Autonomni mobilni roboti

Rajković, Marko

Sveučilište Josipa Jurja Strossmayera u Osijeku

Fakultet primijenjene matematike i informatike

Studij
Sveučilišni diplomski studij matematike
modul: matematika i računarstvo

Autonomni mobilni roboti

Diplomski rad

Mentor:
doc. dr. sc. Domagoj Ševerdija

Komentor:
dr. sc. Jurica Maltar

Student:
Marko Rajković

Osijek, 2024

Sadržaj

1 | Uvod

Robotski su sustavi postali vrlo popularni te ih se može prepoznati u raznim primjenama, a najviše u industrijskoj proizvodnji. Tu se najviše misli na robotske ruke koje se mogu gibati vrlo efikasno i precizno za obavljanje namijenjenih zadataka. Uspješne se primjene mogu naći i u problemima istraživanja raznih područja (kao što su planeti i ostala nepristupačna/teško pristupačna područja za čovjeka), autonomnim vozilima, komercijalnim primjenama (roboti za čišćenje). Veliki je broj primjena robotskih sustava moguće ostvariti kada su robotski sustavi mobilni, odnosno kada postoji mogućnost kretanja (pomoću kotača, robotskih nogu ili letom). Dizajniranje mobilnih robota obuhvaća veliki raspon područja iz znanosti te se time mobilnu robotiku čini velikim interdisciplinarnim područjem. Tu je potrebno razumjeti kinematiku, ponašanje dinamičkih sustava i teoriju upravljanja. Da bi se napravio robustan perceptivni sustav, potrebno je poznavati područje obrade signala i ostala specijalizirana područja kao što je u posljednje vrijeme računalni vid. Lokalizacija i navigacija zahtijevaju poznavanje specijaliziranih računalnih algoritama, algoritama umjetne inteligencije, ali i teoriju vjerojatnosti. Teorija vjerojatnosti je vrlo bitna jer roboti moraju biti sposobni nadomjestiti velike nepreciznosti i nesigurnosti koje postoje u fizičkom svijetu. Primjerice, senzori kojima se robot koristi podložni su različitim šumovima pa mjerenja koja oni vrate mogu biti nepredvidiva te se tako ograničavaju informacije koje mogu biti dobivene. Stoga ćemo u ovom radu proći kroz te gradivne cjeline mobilnih robota na kotače i pripadne tehnologije koje omogućuju robusnu mobilnost robota. Tako ćemo se u drugom poglavlju osvrnuti na kinematiku mobilnog robota te analizirati različite konfiguracije mobilnog robota te dati primjer upravljačkog sustava na jednoj takvoj konfiguraciji. U trećem ćemo poglavlju analizirati kako se mobilni robot može lokalizirati, odnosno održavati znanje o svojoj poziciji u prostoru te kako izgraditi mapu tog prostora. To će nas dovesti do problema istovremenog lokaliziranja i mapiranja (eng. *SLAM*) koji postavlja pitanje kako lokalizirati robota u nepoznatom prostoru te kako istovremeno izgrađivati mapu tog prostora. U četvrtom ćemo se poglavlju dotaknuti algoritma navigacije koji predstavlja kognitivnu razinu robota. U zadnjem ćemo poglavlju napraviti simulaciju mobilnog robota u kojem ćemo primijeniti obrađene dijelove ovog rada te vidjeti kako se mobilni robot ponaša u većinski nepoznatom prostoru ne znajući gdje se inicijalno nalazi.

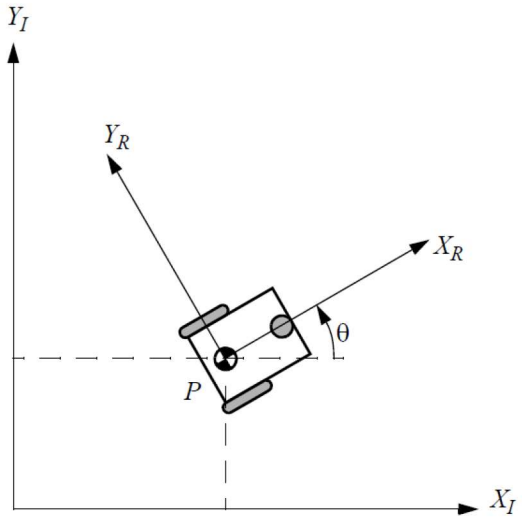
2 | Kinematika mobilnih robota

Kinematika se bavi proučavanjem ponašanja mehaničkih sustava. Tako je u području mobilnih robota potrebno poznavati mehaničko ponašanje robota da bi se moglo dizajnirati mobilne robote za pojedine zadatke i kreirati odgovarajući softver. Stoga se postavljaju mnoga važna pitanja vezana uz kinematiku. Tu je vrlo bitno analizirati radno okruženje (eng. *workspace*) mobilnog robota jer se time definira raspon mogućih stanja koje robot može postići. Upravljivost mobilnog robota definira moguće putove i trajektorije u radnom okruženju, dok dinamika mobilnog robota dodatno postavlja ograničenja na njegove trajektorije i radno okruženje zbog razmatranja mase i dodatnih sila koje djeluju na robota. Dodatno, bitno je procijeniti stanje mobilnog robota budući da se ono ne može direktno izmjeriti prilikom gibanja robota, a što se može postići integracijom gibanja kroz vrijeme. Zbog mogućih proklizavanja robota ili drugih razloga, u procesu procjene stanja treba pripaziti na određene nepreciznosti. Vidimo da sva ova pitanja kinematiku mobilnih robota čine izazovnim područjem.

2.1 Reprezentacija konfiguracije robota

Kako bismo izveli model za gibanje mobilnih robota, prvo je potrebno uvesti terminologiju koju ćemo cijelo vrijeme koristiti. Modelirat ćemo mobilnog robota kao kruto tijelo na kotačima na horizontalnoj ravnini te ćemo uvesti globalni referentni okvir i lokalni referentni okvir robota. Globalni se referentni okvir odnosi na koordinatni sustav koji ima fiksno inertno ishodište koje odabiremo u ravnini u kojoj se robot nalazi. Definirat ćemo konfiguraciju robota kao njegovu poziciju u ravnini zajedno s orijentacijom. Kako bismo odredili tu konfiguraciju, odredit ćemo jednu točku P na šasiji robota kao njegovu referentnu točku položaja. Ta će točka P ujedno biti ishodište lokalnog koordinatnog sustava robota koje čini lokalni referentni okvir kao što je prikazano na slici 2.1. Vidimo da su koordinatne osi referentnih okvira indeksirane s I (globalni) i s R (lokalni) tako da je jasno na temelju tih indeksa o kojem se referentnom okviru radi. Tada točku P u globalnom referentnom okviru možemo definirati s koordinatama x i y te sa θ što predstavlja kutni odmak lokalnog referentnog okvira od globalnog:

$$\xi_I = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}.$$



Slika 2.1: Globalni i lokalni referentni okvir [6].

Definirat ćemo ortogonalnu matricu rotacije koja će preslikavati konfiguraciju robota iz globalnog referentnog okvira u lokalni referentni okvir:

$$R(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.1}$$

Označavat ćemo ju s $R(\theta)$ jer njezini elementi ovise samo o θ, a jer je matrica ortogonalna vrijedi $R(\theta)^{-1} = R(\theta)^T$. Tada konfiguracija i gibanje robota u lokalnom referentom okviru mogu biti dobiveni sljedećim preslikavanjima:

$$\xi_R = R(\theta)\xi_I$$
$$\dot{\xi}_R = R(\theta)\dot{\xi}_I.$$

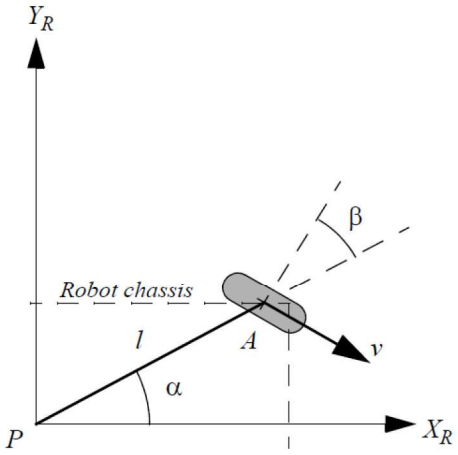
2.2 Kinematika kotača

Modeli gibanja mobilnih robota počinju s razumijevanjem kako svaki pojedini kotač na mobilnom robotu doprinosi tom gibanju. Osim što svaki kotač daje svoj doprinos gibanju robota, on također postavlja određena ograničenja tom gibanju. Tako svi kotači koji su postavljeni na šasiju robota formiraju skup ograničenja koji utječu na sveukupnu kretnju robota. Da bismo te doprinose i ograničenja mogli izraziti potrebno je imati konzistentan i pregledan referentan okvir. Stoga je bilo potrebno definirati referentne okvire koje smo uveli u prethodnom potpoglavlju zajedno s preslikavanjem (2.1). Prije nego što budemo prezentirali svaku vrstu kotača, napravit ćemo neke pretpostavke koje će pojednostaviti analizu utjecaja svakog kotača. Pretpostavit ćemo da je ravnina kotača uvijek vertikalna i da svaki kotač ima jednu točku dodira s ravninom podloge te da kotač ne proklizuje u toj točki dodira. Uz te će pretpostavke svaka vrsta kotača pridonositi gibanju robota

preko uvjeta vrtnje i lateralnom neproklizavanju preko uvjeta neproklizavanja. Uvjet vrtnje govori da se kotač mora kotrljati na odgovarajući način da bi uzrokovao gibanje robota. Drugi se uvjet odnosi na proklizavanje kotača, odnosno da kotač ne smije lateralno proklizavati, tj. ortogonalno u odnosu na njegovu ravninu gibanja.

2.2.1 Fiksni standardni kotač

Fiksni standardni kotač nema vertikalnu os rotacije za upravljanje. On ima fiksnu poziciju na šasiji robota i ograničen je na gibanje naprijed-natrag duž njegove ravnine. Na slici 2.2 možemo vidjeti položaj kotača u odnosu na lokalni referentni okvir robota.

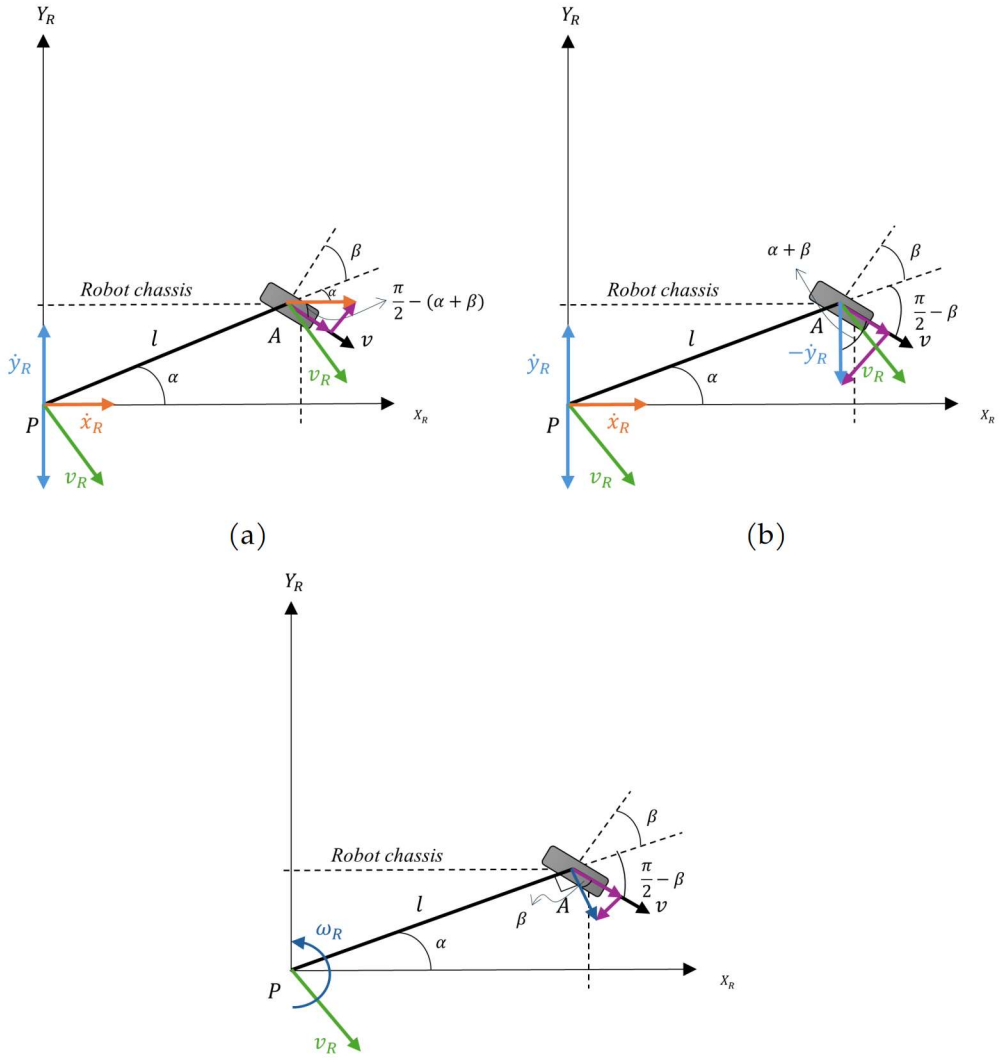


Slika 2.2: Fiksni standardni kotač [6].

Pozicija je kotača A izražena pomoću polarnih koordinata s udaljenosti l i kutom α. Kut između ravnine kotača i šasije robota označen je s β te je u ovom slučaju fiksan jer fiksni standardni kotač nije upravljiv. S r ćemo označiti njegov radijus, a kutni položaj oko horizontalne osovine kao funkciju $\varphi(t)$. Uvjet vrtnje za ovaj kotač govori da svako gibanje duž ravnine kotača mora biti popraćeno s odgovarajućom količinom vrtnje tako da se kotač kotrlja u točki dodira:

$$\begin{bmatrix}\sin(\alpha+\beta) & -\cos(\alpha+\beta) & (-l)\cos\beta\end{bmatrix} R(\theta)\dot{\xi}_I - r\dot{\varphi} = 0. \tag{2.2}$$

Da bismo došli do ovog rezultata promotrimo ovu situaciju. Gibanje će kotača prouzrokovati da se robot, odnosno šasija robota, giba brzinom v_R sa smjerom i orijentacijom kao na slikama 2.3. Ta se brzina može rastaviti na horizontalnu brzinu robota $\dot{x}_R$ i vertikalnu brzinu $\dot{y}_R$ u lokalnom referentom sustavu. Te brzine dalje možemo rastaviti na komponente koje odgovaraju smjeru putanje kotača i normale na tu putanju. Tada vidimo da komponenta od $\dot{x}_R$ koja smjerom i orijentacijom odgovara putanji kotača iznosi $\cos(\frac{\pi}{2} - (\alpha+\beta))\dot{x}_R = \sin(\alpha+\beta)\dot{x}_R$ kao što možemo vidjeti na slici 2.3a. Slično promatrajući $\dot{y}_R$ vidimo da komponenta koja smjerom i orijentacijom odgovara putanji kotača iznosi $-\cos(\alpha+\beta)\dot{y}_R$ (slika



Slika 2.3: Analiza fiksnog standardnog kotača.

