegavanje prepreka može se koristiti sigurnosno područje koje se može postaviti naredbom *sfGradObsParams()*.

Postoje lokalni kontroleri implementirani kao akcija (*SfGradAction*) koji voze robota po gradijentnoj stazi. Ova akcija kontrolira brzinu gibanja robota koja se može postaviti naredbom *sfGradSetSpeed*. Daljnja finoća za robote kvadratnog oblika dobije se mogućnošću da se giba unatrag kada je to potrebno, a kontrolira se naredbama *sfGradSetCanBack* i *sfGradSetTurnRadius*. Ako se robot treba zakrenuti za više od 45° pogledat će je li njegov polumjer zakretanja slobodan. Ako nije slobodan tada će se probati pomaknuti unatrag ako je moguće i okrenuti se čim to bude moguće.

5.4.1. Učitavanje gradijentnog modula

Ključne datoteke za gradijentni algoritam prikazane su u sljedećoj tablici.

Ime datoteke	**Opis**
`lib/sfGrad.so` `bin\sfGrad.dll`	Ključne datoteke s rutinama za API sučelje za gradijentni algoritam s laserskim senzorom ili sonarima.
`lib/sfGradF1.so` `bin\sfGradF1.dll`	Datoteke za grafičko sučelje. Dodaju padajuće izbornike i prozore za mijenjanje performansi gradijentnog postupka.
`colbert/flgrad.act`	Ovom datotekom se inicijalizira gradijentni postupak bilo za laserski senzor ili sonare ili oboje.
`colbert/scan.act`	Koristi se za inicijalizaciju ML s laserskim senzorima, učitavanje karte napravljene u *ScanStudiu* i gradijentnog modula.

Tablica 5.1. Datoteke koje se koriste pri pokretanju gradijentnog algoritma

Normalno se gradijentni postupak pokreće zajedno s kartom i lokalizacijskim postupkom iako se može pokrenuti i zasebno za izbjegavanje prepreka.

Najjednostavniji način pokretanja jest da se učita datoteka flloc.act, koristeći Colbert naredbu *load flloc*. Ona će učitati kartu *aic.wld*, ključne datoteke za Markovljevu lokalizaciju i grafičko sučelje, te pokrenuti ML proces. Nakon toga se učitaju ključne datoteke za gradijentni modul pomoću datoteke *flgrad.act* s Colbert naredbom *load flgrad*. Zatim je potrebno pokrenuti simulator robota i u njega učitati istu *aic.wld* kartu te se sa Saphirom spojiti na njega. Nakon toga bi crveni skup točaka trebao slijediti ispravnu stvarnu poziciju

robota. S padajućeg izbornika odabere se opcija "Localize->Update Position" i Saphira klijent bi trebao pratiti položaj robota. Nakon toga potrebno je natjerati gradijentni postupak da obrati pažnju na zidove i ostale objekte na karti što se može napraviti pomoću opcije "Gradient->Use Artifacts"

U ovom trenutku aplikacija je spremna za pokretanje. Ciljne točke se postavljaju s klikom miša na lijevi gumb i pri tom držeći tipku SHIFT u Saphira grafičkom prozoru (prije toga je potrebno samo jednom kliknuti unutar tog prozora da bi postao aktivan). Tada bi se trebala vidjeti staza kao na *slici 4.4.* Ako je spojen robot ili simulator i ako su uključeni *Behaviors* (tipka sa signalnom lampicom na lijevoj strani glavnog prozora Saphire) tada bi se robot trebao početi gibati prema ciljnoj točki. Gradijentni modul uključuje programe koji izračunavaju optimalnu brzinu tijekom gibanja po stazi tako da robot usporava u naglim zavojima i ubrzava na ravnim dijelovima.

Ciljna točka može biti promijenjena u bilo kojem trenutku s klikom miša uz tipku SHIFT u grafičkom prozoru Saphire. Staza će se promijeniti istog trenutka.

5.4.2. Opis gradijentnog padajućeg izbornika

Koristeći gradijentni padajući izbornik može se utjecati na nekoliko aspekata prikaza i performansi gradijentnog postupka.

- "Gradient->Display" – uključuje ili isključuje prikaz staze, cilja, akceleracije ili gradijenta.
- "Gradient->Left Click Goes To Goal " – omogućuje da se klikom na lijevi gumb miša odredi ciljna točka.
- "Gradient->Use Sonars" – uključuje ili isključuje senzore sonara za izbjegavanje prepreka.
- "Gradient->Use LRF" – uključuje ili isključuje laserski senzor za izbjegavanje prepreka.
- "Gradient->Use Artifacts" – omogućuje korištenje WLD karte.
- "Gradient->Use Scan Map" – omogućuje korištenje MAP karte.

5.5. Eksperimentalni rezultati

5.5.1. Provjera rada gradijentnog modula

Prije odrađivanja stvarnog pokusa, sve korake isprobat ćemo na simulatoru robota.

Prvo je potrebno otići u mapu $SAPHIRA/colbert i preurediti datoteku s funkcijama za učitavanje gradijentnog modula *scan.act*. Podatke je potrebno pohraniti u novu datoteku s nazivom *gradient.act*. Preduvjet koji smo trebali ispuniti prije preuređivanja te datoteke jest da smo pri instalaciji ispravno podesili sustavsku varijablu SAPHIRA_LOAD za koju smo stavili vrijednost "C:\Program Files\Saphira" (u Windows * OS-u) ili "/usr/local/Saphira" (u Linux OS-u).

U datoteci *gradient.act* nalaze se funkcije za inicijalizaciju gradijentnog i laserskog modula, učitavaju se inicijalizacijske datoteke, inicijaliziraju se i pokreću sonari i laser te im se zadaju odgovarajući parametri. Također se inicijalizira i učitava MAP karta te WLD karta koja u ovom slučaju služi kao ograničenje za područja u kojima ne želimo robotu dopustiti da se giba. Nakon učitanja WLD datoteke potrebno je pozvati naredbu *sfGradUseArtifacts(1)* koja će učiniti da gradijentni postupak izbjegava linije koje smo postavili u WLD datoteku, s tim da neće utjecati na (ometati) lokalizaciju robota. U WLD datoteci dodaju se linije koje ne postoje na karti, a nalaze se u blizini objekata ili mjesta blizu kojih ne želimo da robot dođe. Izvorni kod datoteke *gradient.act* dan je u *dodatku 10*.

Kad su sve potrebne funkcije za rad s gradijentnim modulom upisane u datoteku *gradient.act*, potrebno je pokrenuti Saphiru i spojiti se na simulator robota ili na stvarni robot. Prije spajanja na stvarni robot potrebno ga je pozicionirati na mjesto s kojeg smo ga pokretali kad smo radili mapiranje (Home Position). Nakon toga je potrebno učitati iz Colbert interaktivnog prozora datoteku *gradient.act* naredbom `load gradient.act`. Navedena datoteka će pri učitavanju pokrenuti lokalizaciju i gradijentne alate te će postaviti odgovarajuće izbornike u Saphiru.

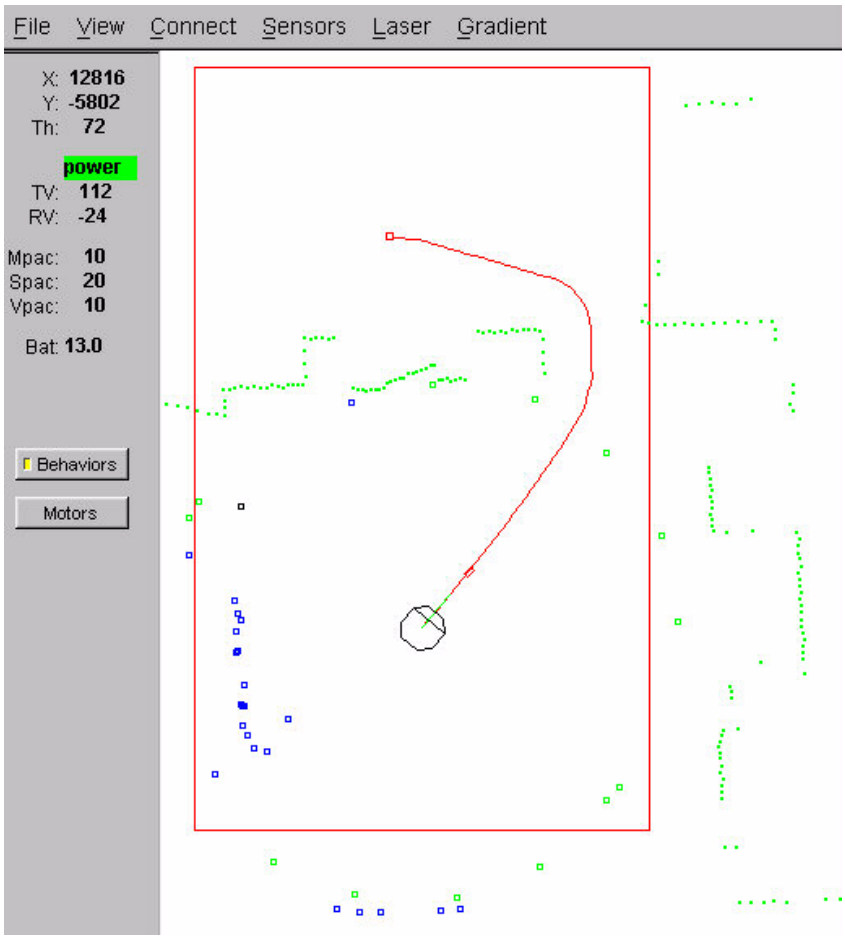
Uspješnost učitavanja može se provjeriti dojavnim naredbama u interaktivnom tekstualnom prozoru Saphire. U Saphirinom LPS prozoru bi se tada trebala pojaviti slika robota s očitanjima lasera i sonara. Napomenut ćemo samo da je, kao i u ML eksperimentima na stvarnom robotu, potrebno pričekati kad se učita datoteka *gradient.act* da se laser spoji na server robota. To traje nekih 45 do 60 sekundi i za to vrijeme se ne smije ništa raditi niti u Saphiri niti na robotu. Nakon spajanja lasera potrebno je još samo omogućiti rad motora naredbom (tipkom) "Enable Motors" (nije potrebno ako se spajamo na simulator) s lijeve strane prozora i kliknuti u prozor Saphire na određeno (ciljno) mjesto na karti držeći pri tom tipku SHIFT. Robot bi prema ciljnom mjestu na karti trebao isplanirati putanju i odvesti se do tog mjesta po proračunatoj putanji.

