Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Integrovaný navigačný systém pre mobilného robota

Bakalárska práca

2017
Radoslav Zeman

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Integrovaný navigačný systém pre mobilného robota

Bakalárska práca

Študijný program:	Informatika
Študijný odbor:	2508 Informatika
Školiace pracovisko:	Katedra aplikovanej informatiky
Školiteľ:	Mgr. Pavel Petrovič, PhD.

Bratislava, 2017
Radoslav Zeman

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Radoslav Zeman
Študijný program: informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Integrovaný navigačný systém pre mobilného robota
Integrated Navigation System for a Mobile Robot

Cieľ: Problém spoľahlivej navigácie v zložitom dynamickom prostredí si vyžaduje využitie informácií z rozličných senzorov. V predchádzajúcich prácach a študentských projektoch bola navrhnutá a čiastočne overená navigácia zvlášť pomocou kamery a zaraďovania význačných bodov z obrazu do mapy, zvlášť pomocou ultrazvukových senzorov i pomocou laserového senzora. Úlohou je vytvoriť spoľahlivý a využiteľný doručovací mobilný robotický systém do pavilónu, ktorý bude využívať dáta zo všetkých dostupných senzorov: gyroskopu, akcelerometra, kompasu, laserového range senzora, ultrazvukových a infračervených senzorov, otáčkových senzorov a kamery.

Literatúra:
Marína Madová: Robot Poštár, diplomová práca, FMFI UK, 2014.
Michal Moravčík: Autonómny mobilný robot pre súťaž Robotour, diplomová práca, FMFI UK, 2015.
S. Thrun, W. Bugard, D. Fox: Probabilistic Robotics, MIT Press, 2005.
R. Murphy: An Introduction to AI Robotics, A Bradford Book, 2000.

Vedúci: Mgr. Pavel Petrovič, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: prof. Ing. Igor Farkaš, Dr.

Dátum zadania: 03.10.2016

Dátum schválenia: 24.10.2016

doc. RNDr. Daniel Olejár, PhD.
garant študijného programu

.....
študent

Pavel Petrovič
.....
vedúci práce

Poďakovanie: Ďakujem môjmu školiteľovi Mgr. Pavlovi Petrovičovi, PhD. za čas, ktorý venoval tejto práci a za odbornú radu.

Abstrakt

Doba, v ktorej bola realizovaná táto bakalárska práca, je revolučná rozsahom v akom sa do výroby i do denného života dostávajú robotické technológie. Do popredia sa dostávajú systémy, ktoré pracujú v zložitom prirodzenom prostredí a nie sú len predprogramované na opakovanie rovnakých pohybov na výrobnej linke. Bez toho, aby boli schopné v tomto prostredí konať bez toho, aby ohrozovali seba, či svoje okolie by ich nasadenie nebolo možné. Trieda algoritmov, ktorých účelom je stanoviť počas pohybu robota v prostredí čo najpresnejšie jeho polohu sa označuje ako Lokalizačné algoritmy. Vzhľadom na povahu prostredia je presné určenie polohy nemožné. Sú potrebné také reprezentácie, ktoré dokážu pracovať mieru neurčitosti, nielen neurčitosť samotnú - prichádzajú do úvahy najmä fuzzy systémy a pravdopodobnostné prístupy. Zatiaľ čo fuzzy primárne vychádza z ad-hoc stanovených funkcií príslušnosti a operátoroch, pravdopodobnostné prístupy stavajú na dobre vybudovanom matematickom aparáte v ktorom reprezentácia neurčitosti prispieva k zlepšovaniu celkovej presnosti. V tejto práce riešime problém navigácie a lokalizácie využitím pokročilých metód prehľadávania stavového priestoru a pravdepodobnostnej Monte-Carlo lokalizácie využitím viacerých senzorov - otáčkových senzorov motorov, kompasového senzora a laserového senzora na meranie vzdialenosti. Podarilo sa nám úspešne vybudovať a otestovať jednotlivé komponenty systému, ktoré sú ľahko znovupoužiteľné v ďalších projektoch. Navrhli sme architektúru pre ucelený riadiaci systém doručovacieho robota založeného na našom riešení a predbežné testy naznačujú, že jeho prevádzka je možná, čím sme naplnili ciele práce.