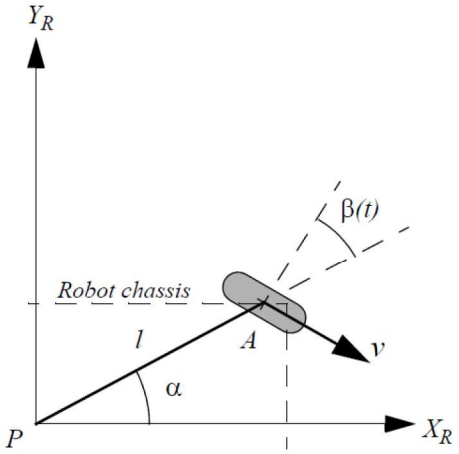
2.3b). Dodatno u točki A imamo translacijsku brzinu zbog koje se šasija robota rotira kutnom brzinom $\dot{\theta}_R$. Pri tome l promatramo kao polumjer kružnice čije je središte P u kojoj se događa rotacija. Iznos te translacijske brzine čiji je smjer i orijentacija prikazan na slici 2.3c iznosi $-l\dot{\theta}_R$, a nju isto možemo rastaviti na komponente koje odgovaraju smjeru putanje kotača i normali na tu putanju. Komponenta koja smjerom i orijentacijom odgovara putanji kotača je dana s $-l\dot{\theta}_R \cos\beta$. Suma ovih komponenti gibanja promatrajući iz središta kotača A mora biti jednaka gibanju $r\dot{\varphi}$ koje uzrokuje vrtnja kotača, što nas dovodi do formule 2.2.
Analogno promatramo komponente koje smjerom i orijentacijom odgovaraju normali na ravninu gibanja kotača te zbrajamo njihove doprinose (slike 2.3a, 2.3b i 2.3c). Suma tih doprinosa mora biti jednaka 0 jer se tada onemogućuje da kotač proklizuje što se može iskazati kao

$$\begin{bmatrix} \cos(\alpha+\beta) & \sin(\alpha+\beta) & l\sin\beta \end{bmatrix} \dot{\xi}_R = 0.$$

2.2.2 Usmjerljiv standardni kotač

Jedina razlika u odnosu na fiksni standardni kotač jest ta da se usmjerljiv standardni kotač sa slike 2.4 može rotirati oko vertikalne osi koja prolazi kroz središte kotača i njegove dodirne točke s podlogom. Zbog toga parametar β više nije fiksan, nego ga promatramo kao funkciju kroz vrijeme $\beta(t)$. No istom analizom kao u fiksnom standardnom kotaču možemo vidjeti da će uvjeti vrtnje i neproklizavanja biti isti osim što parametar β promatramo kao funkciju:

$$\begin{bmatrix}\sin(\alpha+\beta) & -\cos(\alpha+\beta) & (-l)\cos\beta\end{bmatrix} R(\theta)\dot{\xi}_I - r\dot{\varphi} = 0.$$
$$\begin{bmatrix}\cos(\alpha+\beta) & \sin(\alpha+\beta) & l\sin\beta\end{bmatrix} R(\theta)\dot{\xi}_I = 0.$$



Slika 2.4: Usmjerljiv standardni kotač [6].

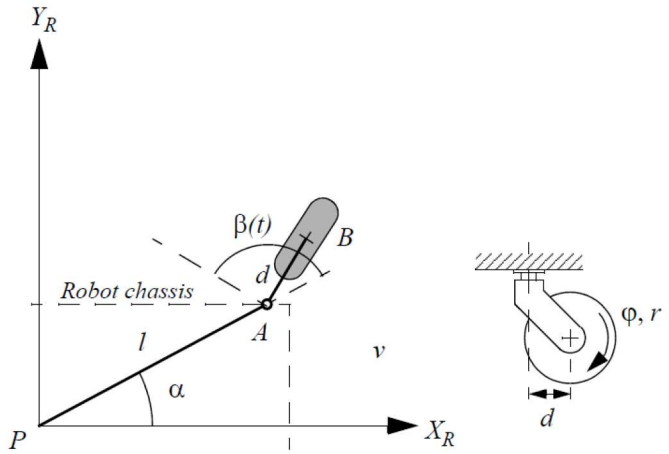
2.2.3 Castor kotač

Castor kotači mogu se rotirati oko vertikalne osi, ali ta vertikalna os ne prolazi kroz točku dodira s podlogom kao u slučaju usmjerljivog standardnog kotača. Kao što je prikazano na slici 2.5, možemo vidjeti da je točka B točka dodira s podlogom koja je povezana s točkom A s poveznicom fiksne duljine d. Uvjet vrtnje za ovaj kotač isti je kao u usmjerljivom standardnom kotaču zato što taj pomak vertikalne osi oko koje se kotač rotira ne predstavlja nikakvu ulogu u analizi komponenti gibanja robota koje smjerom odgovaraju putanji gibanja kotača (više u [6]):

$$\begin{bmatrix}\sin(\alpha+\beta) & -\cos(\alpha+\beta) & (-l)\cos\beta\end{bmatrix} R(\theta)\dot{\xi}_I - r\dot{\varphi} = 0.$$

No geometrija castor kotača ima utjecaja na uvjet neproklizavanja. Njezinom se analizom (vidjeti [6]) može ustvrditi da svako okomito gibanje u odnosu na ravninu kotača mora biti balansirano s ekvivalentnom i suprotnom količinom gibanja koje se generira njegovim zakretanjem:

$$\begin{bmatrix}\cos(\alpha+\beta) & \sin(\alpha+\beta) & d+l\sin\beta\end{bmatrix} R(\theta)\dot{\xi}_I = -d\dot{\beta}.$$

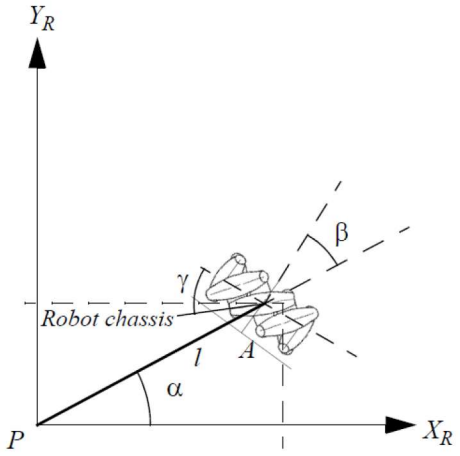


Slika 2.5: Castor kotač [6].

Iz tih se dvaju uvjeta može zaključiti da za bilo koje dano gibanje $\dot{\xi}_I$ postoji kutna brzina kotača $\dot{\varphi}$ i rotacijska brzina $\dot{\beta}$ tako da su dani uvjeti zadovoljeni. Stoga se roboti s castor kotačima mogu gibati bilo kojom brzinom u bilo kojem smjeru, a takve robote nazivamo svesmjernim robotima.

2.2.4 Švedski kotači

Švedski se kotači također mogu gibati u bilo kojem smjeru kao castor kotači. To im omogućuje njihova konstrukcija koja se sastoji od fiksnog standardnog kotača i valjaka pričvršćenih na perimetar kotača čije su osi oko koje se oni vrte pomaknute za γ u odnosu na ravninu kotača. Uvjeti vrtnje i neproklizavanja glase (vidjeti [6]):



Slika 2.6: Švedski kotač [6].

$$\begin{bmatrix}\sin(\alpha+\beta+\gamma) & -\cos(\alpha+\beta+\gamma) & (-l)\cos(\beta+\gamma)\end{bmatrix} R(\theta)\dot{\xi}_I - r\dot{\varphi}\cos\gamma = 0$$
$$\begin{bmatrix}\cos(\alpha+\beta+\gamma) & \sin(\alpha+\beta+\gamma) & l\sin(\beta+\gamma)\end{bmatrix} R(\theta)\dot{\xi}_I - r\dot{\varphi}\sin\gamma - r_{sw}\dot{\varphi}_{sw} = 0.$$

Upravo ta slobodna varijabla $\dot{\varphi}_{sw}$ koja predstavlja kutnu brzinu pričvršćenih valjaka omogućuje da uvjet neproklizavanja uvijek vrijedi.

2.2.5 Sferni kotač

Sferni kotač ne postavlja nikakve uvjete za gibanje i neproklizavanje zbog toga što sferni kotač nema principalnu os rotacije. Zbog toga može se gibati u bilo kojem smjeru kao castor i švedski kotač.

$$\begin{bmatrix}\sin(\alpha+\beta) & -\cos(\alpha+\beta) & (-l)\cos\beta\end{bmatrix} R(\theta)\dot{\xi}_I - r\dot{\varphi} = 0.$$
$$\begin{bmatrix}\cos(\alpha+\beta) & \sin(\alpha+\beta) & l\sin\beta\end{bmatrix} R(\theta)\dot{\xi}_I = 0.$$

Gore dane jednadžbe, koje su identične kao za fiksni standardni kotač, imaju drugačiju interpretaciju. Prva jednadžba opisuje brzinu rotacije sfernog kotača u točki A u smjeru putanje kotača, dok druga jednadžba slijedi iz definicije neproklizavanja u kojoj je β slobodna varijabla. Kut β predstavlja otklon normale na smjer gibanja kotača od šasije robota te se može izračunati na temelju ostalih podataka iz druge jednadžbe, a onda iz prve se jednadžbe može zaključiti brzina rotacije sfernog kotača.

2.3 Kinematski uvjeti za robota

Svaki će kotač davati određeni broj uvjeta na kretnju robota te će upravo ta kombinacija definirati kinematske uvjete za šasiju robota. Primijetimo da castor kotač, švedski kotač i sferni kotač ne daju nikakve kinematske uvjete za šasiju robota, odnosno ne postoje nikakva ograničenja na $\dot{\xi}_I$ zbog njihove geometrije koja daje dodatan stupanj slobode u gibanju. Stoga jedino fiksni i usmjerljiv standardni kotač imaju utjecaj na kinematiku šasije robota. Pretpostavimo da robot ima N standardnih kotača, od toga N_f fiksnih, a N_s usmjerljivih standardnih kotača. S $\beta_s(t)$ označit ćemo kutove zakrenutosti od N_s usmjerljivih standardnih kotača u trenutku t, dok ćemo s β_f označiti orijentacije od N_f fiksnih standardnih kotača. S $\varphi_f(t)$ i $\varphi_s(t)$ označit ćemo kutne položaje s obzirom na osovinu fiksnih i usmjerljivih standardnih kotača te ćemo ih skupiti u matricu $\varphi(t)$:

$$\varphi(t) = \begin{bmatrix}\varphi_f(t)\\ \varphi_s(t)\end{bmatrix}.$$

Tada se uvjeti vrtnje svih kotača mogu zapisati kao:

$$J_1(\beta_s)R(\theta)\dot{\xi}_I - J_2\dot{\varphi} = 0, \tag{2.3}$$

pri čemu je J_2 konstantna dijagonalna $N \times N$ matrica čiji su elementi na dijagonali radijusi standardnih kotača, a $J_1(\beta_s)$ $N \times 3$ matrica čiji redci sadrže projekcije na ravninu gibanja svakog kotača. Tada $J_1(\beta_s)$ možemo zapisati kao:

$$J_1(\beta_s) = \begin{bmatrix} J_{1f} \\ J_{1s}(\beta_s) \end{bmatrix},$$

gdje je J_{1f} konstantna $N_f \times 3$ matrica s projekcijama za fiksne standardne kotače, dok je $J_{1s}(\beta_s)$ $N_s \times 3$ matrica s projekcijama za usmjerljive standardne kotače koja ovisi o parametru β_s. Analogno možemo zapisati uvjete neproklizavanja:

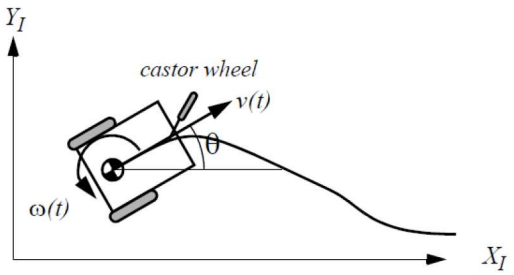
$$C_1(\beta_s)R(\theta)\dot{\xi}_I = 0, \tag{2.4}$$

$$C_1(\beta_s) = \begin{bmatrix} C_{1f} \\ C_{1s}(\beta_s) \end{bmatrix},$$

gdje su C_{1f} i $C_{1s}(\beta_s)$ $N_f \times 3$ i $N_s \times 3$ matrice koje sadrže projekcije okomite na ravninu gibanja kotača za fiksne i usmjerljive standardne kotače redom.

2.3.1 Direktna kinematika za robota na diferencijalni pogon

Pogledajmo kako izvesti direktnu kinematiku na primjeru robota na diferencijalni pogon 2.7. Pod pojmom direktne kinematike smatramo ovisnost između kutnih brzina kotača i gibanja šasije robota, odnosno kako kutne brzine kotača utječu na gibanje robota.



Slika 2.7: Robot na diferencijabilni pogon kotač [6].

Taj robot sadrži tri kotača: dva fiksna standardna kotača na pogon i jedan castor kotač. Castor kotač nema pogon te se može gibati u bilo kojem smjeru, stoga ne pridonosi nikakva ograničenja na robota. Kako su oba kotača fiksna standardna, to znači da se u uvjetima vrtnje robota (2.3) i uvjetima neproklizavanja robota (2.4) matrice $J_1(\beta_s)$ i $C_1(\beta_s)$ pojednostavljuju na J_{1f} i C_{1f}. Pretpostavimo da je $\theta = \frac{\pi}{2}$ odnosno da se robot giba u smjeru ξ_R, tj. y_I osi te da su standardni kotači udaljeni od središta šasije za l i da su jednakog polumjera r. Tada za desni kotač imamo $\alpha = -\frac{\pi}{2}, \beta = \pi$, a za lijevi $\alpha = \frac{\pi}{2}, \beta = 0$. Sada možemo računati retke matrica J_{1f} i C_{1f}:

$$J_{1f} = \begin{bmatrix} \sin(-\frac{\pi}{2}+\pi) & -\cos(-\frac{\pi}{2}+\pi) & (-l)\cos\pi \\ \sin(\frac{\pi}{2}+0) & -\cos(\frac{\pi}{2}+0) & (-l)\cos 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & l \\ 1 & 0 & -l \end{bmatrix}$$
$$C_{1f} = \begin{bmatrix} \cos(-\frac{\pi}{2}+\pi) & \sin(-\frac{\pi}{2}+\pi) & l\sin\pi \\ \cos(\frac{\pi}{2}+0) & \sin(\frac{\pi}{2}+0) & l\sin 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Primijetimo da u matrici C_{1f} imamo dva zavisna (ista) retka, što je posljedica toga što se kotači nalaze na istoj osovini, odnosno paralelni su. Svođenjem uvjeta (2.3) i (2.4) u jednu matričnu jednadžbu dobijemo:

$$\begin{bmatrix} 1 & 0 & l \\ 1 & 0 & -l \\ 0 & 1 & 0 \end{bmatrix} R(\theta)\dot{\xi}_I = \begin{bmatrix} r\dot{\varphi}_l \\ r\dot{\varphi}_r \\ 0 \end{bmatrix},$$

te iz nje možemo izvesti formulu za direktnu kinematiku robota pri čemu vidimo kako upravljanjem kutnim brzinama kotača možemo mijenjati stanje robota:

$$\dot{\xi}_I = R(\theta)^{-1} \begin{bmatrix} 1 & 0 & l \\ 1 & 0 & -l \\ 0 & 1 & 0 \end{bmatrix}^{-1} \begin{bmatrix} r\dot{\varphi}_l \\ r\dot{\varphi}_r \\ 0 \end{bmatrix}. \tag{2.5}$$

2.4 Manevrabilnost mobilnog robota

Mobilnost se robota odnosi na sposobnost kretanja u vlastitom okruženju, a uvjeti neproklizavanja kotača mogu ograničiniti tu mobilnost. Kotači mogu utjecati na kretanje robota njihovim okretanjem čime se definira stupanj skretljivosti robota. Kombinacijom stupnja mobilnosti robota na temelju uvjeta neproklizavanja i stupnja skretljivosti kotača definirana je manevrabilnost mobilnog robota.

2.4.1 Stupanj mobilnosti

Iz jednadžbe (2.4) možemo iščitati da vektor gibanja $R(\theta)\dot{\xi}_I$ mora biti unutar nul-prostora projekcijske matrice $C_1(\beta_s)$ da bi se izbjeglo proklizavanje. Kao što smo vidjeli u primjeru robota na diferencijalni pogon, može se dogoditi da kotači daju zavisne uvjete neproklizavanja. No to ne doprinosi kinematici robota te se zato kinematika robota zasniva na skupu nezavisnih uvjeta neproklizavanja kotača. Broj nezavisnih uvjeta možemo iščitati iz (2.4) tako da promatramo rang matrice $C_1(\beta_s)$. Što je više nezavisnih uvjeta, to je mobilnost robota više ograničena. Primijetimo da za rang matrice $C_1(\beta_s)$ vrijedi $0 \leq rang[C_1(\beta_s)] \leq 3$ jer je matrica $C_1(\beta_s)$ dimenzija $N \times 3$. Sada možemo definirati stupanj mobilnosti robota δ_m:

$$\delta_m = dim[N(C_1(\beta_s))] = 3 - rang[C_1(\beta_s)].$$

Dimenzija nul-prostora matrice $C_1(\beta_s)$, odnosno stupanj mobilnosti pokazuje na broj stupnjeva slobode robota koji se mogu direktno upravljati kroz promjene kutnih brzina kotača. Za $\delta_m = 0$, tj. $C_1(\beta_s) = 3$, robot je potpuno ograničen te se ne može uopće gibati. U primjeru robota na diferencijalni pogon možemo vidjeti da

je $\delta_m = 2$ odnosno $C_1(\beta_s) = 1$ jer robot uistinu može mijenjati svoju orijentaciju te kretati se naprijed-nazad manipuliranjem kutnih brzina kotača. Najbolji primjer za $\delta_m = 1$ odnosno $C_1(\beta_s) = 2$ nam daje bicikl koji se može kretati samo duž ravne linije ili po nekom kružnom luku. Kutnom se brzinom jedinog kotača na pogon bicikl može kretati samo prema naprijed ili nazad (ako je nazad moguće). Maksimalan stupanj mobilnosti odnosno $\delta_m = 3$ znači da se robot može gibati u bilo kojem smjeru. To se događa kod robota koji se sastoje samo od svesmjernih kotača kao što su castor ili švedski kotači.