Ako robot pri gibanju zaglavi može se pomaknuti s mjesta ručno, pomoću tipkovnice ili naredaba u interaktivnom prozoru Saphire.

Pri eksperimentima za provjeru rada gradijentne metode na zavodu robot se prema očekivanjima dobro gibao po optimalnoj stazi maksimalno dopuštenom ograničenom brzinom. Gibanje robota P2DX prikazano je *slikom 4.5*. Ograničenja translacijske i rotacijske brzine prvo smo ograničavali pomoću naredbe `setMaxTransVel(int Vel)` u datoteci *mySaphira.cpp*, a zatim pomoću naredbe `sfGradSetSpeed(int high, int mid, int back)` u datoteci *gradient.act*. Pomoću druge naredbe mogu se zadati ograničenja brzina u

različitim okolnostima: parametar *high* označava brzinu gibanja robota kada na putu nema nikakvih prepreka, parametar *mid*brzinu kada je prostor prenatrpan predmetima ili su u blizini prepreke, a parametar *back* maksimalnu brzinu za putovanje unatrag (u tom slučaju gibanje unatrag mora biti omogućeno naredbom `sfGradSetCanBack(int on)`).

Pošto je ovo bio eksperiment samo za provjeru rada gradijentnog modula nisam isprobavao brzinu gibanja robota u slučajevima kada bi robot trebao stići do istog cilja kao i gradijentnom metodom, ali upravljan pomoću palice za igru. Robot P2DX na zavodu za APR je preskup da bi se s njim igrale igre nadmetanja čovjeka i gradijentne metode. Za očekivati je da bi gradijentna metoda nadmašila ljudsko planiranje u svim uobičajenim slučajevima. Iskusni eksperti su pri ovakvim pokusima doživljavali prosječno jedan sudar robota s preprekom u tri gibanja upravljanog palicom za igru pa se za rezultate možemo osloniti na njihove eksperimente.



Slika 5.5. Kretanje robota s gradijentnom metodom po optimalnoj stazi

5.5.2. Eksperimenti Kurt Konoligea

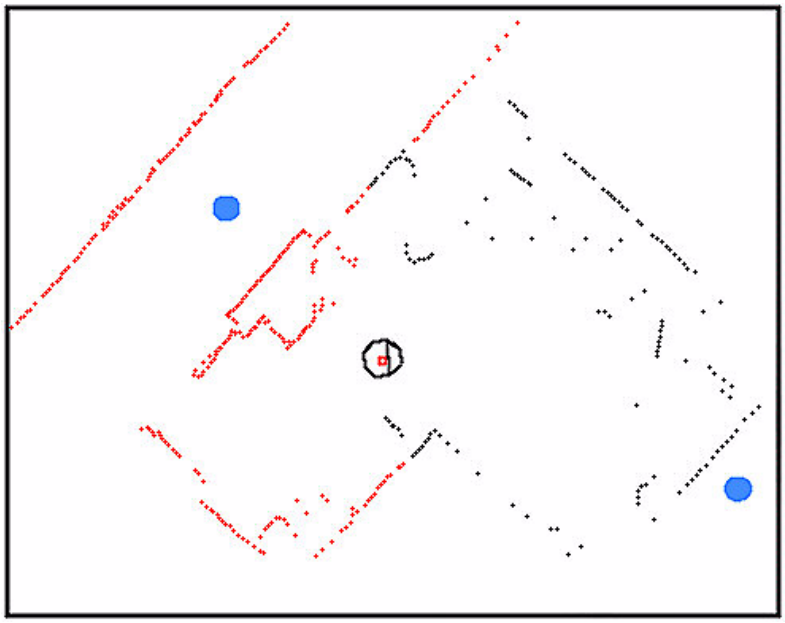
Kurt Konolige je sa svojom ekipom proveo je manji broj eksperimenata da bi provjerio valjanost gradijentne metode i pripadajuće LPS konstrukcije prostora. Ovi eksperimenti su korisn iako nisu statistički obrađeni. Dva glavna pitanja koja su se htjela provjeriti ovim eksperimentima su:

1. Kako dobro radi gradijentna metoda u odnosu na operatere koji upravljaju gibanjem robota u nepoznatoj okolini.

2. Kako dobro radi gradijentna metoda u odnosu na operatere koji upravljaju gibanjem robota u poznatoj okolini.

Očekuje se da rezultati budu različiti jer i ljudi i roboti mogu optimizirati upravljanje kada je okolina unaprijed poznata. Prvi eksperiment se više usmjeruje na efikasno istraživanje prostora dok drugi pridaje veću težinu brzini upravljanja robotom.

Osnovni zadatak sastoji se u tome da se upravlja gibanjem robota od početne do ciljne točke s udaljenosti od kojih 10 m što je brže moguće (*slika 4.6*).



Slika 5.6. Gibanje robota od jedne do druge ciljne točke tijekom eksperimenta

Optimalna staza prolazi kroz veliku sobu s kliznim vratima. Soba sadrži brojne, za senzore zbunjujuće prepreke, kao što su stolice, stolovi i ostali namještaj te povremeno ljudi u prolazu. Iste informacije su dane na raspolaganje i operateru i robotu preko LPS prikaza okoline na ekranu.

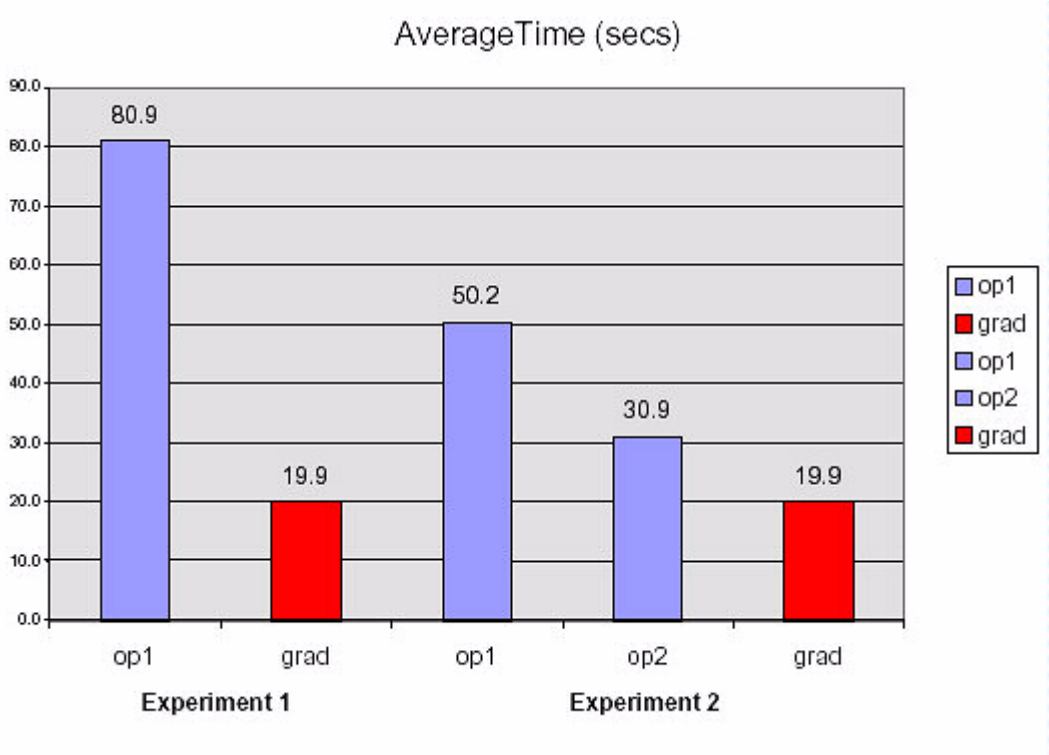
U prvom eksperimentu robot je pri pokretanju s početne pozicije bio okrenut na drugu stranu od sobe, tako da niti jedan dio okoline između početne i ciljne pozicije nije bio vidljiv. I početna i ciljna točka su bile prisutne u LPS-u cijelo vrijeme tako da su i operater i robot mogli vidjeti kuda bi se robot trebao gibati. Vrlo je teško dobiti podatke bez pomaka koji bi bili dovoljno dobri za usporedbu rezultata jer se robot svaki put mora staviti u novu okolinu.

U drugom eksperimentu robot je opet bio pozicioniran na sličan način, ali su očitanja senzora od prošlog gibanja ostala sačuvana u robotu tako da je bilo jednostavno vidjeti kuda se robot trebao gibati. Na ovaj način snimljeno je nekoliko gibanja operatera i robota.

Operater je upravljao robotom pomoću palice za igru tako da je imao mogućnost upravljanja brzinom i smjerom robota. Eksperiment su obavljali dvojica operatera. Jedan je imao solidno znanje i sposobnosti o upravljanju robotom dok je drugi bio ekspert. U svim eksperimentima bile su fiksirane kutna i translacijska akceleracija robota, a maksimalna brzina bila je postavljena na 1 m/s.

U prvom eksperimentu bilo je jednostavno upravljati robotom nekoliko puta da bi se dobili statistički podaci, ali je bilo nemoguće izbrisati memoriju (pamćenje) operatera. Prikazani

rezultati prikazuju samo jedno upravljanje operatera u ovom eksperimentu (radi se o operateru koji nije ekspert).



Slika 5.7. Prosječno vrijeme gibanja od početne do ciljne točke za oba eksperimenta

Vrijeme je ovdje osnovna performansa za razmatranje. U prvom eksperimentu operater je imao poteškoća u odlučivanju kojim bi smjerom robota trebalo upravljati, a u jednom trenutku se robot i sudario s vratima dok je operater pokušavao manevrirati oko njih. To vrijeme je 4 puta lošije od automatskog gradijentnog algoritma. U drugom primjeru su se operateri dosta popravili, no svejedno su za klasu bili lošiji od gradijentnog algoritma. U svim eksperimentima rezultati su bili nepromjenjivi (konzistentni) s varijancama manjim od 2 sekunde.

Ne-sudaranje s robotom je vrlo važan zadatak koji treba odraditi. U prvom eksperimentu s operaterom robot se pri prolazu kroz vrata sudario u njih. U drugom eksperimentu bio je prosječno jedan sudar u 3 gibanja.