Kľúčové slová: lokalizácia, robotika, monte carlo

Abstract

The time in which this thesis was realized, is revolutionary in range of aplying robotic technologies into production and daily life. Systems working in complex natural environment are getting to the foreground. These systems are not just programmed for repeated movements on a production line. Safe acting without threating surroundings is necessary to deploy these systems. Algorithms which purpose is to determine the robot's location during its motion in the environment are called Localization algorithms. Precise positioning is impossible considering the nature of the environment We need representations which can process uncertainty and its measure. Fuzzy systems and probabilistic approaches comes to mind. While fuzzy systems are based on ad-hoc functions and operands, probabilistic approaches are based on well build mathematical apparatus where representation of uncertainty contributes to accuracy improvement. In this thesis, we solve problem of navigation and localization using advanced methods of state space searching and probabilistic Monte-Carlo localization using multiple sensors - engine speed sensors, compass and laser rande sensor for disntance measurement. We have succesfully built and tested system components that are easily reusable in other projects. We have designed architecture of complete control system of delivery robot based on our solution. Preliminary tests indicate that robot operation is possible so we accomplished the goals of the thesis.

Keywords: localization, robotics, monte carlo

Obsah

Zoznam obrázkov

Zoznam ukážok kódu

Zoznam tabuliek

Úvod

Robotika je veda zaoberajúca sa vnímaním a interakciou s fyzickým svetom pomocou počítačom riadených prístrojov. Medzi príklady robotických systémov môžeme uviesť robotické ramená na priemyselných výrobných linkách, autá so samostatnou navigáciou či robotických chirurgických asistentov.

Robotické systémy umiestňujeme do reálneho sveta, kde vnímajú informácie z okolitého prostredia pomocou rozličných senzorov a interagujú s ním. Okrem nespočetného množstva úloh, ktoré takýto robotický systém musí riešiť, je požiadavka na *autonómnosť* celého systému. Podľa [5]: „**Autonómnosť** z pohľadu robotiky znamená, že robot musí vykonávať určené úlohy bez vonkajšieho zásahu človeka. Pre mobilné roboty je to predovšetkým schopnosť pohybovať sa a orientovať v (neznámom) prostredí."

Prostredie, hlavne ak sa v ňom pohybujú aj ľudia, je veľmi nepredvídateľné a môžeme očakávať jeho vysokú premenlivosť. Aby bol mobilný robot schopný orientovať sa v takomto prostredí, musí v každom okamihu vedieť odpovedať na tri základné otázky:

- Kde som?
- Kam chcem ísť?
- Ako sa tam dostanem?

Odpovede na tieto tri otázky predstavuje súbor úloh, ktoré sa v robotike nazývajú *lokalizácia* a *navigácia*. Podľa [5]: „**Lokalizácia** je taký súbor úloh, ktorý vedie k stanoveniu polohy objektu (robota) v prostredí. Bez toho, aby robot vedel, kde sa v prostredí nachádza, nie je schopný sa v prostredí orientovať a vykonávať zmysluplnú činnosť. Ukazuje sa však, že práve najdôležitejšia odpoveď na otázku ‚Kde som¿nemá v robotike univerzálne riešenie. Dôvodom je hlavne nepresnosť v meraní snímačov používaných na lokalizáciu mobilného robota."Senzory sú limitované v tom, čo môžu vnímať viacerými faktormi a rozruch okolo senzorov nepredvídateľne skresľuje nasnímané dáta.

Spracovávanie nepresností z reálneho sveta je možno najdôležitejším krokom k odolným robotickým systémom.

Cieľom práce je vytvoriť spoľahlivý a využiteľný doručovací mobilný robotický systém do pavilónu, ktorý bude využívať dáta zo všetkých dostupných senzorov: gyro-

skopu, akcelerometra, kompasu, laserového range senzora, ultrazvukových a infračervených senzorov, otáčkových senzorov a kamery.

V našej práci sa preto budeme zaoberať hlavne lokalizáciou robota v interiéri budovy. V kapitole 1 popíšeme teoretické základy globálnej lokalizácie a aplikujeme ich do lokalizačných algoritmov. Kapitola 2 popisuje stavbu robota a použité technológie. Ďalej si v kapitole 3 predstavíme podobné systémy. V druhej časti práce si zadefinujeme ciele práce a v kapitole 5 podrobne popíšeme návrh riešenia. Ďalšie dve kapitoly 6 a 7 opisujú experimenty a ich výsledky.