2.4.2 Stupanj skretljivosti

Kako promjene u kutnim brzinama kotača mogu utjecati na gibanje robota, tako i zakretanje kotača ima utjecaj na gibanje robota, iako je on indirektan iz razloga što se on tek vidi kada se robot giba. Zato definiramo broj nezavisnih upravljivih parametara za skretanje kao stupanj skretljivosti

$$\delta_s = rang[C_{1s}(\beta_s)].$$

Možemo definirati mogući raspon od δ_s: $0 \leq \delta_s \leq 2$. Možemo primijetiti da kako $C_1(\beta_s)$ uključuje $C_{1s}(\beta_s)$, to znači da usmjerljiv standardni kotač istovremeno može smanjivati stupanj mobilnosti i povećavati stupanj skretljivosti. Njegova trenutna orijentacija može ograničiti kinematiku robota, ali njegova sposobnost mijenjanja orijentacije može omogućiti robotu gibanje novim trajektorijama. Degenerativni slučaj $\delta_s = 3$ nećemo promatrati jer je tada gibanje nemoguće. Kada robot nema usmjerljivih standardnih kotača tada je $\delta_s = 0$. Slučaj $\delta_s = 2$ jedino je moguć kada robot nema fiksne standardne kotače. Kada robot sadrži jedan ili više usmjerljivih standardnih kotača onda je najčešće $\delta_s = 1$. To se može vidjeti na primjeru automobila koji sadrži dva fiksna standardna kotača i dva usmjerljiva standardna kotača. Kako su dva fiksna kotača na istoj osovini, tada je $rang[C_{1f}] = 1$. Računom se može se vidjeti da je $rang[C_{1s}(\beta_s)] = \delta_s = 1$ i da je $\delta_m = 1$.

2.4.3 Manevrabilnost

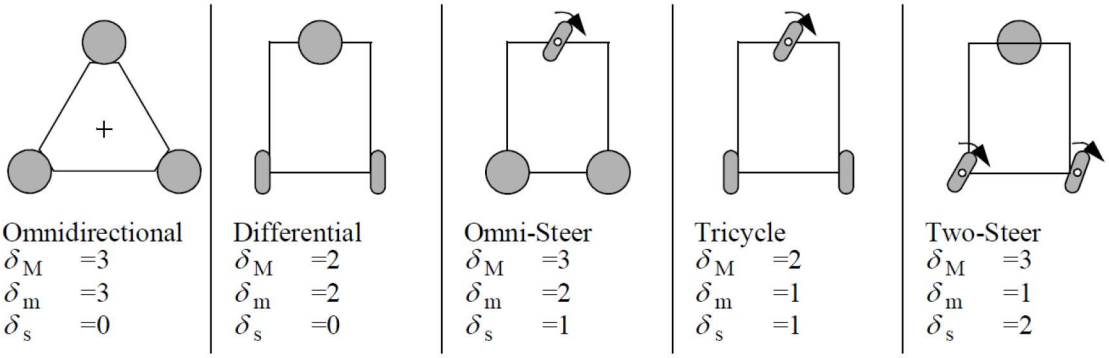
Spremni smo definirati stupanj manevrabilnosti robota kao:

$$\delta_M = \delta_m + \delta_s.$$

Vidimo da manevrabilnost uključuje stupnjeve slobode kojima robot može direktno upravljati samo kroz kutne brzine kotača te stupnjeve slobode kojima može indirektno upravljati okretanjem kotača i njihovim gibanjem. Na slici 2.8 možemo vidjeti različite primjere konfiguracija s tri kotača pri čemu sferne kotače možemo zamijeniti castor ili švedskim kotačima, a da stupnjevi ostanu nepromijenjeni.

2.5 Upravljanje mobilnog robota

Ovi teorijski rezultati kinematike samo su uvertira u složenije područje. U stvarnim je primjerima potrebno osim ovih rezultata uklopiti i brojne druge dinamičke



Slika 2.8: Primjeri konfiguracije s 3 kotača [6].

uvjete koji mogu utjecati na gibanje robota. Bitno pitanje koje se dodatno javlja je pitanje upravljanja robota, odnosno na koji način dovesti robota u željenu konfiguraciju u nekom vremenu. Za to je potrebno i znanje iz područja teorije upravljanja. Zato ćemo pokazati pojednostavljeni primjer upravljanja robota na diferencijalni pogon kojeg smo već uzimali u primjeru izvođenja njegove direktne kinematike.

2.5.1 Sustavi

Kada promatramo neki sustav možemo ga generalno modelirati sljedećom diferencijalnom jednadžbom:

$$\dot{x}(t) = f(x(t), u(t)),$$

pri čemu je $x(t) \in \mathbb{R}^n$ stanje sustava, a $u(t) \in \mathbb{R}^m$ upravljački ulaz. Sustav može davati izlazne signale koje možemo opažati, a što se modelira sljedećom jednadžbom:

$$y(t) = h(x(t), u(t)),$$

pri čemu $y(t) \in \mathbb{R}^p$. Pri tome poželjno je da sustavi budu stabilni i upravljivi. Upravljivost nam je bitna jer onda možemo dizajnirati kontrolere/regulatore koji će određivati upravljanje $u(t)$ u nekom vremenskom rasponu i dovoditi sustav u željeno stanje. Jedan takav regulator je i PID regulator.

2.5.2 PID regulator

PID regulator gleda pogreške sustava u odnosu na željenu vrijednost. Tu pogrešku iskazujemo u obliku funkcije $e(t)$. Tada regulator definira upravljanje koje se osniva na proporcionalnom, integralnom i derivacijskom izrazu (od čega dolazi kratica PID na engleskom jeziku). To se može iskazati sljedećim izrazom:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)\, d\tau + K_d \frac{de(t)}{dt},$$

pri čemu su K_p, K_i i K_d proizvoljne konstante. Proporcionalni izraz uzima u obzir trenutnu pogrešku sustava $e(t)$ s konstantom K_p. Integralni izraz $K_i \int_0^t e(\tau)\, d\tau$ uzima u obzir i prethodne pogreške koje je sustav imao. Derivacijski izraz $K_d \frac{de(t)}{dt}$ "predviđa" kako bi se funkcija $e(t)$ trebala ponašati. Prilikom praktične primjene za izračun integrala i derivacije koriste se numeričke metode.

2.5.3 Upravljanje mobilnog robota na diferencijalni pogon

Pretpostavimo da se robot giba uvijek u $+\xi_R$ smjeru nekom konstantnom brzinom $v_R \geq 0$ koju mi zadajemo. Pretpostavimo da je trenutna konfiguracija robota dana s ξ_I te da je konačna konfiguracija dana s g_I u globalnom referentnom okviru. Ono što želimo postići jest da se robot giba prema cilju, odnosno da ima orijentaciju $\xi^*_{I_\theta}$. Ta se željena orijentacija postiže:

$$\xi^*_{I_\theta} = \arctan\left(\frac{g_{I_y} - \xi_{I_y}}{g_{I_x} - \xi_{I_x}}\right),$$

s time da uvijek uzimamo $\xi^*_{I_\theta} \in \langle -\pi, \pi]$, a rubnu situaciju za $g_{I_x} = \xi_{I_x}$ kada $g_{I_y} \neq \xi_{I_y}$ posebno definiramo tako da stavimo da je $\xi^*_{I_\theta} = \frac{\pi}{2}$. Tada možemo definirati grešku sustava kao $e(t) = \xi^*_{I_\theta} - \xi_{I_\theta}$ za koju ćemo također zahtijevati $e(t) \in \langle -\pi, \pi]$. Sada možemo definirati upravljački ulaz kao kutnu brzinu robota ω_R jer je to jedino što je preostalo definirati u robotu kako bi se on mogao gibati. Kutnu brzinu ω_R možemo dobiti preko PID regulatora:

$$\omega_r = K_p e(t) + K_i \int_0^t e(\tau)\, d\tau + K_d \frac{de(t)}{dt}.$$

No zbog fizičkih ograničenja kotača nije uvijek moguće očekivati brzine v_r i ω_r na robotu. Zato ćemo definirati izvedive brzine $v_{r_{izv}}$ i $\omega_{r_{izv}}$ odnosno one kojima će se robot zapravo gibati. Pretpostavljajući da su oba kotača jednaka u smislu maksimalne rotacijske brzine i polumjera i da se okreću u istom smjeru uvest ćemo

$$\dot{\varphi}_{L_{max}} = \dot{\varphi}_{R_{max}} = \dot{\varphi}_{max}$$
$$\dot{\varphi}_{L_{min}} = \dot{\varphi}_{R_{min}} = \dot{\varphi}_{min} = 0.$$

Promotrimo desni kotač na robotu. Kada se on vrti prema naprijed kutnom brzinom $\dot{\varphi}_R$, tada će se središte šasije P, koje se nalazi na polovici udaljenosti od oba kotača, gibati brzinom $\frac{r\dot{\varphi}_R}{2}$, a analogno vrijedi i za lijevi kotač. Ukupna će brzina u središtu šasije P iznositi $\frac{r\dot{\varphi}_R}{2} + \frac{r\dot{\varphi}_L}{2}$. Kada se pokreće samo desni kotač, tada će se šasija rotirati oko lijevog kotača kutnom brzinom $\omega_R = \frac{r\dot{\varphi}_R}{2l}$ (obrnuto od smjera kazaljke na satu). No to znači da će se i u točki P također javljati ta kutna brzina. Analogno vrijedi i za lijevi kotač samo što se tu rotacija događa u smjeru kazaljke na satu pa je ukupna kutna brzina šasije u točki P jednaka $\frac{r\dot{\varphi}_R}{2l} - \frac{r\dot{\varphi}_L}{2l}$.
Tada možemo vidjeti da vrijedi:

$$\omega_{r_{max}} = \frac{r\dot{\varphi}_{max}}{2l}, \quad \omega_{r_{min}} = -\frac{r\dot{\varphi}_{max}}{2l}$$
$$v_{r_{max}} = r\dot{\varphi}_{max}, \quad v_{r_{min}} = r\dot{\varphi}_{min} = 0.$$

Iz formule da direktnu kinematiku (2.5) možemo izvesti formulu za inverznu kinematiku te onda napraviti preslikavanje $(v_r, \omega_r) \mapsto (\dot{\varphi}_L, \dot{\varphi}_R)$:

$$\begin{bmatrix} \dot{\varphi}_L \\ \dot{\varphi}_R \\ 0 \end{bmatrix} = \frac{1}{r}\begin{bmatrix} 1 & 0 & l \\ 1 & 0 & -l \\ 0 & 1 & 0 \end{bmatrix} \dot{\xi}_R = \frac{1}{r}\begin{bmatrix} 1 & 0 & l \\ 1 & 0 & -l \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} v_r \\ 0 \\ \omega_r \end{bmatrix}.$$

No kako dobivene kutne brzine kotača ne moraju biti unutar $[\dot{\varphi}_{min}, \dot{\varphi}_{max}]$, možemo definirati algoritam 1 koji će napraviti potencijalnu korekciju kutnih brzina.

Algoritam 1

procedure CORRECTION($\dot{\varphi}_L, \dot{\varphi}_R, \dot{\varphi}_{min}, \dot{\varphi}_{max}$)
  $\dot{\varphi}_{LRMAX} = \max(\dot{\varphi}_L, \dot{\varphi}_R)$
  $\dot{\varphi}_{LRMIN} = \min(\dot{\varphi}_L, \dot{\varphi}_R)$
  if $\dot{\varphi}_{LRMAX} > \dot{\varphi}_{max}$ **then**
    $\dot{\varphi}_L = \dot{\varphi}_L - (\dot{\varphi}_{LRMAX} - \dot{\varphi}_{max})$
    $\dot{\varphi}_R = \dot{\varphi}_R - (\dot{\varphi}_{LRMAX} - \dot{\varphi}_{max})$
  else
    if $\dot{\varphi}_{LRMIN} < \dot{\varphi}_{min}$ **then**
      $\dot{\varphi}_L = \dot{\varphi}_L + (\dot{\varphi}_{min} - \dot{\varphi}_{LRMIN})$
      $\dot{\varphi}_R = \dot{\varphi}_R + (\dot{\varphi}_{min} - \dot{\varphi}_{LRMIN})$
  $\dot{\varphi}_L = \max(\dot{\varphi}_{min}, \min(\dot{\varphi}_L, \dot{\varphi}_{max}))$
  $\dot{\varphi}_R = \max(\dot{\varphi}_{min}, \min(\dot{\varphi}_R, \dot{\varphi}_{max}))$

Nakon korekcije možemo se vratiti na formulu direktne kinematike (2.5) te dobiti konačnu translacijsku $v_{r_{izv}}$ i rotacijsku $\omega_{r_{izv}}$ brzinu kojima se robot giba. Tako se postupak ponavlja sve dok ne dođemo do cilja.

$$\dot{\xi}_R = \begin{bmatrix} v_{r_{izv}} \\ 0 \\ w_{r_{izv}} \end{bmatrix} = \begin{bmatrix} 1 & 0 & l \\ 1 & 0 & -l \\ 0 & 1 & 0 \end{bmatrix}^{-1} \begin{bmatrix} r\dot{\varphi}_L \\ r\dot{\varphi}_R \\ 0 \end{bmatrix}.$$

3 | Lokalizacija i mapiranje

Jedna je od izazovnih stvari na mobilnom robotu njegova navigacija. Da bi se robot uspješno mogao voditi kroz prostor potrebno je prije toga savladati nekoliko bitnih stavki. Prva od njih je percepcija, tj. robot mora znati interpretirati podatke sa senzora da bi dobio smislene informacije o okolini. Zatim je bitna lokalizacija, odnosno robot mora znati odrediti svoju poziciju u okolini u kojoj se nalazi. Upravo se najviše istraživanja usmjerilo na tehnike lokalizacije. No i sam postupak lokalizacije uključuje više ostalih procesa. Tako percepcija ima važnu ulogu u procesu lokalizacije, a fundamentalnu ulogu u percepciji imaju senzori. Senzore možemo svrstati u dvije kategorije, one koji promatraju interno stanje robota (npr. odometrijski senzori) i one koji promatraju okolinu u kojoj se robot nalazi (npr. ultrazvučni ili laserski senzori udaljenosti). Senzori nisu uvijek točni i precizni zbog raznih fizičkih svojstava, stoga se ne možemo u potpunosti osloniti na njih. Kada se te pogreške akumuliraju kroz određeno vrijeme, to može biti loše za lokalizaciju.
Zato se u lokalizaciji razvijaju modeli koji stanje robota iskazuju u obliku vjerojatnosne distribucije. Tu se javljaju dva načina reprezentiranja mogućih konfiguracija robota. Prvi se način zasniva na korištenju samo jedne hipotetske pretpostavke o stanju robota (eng. *single-hypothesis belief system*). Primjerice, stanje robota možemo modelirati jednom Gaussovom distribucijom. U drugom se načinu koristi više (moguće i beskonačno mnogo) hipotetskih pretpostavki o stanju robota (eng. *multiple-hypothesis belief system*). Tada se konfiguracije robota mogu modelirati s više Gaussovih distribucija ili mogu se diskretizirati stanja robota pa svakom stanju dodijeliti vjerojatnost da je to pravo stanje robota. Glavna je prednost prvog načina ta da se olakšava donošenje odluka o budućim akcijama robota na njegovoj kognitivnoj razini (npr. u procesu planiranja putanje) jer imamo samo jednu hipotetsku pretpostavku konfiguracije robota. No glavna je mana ta što se prilikom ažuriranja konfiguracije robota poslije upravljačkih ulaza jako teško može postići zaključak o sljedećem stanju koje bi se temeljilo samo na jednoj hipotetskoj pretpostavci zbog svih mogućih pogrešaka/nesigurnosti u senzorima koje se mogu dogoditi. Glavna je prednost u drugom načinu, u kojem robot prati određeni skup konfiguracija, sposobnost procjenjivanja nesigurnosti vlastitog stanja što je vrlo korisno kada se procjenjuje buduće stanje na temelju upravljačkih ulaza. Ono što je nedostatak jest to da je problematično donositi odluke o tome kako će se robot gibati ako nije siguran gdje se nalazi. Tu se može doskočiti da se uzima najvjerojatnija konfiguracija robota, ali to znači da se mora za svaku moguću konfiguraciju računati vjerojatnost, što pridodaje računskoj složenosti. Moguće je također planirati trajektorije koje će svesti tu nesigurnost na najmanju moguću mjeru, no to

opet pridodaje računskoj složenosti.