Pod svim uvjetima i u svim kategorijama gradijentni algoritam je zadatak obavio znatno bolje od ljudskih operatera, čak i u slučajevima kad su oni bili eksperti za upravljanje robotom te uvježbani u specifičnom prostoru. Razlog tomu je što robot prvo proračuna stazu do ciljne pozicije i kada ima stazu tada se može skoncentrirati samo na proračun brzine kojom se mora gibati u svakoj točki staze da bi do cilja stigao u najkraćem vremenu, osvježavajući pri tom svoju brzinu svakih 100 ms. On može također precizno upravljati brzinom kojom prilazi preprekama tako da se može gibati velikom brzinom dok ne dođe blizu prepreke kao što su vrata i tada usporiti da bi prošao kroz njih.

S druge strane, iako operateri imaju neke bolje sposobnosti zapažanja od robota, oni imaju poteškoća s procjenjivanjem brzine kojom bi se robot trebao gibati, kada dođe do kompliciranog manevriranja kao što je zaobilaženje kutova zidova. Izbor koji je ljudima nametnut je ili da uspore i odigraju na sigurno ili da idu brzo i riskiraju sudar.

6. ZAKLJUČAK

Za autonomnu navigaciju mobilnog robota kroz prostor, navigacijski sustav robota mora u svakom trenutku znati položaj robota u prostoru s velikom točnošću. Algoritmi rješavanja ovog problema uglavnom se zasnivaju na fuziji informacija percepcijskih i propriocepcijskih senzora robota. U tom smislu u ovom je radu detaljno istražena Markovljeva lokalizacija koja estimira položaj robota u njegovoj okolini. Ovaj postupak je razrađen i implementiran u Saphira 8 okruženju.

Da bi estimacija položaja robota bila moguća, potrebno je napraviti model prostora robota u obliku karte. Karta se može dobiti korištenjem programa *sickLogger* instaliranog u *Arii* i informacija dobivenih laserskim senzorom robota. Postupak kreiranja karte detaljno je opisan, a u *dodatku 13* dana je i gotova karta zavoda za APR dobivena na ovaj način.

Na kraju je razrađen i gradijentni postupak koji koristi navigacijsku funkciju za generiranje gradijentnog polja, a ono predstavlja optimalnu stazu s najmanjim troškom do cilja iz bilo koje točke u prostoru. Na taj način se omogućuje postizanje optimalne brzine robota čak i kada istražuje nova područja ili nailazi na neočekivane prepreke. Prethodno dobivena karta prostora robota iskorištena je i za pokretanje gradijentnog modula u Saphira 8 okruženju.

LITERATURA

[1] Johann Borenstein, H. R. Everett, Liqiang Feng: Navigating Mobile Robots Systems and Techniques, A K Peters, Wellesley, Massachusetts, 1996.

[2] Dimitri van Heesch: Aria Sick Logger Documentation, Doxygen, 2002.

[3] ActivMedia Robotics Team: MapperProEditor Documentation, ActivMedia Robotics, LLC, 2002.

[4] Steffen Gutmann: ScanStudio Documentation, ActivMedia Robotics, LLC, 2002.

[5] ActivMedia Robotics Team: Mapping and Navigation Overview, ActivMedia Robotics, LLC, 2002..

[6] Dimitri van Heesch: Aria Reference Manual, Doxygen, 2002.

[7] Dimitri van Heesch: Saphira Reference Manual, Doxygen, 2002.

[8] Kurt G. Konolige: Robotics: Saphira 8 Software Manual, SRI International, Menlo Park, California, 2001.

[9] Kurt G. Konolige: A Gradient Method for Realtime Robot Control, SRI International, Menlo Park, California, 2001.

[10] Dieter Fox: Markov Localization for Mobile Robots in Dynamic Environments, AI Access Foundation and Morgan Kaufmann Publishers, 1999.

[11] Kurt Konolige: Robot Motion, SRI International, Menlo Park, California, 2001.

SAŽETAK

Naslov: Estimacija položaja mobilnog robota temeljena na karti prostora

Opis: U Saphira 8 okruženju razrađeni su i implementirani Markovljev algoritam estimacije položaja mobilnog robota i gradijentni algoritam za planiranje optimalne staze robota. Oba algoritma zasnivaju se na karti prostora, a postupak njezinog dobivanja je detaljno razrađen. Dane su upute za Saphiru i simulator robota Pioneer kao i za ostale, ovdje korištene aplikacije. Eksperimenti su izvedeni na stvarnom robotu Pioneer 2DX na zavodu za APR na Fakultetu elektrotehnike i računarstva.

Ključne riječi: estimacija pozicije, snimanje karte prostora, lokalizacija, navigacija, Markovljeva lokalizacija, gradijentna metoda, lokalizacijski modul, gradijentni modul, upute za Saphiru, ML, P2DX, mobilni robot, autonomno vozilo, Saphira, Colbert, SickLogger, Makefile, ScanStudio, Pioneer simulator robota.

ABSTRACT

Title: Map based mobile robot position estimation.

Description: Markov loclization algorithm for mobile robot position estimation, and Gradient algorithm for planing optimal path, have been elaborated and implemented in Saphira 8 environment. Both algorithms are based on a world map and procedure that describes how to extract it has been given in great detail. Saphira and Pioneer robot simulator tutorials, as well as tutorials for other used applications has also been given. All experiments are made on real Pioneer 2DX robot at Faculty of Electrical Engineering and Computing in Zagreb.

Ključne riječi: position estimation, map matching, localization, navigation, Markov localization, gradient method, localization module, gradient module, Saphira tutorials, ML, P2DX, mobile robot, autonomus vehicle, Saphira, Colbert, SickLogger, Makefile, ScanStudio, Pioneer robot simulator.

ŽIVOTOPIS

Rođen sam 31.5.1978. u Sisku, kao najstariji sin od trojice u obitelji. Tijekom djetinjstva doživio sam par selidbi zbog toga jer je mama bila teško bolesna tako da nije bilo lako, ali srećom nije se jako odrazilo.

Do 31.8.2003. radio sam u studentskoj organizaciji STEP kao "web master" stranice www.step.hr. Tim poslom bavio sam se dvije godine, a posjećenost stranice je za vrijeme mog rada porasla sa 6 000 na 15 000 studenata mjesečno.

Poslovi kojima sam se bavio prije (ili za vrijeme) ovoga bili su: rad na porti u studentskom domu kroz dvije godine; generalno čišćenje aviona Croatia Airlinesa kroz tri godine; povremeno davanje instrukcija te rad na zidarskim poslovima honorarno.

Sretno sam oženjen od 13.7.2002. s prvom i jedinom djevojkom. Djece još nemamo. Živimo u stanu koji smo dobili na čuvanje od jednog gospodina iz Njemačke. Pošto je stan bio rupa mi smo ga trebali u potpunosti preurediti tako da sad po ugovoru u njemu stanujemo do 1.1.2010. godine.

Imam vrlo dobro znanje Windows i osnovno poznavanje Linux platforme. Što se tiče programskih jezika tu bih izdvojio HTML, Flash ActionScript, PHP, MySQL, C++. Programi koje sam dobro savladao su: Photoshop, Dreamveawer, Flash, Matlab, Office.

Diplomirao sam na FER-u s prosjekom 4.1 na smjeru automatike. Sve godine sam redovno upisao, a u 7. semestru sam dobio mogućnost pisanja znanstvenog diplomskog koju sam prihvatio, no kasnije sam se predomislio jer sam učinio kompromis koji mi se učinio puno isplativijim (stipendija, jeftina hrana, prijevoz i stanovanje, a uz to sam već bio u radnom odnosu) tako da sam ipak odustao od znanstvenog diplomskog, a za nagradu dobio dva ispita kojih sam bio prethodno oslobođen. Posljednje dvije godine primao sam stipendiju tako da sam svoja studentska prava kao apsolvent iskoristio maksimalno. Za cijelo vrijeme studiranja boravio sam u studentskom domu Stjepan Radić na Savi.

Od 1997. godine imam položen vozački ispit B kategorije (bez ijednog negativnog boda na svim ispitima).

Završio sam srednju tehničku školu u Sisku kao tehničar za elektroniku, a moja generacija je slovila kao najbolja u povijesti TŠ Sisak. Bio sam uvijek odličan učenik, a jedine ocjene koje nisu bile petice bile su iz štreberskih predmeta. Kroz srednju školu stekao sam dosta dobro znanje engleskog jezika samostalnim učenjem i preko tečajeva koje sam pohađao. Također sam bio uključen u klub elektrotehničara SEL gdje sam izrađivao elektroničke konstrukcije. Na raznim natjecanjima sakupio sam petnaestak diploma i pohvalnica od kojih bih izdvojio HI-FI pojačalo i drugo za gitaru s kojima sam osvojio 2. mjesto u državi 1997. godine na natjecanju u Zadru i 6. mjesto u državi 1996. u Varaždinu tako da sam se par puta našao i u novinama.

Kao kršćanin aktivno sudjelujem u radu baptističke crkve u Radićevoj ulici. Trenutno sam zadužen za rad na mix pultu i sviranje gitare. Bio sam uključen i u rad s djecom po raznim kampovima u Hrvatskoj i Njemačkoj. Uglavnom se tu radilo o sportskim aktivnostima. Aktivnosti u crkvi su osim posla i turizma bili razlozi proputovanja gotovo svih evropskih država između Španjolske, Norveške i Hrvatske.

Dosta aktivno sam se bavio nogometom, stolnim tenisom, biciklizmom i baseballom. Baseball sam trenirao za klub Sisak. Kao vođa ekipe imao sam najodgovorniju poziciju tako da su utakmice većinom ovisile o meni. Bio sam nominiran i za hrvatsku reprezentaciju, no ipak nisam upao unatoč dobrim rezultatima (valjda nisam imao dobre veze).

Sportske aktivnosti morao sam malo smanjiti dok sam okrenuo koljeno na 1. godini faksa 1998. g. pa sam morao na operaciju što je bilo samo privremeno rješenje. Zbog toga sam 2003. g. u 5. mj. išao na još jednu da mi rekonstruiraju ligament da bih opet mogao biti u dobroj formi.

Od hobija kojima se bavim izdvojio bih osim sviranja gitare i sportskih aktivnosti još i fotografiranje. Ranije sam se bavio i uzgojom malih ptica i golubova tako da sam već tada krenuo sudjelovati po natjecanjima i osvojio jednom 1. mjesto u državi (ustvari ne ja, nego golub). Krletke za ptice izrađivao sam sam po svojim nacrtima tako da sam postupno stjecao vještinu izrađivanja raznih konstrukcija.