Kapitola 1

Teória

Táto kapitola opisuje základné matematické koncepty, ktoré v práci využívame, hlavne pravdepodobnostné koncepty ako náhodné premenné a funkcie hustoty pravdepodobnosti. Tieto predpoklady potom rozšírime na Bayesovu vetu a Bayesovo pravidlo. Na záver poznatky aplikujeme do lokalizačných algoritmov, na ktorých staviame v našej práci. Vychádzať budeme najmä z odbornej literatúry zaoberajúcej sa pravdepodobnostnou robotikou [16].

1.0.1 Pravdepodobnostná robotika

Pravdepodobnostná robotika je relatívne novým odvetvím robotiky, ktoré sa snaží vysporiadať práve s nepresnosťami v robotovom vnímaní a interakcii s reálnym svetom. Kľúčovou ideou je reprezentovať nepresnosť pravdepodobnostnými výpočtami. Namiesto spoliehania sa na jeden najlepší tip zobrazíme množinu pravdepodobných tipov. Pravdepodobnostné algoritmy reprezentujú informáciu pravdepodobnostnou distribúciou na celom definičnom obore. Takto môžeme reprezentovať viacznačnosť a vieru v daný tip v matematickom podaní.

Využitie pravdepodobnostnej robotiky pri lokalizácii mobilného robota, kde má robot k dispozícii mapu a na určenie svojich súradníc konzultuje senzorické dáta ilustrujeme neskôr.

1.1 Základné koncepty pravdepodobnosti

V nasledujúcej časti vysvetlíme základné pravdepodobnostné koncepcie, ktoré sa v pravdepodobnostnej robotike, a takisto v našej práci, využívajú.

1.1.1 Náhodné premenné

V pravdepodobnostnej robotike sú všetky kvantitatívne veličiny ako merania snímačov, riedenie pohybu, a všetky stavy robota a jeho prostredia, modelované ako **náhodné premenné**. Náhodné premenné môžu nadobúdať rôzne hodnoty podľa špecifických „pravdepodobnostných zákonov“. Pravdepodobnostným odvodením je potom proces počítania týchto zákonov z náhodných premenných, ktoré sú odvodené z iných náhodných premenných ako napíklad už spomenuté merania snímačov.

Nech X označuje náhodnú premennú a x označuje konkrétnu hodnotu, ktorú môže X nadobudnúť.

Ak je množina všetkých hodnôt, ktoré môže X nadobudnúť, diskrétna, píšeme

$$p(X = x) \tag{1.1}$$

čo označuje pravdepodobnosť, že náhodná premenná X nadobudla hodnotu x. Súčet všetkých diskrétnych pravdepodobností je rovný 1. Ako príklad môžeme uviesť hod mincou, kde $p(X = \text{líce}) = p(X = \text{rub}) = \frac{1}{2}$.

$$\sum_x p(X = x) \;=\; 1 \tag{1.2}$$

Pravdepodobnosť je samozrejme vždy nezáporná, tz. $p(X = x) \geq 0$.

Náhodné premenné je zvykom označovať veľkými písmenami z konca abecedy a ich možné hodnoty zodpovedajúcimi malými písmenami.

Pre zjednodušenie zápisu budeme vynechávať explicitné pomenovanie náhodnej premennej vždy, keď to bude možné, a použijeme bežné skrátenie na $p(x)$ namiesto $p(X = x)$.

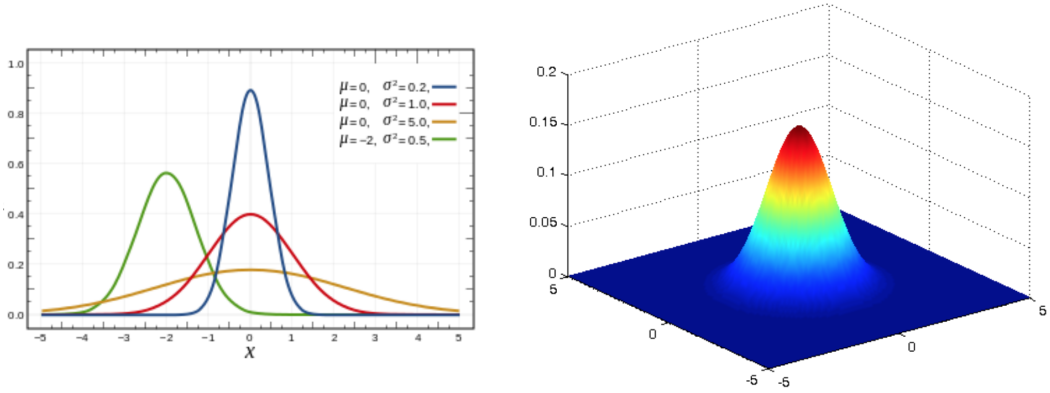
1.1.2 Funkcia hustoty pravdepodobnosti (FHP)