3.1 Reprezentacija mape

Reprezentacija mape također ima veliku ulogu u lokalizaciji. Često odabir određene reprezentacije mape utječe na odabir reprezentacije konfiguracije robota, a isto tako utječe na računsku složenost procesa mapiranja, lokalizacije i navigacije.

3.1.1 Kontinuirana reprezentacija mape

Jedan od načina reprezentacije mape je kontinuirani pristup pri čemu se na kontinuiran i precizan način prikazuju značajke u okolini. Pod pojmom značajke smatramo sve prepoznatljive objekte u prostoru. Te se značajke mogu prepoznati posebnim algoritmima koji pokušavaju na temelju podataka sa senzora izvući smislene informacije. Ovakav se pristup najčešće koristi u 2D reprezentacijama. Jedan takav primjer je pojednostavljeno prikazivanje objekata linijama koje ne moraju čak ni u potpunosti ocrtavati objekt nego samo one dijelove koje senzori mogu očitati. Velika prednost u ovakvom pristupu je visoka preciznost i izražajnost koja se može dobiti u razumnom vremenu. Mapu tada možemo zapisati kao

$$m = \{m_1, m_2, \ldots, m_N\},$$

gdje je N ukupan broj objekata u prostoru, a najčešće svaki $m_n, 1 \leq n \leq N$ predstavlja koordinate objekata.

3.1.2 Dekompozicija mape

Drugi pristup u reprezentiranju mape je diskretiziranje okoline, odnosno radi se dekompozicija u kojoj se stvarna okolina podijeli na dijelove. Prednost u ovakvom pristupu je ta da neke dekompozicije mogu odgovarati algoritmima navigacije pa se tu mogu postići brže i bolje performanse. Ono što jest nedostatak što se prilikom dekompozicije može izgubiti određena točnost i pouzdanost. Vrlo popularan primjer u načinu reprezentiranja okoline su mrežaste mape zauzetosti u kojima se okolina prikazuje pomoću mreže ćelija, a obojenost tih ćelija označava zauzetost.

3.2 Bayesovi filtri

U lokalizaciji vjerojatnost ima vrlo važnu ulogu jer svi ulazni podaci sa senzora bilo koje vrste sa sobom nose određenu dozu pogreške kao i upravljački ulaz koji ne mora odgovarati stvarnom gibanju. Stoga je jako teško deterministički procjenjivati konfiguracije robota na temelju senzora i upravljačkog ulaza u stvarnim uvjetima. Vjerojatnost ima fundamentalnu ulogu ovdje jer se sve te nesigurnosti mogu vjerojatnosno modelirati, odnosno procjenjivanje se konfiguracije robota može modelirati pomoću vjerojatnosnih distribucija što je i dovelo do uspjeha mobilnih robota.

S $\Xi_t = \{\xi_0, \xi_1, \ldots, \xi_T\}$ označimo put robota do trenutka T. S u_t ćemo označiti upravljački ulaz, no taj izraz može predstavljati podatke sa senzora (npr. odometrijskih senzora kotača). Kao što je već rečeno, senzori sa sobom nose određene nesigurnosti, odnosno šum koji nam onemogućuje da zapravo samo iz upravljačkih ulaza $U_t = \{u_0, u_1, \ldots, u_T\}$ deterministički odredimo konfiguraciju robota. S z_t ćemo označiti mjerenje sa senzora (a može uključivati i podatke sa više senzora) u trenutku t, a tada niz mjerenja možemo označiti s $Z_t = \{z_0, z_1, \ldots, z_T\}$ pri čemu ta mjerenja mogu predstavljati koordinate točaka, linija ili ravnina u referentom okviru senzora. Kao što smo rekli, konfiguraciju ćemo robota morati procjenjivati na temelju podataka sa senzora, a najbolju ćemo pretpostavku o stanju robota nazivati uvjerenjem. Uvjerenje možemo modelirati aposteriornom distribucijom nad stanjima na temelju svih mjerenja i upravljačkih ulaza do trenutka t:

$$bel(\xi_t) = p(\xi_t | z_{1\to t}, u_{1\to t}).$$

No također nam je bitno uvjerenje prije uključivanja najnovijeg mjerenja z_t odmah poslije upravljačkog ulaza u_t:

$$\overline{bel}(\xi_t) = p(\xi_t | z_{1\to t-1}, u_{1\to t}).$$

Često se distribucija $\overline{bel}(\xi_t)$ naziva predikcija jer se konfiguracija robota predviđa na temelju upravljačkog ulaza i prethodnih mjerenja ili akcija jer se tijekom ove faze robot giba. Slično se distribucija $bel(\xi_t)$ naziva korekcija jer se na temelju mjerenja stanje robota ažurira. Algoritam 2 opisuje općeniti algoritam za računanje uvjerenja koji se još naziva Bayesov filter.

Algoritam 2 Bayesov filter

1: **procedure** BAYESFILTER($bel(\xi_{t-1}), u_t, z_t$)
2: **for** $\forall \xi_t$ **do**
3: $\overline{bel}(\xi_t) = \int p(\xi_t | u_t, \xi_{t-1})\, bel(\xi_{t-1})\, d\xi$
4: $bel(\xi_t) = \eta\, p(z_t | \xi_t)\, \overline{bel}(\xi_t)$
5: **return** $bel(\xi_t)$

Primijetimo da u Bayesovom filteru nemamo konkretne informacije o prostoru, odnosno mapi. On je rekurzivan, odnosno vidimo da se distribucija uvjerenja $bel(\xi_t)$ u trenutku t računa na temelju $bel(\xi_{t-1})$ u trenutku $t-1$ uz najnoviji upravljački ulaz u_t i mjerenje z_t. On se sastoji od dva koraka: predikcije i korekcije. Predikcija se sastoji integralne sume umnoška dviju distribucija: apriorne distribucije uvjerenja nad ξ_{t-1} i vjerojatnosne distribucije gibanja (vjerojatnost da u_t inducira tranziciju iz stanja ξ_{t-1} u ξ_t). Korekcija se sastoji od računanja umnoška uvjerenja $\overline{bel}(\xi_t)$ s vjerojatnosnom distribucijom mjerenja (vjerojatnost da se dogodilo mjerenje z_t s obzirom na stanje ξ_t). Taj umnožak ne čini vjerojatnosnu distribuciju jer nije normaliziran, zbog čega se pojavljuje normalizacijska konstanta η. Kako se distribucija uvjerenja računa rekurzivno, tada je potrebno definirati početno uvjerenje $bel(\xi_0)$ gdje se najčešće javljaju dva slučaja. Prvi je slučaj kada točno znamo gdje se robot nalazi. Tada distribuciju uvjerenja $bel(\xi_0)$ modeliramo s δ distribucijom u točki u kojoj se robot nalazi. Drugi je slučaj kada uopće ne znamo gdje se

robot nalazi te tada možemo koristiti uniformnu distribuciju. Moguće su i druge varijante gdje imamo djelomične informacije o početnom stanju robota pa se tu mogu koristiti i druge distribucije. Jedna od bitnih pretpostavki za ovaj algoritam je Markovljeva pretpostavka koja govori da buduće stanje ξ_t ovisi samo o prethodnom stanju ξ_{t-1}, odnosno da su sve informacije o prethodnim mjerenjima i upravljačkim ulazima sadržane u ξ_{t-1}. U stvarnim primjerima to nije istina jer zapravo trenutno stanje može ovisi o prethodnim stanjima, no pokazalo se da je ova pretpostavka uspješna i robusna na situacije u kojem Markovljeva pretpostavka ne vrijedi, a da pri tome ta pretpostavka smanjuje računalnu složenost.
Pokažimo korektnost Bayesovog filtra, odnosno da se distribucija $bel(\xi_t) = p(\xi_t|z_{1\to t}, u_{1\to t})$ dobije iz $bel(\xi_{t-1}) = p(\xi_{t-1}|z_{1\to t-1}, u_{1\to t-1})$ pri čemu znamo početnu distribuciju $bel(\xi_0)$. Koristeći Bayesovo pravilo imamo:

$$\begin{aligned} p(\xi_t|z_{1\to t}, u_{1\to t}) =& \frac{p(z_t|\xi_t, z_{1\to t-1}, u_{1\to t})\, p(\xi_t|z_{1\to t-1}, u_{1\to t})}{p(z_t|z_{1\to t-1}, u_{1\to t})} \\ =& \eta\, p(z_t|\xi_t, z_{1\to t-1}, u_{1\to t})\, p(\xi_t|z_{1\to t-1}, u_{1\to t}). \end{aligned}$$

Uz Markovljevu pretpostavku ($p(z_t|\xi_t, z_{1\to t-1}, u_{1\to t}) = p(z_t|\xi_t)$) dobijemo

$$p(\xi_t|z_{1\to t}, u_{1\to t}) = \eta\, p(z_t|\xi_t)\, p(\xi_t|z_{1\to t-1}, u_{1\to t}),$$

odnosno

$$bel(\xi_t) = \eta\, p(z_t|\xi_t)\, \overline{bel}(\xi_t).$$

Koristeći teorem potpune vjerojatnosti za $\overline{bel}(\xi_t)$ dobijemo:

$$\begin{aligned} \overline{bel}(\xi_t) =& p(\xi_t|z_{1\to t-1}, u_{1\to t}) \\ =& \int p(\xi_t|\xi_{t-1}, z_{1\to t-1}, u_{1\to t})\, p(\xi_{t-1}|z_{1\to t-1}, u_{1\to t})\, d\xi_{t-1}, \end{aligned}$$

Primijetimo da u izrazu $p(\xi_{t-1}|z_{1\to t-1}, u_{1\to t})$ stanje ξ_{t-1} ne ovisi o u_t. Uz Markovljevu pretpostavku za stanje ξ_{t-1} dobijemo:

$$p(\xi_t|\xi_{t-1}, z_{1\to t-1}, u_{1\to t}) = p(\xi_t|\xi_{t-1}, u_t),$$

što će sveukupno dati predikciju algoritma:

$$\begin{aligned} \overline{bel}(\xi_t) &= \int p(\xi_t|\xi_{t-1}, u_t)\, p(\xi_{t-1}|z_{1\to t-1}, u_{1\to t-1})\, d\xi_{t-1} \\ &= \int p(\xi_t|\xi_{t-1}, u_t)\, bel(\xi_{t-1})\, d\xi_{t-1}. \end{aligned}$$

No ovakav se generalni oblik algoritma ne može implementirati pa se zbog toga koriste aproksimacije i varijante koje se biraju s obzirom na efikasnost, točnost i

praktičnost. Primjerice, često se u 2D gibanju koristi dekompozicija prostora stanja pomoću mreže s fiksnom veličinom ćelija (iako je moguće koristiti tzv. adaptivnu dekompoziciju za bolju računalnu složenost) pa se time dobije trodimenzionalan diskretan prostor stanja. U takvim je varijantama potrebno definirati početnu distribuciju uvjerenja $bel(\xi_0)$, distribuciju gibanja $p(\xi_t|u_t, \xi_{t-1})$ i distribuciju mjerenja $p(z_t|\xi_t)$.
Kao što je rečeno na početku, mi smo ovdje modelirali distribuciju uvjerenja na temelju najnovijeg upravljačkog ulaza i podataka sa senzora, ali bez ikakvih saznanja o okolini, tj. o mapi. Kada bismo uključivali informacije o prostoru, tada bismo promatrali problem $p(\xi_t|z_{1\to t}, u_{1\to t}, m)$, a tada govorimo o problemu lokalizacije.

3.2.1 Vjerojatnosni modeli gibanja

Vjerojatnosni modeli gibanja opisuju aposteriornu distribuciju stanja robota na temelju upravljačkog ulaza u_t (što ne mora nužno označavati upravljanje nego i mjerenje s odometrijskih senzora) i njegove trenutne konfiguracije ξ_{t-1}:

$$p(\xi_t|u_t, \xi_{t-1}).$$

Navest ćemo dva modela gibanja za mobilne robote koji se gibaju u 2D prostoru. Prvi se model zasniva na tome da u_t predstavlja upravljačku naredbu za translacijsku i kutnu brzinu robota (ili se pretpostavlja da će se u najboljem slučaju robot tako gibati). Drugi se model osniva na tome da u_t predstavlja odometrijska mjerenja. U praksi se pokazalo da su odometrijski modeli više precizniji nego modeli osnovani na brzini jer se dani upravljački ulaz aktualizira s manjom preciznošću od odometrijskog mjerenja. No kako odometrijski model predstavlja naknadne informacije, on se ne može koristiti za planiranje gibanja, nego se koristi samo za procjenjivanje, dok se modeli zasnovani na brzini koriste u modelima planiranja.

Model zasnovan na brzini

U modelu zasnovanom na brzini upravljački ulaz u_t predstavlja translacijsku i kutnu brzinu kojima se robot upravlja:

$$u_t = \begin{bmatrix} v_t \\ \omega_t \end{bmatrix}.$$

Stoga možemo dati algoritam 3 u zatvorenoj formi za računanje vjerojatnosti $p(\xi_t|u_t, \xi_{t-1})$ pri čemu ulazni parametri su inicijalna poza $\xi_{t-1} = [x, y, \theta]^T$, upravljački ulaz $u_t = [v, \omega]^T$ i pretpostavljena konfiguracija $\xi_t = [x', y', \theta']^T$.
Ovaj model pretpostavlja da se robot giba konstantnom translacijskom i rotacijskom brzinom po kružnici sa središtem u (x^*, y^*) i radijusa r^* u intervalu $\Delta t = \langle t-1, t]$. Ta će pretpostavka biti dobra samo ako se uzimaju male vrijednosti za Δt. Na temelju geometrijske analize (više u [8]) možemo dobiti potreban upravljački ulaz $\hat{u} = [\hat{v}, \hat{\omega}]^T$ koji će napraviti tranziciju iz $[x, y]^T$ u $[x', y']^T$. Također treba uzeti u obzir i to da zbog šuma u gibanju i grube pretpostavke o gibanju po

Algoritam 3 Model gibanja zasnovan na brzini

procedure MOTIONMODELVELOCITY(ξ_{t-1}, u_t, ξ_t)
  $\mu = \frac{1}{2}\frac{(x-x')\cos\theta+(y-y')\sin\theta}{(y-y')\cos\theta-(x-x')\sin\theta}$
  $x^* = \frac{x+x'}{2} + \mu(y-y')$
  $y^* = \frac{y+y'}{2} + \mu(x'-x)$
  $r^* = \sqrt{(x-x^*)^2+(y-y^*)^2}$
  $\Delta\theta = \arctan\frac{y'-y^*}{x'-x^*} - \arctan\frac{y-y^*}{x-x^*}$
  $\hat{v} = \frac{\Delta\theta}{\Delta t}r^*$
  $\hat{\omega} = \frac{\Delta\theta}{\Delta t}$
  return NORMAL$(v-\hat{v}, \alpha_1|v| + \alpha_2|\omega|) \cdot$ NORMAL$(\omega-\hat{\omega}, \alpha_3|v| + \alpha_4|\omega|)) \cdot$ NORMAL$(\frac{\theta'-\theta}{\Delta t} - \hat{\omega}, \alpha_5|v| + \alpha_6|\omega|))$

kružnici konačna orijentacija nakon gibanja može biti drugačija u odnosu na očekivanu. Uzimajući u obzir šumove pri gibanju i promatrajući greške translacijske brzine, kutne brzine i konačne orijentacije (pri čemu je očekivano da nema dodatne orijentacije robota na kraju gibanja) te pod pretpostavkom da su te greške nezavisne, možemo računati njihove vjerojatnosti pri čemu koristimo normalnu distribuciju s očekivanjem 0 i varijancom koja je proporcionalna upravljačkim brzinama. Parametri $\alpha_i \geq 0, i = 1, \ldots 6$ specifični su za svakog robota te ukoliko je robot manje precizan, tada su ti parametri veći.
Za pojedine će algoritme procjenjivanja stanja (npr. čestični filter) biti potrebno uzorkovati slučajno stanje ξ_t prema modelu gibanja $p(\xi_t|u_t, \xi_{t-1})$ na temelju ulaza u_t i prošlog stanja ξ_{t-1}.