Razne aktivnosti kojima sam se bavio bili su dobar način za učenje odgovornosti, a uz rat, faks, djevojku i operacije bile su samo uzrok preranog odrastanja. Sada su red, rad i disciplina moji aduti, no unatoč organiziranosti i dobrom planiranju ne bih stigao napraviti sve što stižem da ne spavam jako malo.

DODACI:

Dodatak1: Colbert naredbe

Naredbe za gibanje robota:

- **`move(n)`** – giba se za n [mm] naprijed (+) ili natrag (-) *[move (int d)]*
- **`turn (n)`** – okreće se za n [°] ulijevo (+) ili udesno (-) *[setDeltaHeading (int th)]*
- **`turnto (n)`** – okreće se prema smjeru od n [°] *[setHeading (int th)]*
- **`speed (n)`** – giba se naprijed ili natrag brzinom n [mm/s] *[setVel (int v)]*
- **`rotate (n)`** – okreće se brzinom n [°/s] ulijevo (+) ili udesno (-) *[setRotVel (int w)]*
- **`stop`** – zaustavlja se *[stop ()]*

Komunikacijske naredbe:

- **`set baud [<int>]`** – vraća ili postavlja *baud rate* serijskog kanala *[sfSerialBaud]*
- **`set serial [<str>]`** – vraća ili postavlja ime serijskog porta *[sfComSerial]*
- **`set server [<str>]`** – vraća ili postavlja TCP/IP ime servera *[sfComServer]*
- **`set serverport [<int>]`** – vraća ili postavlja TCP/IP port servera *[sfComServerPort]*
- **`set local [<str>]`** – vraća ili postavlja lokalni COM port *[sfComPipe]*
- **`connect serial | local | server [<str>]`** – spaja se na zadani kanal
- **`disconnect`** – odspaja se s robota ili simulatora
- **`enable`** – omogućuje rad motora
- **`exit`** – odspaja se s robota ili simulatora i izlazi iz Saphire

Utility naredbe:

- **`pwd`** – ispisuje trenutnu stazu mape
- **`cd <str>`** – učitava <str> mapu
- **`load [<str>]`** – učitava <str> datoteku akcija iz trenutne mape bez argumenta
- **`loadlib [<str>]`** – učitava dinamičku datoteku <str> bez argumenta
- **`loadworld [<str>]`** – učitava *wld* datoteku
- **`unload [<str>] -`** zatvara *dll* datoteku <str> ili (bez argumenta) zadnju učitanu
- **`trace [<str>]`** – prati akciju <str>

Dodatak 2: Ispis izvornog koda datoteke *Makefile*

```
#
# Makefile for Saphira applications
#

SHELL = /bin/sh

############################################################

INCD = /usr/local/Saphira/ohandler/include/

# find out which OS we have
include $(INCD)os.h

ARIAD = /usr/local/Aria/

CFLAGS =  -g -D$(CONFIG) $(PICFLAG) $(REENTRANT)
CC = gcc
CPP = g++
INCLUDE = -I$(ARIAD)/include

############################################################

all: $(BIND)sickLogger
       touch all

sickLogger.o: sickLogger.cpp
       $(CC) $(CFLAGS) -c $(SRCD)sickLogger.cpp $(INCLUDE) -o sickLogger.o

sickLogger: sickLogger.o
       $(CPP) sickLogger.o -o sickLogger -L$(ARIAD)/lib -lAria $(LLIBSX)}
```

Dodatak3: I spis izvornog koda datoteke *sickLogger.cpp*

```
#include "Aria.h"

/*
This program will let you joystick the robot around, and take logs for the mapper while you
drive, automatically.
*/

int main(int argc, char **argv) {
  // whether to use the sim for the laser or not, if you use the sim
  // for hte laser, you have to use the sim for the robot too
  bool useSim = false;
  // robot
  ArRobot robot;
  // the laser
  ArSick sick;
  // connection
  ArDeviceConnection *con;
  // Laser connection
  ArSerialConnection laserCon;

  std::string filename = "1scans.2d";
  if (argc > 1)
    filename = argv[1];

  printf("Logging to file %s\n", filename.c_str());
  // configure the laser before we make the logger
  sick.configureShort(useSim, ArSick::BAUD38400, ArSick::DEGREES180,
                      ArSick::INCREMENT_HALF);


  // mandatory init
  Aria::init();

  printf("Pausing 5 seconds so you can disconnect VNC if you are using it.\n");
  ArUtil::sleep(5000);
  // attach the laser to the robot
  robot.addRangeDevice(&sick);

  // if we're not using the sim, make a serial connection and set it up
  if (!useSim)
  {
    ArSerialConnection *serCon;
    serCon = new ArSerialConnection;
    serCon->setPort();
    con = serCon;
  }
  // if we are using the sim, set up a tcp connection
  else
  {
    ArTcpConnection *tcpCon;
    tcpCon = new ArTcpConnection;
    tcpCon->setPort();
    con = tcpCon;
  }

  if (ArModuleLoader::load("libArInertial", &robot, NULL, true) == 0)
  {
    printf("Loaded the base inertial library\n");
    if (ArModuleLoader::load("ISense_Mod", &robot, (void *)"2", true) == 0)
    {
      printf("The ISense inertial module has been loaded and should be correcting heading
now.\n");
    }
    else if (
    ArModuleLoader::load("ISIS_Mod", &robot,(void *)ArUtil::COM2, false) == 0)
    {
      printf("The ISIS inertial module has been loaded and should be correcting heading
now.\n");
    }
  }
  // set the connection on the robot
  robot.setDeviceConnection(con);
  // try to connect, if we fail exit
  if (!robot.blockingConnect())
```

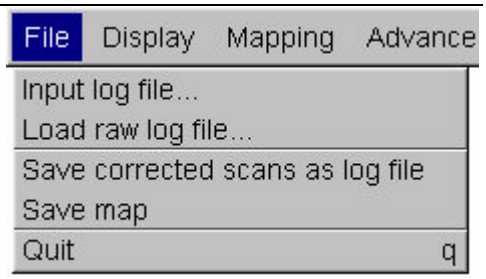
```
  {
    printf("Could not connect to robot... exiting\n");
    Aria::shutdown();
    return 1;
  }

  // if we're not using the sim, set up the port for the laser
  if (!useSim)
  {
#ifdef WIN32
    laserCon.setPort("COM3");
#else
    laserCon.setPort("/dev/ttyS2");
#endif
    sick.setDeviceConnection(&laserCon);
  }

  // give p2os a command to make it respond to the on-microcontroller
  // joystick port
  robot.comInt(ArCommands::JOYDRIVE, 1);

  // This must be created after the robot is connected so that it'll
  // get the right laser pos
  ArSickLogger logger(&robot, &sick, 300, 25, filename.c_str());

  sick.setSensorPosition(0,0,0);


  // add a keydrive action so that the robot can be driven with a
  // keyboard as well, toss in a joystick one too, just in case
  // someone does get a joystick working on a robot somehow

  ArActionJoydrive joydrive;
  joydrive.setStopIfNoButtonPressed(false);

  ArActionKeydrive keydrive;

  robot.addAction(&joydrive, 100);
  robot.addAction(&keydrive, 99);

  // run the robot, true here so that the run will exit if connection lost
  robot.runAsync(true);


  // now that we're connected to the robot, connect to the laser
  sick.runAsync();

  // connect to the laser
  if (!sick.blockingConnect())
  {
    printf("Could not connect to SICK laser... exiting\n");
    robot.comInt(ArCommands::SOUND, 13);
    Aria::shutdown();
    return 1;
  }


  // enable the motors, disable amigobot sounds
  robot.comInt(ArCommands::SONAR, 0);
  robot.comInt(ArCommands::ENABLE, 1);
  robot.comInt(ArCommands::SOUND, 32);
  robot.comInt(ArCommands::SOUNDTOG, 0);
  robot.comInt(ArCommands::JOYDRIVE, 1);
  // just hang out and wait for the end
  robot.waitForRunExit();
  sick.lock();
  sick.disconnect();
  //sick.setRobot(NULL);
  sick.unlock();
  // now exit
  Aria::shutdown();
  return 0;
}
```