V našej práci budeme pracovať nielen s odhadmi v diskrétnom priestore, ale aj s odhadmi v spojitých priestoroch, ktoré sú charakterizované práve spojitými náhodnými premennými. Ďalej budeme predpokladať, že ku všetkým spojitým náhodným premenným existuje **funkcia hustoty pravdepodobnosti (FHP)** (angl. probability density function), ktorá pre spojité náhodné premenné vyjadruje, s akou pravdepodobnosťou sa daná premenná nachádza v intervale (a, b) a počíta sa ako integrál danej funkcie od a po b:

$$p(a < x < b) = \int_a^b f(x)dx. \tag{1.3}$$

Pre spojitú FHP musí platiť

$$\int_{-\infty}^{\infty} f(x)dx = 1, \tag{1.4}$$



Obr. 1.1: Normálne rozdelenie 1D s rôznymi hodnotami μ a σ (vľavo) a 2D (vpravo)

teda plocha pod krivkou funkcie sa rovná jednej.

Bežnou jednorozmernou FHP je Normálne rozdelenie vyjadrené Gaussovou krivkou s priemerom μ a štandardnou odchýlkou σ, ktorá charakterizuje rozloženie výsledkov meraní vzhľadom na referenčnú hodnotu. Príklad Gaussovej krivka je na obrázku 1.1 a vieme ju vyjadriť nasledovne:

$$p(x) \;=\; (2\pi\sigma^2)^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\right\} \tag{1.5}$$

V normálnom rozdelení 1.5 predpokladáme, že x je skalárna veličina. Ak x je viacrozmernou veličinou, potom viacrozmerné normálne rozdelenie charakterizuje FHP nasledujúcej formy:

$$p(x) \;=\; \det(2\pi\Sigma)^{-\frac{1}{2}} \exp\left\{-\tfrac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right\}, \tag{1.6}$$

kde μ je priemerný vektor a Σ je symetrická matica nazývaná *kovariančná matica*. Horný index ^{T} označuje transponovaný vektor. Všimnime si, že výraz 1.6 je len zovšeobecnením výrazu 1.5. Obe definície sú tak v prípade skalárneho x ekvivalentné.

Ako súčet diskrétnej distribúcie pravdepodobnosti je vždy 1, tak aj integrál FHP je vždy rovný 1.

$$\int p(x)\; dx = 1 \tag{1.7}$$

1.1.3 Spoločná distribúcia, podmienená pravdepodobnosť

Spoločnú distribúciu, čiže jav, kedy náhodná premenná X nadobudne hodnotu x *a zároveň* náhodná premenná Y nadobudne hodnoty y označíme

$$p(x,y) \;=\; p(X = x \wedge Y = y). \tag{1.8}$$

Ak X a Y sú **nezávislé**, dostávame rovnosť

$$p(x, y) \;=\; p(x) \cdot p(y) \tag{1.9}$$

Často sa však stáva, že náhodná premenná nesie informáciu aj o inej náhodnej premennej. Predpokladajme, že vieme, že hodnota pre Y je y a chceme vedieť pravdepodobnosť, že X bude x v závislosti na Y. Pre dva javy môžeme povedať: chceme zistiť pravdepodobnosť výskytu javu X za predpokladu výskytu javu Y. Takúto pravdepodobnosť, ako uvidíme aj v Bayesovej vete 1.13, označíme

$$p(x \mid y) \;=\; p(X = x \mid Y = y) \tag{1.10}$$

a budeme ju volať **podmienená pravdepodobnosť**. Ak $p(y) > 0$, tak je podmienená pravdepodobnosť definovaná ako

$$p(x \mid y) \;=\; \frac{p(x, y)}{p(y)}. \tag{1.11}$$

Ak X a Y sú nezávislé premenné, tak dostaneme

$$p(x \mid y) \;=\; \frac{p(x, y)}{p(y)} \;=\; p(x). \tag{1.12}$$

Inými slovami, ak X a Y sú nezávislé, Y nám nič nehovorí o tom, akú hodnotu má X.