Algoritam 4 Uzorkovanje modela gibanja zasnovanog na brzini

procedure SAMPLEMOTIONMODELVELOCITY(u_t, ξ_t)
  $\hat{v} = v +$ SAMPLE$(\alpha_1|v| + \alpha_2|\omega|)$
  $\hat{\omega} = \omega +$ SAMPLE$(\alpha_3|v| + \alpha_4|\omega|)$
  $\hat{\gamma} =$ SAMPLE$(\alpha_5|v| + \alpha_6|\omega|)$
  $x' = x - \frac{\hat{v}}{\hat{\omega}}\sin\theta + \frac{\hat{v}}{\hat{\omega}}\sin(\theta + \hat{\omega}\Delta t)$
  $y' = y + \frac{\hat{v}}{\hat{\omega}}\cos\theta - \frac{\hat{v}}{\hat{\omega}}\cos(\theta + \hat{\omega}\Delta t)$
  $\theta' = \theta + \hat{\omega}\Delta t + \hat{\gamma}\Delta t$
  return $\xi_t = [x', y', \theta']^T$

Možemo vidjeti da se ulazne brzine perturbiraju šumom te se iz perturbiranih brzina definira novo stane. Primijetimo da se pri tome uključuje dodatna perturbacija konačne orijentacije. Funkcija SAMPLE generira slučajan uzorak iz normalne distribucije s očekivanjem 0 i varijancom koja je proporcionalna upravljačkim brzinama.

Model zasnovan na odometriji

Upravljački ulaz odometrijskog modela sastoji se od mjerenja s odometrijskih senzora. Informacije koje dobijemo od tih mjerenja predstavljaju relativni pomak s

$\bar{\xi}_{t-1} = [\bar{x}, \bar{y}, \bar{\theta}]^T$ u $\bar{\xi}_t = [\bar{x}', \bar{y}', \bar{\theta}']^T$. Upravo će taj pomak biti dobar aproksimator između pravih stanja ξ_{t-1} i ξ_t jer ćemo koristiti svojstvo da između svaka dva stanja postoji jedinstven vektor $[\delta_{rot_1}, \delta_{trans}, \delta_{rot_2}]^T$ koji radi tranziciju iz jednog stanja u drugo, odnosno iz prvog se stanja rotacijom i translatiranjem dovodi u drugo stanje nakon čega slijedi radi završna rotacija zbog šumova i grubih pretpostavki gibanja.

Algoritam 5 Model zasnovan na odometriji

procedure MOTIONMODELODOMETRY(ξ_{t-1}, u_t, ξ_t)
  $\delta_{rot_1} = \arctan \frac{\bar{y}' - \bar{y}}{\bar{x}' - \bar{x}} - \bar{\theta}$
  $\delta_{trans} = \sqrt{(\bar{x} - \bar{x}')^2 + (\bar{y} - \bar{y}')^2}$
  $\delta_{rot_2} = \bar{\theta}' - \bar{\theta} - \delta_{rot_1}$
  $\hat{\delta}_{rot_1} = \arctan \frac{y' - y}{x' - x} - \theta$
  $\hat{\delta}_{trans} = \sqrt{(x - x')^2 + (y - y')^2}$
  $\hat{\delta}_{rot_2} = \theta' - \theta - \hat{\delta}_{rot_1}$
  $p_1 = \text{NORMAL}(\delta_{rot_1} - \hat{\delta}_{rot_1}, \alpha_1 \hat{\delta}_{rot_1} + \alpha_2 \hat{\delta}_{trans})$
  $p_2 = \text{NORMAL}(\delta_{trans} - \hat{\delta}_{trans}, \alpha_3 \hat{\delta}_{trans} + \alpha_4 (\hat{\delta}_{rot_1} + \hat{\delta}_{rot_2}))$
  $p_3 = \text{NORMAL}(\delta_{rot_2} - \hat{\delta}_{rot_2}, \alpha_1 \hat{\delta}_{rot_2} + \alpha_2 \hat{\delta}_{trans})$
  return $p_1 \cdot p_2 \cdot p_3$

Algoritam 6 Uzorkovanje odometrijskog modela gibanja

procedure SAMPLEMOTIONMODELODOMETRY(u_t, ξ_{t-1})
  $\delta_{rot_1} = \arctan \frac{\bar{y}' - \bar{y}}{\bar{x}' - \bar{x}} - \bar{\theta}$
  $\delta_{trans} = \sqrt{(\bar{x} - \bar{x}')^2 + (\bar{y} - \bar{y}')^2}$
  $\delta_{rot_2} = \bar{\theta}' - \bar{\theta} - \delta_{rot_1}$
  $\hat{\delta}_{rot_1} = \delta_{rot_1} - \text{SAMPLE}(\alpha_1 \delta_{rot_1} + \alpha_2 \delta_{trans})$
  $\hat{\delta}_{trans} = \delta_{trans} - \text{SAMPLE}(\alpha_3 \delta_{trans} + \alpha_4 (\delta_{rot_1} + \delta_{rot_2}))$
  $\hat{\delta}_{rot_2} = \delta_{rot_2} - \text{SAMPLE}(\alpha_1 \delta_{rot_2} + \alpha_2 \delta_{trans})$
  $x' = x + \hat{\delta}_{trans} \cos(\theta + \hat{\delta}_{rot_1})$
  $y' = y + \hat{\delta}_{trans} \sin(\theta + \hat{\delta}_{rot_1})$
  $\theta' = \theta + \hat{\delta}_{rot_1} + \hat{\delta}_{rot_2}$
  return $\xi_t = [x', y', \theta']^T$

U algoritmu 5 možemo vidjeti kako se vjerojatnosti greške u rotacijama i translaciji računaju. Uspoređuje se vektor tranzicije na temelju mjerenja s odometrijskog senzora $\bar{\xi}_t$ i $\bar{\xi}_{t-1}$ i vektor tranzicije na temelju predviđenog stanja ξ_t i prošlog stanja ξ_{t-1}. Tada računamo vjerojatnosti razlike odgovarajućih elemenata od tih vektora, a te razlike predstavljaju pogreške u odometriji koje su modelirane normalnom distribucijom jer uključujemo šum. Konačnu vjerojatnost dobijemo tako što pretpostavljamo da su pogreške pojedinačnih parametara nezavisne. Koeficijenti $\alpha_i, i = 1, \ldots, 4$ specifični su za svakog robota.
Analogno kao u modelu gibanja zasnovanom na brzini možemo definirati uzorkovanje u kojem će se novo stanje slučajno generirati kao što je prikazano u al-

goritmu 6. Ideja je ista kao u zatvorenoj formi, samo malo drugačije iskazana. Dakle pretpostavljamo da odometrijska mjerenja sadrže šumove modelirane normalnom distribucijom te na temelju jednostavne geometrijske analize dobijemo novo slučajno generirano stanje.

Modeli gibanja s informacijama iz mape

Postavlja se pitanje kako modelirati model gibanja poznavajući mapu m:

$$p(\xi_t|u_t, \xi_{t-1}, m). \tag{3.1}$$

Ako mapa m sadrži informacije koje su bitne za procjenu stanja (kao što je u mrežastim mapama zauzetosti gdje imamo informacije o tome gdje se objekti nalaze) tada vrijedi:

$$p(\xi_t|u_t, \xi_{t-1}) \neq p(\xi_t|u_t, \xi_{t-1}, m).$$

Računanje je tog modela kompleksno jer se mora uzeti u obzir vjerojatnost da postoji neokupirani put između ξ_{t-1} i ξ_t zajedno s vjerojatnošću da će taj put robot uspjeti pratiti na temelju ulaza u_t zbog nesigurnosti/šumova. No pod pretpostavkama da su udaljenosti između ξ_{t-1} i ξ_t dovoljno male i da se ažuriranja stanja rade dovoljno često, tada model 3.1 možemo aproksimirati koristeći Bayesovo pravilo:

$$\begin{aligned} p(\xi_t|u_t, \xi_{t-1}, m) &= \eta_1\, p(m|\xi_t, u_t, \xi_{t-1})\, p(\xi_t|u_t, \xi_{t-1}) \\ &= \eta_1\, p(m|\xi_t)\, p(\xi_t|u_t, \xi_{t-1}) \\ &= \eta_1 \frac{p(\xi_t|m)\, p(m)}{p(\xi_t)}\, p(\xi_t|u_t, \xi_{t-1}) \\ &= \eta_2 \frac{p(\xi_t|m)\, p(\xi_t|u_t, \xi_{t-1})}{p(\xi_t)} \\ &= \eta_3\, p(\xi_t|m)\, p(\xi_t|u_t, \xi_{t-1}) \end{aligned}$$

Često je $p(\xi_t)$ uniformna pa se može također smatrati normalizacijskom konstantom. U mrežastim mapama zauzetosti $p(\xi_t|m)$ možemo računati tako da je postavljamo na 0 kada je ćelija u kojoj bi se robot mogao nalazi zauzeta. Sada možemo na temelju ove aproksimacije dati modele gibanja s informacijama iz mape u zatvorenoj formi 7 i u formi za uzorkovanje 8:

Algoritam 7

1: **procedure** MotionModelWithMap(ξ_{t-1}, u_t, ξ_t, m)
2: **return** $p(\xi_t|u_t, \xi_{t-1}) \cdot p(\xi_t|m)$

Primijetimo da u formi za uzorkovanje imamo računanje težinskog koeficijenta koji je proporcionalan $p(\xi_t|m)$.

3.2.2 Vjerojatnosni modeli mjerenja

Vjerojatnosni model mjerenja definiran je kao uvjetna distribucija $p(z_t|\xi_t, m)$. Takvi modeli opisuju vjerojatnost da će senzori dati mjerenje z_t s obzirom na poznavanje mape m i stanja robota ξ_t. Mjerenje z_t može se sastojati od više mjerenja.

Algoritam 8

1: **procedure** SAMPLEMOTIONMODELWITHMAP(ξ_{t-1}, u_t, m)
2: $\pi = 0$
3: **while** $\pi \not> 0$ **do**
4: $\xi_t = \text{SAMPLEMOTIONMODEL}(u_t, \xi_{t-1})$
5: $\pi = p(\xi_t|m)$
6: **return** (ξ_t, π)

To se može dogoditi kada imamo više senzora (bilo istih ili različitih vrsta) ili kada jedan senzor generira više mjerenja. Tada ćemo broj mjerenja u trenutku t označiti s K pa onda z_t možemo zapisati kao $z_t = \{z_t^1, \ldots, z_t^K\}$. Među najpopularnijim senzorima u robotici su senzori udaljenosti (laserski, ultrazvučni) koji mjere udaljenost do objekata. Stoga ćemo upravo za tu vrstu senzora dati vjerojatnosni model mjerenja. Taj će model sadržavati tri vrste grešaka/nepouzdanosti koje se mogu javljati prilikom mjerenja: šum prilikom mjerenja, greška senzora i slučajnost mjerenja. Željeni se model tada definira kao mješavina tih triju grešaka.

Šum prilikom mjerenja

U idealnom svijetu senzor udaljenosti uvijek bi vraćao točnu izmjerenu vrijednost do najbližeg objekta u mapi. Tu točnu vrijednost možemo označiti s z_t^{k*} prilikom mjerenja z_t^k. No kako su mjerenja sa senzora podložni različitim šumovima teško ćemo znati da je $z_t^k = z_t^{k*}$. Takav šum možemo modelirati Gaussovom distribucijom. Pomoću koordinata objekta izmjerenih s z_t^k i projiciranjem u globalni referentni sustav izračunat ćemo udaljenost d_{obs} do najbližeg objekta na mapi m. Tada vjerojatnost mjerenja možemo označiti s p_{hit} i definirati kao

$$p_{hit}(z_t^k|\xi_t, m) = \varepsilon_{\sigma_{hit}}(d_{obs}), \varepsilon_{\sigma_{hit}} \sim \mathcal{N}(0, \sigma_{hit}^2).$$

Greška senzora

Moguće je da se dogodi da senzor ne uspije očitati objekt. To je moguće iz više razloga, a tada će senzor vratiti maksimalnu moguću vrijednost z_{max}. Kako takvi slučajevi nisu rijetki, tada bismo to trebali modelirati:

$$p_{max}(z_t^k|\xi_t, m) = I(z = z_{max}) = \begin{cases} 1 & ako\ z = z_{max} \\ 0 & inače \end{cases}$$

Primijetimo da p_{max} ne predstavlja kontinuiranu distribuciju, nego diskretnu distribuciju.

Slučajnost mjerenja

Događa se i to da senzori udaljenosti generiraju potpuno neobjašnjiva mjerenja. Takva ćemo mjerenja modelirati uniformnom distribucijom nad intervalom vri-

jednosti mjerenja $[0, z_{max}]$:

$$p_{rand}(z_t^k|\xi_t, m) = \begin{cases} \frac{1}{z_{max}} & ako\ 0 \leq z_t^k < z_{max} \\ 0 & inače \end{cases}.$$

Sada distribuciju $p(z_t^k|\xi_t, m)$ možemo zapisati kao:

$$p(z_t^k|\xi_t, m) = \begin{bmatrix} z_{hit} \\ z_{rand} \\ z_{max} \end{bmatrix}^T \begin{bmatrix} p_{hit}(z_t^k|\xi_t, m) \\ p_{rand}(z_t^k|\xi_t, m) \\ p_{max}(z_t^k|\xi_t, m) \end{bmatrix},$$

pri čemu su z_{hit}, z_{rand} i z_{max} težine za koje vrijedi $z_{hit} + z_{rand} + z_{max} = 1$. One (uključujući i σ_{hit}) se mogu odrediti ručno, a mogu i pomoću algoritama poput metode maksimalne vjerodostojnosti na temelju stvarnih mjerenja (više u [8]). Sada možemo navesti algoritam koji će obuhvatiti sve navedeno:

Algoritam 9 Polja izglednosti za senzore udaljenost

1: **procedure** LIKELIHOODFIELDSFORRANGEFINDERS(z_t, ξ_t, m)
2: $q = 1$
3: **for** $k = 1, \ldots, K$ **do**
4: **if** $z_t^k \neq z_{max}$ **then**
5: $x_{z_t^k} = x + x_{k,sens} \cos\theta - y_{k,sens} \sin\theta + z_t^k \cos(\theta + \theta_{k,sens})$
6: $y_{z_t^k} = y + y_{k,sens} \cos\theta + x_{k,sens} \sin\theta + z_t^k \sin(\theta + \theta_{k,sens})$
7: $d_{obs} = \min_{x',y'} \{(x_{z_t^k} - x')^2 + (y_{z_t^k} - y')^2 | (x', y')$ occupied in $m\}$
8: $q = q \cdot (z_{hit} \cdot \mathcal{N}(d_{obs}; 0, \sigma_{hit}^2) + \frac{z_{rand}}{z_{max}})$
return q

Primijetimo da u linijama 5 i 6 projiciramo mjerenja u globalni referenti okvir na temelju stanja robota i relativne pozicije senzora u odnosu na referentni okvir robota. Najskuplja operacija po pitanju računske složenosti u ovom algoritmu je u liniji 7 gdje se traži najbliža zauzeta pozicija na mapi. Primijetimo da u liniji 8 implicitno koristimo sljedeću aproksimaciju:

$$p(z_t|\xi_t, m) = \prod_{k=1}^{K} p(z_t^k|\xi_t, m).$$

Tu pretpostavljamo da su šumovi mjerenja međusobno nezavisni. To i je istina samo u idealnim slučajevima, ali generalno ta pretpostavka ne vrijedi iako i dalje daje dobre rezultate.