Dodatak4 : Ispis dijela sadržaja datoteke 1scans.2d

```
LaserOdometryLog
#Created by ARIA's ArSickLogger
version: 1
sick1pose: 0 0 0
sick1conf: -90 90 361
#Scan 1
time: 31.617
robot: 0 0 0.00
sick1: 1578 1568 1569 1561 1562 1554 1545 1546 1537 1530 1528 1528 1521 1510 1512 1512 1503
1502 1503 1492 1491 1494 1484 1484 1474 1475 1467 1468 1467 1466 1467 1468 1468 1460 1461 1460
1451 1454 1454 1452 1454 1454 1451 1453 1447 1447 1448 1449 1448 1447 1447 1455 1453 1452 1447
1455 1448 1320 1322 1323 1318 1313 1314 1291 1290 1286 1284 1292 1288 1293 1292 1303 1307 1311
1309 1319 1325 1332 1339 1515 1516 1527 1525 1527 1534 1543 1544 1551 1551 1559 1562 1570 1571
1578 1578 1595 1593 1604 1613 1609 1606 1615 1634 1643 1655 4886 4911 4933 4960 4983 5013 5031
5065 5101 2864 2824 2791 2754 2728 2706 2671 2642 2610 2582 2562 2535 2510 2491 2465 2443 2421
2403 2378 2361 2337 2321 2304 2278 2266 2269 2279 2211 2191 2175 2176 2201 2202 2238 2266 2290
2317 2295 2278 2254 2243 2216 2212 2201 2189 2166 2157 2150 2138 2123 2105 2092 2083 2068 2059
2042 2032 2020 2014 2010 1995 1985 1975 1969 1959 1952 1948 1937 1928 1922 1914 1911 1905 1896
1893 1883 1875 1883 1860 1670 1679 1681 1671 1661 1662 1661 1654 1648 1646 1646 1636 1636 1627
1637 1630 1628 1625 1620 1619 1611 1614 1614 1611 1616 1610 1615 1612 1614 1919 2245 2280 2273
2282 2279 2279 2276 2281 2276 1929 1671 2233 2017 2284 2279 2286 2294 2292 2292 2292 2300 2300
2299 2305 2306 2307 2316 2323 2325 2323 2331 2333 2061 2251 2191 2110 2044 1984 1923 1725 1771
1768 1728 1700 1707 1704 1704 1714 1723 1722 1730 1735 1749 1748 1758 1768 1776 1784 1791 1802
1805 1821 1822 1839 1841 1856 1859 1868 1862 1858 1869 1888 1895 1912 1922 1928 1942 1952 1968
1975 1988 2006 2022 2026 2047 2061 2072 2126 3006 3016 2977 2953 2921 2897 2204 2841 2802 2792
2781 2755 2732 2712 2689 2672 2648 2631 2614 2484 2575 2570 2590 2617 2650 2689 2720 2758 2796
2833 2861 2939 1377 1390 4395 4457 4521 4584 3701 4729 4179 4486 4953 5037 5126 5218 2852 5382
3327 5618
#Scan 2
time: 41.981
robot: 222 4 2.07
sick1: 1537 1531 1524 1521 1516 1514 1504 1499 1497 1491 1487 1480 1473 1474 1474 1466 1466
1454 1455 1452 1443 1445 1437 1438 1438 1429 1431 1423 1428 1431 1421 1421 1421 1420 1415 1413
1415 1415 1406 1406 1406 1408 1407 1400 1403 1405 1402 1405 1400 1398 1401 1298 1308 1301 1295
1272 1251 1235 1236 1231 1228 1235 1232 1254 1251 1256 1257 1246 1258 1254 1268 1270 1285 1435
1438 1447 1445 1452 1455 1455 1462 1463 1463 1472 1472 1471 1478 1480 1489 1489 1506 1505 1514
1513 1523 1524 1524 1519 1539 1550 1561 1557 4575 4580 4628 4785 4816 4836 4874 4879 4917 4936
4968 2777 2749 2708 2685 2648 2625 2581 2569 2529 2507 2473 2456 2432 2405 2376 2355 2336 2310
2287 2270 2243 2228 2204 2188 2172 2161 2174 2188 2105 2092 2065 2075 2084 2090 2114 2160 2177
2205 2209 2170 2165 2143 2137 2117 2100 2099 2074 2063 2045 2053 2017 2027 2012 1996 1981 1973
1956 1948 1934 1925 1905 1903 1895 1889 1874 1875 1866 1858 1848 1848 1835 1828 1817 1812 1805
1803 1789 1789 1774 1776 1767 1769 1678 1571 1562 1571 1564 1564 1553 1555 1544 1547 1539 1544
1536 1536 1526 1530 1527 1529 1521 1527 1520 1521 1504 1516 1506 1507 1500 1509 1497 1511 1496
1518 1796 2172 2173 2176 2168 2173 2171 2180 2179 2176 2178 1578 1597 1923 2180 2188 2189 2187
2188 2196 2196 2207 2205 2213 2211 2218 2220 2229 2227 2238 2240 2246 2244 2127 1972 2137 2080
2021 1972 1908 1867 1798 1654 1718 1696 1648 1619 1622 1625 1638 1634 1649 1644 1660 1660 1671
1670 1690 1693 1704 1710 1727 1724 1741 1739 1754 1755 1775 1783 1792 1794 1809 1793 1812 1821
1830 1839 1856 1861 1889 1893 1911 1919 1935 1941 1959 1966 1989 1998 2021 2039 2947 2932 2906
2875 2855 2829 2193 2589 2763 2746 2719 2703 2685 2658 2643 2634 2610 2594 2573 2451 2544 2551
2581 2606 2654 2676 2728 2749 2809 2836 2948 1788 1807 1794 1788 1792 1381 1378 4102 2114 2085
2093 2291
#Scan 3
time: 42.377
.........................................
.........................................
.........................................
```

Dodatak5: Opis padajućih izbornika u *ScanStudiu*

Izbornik File:



Slika D.1. Izgled File izbornika

File->Load log file...

Otvara pretraživač koji omogućuje da odaberete log datoteku pomoću koje će se izvršiti mapiranje.

File->Load raw log file...

Otvara pretraživač koji omogućuje da odaberete log datoteku koju će *ScanStudio* u ovisnosti o tim vrijednostima pretvoriti u statičnu sliku. Ova opcija je korisna da bi se uočili veliki skokovi zbog proklizavanja kotača robota.

File->Save corrected scans as log file

Otvara pretraživač koji omogućuje da odaberete datoteku u koju će se pohraniti očitanja mapiranog prostora s ispravljenim pogreškama odometrije.

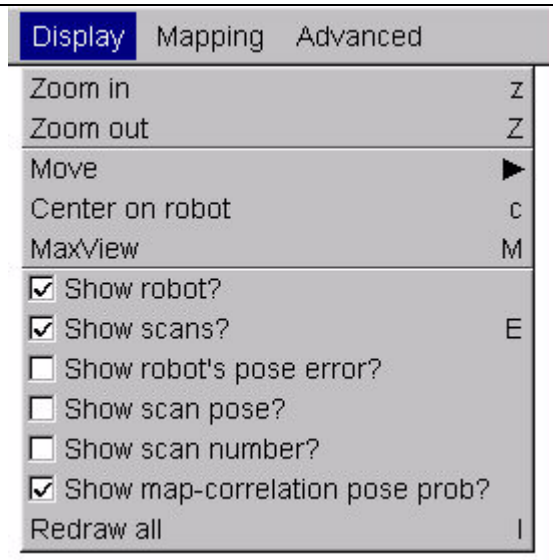
File->Save map

Otvara pretraživač koji omogućuje da odaberete datoteku u koju će se pohraniti napravljena karta prostora.

File->Quit

Izlaz iz programa.

Izbornik Display:



Slika D.2. Izgled Display izbornika

Display->Zoom in

Uvećava se centar ekrana.

Display->Zoom out

Smanjivanje prikaza ekrana.

Display->Move

Tu postoje još 4 podizbornika: Up, Down, Left, Right. Svaki od njih pomiče kartu u jednu stranu: gore, dolje, lijevo ili desno.

Display->Center on robot

Centrira pogled na robota.

Display->MaxView

Ekran se pomiče i povećava tako da bi na njegovu čitavom dijelu bila vidljiva cijela karta..

Display->Show robot?

Ako je kvačicom označena kućica tada će robot biti prikazan na karti na mjestu trenutnog očitanja lasera.

Display->Show scans?

Ako je kvačicom označena kućica tada će sitnim točkicama na karti biti označenja očitanja lasera.

Display->Show robot's pose error?

Omogućuje prikaz nesigurnosti robota o svom položaju. Kada je kućica označena, na mjestu gdje se nalazi robot oko njega se nalazi i elipsa koja nam govori koliko je robot siguran u svoj trenutni položaj. Također se prikazuje i trokut ispred robota koji se širi i sužava ovisno o tome koliko je robot siguran u svoju orjentaciju. Nesigurnost robota o položaju definira korelacijsko područje pretraživanja oko robota pri traženju hipoteza za zatvaranje petlje. Općenito, što dalje se robot giba u neistraženo područje, veća će biti i njegova nesigurnost u položaj.

Display->Show scan pose?

Ako je kvačicom označena kućica tada će na karti biti prikazana lokacija svih mjesta s kojih su uzeta očitanja lasera. Ona će biti prikazana kao male strelice koje označavaju poziciju i smjer robota u kojoj se nalazio dok su očitanja uzeta.

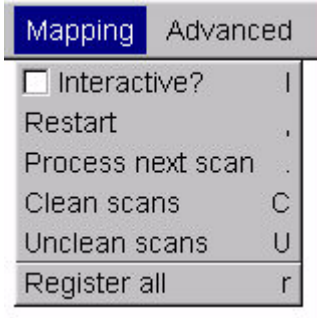
Display->Show scan number?

Ako je označena, pokraj svakog očitanja lasera na karti će biti prikazan njegov redni broj. Potrebno je dosta uvećati kartu da bi se mogli vidjeti pripadni brojevi.

Display->Show map-correlation pose prob?

Omogućuje prikaz rezultata korelacije za zatvaranje petlje. Kada je kućica označena, prikazan je zadnji rezultat korelacije kao mreža vjerojatnosti gdje tamnije ćelije označuju veću vjerojvatnost. Centar mreže je označen crvenim kvadratićem, a ćelija s najvećom vjerojatnosti označena je zelenom oznakom. Pogledajte također opciju "Mapper->Min prob for closing loop".

Izbornik Mapping:



Slika D.3. Izbornik Mapping

Mapping->Interactive?

Ako je ovo označeno tada će se *ScanStudio*zaustaviti nakon procesiranja svakog očitanja lasera, a ako nije označeno tada će se procesiranje nastaviti sve do kraja ili dok se ponovo ne označi ovo polje.

Mapping->Restart

Ova naredba ponovno pokreće mapiranje od prvog očitanja u log datoteci.

Mapping->Process next scan

Ova naredba procesira sljedeće očitanje iz log datoteke. Ako je isključen interaktivni način rada tada će se procesiranje samo nastaviti dok se interaktivni mod ne omogući ili dok ne ponestane očitanja u log datoteci.

Mapping->Clean scans

Nakon uspješnog mapiranja prostora ova naredba se može iskoristiti za pročišćavanje očitanja uzrokovanih gibajućim objektima, npr. ljudima. Nakon što se odabere ova naredba, mora se unijeti još i broj koji definira veličinu ćelije mreže kojom će biti prekriveno cijelo područje karte. Za svaku ćeliju računa se broj puta koje je kraj laserske zrake pogodio ćeliju i broj puta koje je zraka prekrila ćeliju. Točke očitanja koje pripadaju ćelijama kod kojih je drugi broj veći od prvog su vjerojatno uzrokovane dinamičkim objektima, npr. ljudima koji se kreću.

Grubo govoreći, manji broj će isfiltrirati više točaka očitanja nego veći broj. Može se eksperimentirati s veličinom ćelija za dobovanje pogodnog rezultata. Dobre vrijednosti za početak su između 50 i 100.

Može se uočiti da ova naredba dosta mijenja kartu. Zbog toga se karta može prvo prethodno pohraniti kao log datoteka za kasnije procesiranje, ako to već nije učinjeno.

Mapping->Unclean scans

Ova naredba će poništiti samo zadnje pročišćavanje očitanja naredbom "Mapping->Clean scans". Potpuno se vraća karta u stanje u kojem je bila prije pročišćavanja, ali samo za jednu razinu.