1.1.4 Bayesova veta

Podľa [10] je to veta z teórie pravdepodobnosti, ktorá hovorí o podmienenej pravdepodobnosti a jej súvislosti s opačnou podmienenou pravdepodobnosťou. Formálny zápis vety pre dva javy definuje vzťah 1.13.

$$p(A \mid B) = \frac{p(A) \cdot p(B \mid A)}{p(B)} \tag{1.13}$$

kde A a B sú dva náhodné javy, $p(A)$ a $p(B)$ sú ich pravdepodobnosti.

Ako príklad môžeme uviesť nasledujúcu situáciu: V triede máme niekoľko študentov. Niektorí vedia lyžovať (označíme ako jav L), niektorí vedia hrať na klavíri (označíme ako jav K). Chceme vypočítať, aká je pravdepodobnosť, že vyberieme študenta, ktorý vie hrať na klavíri a zároveň aj lyžovať $p(K, L) = p(K \wedge L)$, čo vieme zapísať ako

$$p(K, L) = p(K) \cdot p(L \mid K)$$

a

$$p(K, L) = p(L) \cdot p(K \mid L).$$

Keď tieto výrazy dáme do rovnosti a upravíme, dostaneme presne Bayesovu vetu 1.13:

$$p(K \mid L) = \frac{p(K) \cdot p(L \mid K)}{p(L)}.$$

1.1.5 Bayesovo pravidlo

Bayesovu vetu použijeme na formulovanie Bayesovho pravidla, ktoré vyjadruje súvislosť pdmienenej pravdepodobnosti $p(x \mid y)$ s jej „opačnou“ podmienenou pravdepodobnosťou $p(y \mid x)$. Predpokladáme, že $p(y) > 0$:

$$p(x \mid y) = \frac{p(y \mid x)\ p(x)}{p(y)} = \frac{p(y \mid x)\ p(x)}{\sum_{x'} p(y \mid x')\ p(x')} \tag{1.14}$$

$$p(x \mid y) = \frac{p(y \mid x)\ p(x)}{p(y)} = \frac{p(y \mid x)\ p(x)}{\int p(y \mid x')\ p(x')dx'} \tag{1.15}$$

Výraz 1.14 vyjadruje Bayesovo pravidlo pre diskrétny priestor, zatiaľ čo výraz 1.15 pre spojitý.

Bayesovo pravidlo hrá v robotike dominantnú úlohu. Ak x je veličina, ktorú by sme chceli odvodiť z y, o pravdepodobnosti $p(x)$ bdeme hovoriť ako o apriórnej pravdepodobnostnej distribúcii a y bude označovať dáta (napr. meranie senzora). Pravdepodobnosť $p(x \mid y)$ nazývame posteriórnou pravdepodobnostnou distribúciou nad X. Ako teda môžeme vidieť z výrazu 1.15, Bayesovo pravidlo nám ukazuje praktický spôsob na počítanie posteriórnej pravdepodobnosti $p(x \mid y)$ použitím „opačnej“ pravdepodobnosti $p(y \mid x)$ spolu s apriórnou pravdepodobnosťou $p(x)$. Inými slovami, ak cheme odvodiť veličinu x zo senzorických dát y, umožňuje nám to Bayesovo pravidlo, ktoré určuje pravdepodobnosť $p(y)$ za predpokladu, že sme namerali práve x.

V robotike sa táto „opačná“ pravdepodobnosť často označuje ako `generative model`, pretože abstraktne vyjadruje ako stavová veličina X spôsobuje nameranie Y.

Dôležitým pozorovaním je, že menovateľ v Bayesovom pravidle $p(y)$ nezávisí od x. Preto vo výrazoch 1.14 a 1.15 bude pre každé x tento menovateľ rovnaký. Z tohto dôvodu často namiesto $p(y)^{-1}$ píšeme *normalizačnú konštantu*, všeobecne označenú η:

$$p(x \mid y) = \eta\ p(y \mid x)\ p(x). \tag{1.16}$$

Ak X je disktétne, výraz tohto typu vieme zapísať nasledovne:

$$\forall x : p(x \mid y) = \frac{p(y \mid x)\ p(x)}{\sum_x p(y \mid x)\ p(x)}. \tag{1.17}$$

Týmto spôsobom efektívne vypočítame $p(x \mid y)$, ale namiesto explicitného počítania $p(y)$ iba normalizujeme výsledok na 1.