3.2.3 Monte Carlo lokalizacija

Opisani Bayesov filter 2 daje općeniti oblik za računanje distribucije uvjerenja $bel(\xi_t)$ bez ikakvog saznanja o mapi. Kada uključimo saznanja o mapi tada govorimo o problemu lokalizacije. Monte Carlo lokalizacija je algoritam koji računa

Algoritam 10 Monte Carlo lokalizacija

1: **procedure** MCL($\mathcal{X}_{t-1}, u_t, z_t, m$)
2: $\overline{\mathcal{X}}_t = \mathcal{X}_t = \varnothing$
3: **for** $k = 1, \ldots, M$ **do**
4: $\xi_t^{[k]} = \text{SAMPLEMOTIONMODEL}(u_t, \xi_{t-1}^{[k]})$
5: $w_t^{[k]} = \text{MEASURMENTMODEL}(z_t, \xi_t^{[k]}, m)$
6: $\bar{\mathcal{X}}_{t-1} = \bar{\mathcal{X}}_{t-1} \cup \{(\xi_t^{[k]}, w_t^{[k]})\}$
7: **for** $k = 1, \ldots, M$ **do**
8: draw i with probability $\propto w_t^{[i]}$
9: $\mathcal{X}_t = \mathcal{X}_t \cup \{\xi_t^{[i]}\}$
return $\mathcal{X}_t$

distribuciju uvjerenja $bel(\xi_t)$ pomoću skupa čestica $\mathcal{X}_t = \{\xi_t^{[1]}, \ldots, \xi_t^{[M]}\}, |\mathcal{X}_t| = M$. Svaka čestica $\xi_t^{[k]}$ predstavlja pretpostavku o stanju sustava te što se više čestica nalazi na nekom dijelu prostora stanja, to je više vjerojatno da je pravo stanje robota u tom dijelu. Kada ovakve čestice koristimo u procjenjivanju distribucije uvjerenja bez uključivanja informacija iz mape, tada govorimo o čestičnom filtru koji je varijanta Bayesovog filtra. Čestični filter radi analognu stvar kao i MCL tako da istovremeno možemo pratiti način funkcioniranja oba algoritma objašnjavajući samo jednog od njih. Možemo primijetiti da MCL konstruira distribuciju uvjerenja $bel(\xi_t)$ rekurzivno od distribucije uvjerenja $bel(\xi_{t-1})$ iz prethodnog koraka. To se radi tako da se skup čestica rekurzivno konstruira na temelju prethodnih čestica: čestica $\xi_t^{[k]}$ se kreira na temelju $\xi_{t-1}^{[k]}$ i ulaza u_t tako da se uzorkuje iz distribucije $p(\xi_t|u_t, \xi_{t-1})$. Kada se tako kreira M čestica, one onda predstavljaju $\overline{bel}(\xi_t)$. Pri tome za svaku se česticu računa težinski koeficijent $w_t^{[k]}$ s vjerojatnosnim modelom mjerenja $p(z_t|\xi_t^{[k]}, m)$. Taj će koeficijent imati važnu ulogu u sljedećem koraku gdje se svaka čestica ponovno uzorkuje iz novonastalog privremenog skupa s vjerojatnošću proporcionalnom tom koeficijentu. Tako će nastati skup koji će aproksimirati distribuciju uvjerenja $bel(\xi_t)$. Primijetimo da smo u uzorkovanju iz modela gibanja izostavili informacije iz mape m što nije problem kod ovakvog pristupa kada radimo s velikim brojem čestica i kada računamo težinske koeficijente koji nam govore koliko je releventna pojedina čestica. Moguće je definirati koliko često će se ponovno uzorkovanje događati s glavnim ciljem da se ne izgubi raznolikost uzoraka tako da se definira mjera raspršenosti težinskih koeficijenata. Postoje i druge varijante ovog algoritma gdje se ne radi korak ponovnog uzorkovanja nego se za svaku česticu ažurira težinski koeficijent prema sljedećoj formuli

$$w_t^{[k]} = p(z_t|\xi_t^{[k]}, m) \cdot w_{t-1}^{[k]}.$$

Zbog rekurzivnog je uzorka potrebno definirati inicijalno uvjerenje $bel(\xi_0)$. To se može napraviti tako da se nasumično generiraju čestice iz apriorne distribucije $p(\xi_0)$ i onda im se dodijeli uniformni težinski koeficijent M^{-1}. Velika je prednost MCL-a da može aproksimirati gotovo svaku praktično važnu distribuciju. No da

bi se u tome uspjelo i izbjeglo moguće probleme u aproksimaciji, potrebno je izbjegavati korištenje malog broja čestica (više o tome u [8]).

3.3 Mapiranje

Mapiranje je proces u kojem mobilni robot izgrađuje mapu okoline u kojoj se nalazi. Mapiranje je moguće napraviti tako da se ručno izgrađena mapa preda robotu, a moguće je i da robot autonomno izgrađuje mapu. Izgradnja mapa može biti zahtjevan proces, a to ovisi o više faktora. Ako je prostor veći u odnosu na perceptivni doseg robota, tada je teže izgrađivati mapu kao i onda kada je šum u percepciji i gibanju velik. Problematično je i onda kada se javlja određena dvosmislenost u percepciji, odnosno kada se češće događa da različita mjesta izgledaju slično. U ovom ćemo poglavlju predstaviti algoritam izgradnje jedne vrste mrežastih mapa zauzetosti čija je osnovna ideja prikazati mapu u mreži jednako raspoređenih slučajnih varijabli koje su binarne i koje reprezentiraju zauzetost. Mrežaste mape zauzetosti imaju vrlo važnu ulogu u procesu planiranja putanje jer svojim načinom reprezentacije jako dobro odgovaraju većini algoritama koji postoje za pronalazak putova (npr. u teoriji grafova).

3.3.1 Mrežaste mape zauzetosti

Dakle u izgradnji mrežastih mapa zauzetosti treba odrediti aposteriornu distribuciju mapa na temelju stanja robota i mjerenja:

$$p(m|z_{1\to t}, \xi_{1\to t}). \tag{3.2}$$

Možemo primijetiti da upravljački ulazi $u_{1\to t}$ nisu navedeni jer nam nisu potrebni kada znamo trajektoriju robota. Raspodijelimo kontinuiran prostor kao:

$$m = \{\mathbf{m}_i\},$$

pri čemu je $\mathbf{m}_i$ ćelija s indeksom i koja može poprimiti samo dvije vrijednosti. Tako ćemo definirati vjerojatnost zauzetosti ćelije $p(\mathbf{m}_i = 1) = p(\mathbf{m}_i)$ i vjerojatnost da nije zauzeta $p(\mathbf{m}_i = 0)$. U praksi se veliki broj ćelija koristi za izgradnju mape, što znači da kada imamo mapu od 10000 ćelija, onda ukupan broj mapa iznosi 2^{10000}, što bi značilo da bi bilo nemoguće računati aposteriornu distribuciju 3.2 za svaku mapu m. Zato se u algoritmima za izgradnju mrežastih mapa zauzetosti problem aposteriorne distribucije 3.2 aproksimira:

$$p(m|z_{1\to t}, \xi_{1\to t}) = \prod_i p(\mathbf{m}_i|z_{1\to t}, \xi_{1\to t}).$$

Ta je aproksimacija pogodna jer se time svodi na računanje binarnog problema $p(\mathbf{m}_i|z_{1\to t}, \xi_{1\to t})$, pri čemu pretpostavljamo da se mapa tijekom vremena ne mijenja, iako tada nije moguće reprezentirati ovisnosti susjednih ćelija.
Algoritam 11 reprezentira zauzetost ćelija pomoću omjera vjerojatnosti na koji djeluje logaritamska funkcija (eng. *log odds representation*):

Algoritam 11

1: **procedure** OCCUPANCYGRIDMAPPING($\{l_{t-1,i}\}, \xi_t, z_t$)
2: **for** $\mathbf{m}_i \in m$ **do**
3: **if** $\mathbf{m}_i$ in perception of z_t **then**
4: $l_{t,i} = l_{t-1,i} + \text{INVERSESENSORMODEL}(\mathbf{m}_i, \xi_z, z_t) - l_0$
5: **else**
6: $l_{t,i} = l_{t-1,i}$
return $\{l_{t,i}\}$

$$l_{t,i} = \log \frac{p(\mathbf{m}_i | z_{1\to t}, \xi_{1\to t})}{1 - p(\mathbf{m}_i | z_{1\to t}, \xi_{1\to t})}.$$

Prednost je ovakve reprezentacije u tome da se može izbjeći numerička nestabilnost za vjerojatnosti blizu 1 ili 0. Vjerojatnost zauzetosti ćelije tada možemo dobiti preko:

$$p(\mathbf{m}_i | z_{1\to t}, \xi_{1\to t}) = 1 - \frac{1}{1 + e^{l_{t,i}}}.$$

Algoritam 11 prolazi kroz sve ćelije te ažurira samo one koje su u percepciji mjerenja z_t sa senzora. To se ažuriranje radi preko inverznog modela senzora. Inverzni model senzora implementira inverzni model mjerenja $p(\mathbf{m}_i | z_t, \xi_t)$ u *log odds* reprezentaciji:

$$\text{INVERSESENZORMODEL}(\mathbf{m}_i, \xi_t, z_t) = \log \frac{p(\mathbf{m}_i | z_t, \xi_t)}{1 - p(\mathbf{m}_i | z_t, \xi_t)}$$

Možemo navesti konkretan primjer inverznog modela senzora za senzore udaljenosti.

Algoritam 12 Inverzni model senzora udaljenosti

1: **procedure** INVERSERANGESENZORMODEL($\mathbf{m}_i, \xi_t = [x, y, \theta]^T, z_t$)
2: $x_i, y_i =$ center of m_i
3: $r = \sqrt{(x_i - x)^2 + (y_i - y)^2}$
4: $\phi = \arctan(\frac{y_i - y}{x_i - x}) - \theta$
5: $k = \text{argmin}_j |\phi - \theta_{j,sens}|$
6: **if** $r > \min(z_{max}, z_t^k + \frac{\alpha}{2})$ or $|\phi - \theta_{k,sens}| > \frac{\beta}{2}$ **then**
7: **return** l_0
8: **if** $z_t^k < z_{max}$ and $|r - z_t^k| < \frac{\alpha}{2}$ **then**
9: **return** l_{occ}
10: **if** $r \leq z_t^k$ **then**
11: **return** l_{free}

Ideja je algoritma da svim ćelijama koje se nalaze u rasponu senzora te čija je udaljenost od robota blizu vrijednosti mjerenja proglasi zauzetima. Preko parametra

α definira se ta blizina, odnosno debljina prepreke, a preko parametra β definira se kutni otvor zrake. Prvo se određuje indeks najbliže zrake k i udaljenost r od središta ćelije $\mathbf{m}_i$. Ako se ćelija nalazi izvan zrake senzora ili se nalazi na udaljenosti većoj od $\frac{\alpha}{2}$ od mjerenja z_t^k tada će se vratiti početna vrijednost ćelije. Kada se ćelija nalazi unutar otvorene okoline oko z_t^k pri čemu je ta duljina okoline jednaka α tada će se vratiti l_{occ}. Vrijednost l_{free} vratit će se kada je udaljenost r kraća od z_t^k za više od $\frac{\alpha}{2}$. Primijetimo da je l_0 početna distribucija zauzetosti ćelije koja je dana s:

$$l_0 = \log \frac{p(\mathbf{m}_i = 1)}{p(\mathbf{m}_i = 0)} = \log \frac{p(\mathbf{m}_i)}{1 - p(\mathbf{m}_i)}$$

Izgradnja mape s više tipova senzora

Ukoliko se više tipova senzora nalazi na robotu, postavlja se pitanje kako najbolje integrirati dobivene informacije s tih senzora u jednu mapu. Jedan popularan način je za svaki tip senzora graditi posebnu mapu te onda te mape kombinirati koristeći prikladnu funkciju. Na primjer, ako s $m^k = \{\mathbf{m}_i^k\}$ označimo dobivenu mapu s k-tog tipa senzora i ako su mjerenja sa senzora neovisna, tada ih možemo kombinirati koristeći De Morganov zakon:

$$p(\mathbf{m}_i) = 1 - \prod_k (1 - p(\mathbf{m}_i^k).$$

Možemo pristupiti i na drugi način, odnosno računati maksimum od svih dobivenih mapa pa proglasiti neku ćelijom zauzetom ako barem jedan senzor tako pretpostavi:

$$p(\mathbf{m}_i) = \max_k p(\mathbf{m}_i^k).$$

3.4 SLAM

Sada dolazimo do jednog od vrlo bitnih problema u mobilnoj robotici, a to je problem istovremene lokalizacije i mapiranja (eng. *simultaneous localization and mapping*, abbr. *SLAM*). Taj se problem javlja kada mobilni robot nema pristup mapi okoline niti ne zna gdje se nalazi. Ono što mu je dostupno jesu mjerenja $z_{1\to t}$ i upravljački ulazi $u_{1\to t}$. Tada robot pokušava izgraditi mapu prostora u kojem se nalazi, a pri tome istovremeno se pokušava lokalizirati s obzirom na tu mapu. SLAM je bitno teži problem i od lokalizacije kada je mapa poznata i od mapiranja s poznatim stanjima robota. SLAM možemo podijeliti na dvije forme: *online* SLAM i *full* SLAM. *Online* SLAM problem sastoji se od procjenjivanja aposteriorne distribucije trenutnog stanja i mape:

$$p(\xi_t, m | z_{1\to t}, u_{1\to t}).$$

Mnogi algoritmi za *online* SLAM problem zanemaruju prošla mjerenja i upravljačke ulaze jednom kada su procesirani. *Full* SLAM problem procjenjuje aposte-

riornu distribuciju cijele trajektorije robota $\xi_{1\to t}$ zajedno s mapom m:

$$p(\xi_{1\to t}, m|z_{1\to t}, u_{1\to t}).$$

Upravo ta razlika tih dvaju problema ima određene posljedice u algoritmima koji se mogu primjenjivati za rješavanje. Možemo vidjeti da je *online* SLAM problem rezultat integracije svih prošlih konfiguracija od *full* SLAM problema:

$$p(\xi_t, m|z_{1\to t}, u_{1\to t}) = \int\int\cdots\int p(\xi_{1\to t}, m|z_{1\to t}, u_{1\to t})\, d\xi_1\, d\xi_2 \ldots d\xi_{t-1}.$$

3.4.1 FastSLAM

Jedan je od načina rješavanja SLAM problema korištenje čestičnog filtra s kojim smo se susreli u lokalizaciji (algoritam 10). Tako ćemo dobiti FastSLAM algoritam koji je efikasan, koji ima sposobnost modeliranja nelinearnog gibanja robota te istovremenog rješavanja *full* SLAM i *online* SLAM problema. To je moguće jer je FastSLAM formuliran da računa aposteriornu distribuciju trajektorije, no kako u svakom koraku čestični filter procjenjuje jedno po jedno stanje, tada se time rješava *online* SLAM problem. Mi ćemo se u nastavku baviti primjenom FastSLAM algoritma u kontekstu mrežastih mapa zauzetosti. Promotrimo sljedeću faktorizaciju uvjetne vjerojatnosti:

$$\begin{aligned} p(\xi_{1\to t}, m|z_{1\to t}, u_{1\to t}) &= p(m|\xi_{1\to t}, z_{1\to t}, u_{1\to t})\, p(\xi_{1\to t}|z_{1\to t}, u_{1\to t}) \\ &= p(m|\xi_{1\to t}, z_{1\to t})\, p(\xi_{1\to t}|z_{1\to t}, u_{1\to t}). \end{aligned}$$

Temeljem ove faktorizacije svaka će generirana čestica $\xi_t^{[k]}$ u trenutku t dati svoju aproksimaciju stanja robota i pri tome sadržavati svoju aproksimaciju mape $m_t^{[k]}$. Za to će nam biti potrebne iste funkcionalnosti kao u lokalizaciji gdje ćemo uzorkovanje čestica raditi na temelju prethodnih čestica, a pomoću modela mjerenja računat ćemo težinske koeficijente. Nova je funkcionalnost u odnosu na lokalizaciju ta što ćemo dodatno ažurirati odgovarajuću mapu pomoću te nove čestice i najnovijeg mjerenja.