Mapping->Register all

Ova naredba može oduzeti dosta vremena. Njezino pozivanje uzrokuje da sva očitanja budu poravnata pri čemu se koristi kombinacija algoritma uspoređivanja slika i estimacije položaja. Naredba je korisna u slučajevima kada je prostor mapiran, ali se iz nekog razloga neki dijelovi karte ne poklapaju dobro.

Izbornik Advanced:

Advanced->Line model from scans

Ova opcija pomoću određenog algoritma interpolacije zamjenjuje na karti masu točkica, koje leže na zamišljenom pravcu, s ravnim linijama.

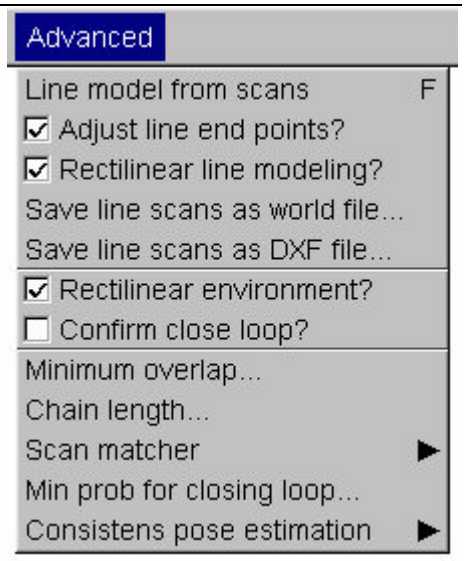
Advanced->Adjust line end points?

Ako je uključena, ova opcija poravnava krajeve linija u uglovima prostorija.

Advanced->Rectilinear line modeling?

Ova opcija služi za modeliranje pravocrtnih linija. Treba je uključiti ako se prostor sastoji od puno ravnih zidova.

Advanced->Save line scans as world file...



Slika D.4. Izbornik Advanced

Ovom opcijom može se karta pohrniti kao *world* datoteka za korištenje u SAPHIRI.

Advanced->Rectilinear environment?

Ova opcija je korisna za prostore koji sadrže puno kutova od 90°. Ako je uključena, algoritam mapiranja ubacuje dodatna ograničenje koja poravnavaju zidove vertikalno ili horizontalno ako je njihov smjer višekratnik od 90°. Cijela karta je također poravnata horizontalno ili vertikalno na ekranu.

Mijenjanje ove vrijednosti ima utjecaj samo kad se mapiranje ponovno pokrene.

Advanced->Confirm close loop?

Kada je ova opcije uključena, prije svakog zatvaranja petlje pojavljuje se prozor koji nas pita želimo li zatvoriti petlju. Postoje tri odgovora: 'Ok', 'Always' i 'No'. Ako pritisnemo 'Ok' petlja će se zatvoriti u označenom položaju. Pritisnemo li 'Always' će također petlju zarvoriti, ali će također spriječiti otvaranje novih prozora. 'No' će odbaciti zatvaranje petlje dok se ne pojavi sljedeće korelacijsko rješenje.

Svrha ovog je da se utvrdi je li algoritam namjerava zatvoriti petlju na pravom mjestu ili ne. Treba na karti provjeriti je li se položaj označen zelenom zastavicom podudara sa stvarnim položajem robota. Zbog toga je potrebno omogućiti opciju "Display->Show pose prob" da bi se mogla vidjeti zelena zastavica.

Advanced->Minimum overlap...

Algoritam mapiranja koristi tehniku poznatu kao usporedba slika za poravnanje različitih očitanja senzora. Ta tehnika koristi inicijalnu estimaciju relativnog položaja jednog para očitanja i efikasno pronalazi novi estimat kojem se dva očitanja bolje poklapaju. Da bi se ova metoda učinila robusnijom, algoritam se sastoji od nekoliko filtara koji odbacuju loša preklapanja i ignoriraju parove očitanja koji nemaju dovoljno početnog kutnog preklapanja.

U ovoj stavki izbornika može se zadati minimum kutnog preklapanja koje bi dva očitanja trebala imati da bi se poklapala. Ovaj algoritam će odbaciti sve parove očitanja koja će imati inicijalno manji broj od ovog kutnog preklapanje. Može se eksperimentirati s ovom vrijednošću. Ako se primjećuju kriva očitanja ovaj broj potrebno je povećati, a ako je karta na kraju loše poravnata, isti broj potrebno je smanjiti.

Dobra vrijednost za laser kuta zrake od 180° iznosi 60°.

Mijenjanje ove vrijednosti ima utjecaj samo kad se mapiranje ponovno pokrene.

Advanced->Chain length...

Minimalni broj prethodnih očitanja koji se koristi za usporedbu snovim očitanjima. Ovaj se broj koristi kad robot istražuje novo područje. Veći broj daje bolji rezultat (bolje poravnanje), dok manji broj ubrzava čitavi proces mapiranja. Dobra vrijednost je 10. Ne bi se smjela staviti na vrijednost manju od 5 osim ako se za sigurno zna što se radi.

Mijenjanje ove vrijednosti ima utjecaj samo kad se mapiranje ponovno pokrene.

Advanced->Scan matcher

Možemo izabrati jednu od tri medote za usporedbu očitanja.

- Cox je varijacija Coxove tehnike usporedbe očitanja i radi dobro samo u okolinama s ravnim zidovima.

- IDC je metoda koju su razvili Lu i Millos i koja je dobra za prostore s bilo kakvim oblicima osim dugih hodnika s ravnim zidovima.

- CSM je kombinacija Cox-a i IDC-a koja sadržava prednosti obiju metoda.

Najbolje je uvijek odabrati CSM jer obično daje najbolje rezultate.

Mijenjanje ove vrijednosti ima utjecaj samo kad se mapiranje ponovno pokrene.

Advanced->Min prob for closing loop...

Kad se robot približi već mapiranom području, započinje traženje da bi saznao gdje se točno nalazi relativno u odnosu na prethodno napravljenu kartu. Metoda traženja uključuje korelaciju zadnjih nekoliko očitanja s prethodno napravljenom kartom unutar područja oko robota definiranog njegovom mjernom nesigurnosti položaja.

Rezultat ovog korelacijskog ispitivanja je probabilistička mreža položaja koja za svaku ćeliju govori koliko dobro zadnjih nekoliko očitanja opisuju kartu (kolika je vjerojatnost da se položaj robota nalazi u toj ćeliji). Ako vrh maksimuma u ovoj mreži prelazi određenu minimalnu vrijednost i ne postoji neki drugi vrh koji prelazi tu vrijednost tada je petlja u tom položaju robota zatvorena.

U ovoj stavki izbornika može se odrediti ta minimalna vjerojatnost za korelacijsku metodu. Teško je dati općeniti savjet za vrijednost tog broja jer ona ovisi o okolini. Dobra početna vrijednost jest 0.8. Ako algoritam mapiranja ne zatvara petlje tada ovu vrijednost treba smanjiti. Ako algoritam zatvara petlje u krivom položaju, tada tu vrijednost treba povećati. Drugi način je da se ta vrijednost postavi na nisku vrijednost, npr. 0.5 te da se aktivira naredba "Mapping->Confirm close loop" i tada će se korisnik pitati da li da se petlja zatvori ili ne tako da se ne bi donesla kriva odluka.

Mijenjanje ove vrijednosti ima utjecaj samo kad se mapiranje ponovno pokrene.

Advanced->Consistens pose estimation

Treća metoda je konzistentna estimacija položaja koja se koristi za poravnanje nekoliko očitanja koristeći rezultate prethodne dvije metode u dvije situacije: kad robot istražuje novi teren i kad zatvara petlju.

- U ovoj stavki izbornika moguće je odabrati između tri različite metode za konzistentnu estimaciju položaja:

- Lu i Millos metoda, nazvana po autorima, općenito pruža točnu estimaciju na koju se može osloniti.

- Kalmanova relaksacijska metoda koja je u nekim slučajevima brža, ali daje rezultate nešto manje točnosti.

Treća metoda je kombinacija prethodne dvije koja daje jednaku točnost kao prva i brzinu kao druga metoda.

Postavite ovu vrijednost na treću, kombiniranu metodu.

Mijenjanje ove vrijednosti ima utjecaj samo kad se mapiranje ponovno pokrene.

Dodatak6 : Ispis dijela sadržaja datoteke *karta_zavoda.map*

```
# Created by Joe's Map Editor, ActivMedia Robotics
2D-Map
MinPos: -8730 -9070
MaxPos: 23630 5740
NumPoints: 42327
Resolution: 1
DATA
8610 -9070
8520 -9060
8530 -9060
8560 -9060
8570 -9060
8590 -9060
9030 -9060
14240 -9060
8550 -9050
8900 -9050
14180 -9050
8940 -9040
12010 -9040
8560 -9030
12040 -9030
12420 -9030
15650 -9030
8550 -9020
12050 -9020
12070 -9020
12390 -9020
12420 -9020
12440 -9020
12450 -9020
12460 -9020
12470 -9020
12480 -9020
13810 -9020
13840 -9020
13890 -9020
15780 -9020
8500 -9010
10630 -9010
............
```

Dodatak7 : Ispis dijela sadržaja datoteke *karta_zavoda.wld*

```
# Created by Steffen's Scan Studio.
origin -6924 -8910
width 30249
height 14674
-446 -1049 532 -1049
687 -1747 1624 -1747
1624 -1747 1624 -1504
1774 -1479 1774 -1039
1774 -1039 5399 -1039
2155 858 86 858
4048 860 3002 860
1624 -1504 1774 -1479
3024 1001 2210 1001
5012 1014 4048 1014
5067 849 5012 1014
6675 -1473 6675 -1031
6051 969 5067 849
10121 876 8594 876
7771 885 6199 885
5553 -1742 6511 -1742
6511 -1742 6511 -1494
6675 -1031 14432 -1031
6199 885 6051 969
6511 -1494 6675 -1473
8594 876 8594 2001
14014 876 10722 876
10942 1019 9836 1019
7678 3124 7149 3124
8315 4953 7625 4953
5399 -1039 5399 -1468
7149 3124 7149 1416
7728 1123 7771 885
14432 -1557 14432 -645
18362 869 14785 869
14847 1016 13941 1016
15724 -1030 20703 -1030
20703 -1030 20703 -124
15724 -1473 15724 -1030
15524 -1818 15579 -1473
14435 -639 14435 -1832
15579 -1473 15724 -1473
.......................
```