1.2 Lokalizácia

Celá práca sa zaoberá globálnou lokalizáciou robota.

Algorithm 1 Bayes filter

1: **function** BAYES_FILTER($bel(x_{t-1}), u_t, z_t$)
2: **for all** x_t **do**
3: $\overline{bel}(x_t) = \int p(x_t \mid u_t, x_{t-1})\ bel(x_{t-1})\ dx$
4: $bel(x_t) = \eta\ p(z_t \mid x_t)\ \overline{bel}(x_t)$
5: **end for**
6: **return** $bel(x_t)$
7: **end function**

1.2.1 Bayes filter

Najvšobecnejší algoritmus na počítanie viery $bel()$ je *Bayes filter*, ktorý je odvodený z Bayesovej vety 1.13. Tento algoritmus počíta vieru pomocou nameraných dát a dát pohybu.

Bayes filter je rekurzívny algoritmus, tz., že vieru $bel(x_t)$ v čase t počítame z viery $bel(x_{t-1})$ v čase $t-1$. Tabuľka 1 opisuje krok rekurzie v čase t, kde na vstupe dostane okrem predchádzajúcej viery aj aktuálne meranie u_t a vykonaný pohyb z_t. Tzv. `update rule`.

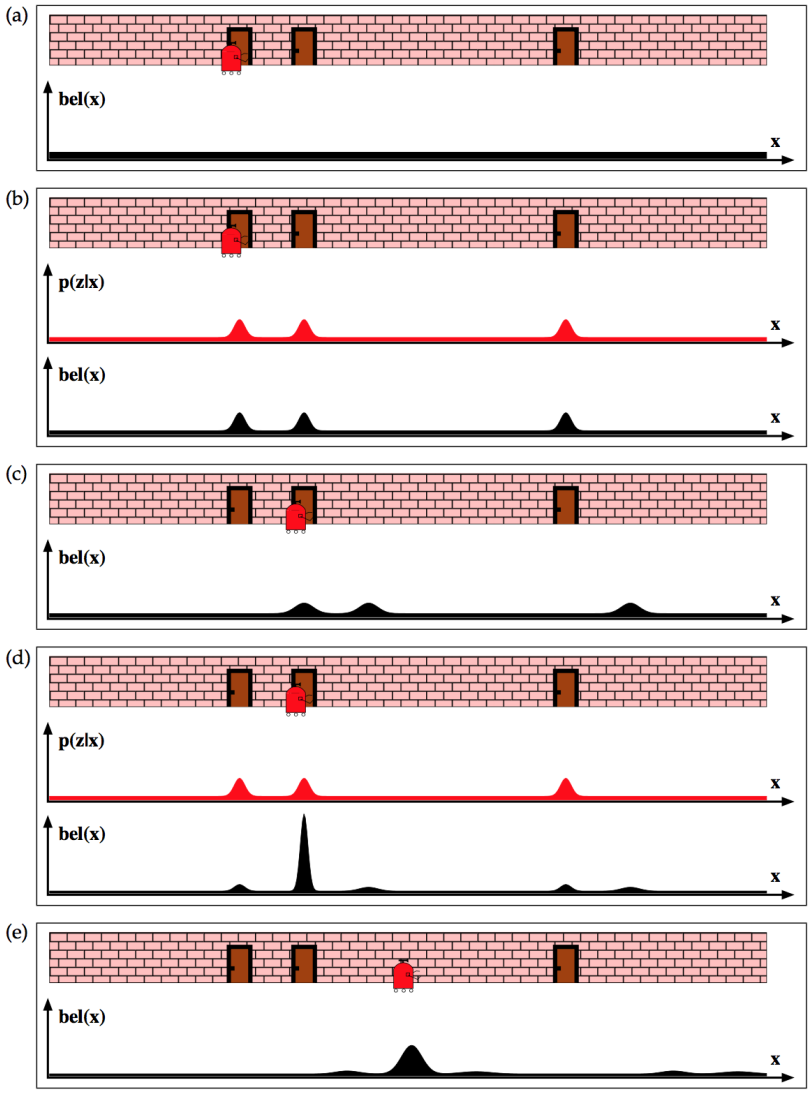
Na riadku 3 vidíme prvý hlavný krok algoritmu, kde spracúvame pohyb u_t tak, že pomocou neho a apriórnej viery $bel(x_{t-1})$ vypočítame $\overline{bel}(x_t)$, teda vieru v stave x_t. Túto vieru vlastne dostaneme spočítaním integrálu (sumy) súčinu dvoch distribúcií: apriórnej distribúcie x_{t-1} a pravdepodobnosti, že pohyb u_t zahŕňa presun zo stavu x_{t-1} do x_t. Tento proces nazývame `control update` alebo `prediction`.