Algoritam 13

1: **procedure** FastSLAM($\mathcal{X}_{t-1}, u_t, z_t, m$)
2: $\bar{\mathcal{X}}_t = \mathcal{X}_t = \varnothing$
3: **for** $k = 1, \ldots, M$ **do**
4: $\xi_t^{[k]} = \text{sampleMotionModel}(u_t, \xi_{t-1}^{[k]})$
5: $w_t^{[k]} = \text{measurmentModel}(z_t, \xi_t^{[k]}, m_{t-1}^{[k]})$
6: $m_t^{[k]} = \text{occupancyGrid}(z_t, \xi_t^{[k]}, m_{t-1}^{[k]})$
7: $\bar{\mathcal{X}}_{t-1} = \bar{\mathcal{X}}_{t-1} \cup \{(\xi_t^{[k]}, w_t^{[k]}, m_t^{[k]})\}$
8: **for** $k = 1, \ldots, M$ **do**
9: draw i with probability $\propto w_t^{[i]}$
10: $\mathcal{X}_t = \mathcal{X}_t \cup \{(\xi_t^{[i]}, m_t^{[k]})\}$
return $\mathcal{X}_t$

4 | Navigacija mobilnog robota

Navigacija je sposobnost robota da na temelju informacija iz okoline i ostalih informacija kojima može pristupiti postupa tako da efikasno i pouzdano dostigne ciljeve. Navigacija sadrži dva bitna pogleda koji čine snažnu komplementarnost: planiranje putanje i izbjegavanje prepreka, odnosno reakcija na prepreke. Algoritmi planiranja putanje pronalaze trajektoriju od početne lokacije do ciljne lokacije na mapi. Algoritmi izbjegavanja prepreka odnosno reakcije pronalaze načine za prilagođavanje postojeće trajektorije u slučaju da robot naiđe na nekakvu prepreku koja se može očitovati na senzorima.
Mnogo algoritama za planiranje putanje koriste diskretne/dekomponirane mape. Ukoliko mape nisu dekomponirane postoje razni načini kako ju dekomponirati. Jedan je od načina fiksna dekompozicija gdje unaprijed podijelimo mapu na fiksan broj dijelova kao što je primjer s mrežastim mapama zauzetosti. Zato ćemo algoritme u nastavku promatrati s tom idejom.

4.1 A* algoritam

A* algoritam razvio se za potrebe projekta Shakey u kojem se željelo omogućiti mobilnom robotu optimalno planiranje njegove putanje. A* algoritam prezentirat ćemo koristeći notaciju grafova jer mnogi se problemi koji trebaju pronaći optimalne putove (npr. putove najkraće udaljenosti) mogu promatrati na grafovima. Osnovni je princip algoritma u svakom trenutku otkrivati optimalni put koji dovodi do cilja minimiziranjem funkcije troška. U svakom trenutku planiranja putanje algoritam sadrži potrebne informacije o do tada kreiranim putovima te se korak po korak ti putovi proširuju sve dok se ne dođe do konačnog cilja. Neka je S početni čvor u grafu i neka je $f(n)$ stvarni trošak optimalnog puta čiji je izvor u S i koji prolazi kroz čvor n. Tada vrijedi $f(n) = f(S)$ za svaki n koji se nalazi na optimalnom putu te $f(n) > f(S)$ za svaki n koji se ne nalazi na optimalnom putu. Funkciju $f(n)$ možemo raspisati na sljedeći način:

$$f(n) = g(n) + h(n),$$

pri čemu je $g(n)$ stvarni trošak optimalnog puta od početnog čvora S do čvora n, a $h(n)$ stvarni trošak optimalnog puta od čvora n do ciljnog čvora. No kako mi tražimo optimalan put, nama nije poznata $f(n)$ te ćemo stoga uvesti procjenu $\hat{f}(n)$. Kako bismo to uveli, uvest ćemo procjenu $\hat{g}(n)$ za $g(n)$ koja predstavlja najmanji trošak puta od početnog čvora S do čvora n što je algoritam uspio pronaći. Primijetimo da tada vrijedi $\hat{g}(n) \geq g(n)$. Zatim ćemo uvesti procjenu funkcije $h(n)$

heuristikom $\hat{h}(n)$. Taj odabir ovisi o problematici koju želimo riješiti. Primjerice, u slučaju kada želimo pronaći najkraću udaljenost između 2 grada koja povezuju ceste, zračna udaljenost tih dvaju gradova čini se jako dobrim odabirom. U tom slučaju primijetimo da vrijedi $\hat{h}(n) \leq h(n)$ što je inače generalno poželjno svojstvo za algoritam. Dakle funkcija troška $\hat{f}(n)$ definirana je kao:

$$\hat{f}(n) = \hat{g}(n) + \hat{h}(n)$$

te možemo prezentirati pseudokod algoritma.

Algoritam 14

```
1:  procedure A*(S, G)
2:      openSet = {S}
3:      closedSet = ∅
4:      while openSet not empty do
5:          n = pick from openSet with lowest f̂(n)
6:          if n = G then return True
7:          closedSet = closedSet ∪ {n}
8:          for neighbour x of n that is not in closedSet do
9:              if x in openSet then
10:                 if ĝ(n) + d(n, x) < ĝ(x) then update f̂(x)
11:             else
12:                 openSet = openSet ∪ {x}
        return False
```

Dobar odabir funkcije $\hat{h}(n)$ tako da vrijedi $\hat{h}(n) \leq h(n)$ rezultira time da će A* sigurno pronaći optimalan put u grafu ukoliko takav put postoji. Ukoliko vrijedi i $h(m,n) + \hat{h}(n) \geq h(m)$ za dva čvora m, n pri čemu je $h(m,n)$ stvarni trošak optimalnog puta između čvorova m i n, onda znači da vrijedi $\hat{g}(n) = g(n)$. To bi značilo da algoritam ne mora vraćati čvor iz skupa istraženih čvorova (closedSet) u skup neistraženih čvorova (openSet) jer svaki put kada se čvor postavi u skup istraženih čvorova tada je optimalni put od početnog čvora S do tog čvora pronađen. Uz ova dva svojstva te ukoliko vrijedi da je $\hat{f}(n) \neq \hat{f}(m), \forall m, n$, tada se može pokazati da će A* algoritam koristiti najmanji mogući broj čvorova za istraživanje. Više o ovim svojstvima može se naći u [2]. Moguće je odabrati i heuristiku $\hat{h}(n)$ za koju bi vrijedilo $\hat{h}(n) \geq h(n)$. Tada A* ne bi više bio optimalan u smislu da će sigurno pronaći optimalan put u grafu. Stoga rješenje koje bi se dobilo možda ne bilo optimalno, ali bi zato možda bilo puno brže. To nas dovodi do zaključka da se odabirom $\hat{h}(n)$ bira kompromis između optimalnosti i efikasnosti rješenja.

5 | Simulacija

U ovom ćemo poglavlju pogledati kako su obrađeni algoritmi implementirani tako da simuliramo gibanje robota u prostoru u kojem jedino znamo gdje se određeni dio prepreka nalazi. Tako će se robot kroz vrijeme trebati uspjeti lokalizirati i izgraditi čitavu mapu prostora, a pri tome će si samostalno izgrađivati putanju po unaprijed zadanim točkama prostora te određivati kutnu i translacijsku brzinu da bi se tom putanjom mogao gibati.

5.1 Glavna petlja

Prvo pogledajmo implementaciju glavne petlje 5.1. Na početku se svake iteracije traži novi upravljački ulaz za robota. Zatim se taj upravljački ulaz kroz kinematiku robota primjenjuje na simuliranog "teorijskog" robota bez šuma. Nakon toga simuliramo čitanje s enkodera te dobivene brzine primjenjujemo na robota koji nam predstavlja "stvarnog" robota. Namjerno prikazujemo robota bez šuma i robota sa šumom, odnosno "stvarnog" robota da pokažemo koliko se stvarni robot može razlikovati u odnosu na ono što mi želimo da bude. S enkodiranjem mi zadržavamo određenu preciznost i sigurnost pozicije robota, stoga ćemo na njemu simulirati mjerenje na koje ćemo se osvrnuti kasnije. Zatim grafički ažuriramo prikaz ultrazvučnih senzora na temelju simuliranih mjerenja zajedno s "teorijskog" robota i "stvarnog" robota. Nakon toga, da bismo ažurirali čestice s novim konfiguracijama, koristimo obrađeni odometrijski model gibanja 6. Za te nove čestice potrebno je izračunati težine pa se to odrađuje preko modela mjerenja 9. Svaka će čestica izgrađivati svoju mapu preko algoritma 11. Tu je dobro napomenuti da smo stavili gornju i donju među na vrijednosti koje ćelija može poprimiti u *log odds* reprezentaciji radi onemogućavanja poprimanja vrlo velikih, odnosno malih vrijednosti. Sada je potrebno ponovno uzorkovati čestice na temelju njihovih težina, no to ćemo raditi svaku petu iteraciju ili ako su čestice postale neefikasne u reprezentaciji stvarnog stanja, odnosno kada je varijanca njihovih težina postala velika (više u [1]). To se mjeri tako da se koriste normalizirane težine $\hat{w}$:

$$\frac{1}{\sum_{i=1}^{M} \hat{w}_i^2} < \frac{M}{2}.$$

Nakon toga najbolje će čestice procijeniti konfiguraciju robota i cjelokupnu mapu prostora. Postoji opasnost da čestice konvergiraju u jednu točku i da ta konvergencija bude kriva pa ćemo to probati izbjeći tako da ćemo na kraju glavne petlje svakih 60 iteracija probati raspršiti čestice u okolinu trenutne procjene/konvergencije

(u početku će ta okolina biti veća, a kasnije manja).

Implementacija 5.1: Glavna petlja

```
counter_resample = 0
def loop(i, robot, estimated_robot, enc_robot, particles,
         estimated_map, shapes, full_known_lines_env, dt):
  global counter_resample
  R_xi_dot = path_planner(np.array([enc_robot.z_t[2]], dtype=float),
                          enc_robot.I_xi, Map())

  phis_dot = robot.R_inverse_kinematics(R_xi_dot)
  I_xi_dot = robot.forward_kinematics(phis_dot)
  robot.update_state(I_xi_dot, dt)

  deltas_c = enc_robot.read_encoders(phis_dot, dt)
  phis_dot_enc = enc_robot.enc_deltas_to_phis_dot(deltas_c, dt)
  I_xi_dot_enc = enc_robot.forward_kinematics(phis_dot_enc)

  I_xi_prev = enc_robot.I_xi.copy()
  enc_robot.update_state(I_xi_dot_enc, dt)
  I_xi = enc_robot.I_xi.copy()

  measurement = enc_robot.simulate_measurement(full_known_lines_env)
  enc_robot.update_cones(measurement)

  update_wedge(shapes[1], robot.I_xi)
  update_wedge(shapes[3], enc_robot.I_xi)

  particles.I_xis = sample_motion_model_odometry(
    np.vstack([I_xi_prev, I_xi]), particles.I_xis)
  particles.normalize_angles()
  particles.update_colors(full_known_lines_env)
  shapes[4].set_offsets(particles.I_xis[:, :2])
  shapes[4].set_color(particles.particle_colors)

  particles.measurement_model(enc_robot)
  particles.weights[
    ~inside_polygon(particles.I_xis, full_known_lines_env[0:4])
  ] = 0.0
  particles.normalize_weights()

  particles.occupancy_grid_mapping(enc_robot)

  if (less_than(1 / np.sum(particles.weights ** 2), particles.M / 2) and i > 2) \
      or counter_resample == 4:
    random_resampling(particles)
    counter_resample = 0
  else:
    counter_resample += 1

  num_of_predictors = robot_settings["num_of_approximators"]
  best_found = np.argpartition(
    particles.weights, -num_of_predictors)[-num_of_predictors:]
  mean = np.mean(particles.I_xis[best_found], axis=0)
  mean[2] = normalize_angle(mean[2])
  estimated_robot.I_xi = mean

  estimated_map.log_odds_probabilities = np.zeros(
    (estimated_map.num_of_rows, estimated_map.num_of_columns),
    dtype=float)
  for particle_map in particles.maps[best_found]:
    estimated_map.log_odds_probabilities += particle_map.log_odds_probabilities
  estimated_map.log_odds_probabilities /= num_of_predictors
  estimated_map.synchronize_map()

  shapes[0].set_data(
    np.vectorize(log_odds_to_prob)(estimated_map.log_odds_probabilities))
  if i % 60 == 0 and i > 0:
```

```
66      n = int(particles.M * 0.6)
67      choice = np.random.choice(particles.M, n)
68      offset = 0.75
69      if i > 400:
70        offset = 0.35
71      particles.I_xis[choice, :2] = estimated_robot.I_xi[:2] \
72        + np.random.uniform(-offset, offset, (n, 2))
```

5.2 Planiranje putanje

U početku robot će se gibati prema naprijed dok to može. Kada bude vidio da su mu prepreke preblizu, onda će si kreirati novi put. Radi jednostavnosti cjelokupnog planiranja, unaprijed smo zadali točke na mapi pomoću kojih robot treba isplanirati kretanje. Put će do svake točke od trenutne pozicije računati preko A* algoritma 14. Taj će put biti reprezentiran u obliku liste ćelija pa je još potrebno definirati kojom se translacijskom i kutnom brzinom gibati između tih ćelija. To se radi u linijama 26-48 s idejom da robot prolazi po 3 ćelije u sekundi. Nakon tih izračuna on spremi te upravljačke ulaze tako da se pri svakom sljedećem pozivu vrati prvi sljedeći upravljački ulaz. Ukoliko se prepreka nađe blizu ili su svi upravljački ulazi za trenutnu točku na mapi iskorišteni, potrebno je ponovno isplanirati putanju. Kada su sve točke u prostoru prijeđene, simulacija će završiti.

Implementacija 5.2: Planiranje putanje

```
1  target_cells = [(30, 50), (49, 52), (50, 30), (51, 15),
2                  (12, 10), (8, 50), (30, 48), (30, 30)]
3  going_forward = True
4  inputs = []
5  def path_planner(z_t, estimated_state, estimated_map):
6    global target_cells, going_forward, inputs
7    if len(target_cells) == 0 and len(inputs) == 0:
8      exit()
9    if going_forward and obstacle_close(z_t, distance=0.75):
10     going_forward = False
11   elif going_forward:
12     # just keep going forward initially
13     return np.array([0.7, 0, 0], dtype=float)
14
15   if not obstacle_close(z_t, distance=0.15) and len(inputs) != 0:
16     return inputs.pop(0)
17   elif obstacle_close(z_t, distance=0.15):
18     target_cell = target_cells[0]
19   else:
20     target_cell = target_cells.pop(0)
21
22   start_cell = estimated_map.coordinates_2_indices(*estimated_state[:2])
23   map_for_navigation = prepare_map(estimated_map)
24   path = A_star(start_cell, target_cell, map_for_navigation)
25
26   estimated_orientation = estimated_state[2]
27   target_cell_visited = False
28   pairs = []
29   k = 3
30   previous_cell = start_cell
31   for index in range(k, len(path), k):
32     pairs.append((previous_cell, path[index]))
33     previous_cell = path[index]
34     if path[index] == target_cell:
```

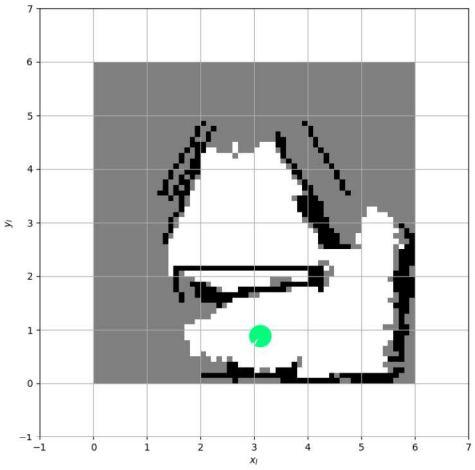
```
35        target_cell_visited = True
36    if not target_cell_visited:
37      pairs.append((previous_cell, target_cell))
38    for cellA, cellB in pairs:
39      cellA_x, cellA_y = estimated_map.indices_2_coordinates(*cellA)
40      cellB_x, cellB_y = estimated_map.indices_2_coordinates(*cellB)
41      delta_theta = normalize_angle(
42        np.arctan2(cellB_y - cellA_y, cellB_x - cellA_x) - estimated_orientation)
43      estimated_orientation = normalize_angle(estimated_orientation + delta_theta)
44      # we want to travel through k cells in 0.5s, 1s = 30 frames => 0.5s = 15 frames
45      # 1 cell = 0.1 m, so we need to travel with 0.2m/s to pass 1 cell
46      for _ in range(15):
47        # do not normalize angle
48        inputs.append(np.array([k * 0.1 * 2, 0, delta_theta * 2], dtype=float))
49    return inputs.pop(0)
```

5.3 Algoritam za pronalazak sjecišta i Bresenhamov algoritam

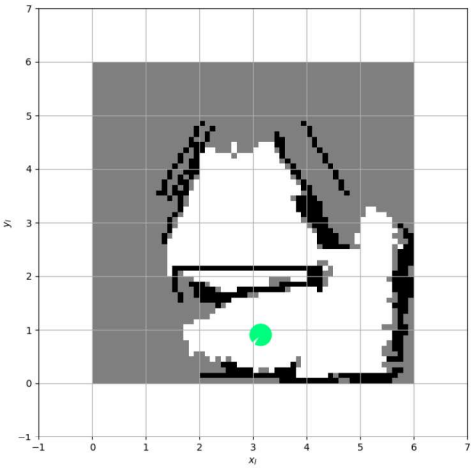
Dobro je za izdvojiti još dva algoritma koja smo koristili u ovoj simulaciji. Oba su se koristila u simulaciji senzora, ali s različitim namjerama. Ultrazvučni senzor u ovoj implementaciji aproksimirali smo s više laserskih zraka koje su jednako razmaknute. Da bismo pronašli sjecišta tih laserskih zraka s postavljenim preprekama koristili smo formule za pronalazak sjecišta dvaju pravaca poznavajući dvije točke na svakom pravcu [10]. Tada bi minimalna pronađena udaljenost bila definirana kao mjerenje od senzora. Bresenhamov smo algoritam [9] koristili u izgradnji mrežaste mape zauzetosti 11 kada smo trebali definirati koje su ćelije u percepciji nekog senzora. Bresenhamov algoritam je tada odredio ćelije koje najbolje aproksimiraju zraku senzora koja putuje od pozicije senzora do točke sjecišta. Ostatak implementacije i ostali detalji mogu se naći na [4].