Dodatak8: Ispis izvornog koda datoteke *laser.act*

```
//  Datoteka za rad s Markovljevom lokalizacijom (ML)

//  Pomoću ove naredbe može se učitati WLD datoteka okruženja robota
    loadworld ../worlds/karta_zavoda.wld;

//  Učitavanje inicijalizacijskih datoteka
    loadlib sfLoc;
    loadlib sfLocFl;
    loadlib sfLocLaser;

//  Funkcije za inicijalizaciju Sonara i Lasera
    mcSonarInit();
    mcSonarInitRes(100);
    mcLrfInit();
    mcLrfScanInit();

//  Spajanje s Laserom lokalno ili na robotu
    sfStartLaser("");
//  sfStartLaser("/dev/ttyS2");

//  P A R A M E T R I    M L    L O K A L I Z A C I J E:

//  Postavlja pojačanje informacija senzora (obično između 10 i 50%)
    mcSetGain(50);
//  Postavlja pomak udaljenosti i kuta nakon kojih se obavlja osvježavanje ML
    sfJumpRobotAbs(0, 0, 0);
    mcSetGauss(400.0, 20.0);
    mcSetMove(20, 300, 30);
//  Postavlja broj ML točaka (uzoraka)
    mcSetNumSamples(3000);

//  P A R A M E T R I    S E N Z O R A:

//  inicijalizira lokalizaciju koristeći određenu veličinu mreže Sonara i Lasera
    mcSonarInitRes(200); // Default je: current=30, accumulated=100
    mcLrfInitRes(100); // Default je: current=30, accumulated=100

//  Da bi se robot pomicao prema mjestu najboljeg estimata položaja dobivenog pomoću ML
    mcUpdateRobotPose(1);
```

Dodatak9: Ispis izvornog koda datoteke *laser_robot.act*

```
//  Datoteka za rad s Markovljevom lokalizacijom (ML)

//  Pomoću ove naredbe može se učitati WLD datoteka okruženja robota
//  loadworld ../worlds/karta_zavoda.wld;

//  Učitavanje inicijalizacijskih datoteka
    loadlib sfLoc;
    loadlib sfLocFl;
    loadlib sfLocLaser;

//  inicijalizira lokalizaciju koristeći određenu veličinu mreže Sonara i Lasera
    mcSonarInitRes(200); // Default je: current=30, accumulated=100
    mcLrfInitRes(100); // Default je: current=30, accumulated=100

//  Funkcije za inicijalizaciju Sonara i Lasera
    mcSonarInit();
    mcSonarInitRes(100);
    mcLrfInit();
    mcLrfScanInit();

//  Učitaj MAP kartu. Ovo se poziva nakon funkcije mcLrfScanInit()
    mcLoadScanMap("worlds/karta_zavoda.map");

//  Postavlja objekt u kojem se nalazi MAP karta
    sfGradSetMap(mcGetObject());

//  Potrebno je omogućiti da se koriste artifakti da ne dođe do sudara s preprekama
    sfGradUseArtifacts(1);

//  P A R A M E T R I    M L    L O K A L I Z A C I J E:

//  Postavlja pojačanje informacija senzora (obično između 10 i 50%)
    mcSetGain(50);
//  Postavlja pomak udaljenosti i kuta nakon kojih se obavlja osvježavanje ML
    sfJumpRobotAbs(0, 0, 0);
    mcSetGauss(400.0, 20.0);
    mcSetMove(20, 1000, 50);
//  Postavlja broj ML točaka (uzoraka)
    mcSetNumSamples(2000);

//  Da bi se robot pomicao prema mjestu najboljeg estimata položaja dobivenog pomoću ML
    mcUpdateRobotPose(1);

//  Spajanje s Laserom lokalno ili na robotu
//  sfStartLaser("");
    sfStartLaser("/dev/ttyS2");
```

Dodatak10 : Ispis izvornog koda datoteke *gradient.act*

```
//  Datoteka za rad s gradijentnim modulom


//  Postavljanje lokalizacijskih rutina iz MAP karte napravljene u ScanStudiu
    loadlib sfLoc;
    loadlib sfLocLaser;
//  Postavljanje rutina za gradijentni modul
    loadlib sfGrad;
    loadlib sfGradFl; // vizualno sučelje za gradijentni modul u Saphiri

//  Inicijalizira lokalizaciju
    mcLrfInit();
//  Dodaje na kartu kumulativni laserski buffer
    mcLrfScanInit();
//  Veličina mreže
    mcLrfInitRes(300);

    mcSonarInit();
    mcSonarInitRes(200);

//  Inicijalizira i pokreće gradijentni modul
    sfGradInit();

//  Učitava MAP kartu. Ovo se poziva nakon funkcije mcLrfScanInit()
    pwd
    mcLoadScanMap("worlds/karta_zavoda.map");

//  Učitaj artifakte koji predstavljaju zabranjena područja
  loadworld ../worlds/forbidden.wld;

//  Postavlja objekt u kojem se nalazi MAP karta
    sfGradSetMap(mcGetObject());

//  Poziva se ako želimo da gradijentna metoda koristi MAP kartu
    sfGradUseMap(1);

//  Potrebno je omogućiti korištenje artifakata da ne dođe do sudara s preprekama
    sfGradUseArtifacts(1);

//  Ako želimo da gradijentna metoda koristi podatke sa sonara
    sfGradUseSonar(1);

//  Ako želimo da gradijentna metoda koristi podatke s lasera
    sfGradUseLaser(1);

//  Maksimalna veličina prozora za pretraživanje (već postavljeno u sfGradSetMap)
    sfGradSetMax(10000,10000);

//  Onemogući ispisivanje na ekranu prilikom osvježavanja
    mcPrintDuringUpdates(0);
//  Da bi se robot pomicao prema mjestu najboljeg estimata položaja
    mcUpdateRobotPose(1);

//  Postavlja delta kut (da), delta udaljenost (ds) i delta vrijeme (tm) prije nego se
lokalizacija ponovno izvrši
    mcSetMove(10, 100, 20); // 10 deg, 100 mm, 2 sec to update on stopping
//  Omogućuje da se robot može gibati prema natrag
    sfGradSetCanBack(1);
//  Postavljanje brzine za gradijentnu metodu u različitim okolnostima
    sfGradSetSpeed(800,300,0); // fw_max = 800 mm/s, fw_mid = 300 mm/s, back_max = 100 mm/s

//  Postavlja pojačanje informacija senzora (obično između 10 i 50%)
    mcSetGain(50);

//  Spajanje s Laserom lokalno ili na robotu
    sfStartLaser("");
//  sfStartLaser("/dev/ttyS2");
```

Dodatak11 : Ispis izvornog koda datoteke *mySaphira.cpp*

```
/*########################################
 * The Saphira application: all the basic processes loaded
 * Then loads colbert/startup.act
 *########################################
 */



#include "Saphira.h"
#include <stdio.h>


// main function, sets up a robot and inits ARIA and Saphira
// only returns when exit is explicitly signaled
#ifndef WIN32
int
main(int argc, char **argv)
#else
int PASCAL
WinMain(HINSTANCE hInst, HINSTANCE hPrevInstance, LPSTR lpszCmdLine, int nCmdShow)
#endif
{
  Sf::init();                 // start up Aria and Saphira
  Sf::robot()->runAsync(false);       // start up the robot, don't need a connection

  sfLoadActivityFile("sysStartup.act"); // this loads basic system files...
  sfLoadActivityFile("myStartup.act"); // this loads anything we want...

// Added by Metikos Marijan
  while (Sf::robot()->isRunning()) {

    if(SfROBOT->isConnected()) {

      // wait a few seconds for a stable connection
      ArUtil::sleep(2000);

      // enable the motors
      SfROBOT->enableMotors();

      // set the maximal translation and rotational velocity
      SfROBOT->setMaxTransVel(200);
      SfROBOT->setMaxRotVel(50);

      break;
    }
  }



  while (Sf::robot()->isRunning())
    ArUtil::sleep(100);
  Aria::shutdown();           // after we finish all processing
  return 0;
}
```

Dodatak12 : Naredbe za upravljanje modulima

Lokalizacijski modul

Ovaj modul koristi lokalizaciju i samostalno se može koristiti samo sa sonarima da bi se robot lokalizirao na karti s vektorskim linijama. Uz lokalizaciju pomoću modula s laserskim senzorima može se koristiti u karti generiranoj pomoću robota.

- **`void mcSonarInit (void)`** – inicijalizira lokalizaciju
- **`void mcSonarInitRes (int res)`** – inicijalizira lokalizaciju koristeći određenu veličinu mreže.
- **`void * mcGetObject (void)`** – preuzima *mc* objekt uglavnom za korištenje u funkciji `sfGradSetMap`.
- **`void mcSetMove (int da, int ds, int tm)`** – postavlja delta kut (*da*), delta udaljenost (*ds*) i delta vrijeme (*tm*) prije nego se lokalizacija ponovno izvrši. Parametar *da* označava vrijednost u stupnjevima vraćenu prije ponovne lokalizacije, parametar *ds* je udaljenost u milimetrima koju robot prijeđe prije ponovne lokalizacije, a *tm* označava broj ciklusa koji treba proći prije ponovne lokalizacije kada robot stoji na mjestu (ako je vrijednost ovog parametra 0 tada pri mirovanju neće raditi lokalizacija).
- **`void mcUpdateRobotPose (int on)`** – ovu je funkciju potrebno pozvati s nekom vrijednosti parametra da bi se tijekom lokalizacije osvježavao položaj robota.
- **`void mcPrintDuringUpdates (int on)`** – ako je pozvana s nekom vrijednosti parametra, ova funkcija će ispisivati trenutne korake tijekom lokalizacije.
- **`void mcSetGain (int pct)`** – postavlja pojačanje informacija senzora u koraku osvježavanja na vrijednost postotka *pct*. Ako je *pct* 0 ne koriste se informacije sa senzora. Razumne vrijednosti su od 10-50 %, ovisno o okolini, aplikaciji i senzorima.
- **`void mcSetGauss (float dx, float dth)`** – centrira skup ML točaka (uzoraka) na mjesto gdje se robot nalazi. Vjerojatno će biti potrebno pozvati funkciju za mijenjanje položaja robota *sfJumpRobotAbs* i nakon toga pozvati ovu funkciju. Parametar *dx* označava veličinu kvadrata u koji se postavljaju uzorci, a parametar *dth* je razlika u kutu unutar koje se postavljaju uzorci.
- **`void mcSetNumSamples (int n)`** – postavlja broj ML točaka (uzoraka). Svi uzorci se prvo resetiraju na vrijednost 0 nakon čega se može pozvati funkcija *mcSetGauss* da bi se skup uzoraka ponovno centrirao oko robota. Parametar *n* označava broj uzoraka za korištenje pri lokalizaciji.