Druhý krok Bayes filtra sa volá `measurement update`. V riadku 4 násobíme vieru $\overline{bel}(x_t)$ pravdepodobnosťou, že sme mohli namerať z_t. Nakoniec treba tento výraz normalizovať konštantou η, čím dostaneme výslednú vieru $bel(x)$.

Tieto dva kroky sa v algoritme udejú pre každý hypotetický posteriórny stav x_t.

Na toto rekurzívne počítanie, potrebuje algoritmus vedieť počiatočnú distribúciu viery $bel(x_0)$ v čase $t = 0$. Buď vieme hodnotu x_0, teda vieme, kde sa robot nachádza, a vtedy by sa mala všetka pravdepodobnosť sústreďovať na tomto bode a inde priradíme nulovú pravdepodobnosť; alebo totálne odignorujeme počiatočný stav x_0 a vtedy nastavíme všade univerzálnu distribúciu; alebo pravdepodobnosť jednotlivých častí vieme odhadnúť.

V tejto forme vieme Bayes filter implementovať iba pre veľmi jednoduché odhadovacie problémy, pretože musíme vedieť vypočítať integrál v riadku 3 a násobenie v riadku 4 v uzavretej forme. Alebo, ako uvidíme nižšie, musíme sa obmedziť na konečný stav, a budeme počítať namiesto integrálu (konečnú) sumu.



Obr. 1.2: Základná idea Markovovej lokalizácie, prevzaté z [16]

1.2.2 Markovova lokalizácia

Pravdepodobnostné lokalizačné algoritmy sú často obmenou Bayes filtra. Ukážeme si teraz priamu aplikáciu Bayes filtra v podobe *Markovovej lokalizácie*. Algoritmus Bayes filtra upravíme tak, že ako vstup bude vyžadovať aj mapu m, ktorá bude často, ale nie vždy, zahrnutá do počítania `motion modelu`, a teda riadok 3 bude vyzerať nasledovne: $\overline{bel}(x_t) = \int p(x_t \mid u_t, x_{t-1})\ bel(x_{t-1})\ dx$.

Bez podrobnejšieho vysvetlenia, si môžeme rovno uviesť príklad Markovovej lokalizácie v prostredí jednorozmernej chodby, na ktorej sa nachádzajú tri nerozlíšiteľné dvere.

V diagrame (a) na obrázku 1.2 vidíme uniformnú distribúciu na všetkých možných pozíciach. Predpokladajme teraz, že robot práve po prvýkrát zaznamenal svojimi sen-

zormi dvere. Využitím tejto informácie robot aktualizuje svoju vieru. Následná viera je naznačená na obrázku 1.2 v diagrame (b). Vypočítali sme zvýšenú pravdepodobnosť na miestach v blízkosti dverí a nižšiu pravdepodobnosť na miestach pri stenách. Poznamenávame, že táto distribúcia má tri vrcholy, každý zodpovedajúci jedným z nerozlíšiteľných dverí v zadanom prostredí. Preto teda robot v žiadnom prípade ešte nemôže *vedieť*, kde sa nachádza. Namiesto toho má teraz tri neprekrývajúce sa hypotézy, ktoré sú navzájom rovnocenné. Všimnime si tiež, že robot priradil kladnú pravdepodobnosť aj na miesta nie priamo pri dverách, čo je zapríčinené chybami snímania – s malou, ale nenulovou, pravdepodobnosťou sa mohol robot pomýliť pri získavaní údajov zo senzorov.

Predpokladajme ďalej, že robot vykoná pohyb. Diagram (c) na obrázku 1.2 ukazuje efekt tohto pohybu na robotovu vieru. Viera bola posunutá v smere pohybu a tiež sa rozšírili a znížili vrcholy pravdepodobnosti, čo reflektuje nepresnosti z robotovho pohybu. Diagram (d) na obrázku 1.2 znázorňuje vieru po zaznamenaní ďalších dverí. toto spozorovanie dverí vedie nášho robota k tomu, aby priradil najväčšiu pravdepodobnosť ku jedným konkrétnym dverám, a to práve k tým, pri ktorých sa nachádza. Robot si je teraz celkom istý, kde sa nachádza.