5.4 Rezultati

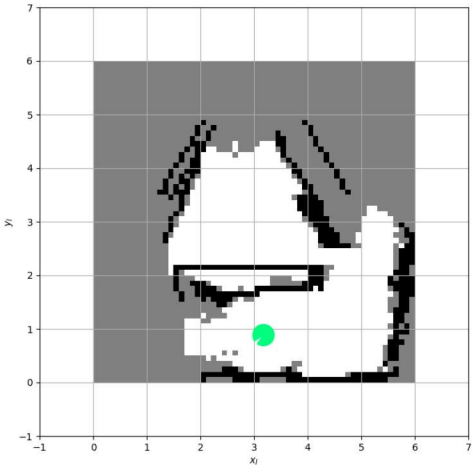
Korištenjem 400 čestica s opisanim algoritmima na slikama 5.1 možemo vidjeti kako su jedne od najboljih 20 čestica izgradile prostor i procijenile gdje se robot uistinu nalazi u određenim trenucima. Zajedno s njima prikazana je izgrađena mapa i procijenjeno stanje aproksimiranog robota u istim trenucima (ljubičasti robot predstavlja "stvarno" stanje robota).



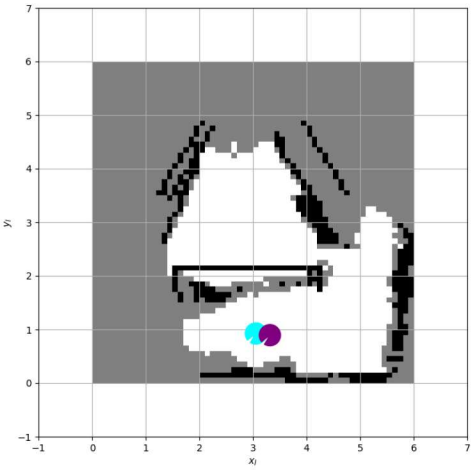
(a) Čestica 1, 7. sekunda



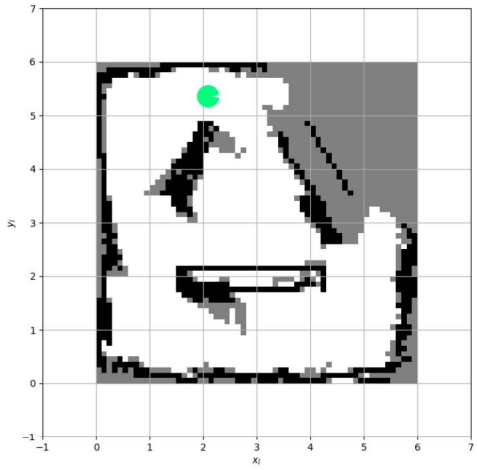
(b) Čestica 2, 7. sekunda



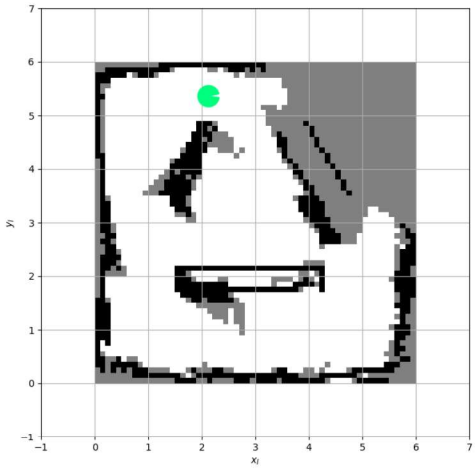
(c) Čestica 3, 7. sekunda



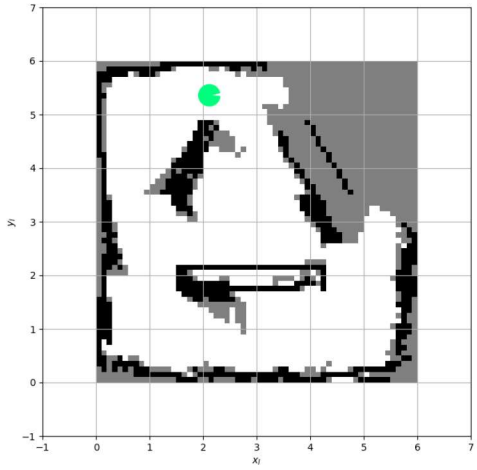
(d) Robot, 7. sekunda



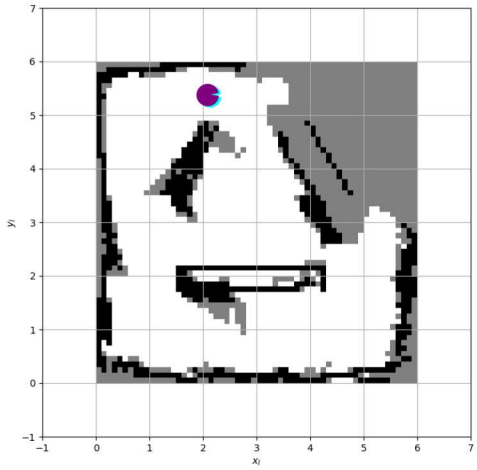
(e) Čestica 1, 14. sekunda



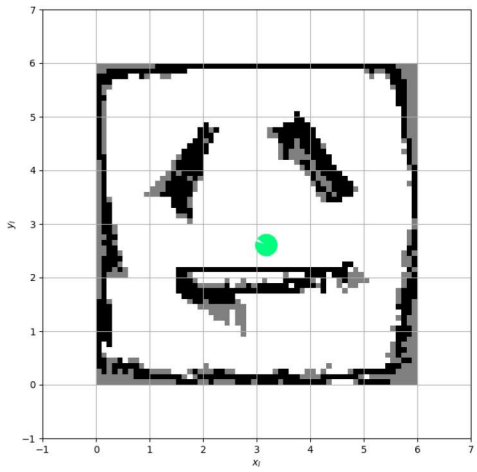
(f) Čestica 2, 14. sekunda



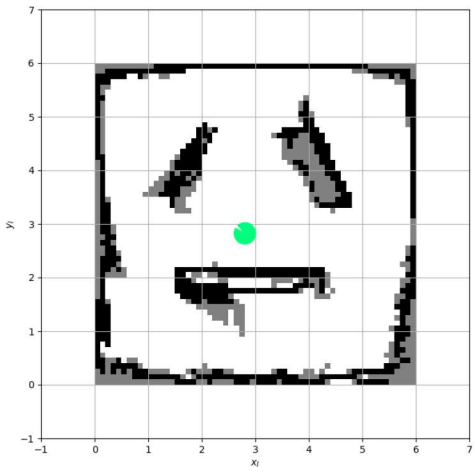
(g) Čestica 3, 14. sekunda



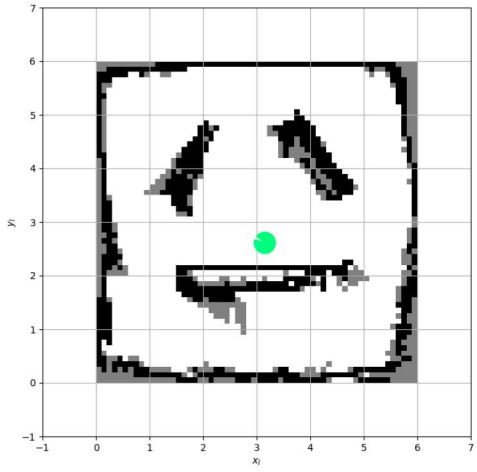
(h) Robot, 14. sekunda



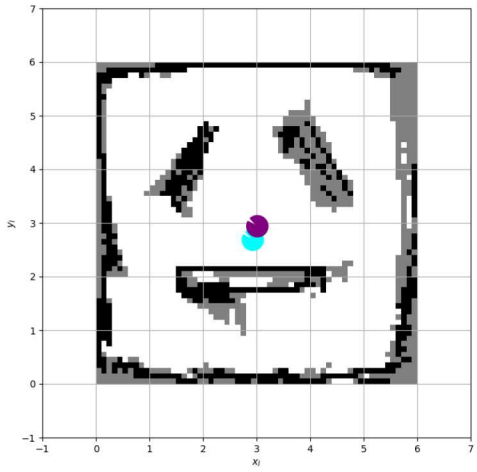
(i) Čestica 1, kraj



(j) Čestica 2, kraj



(k) Čestica 3, kraj



(l) Robot, kraj

Slika 5.1: Proces izgradnje mape i lokalizacije

5.5 Zaključak

Čitava simulacija implementirana prema ovome radu traje 20 sekundi i može se pronaći u [4] u GIF datoteci. Robot se u toj simulaciji uspio lokalizirati i izgraditi čitavu mapu prostora kao što smo mogli vidjeti u rezultatima. U toj simulaciji u prvih nekoliko sekundi čestični filter nije mogao procijeniti točno stanje robota, ali je ipak kasnije uspio. Ono što je malo bunilo robota u početku u lokaliziranju jest sličnost pojedinih mjesta na mapi, što je inače problem koji se javlja u lokalizaciji. Kada se robot uspio lokalizirati, tada je uspješno pratio svoju lokaciju sve do kraja simulacije. Mapa u konačnici izgleda malo neprecizno na pojedinim područjima, no to bi se moglo poboljšati tako da robot prođe bar još jedan krug. Za dodatnu analizu napravili smo još dvije simulacije s različitim postavkama. Prvo smo probali izbaciti završni korak u glavnoj petlji gdje smo konvergirane čestice probali malo raštrkati da robot ne bi konvergirao u krivo stanje. Tada se robot uspio lokalizirati i izgraditi mapu prostora, ali s korištenjem 700 čestica. U drugom smo načinu također izbacili taj korak, ali malo smo pomogli robotu tako da smo mu rekli koja je njegova točna početna orijentacija. Tada se robot uspije lokalizirati i mapirati prostor s 300 čestica. Možemo zaključiti da FastSLAM u osnovnoj verziji radi, ali isto tako postoje poboljšane verzije algoritma koje uspiju lokalizirati robota i mapirati prostor koristeći puno manje čestica, a time i procesorske snage (više u [1]).

Literatura

[1] G. Grisetti, C. Stachniss, W. Burgard, *Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters*", IEEE Transactions on Robotics, vol. 23, no. 1, pp. 34-46, Feb. 2007.

[2] P. E. Hart, N. J. Nilsson, B. Raphael, *A Formal Basis for the Heuristic Determination of Minimum Cost Paths*, IEEE Transactions on Systems Science and Cybernetics, vol. 4, no. 2, pp. 100-107, July 1968.

[3] J. Hespanha, *Linear Systems Theory*, Princeton University Press, 2009.

[4] M. Rajković. (2024). PF-SLAM [Software]. GitHub. https://github.com/mrajkovi/PF-SLAM

[5] N. Sarapa, *Teorija vjerojatnosti*, Školska knjiga, Zagreb, 2002.

[6] R. Siegwart, I. R. Nourbakhsh, D. Scaramuzza, *Introduction to Autonomous Mobile Robots*, The MIT Press, London, 2011.

[7] S. Thrun, *Particle filters in robotics*, In Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence (UAI'02), Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.

[8] S. Thrun, W. Burgard, D. Fox, *Probabilistic Robotics*, The MIT Press, London, 2006.

[9] *Bresenham's line algorithm*, dostupno na `https://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm` (12.10.2024.)

[10] *Line-line intersection*, dostupno na `https://en.wikipedia.org/wiki/Line%E2%80%93line_intersection` (12.10.2024.)

Sažetak

U ovom ćemo se radu upoznati s autonomnim mobilnim robotima. Krenut ćemo od kinematike mobilnih robota koja će nam opisati kako pojedine vrste kotača mogu utjecati na gibanje samog robota. Tu ćemo dati jednostavan primjer upravljanja robotom pomoću PID regulatora koristeći dobivene kinematske uvjete. Zatim ćemo se baviti lokalizacijom gdje ćemo vidjeti kako se pozicija i orijentacija mobilnog robota može vjerojatnosno modelirati u prostoru. Tu ćemo se dotaknuti različitih reprezentacija mape te opisati postupak mapiranja, odnosno izgradnje mape za jednu vrstu dekomponiranih mapa. Zatim ćemo se baviti problemom istovremene lokalizacije i mapiranja gdje se robot u nepoznatom prostoru bavi istovremeno izgradnjom mape i lokaliziranjem. Nakon toga obradit ćemo navigaciju gdje ćemo prezentirati algoritam koji pronalazi najkraći put između dvije točke na dekomponiranim mapama. Zadnja će stavka ovog rada biti konkretna implementacija navedenih algoritama gdje ćemo omogućiti simuliranom robotu da se potpuno lokalizira i kreira potpunu mapu prostora, pri tome ne poznavajući u početku gdje se nalazi i djelomično poznavajući objekte u tom prostoru u početku. Kroz prostor će se gibati koristeći obrađeni algoritam navigacije po unaprijed zadanim točkama prostora.

Ključne riječi

autonomni mobilni roboti, kinematika, teorija upravljanja, lokalizacija, mapiranje, navigacija, simulacija, vjerojatnosni model gibanja, vjerojatnosni model mjerenja, mrežaste mape zauzetosti, SLAM, A*

Autonomous mobile robots

Summary

In this paper we will introduce ourselves to autonomous mobile robots. We will start with mobile robot kinematics, which will explain to us how different types of wheels affect motion of a mobile robot. In this section we will give one simple example of how to control a robot using PID regulator while incorporating obtained kinematics constraints. Then we will deal with localization where we will see how position and orientation of a mobile robot can be probabilistically modelled in space. There we will see different map representations and describe mapping process for one type of decomposed maps. After that, we will get into the simultaneous localization and mapping problem (SLAM) which happens when a mobile robot finds himself in an unexplored area and needs to explore the area while localizing in that area in parallel. Then we will see one navigation technique commonly used in decomposed types of maps. The last part of this paper will contain the implementation of the processed algorithms, which will enable a simulated mobile robot to successfully localize and build a map of an area with partial knowledge of obstacles and unknown initial position. The robot will move through the area using the obtained navigation technique which will build paths through preset points.

Keywords

autonomous mobile robots, kinematics, control theory, localization, mapping, navigation, simulation, probabilistic motion model, measurement model, occupancy grid maps, SLAM, A*

Životopis

Rođen sam 16.2.2000. u Osijeku. Pohađao sam Prirodoslovno-matematičku gimnaziju u Osijeku te sam kao srednjoškolac sudjelovao na natjecanjima iz matematike, fizike i informatike. U 2019. godini osvojio sam nagradu Sakač za najbolje napisani ispit iz informatike na državnoj maturi te dobio priznanje od Zajednice tehničke kulture Osječko-baranjske županije za značajan doprinos u razvoju i promicanju tehničke kulture. Iste sam godine upisao prijediplomski studij Matematika i računarstvo na Odjelu za matematiku (sada Fakultetu primijenjene matematike i informatike) za vrijeme kojeg sam dva puta dobio pohvalu za uspješnost u studiranju te jednom Pročelnikovu nagradu. Studij sam završio 2022. godine s najvišim uspjehom (*summa cum laude*). Iste sam godine upisao diplomski studij Matematike, modul Matematika i računarstvo gdje sam jednom bio nagrađen s pohvalom za uspješnost u studiranju. Za vrijeme sam diplomskog studija radio u tvrtki Ericsson Nikola Tesla kao softverski inženjer.