Laserski modul

Ovaj modul omogućuje Saphiri korištenje *ArSick* (*SICK laser*) klase u Arii za korištenje ekrana i gradijentnog modula. U normalnom slučaju lokalizacija ne koristi laser kao zasebni proces što se može vidjeti u *laserskom lokalizacijsko-navigacijskom modulu*. Prikaz lasera je sa zelenim točkicama razbacanim po ekranu na mjestima očitanja lasera u stvarnosti. Može se primjetiti da je podešeno da su u Saphiri filtrirana očitanja lasera koja su previše blizu jedna drugom.

- **`void sfStartLaser (char *port)`** – spaja se s laserom. Vrijednost parametra *NULL* ili *""* znači spajanje na simulator robota.
- **`void sfStartLaserTcp (char *host, int port)`** – spaja se s laserom na TCP port. Ova opcija ne koristi se za simulator robota.
- **`void sfStopLaser ()`** – odspaja se s lasera.

Laserski lokalizacijsko-navigacijski modul

Ovaj modulomogućuje Saphiri korištenje lasera za lokalizaciju.

- **`void mcLrfInit ()`** – inicijalizira lokalizaciju.
- **`void mcLrfInitRes (int res)`** – inicijalizira lokalizaciju s određenom veličinom mreže.
- **`void mcLrfScanInit ()`** – dodaje na kartu kumulativni laserski buffer.
- **`void mcLoadScanMap (char *name)`** – učitava kartu prostora u MAP formatu.

Gradijentno-navigacijski modul

Ovaj modul služi za efikasno planiranje trajektorije robota u stvarnom vremenu, a zasniva se na gradijentnoj metodi uz pomoć karte prostora.

- **`void sfGradInit (void)`** – inicijalizira gradijentni modul. Potrebno ju je pozvati odmah nakon pozivanja ključnih gradijentnih *.dll* ili *.so* datoteka.
- **`void sfGradInitRes (int res, int turnRadius)`** – inicijalizira gradijentni modul. Parametar *res* služi za rezoluciju ćelije mreže. Ako je parametar *turnRadius* postavljen na 0 tada će se koristiti normalni parametri i akcije za gibanje robota, a ako nije 0 tada će robot ustuknuti ako se treba gibati unatraške i oko njega će se stvoriti krug polumjera *turnRadius*.
- **`void sfGradSetMap (void *p)`** – postavlja MAP kartu koju će koristiti gradijentna metoda. Obično se ta karta uzima iz lokalizacijskog modula koristeći poziv funkcijom *mcGetObject()*, tako da gradijentna metoda koristi istu kartu koju koristi i lokalizacijka metoda. Treba napomenuti samo da rezolucija karte mora biti ista kao i kod korištenja gradijentnih rutina (kod poziva funkcije *sfGradInit*).

- **`void sfGradUseArtifacts (int useArtifacts)`** – poziva se ako želimo da gradijentna metoda koristi WLD kartu.

- **`void sfGradUseMap (int useMap)`** – poziva se ako želimo da gradijentna metoda koristi MAP kartu.

- **`void sfGradUseSonar (int useSonar)`** – poziva se ako želimo da gradijentna metoda koristi podatke sa sonara.

- **`void sfGradUseLaser (int useLaser)`** – poziva se ako želimo da gradijentna metoda koristi podatke s laserskih senzora.

- **`voidsfGradSetGoal (float x, float y)`** – postavlja cilj gradijentne metode u globalnu točku zadanu u milimetrima. Cilj može biti promijenjen u bilo kojem trenutku.

- **`void sfGradSetSpeed (int high, int mid, int back)`** – postavlja brzine koje će koristiti gradijentna metoda u različitim okolnostima. Parametar *high* označava brzinu gibanja robota kada na putu nema nikakvih prepreka, parametar *mid* brzinu kada je prostor prenatrpan predmetima ili su u blizini prepreke, a parametar *back* maksimalnu brzinu za putovanje unatrag (pogledajte *sfGradInitRes*). Gibanje natraške će biti i sporije od zadanog ovim parametrom ako je to potrebno i koristit će se isti parametri *high* i *mid*

- **`void sfGradStop (void)`** – zaustavlja robota pri gibanju prema nekom cilju.

- **`int sfGradStatus (void)`** – prikazuje status gradijentnog modula. Ako je vraćena vrijednost 0 tada je robot u praznom hodu; 1 robot je aktivan, 2 gibanje je obavljeno, 3 zadatak nije uspio, 4 robot pretražuje područje.

- **`int sfGradIsActive (void)`** – vraća da li je gradijentni modul aktivan ili ne.

- **`void sfGradObsParams (int keepout, int decay)`** – postavlja udaljenost od prepreka koje će se gradijentni modul držati. Vrijednost parametra *keepout* označava udaljenost u milimetrima unutar koje se robot nikad ne smije približiti preprekama. Parametar *decay* označava vrijednost udaljenosti od prepreka u milimetrima koje se robot treba držati ako je moguće.

- **`void sfGradSetDone (int close, int done)`** – određuje koliko blizu cilja robot mora biti da bi usporio ili završio s približavanjem. Parametar *close* određuje udaljenost robota od cilja u milimetrima kad se on prebacuje u mod približavanja (*close*). Parametar *done* označava udaljenost robota od cilja u milimetrima kad se postupak približavanja smatra završenim.

- **`int sfGradGetCanBack (void)`** – vraća vrijednost da li se gradijentni postupak može pomaknuti unatrag. Ako je vraćeni rezultat 0 tada se gradijentni postupak nikad ne vraća natrag, ako je rezultat 1 tada će se vratiti kada je to prikladno (pogledajte naredbu *sfGradInitRes*), a kada je rezultat 2 tada će se uvijek vraćati natrag.

- **`void sfGradSetCanBack (int canBack)`** – postavlja vrijednost da li se gradijentni postupak može pomaknuti unatrag. Ako se za vrijednost *canBack* postavi 0 tada se gradijentni postupak nikad ne vraća natrag, ako se postavi 1 tada će se vratiti kada je to prikladno (pogledajte naredbu *sfGradInitRes*), a ako se postavi na vrijednost 2 tada će se uvijek vraćati natrag.

- **`void sfGradSetTurnRadius (int turnRadius)`** – postavlja polumjer okretanja robota potreban da bi se robot okrenuo umjesto gibanja unatrag. Za gibanje natrag treba pozvati funkciju *sfGradSetCanBack*. Vrijednost parametra *turnRadius* određuje polumjer koji mora biti slobodan da bi se robot zakrenuo za više od 45°.

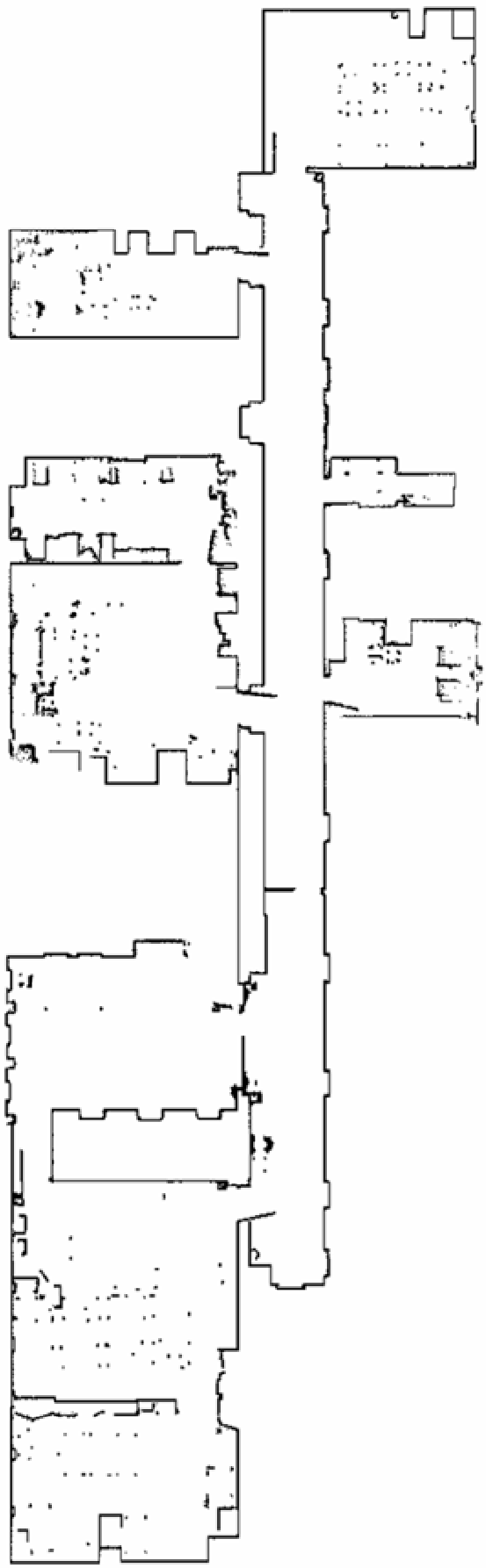
- **`int sfGradGetTurnRadius (void)`** – vraća vrijednost polumjera zakretanja potrebnog da bi se robot zakrenuo umjesto gibanja unatrag. Potrebno je samo se uvjeriti da je funkcijom *sfGradGetCanBack* omogućeno gibanje robota unatrag. Naredba vraća vrijednost polumjera koji mora biti slobodan da bi se robot zakrenuo za više od 45°.

- **`void sfGradSetAcc (int acc)`** – postavlja akceleraciju koja se koristi pri gibanju robota.

- **`void sfGradSetPnum (int n)`** – postavlja broj propagacija (uglavnom za pokuse).

- **`void sfGradSetMax (int width, int height)`** – postavlja veličinu gradijentnog prozora koja predstavlja okolinu gradijentnog modula. Okolina će se priširivati do veličine zadane ovom naredbom pri pretraživanju odgovarajuće trajektorije. Parametrom *width* određuje se širina okoline u milimetrima, a sa *height* duljina okoline također u milimetrima.

Dodatak 13: Slika karte zavoda



Slika D.5. Karta zavoda za APR