Na záver diagram (e) obrázku 1.2 reprezentuje vieru po tom, ako sa robot hýbe ďalej po chodbe.

1.3 Monte Carlo lokalizácia (MCL)

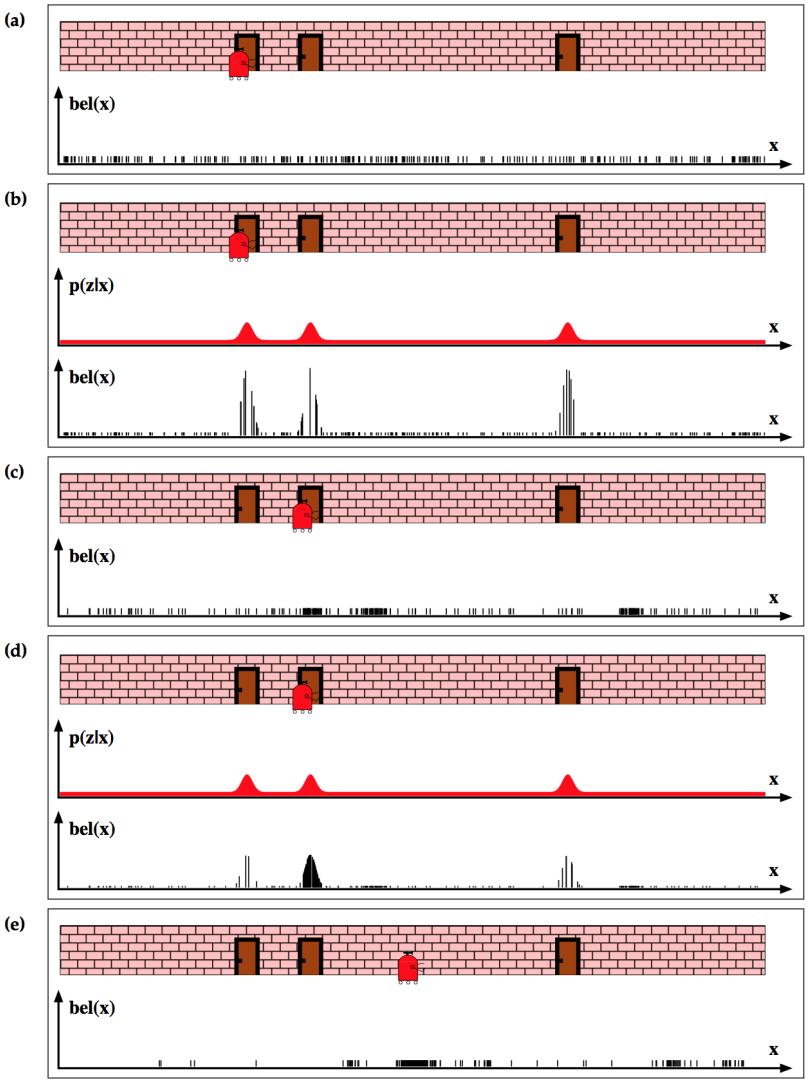
Táto časť opisuje algoritmus Monte Carlo lokalizácie, ktorý je pomerne mladým, zato však jedným z najpopulárnejších algoritmov na lokalizáciu v robotike. Je ľahko implementovateľný a aplikovateľný na množstvo globálnych aj lokálnych lokalizačných problémov. Vieru $bel(x_t)$ reprezentuje nie pomocou spojitej funkcie ale v diskrétnych časticiach (angl. particles).

1.3.1 Ilustrácia MCL

Obrázok 1.3 reprezentuje MCL, rovnako ako v predchádzajúcej časti, na príklade jednorozmerného prostredia chodby s tromi nerozlíšiteľnými dverami. Úvodnú nepresnosť dostaneme vygenerovaním množiny častíc (angl. particle set) uniformne náhodne cez celý priestor (angl. pose space), ako je vidieť na obrázku 1.3a.

Po zaznamenaní dverí MCL algoritmus priradí dôležitosť každej častici. Ako je vidno na obrázku 1.3b, výška každej častice zodpovedá jej dôležitosti. Upozorňujeme na to, že množina častíc je totožná s množinou častíc v prípade 1.3a, líši sa len váha dôležitosti jednotlivých častíc.

Obrázok 1.3c ukazuje množinu častíc po *prevzorkovaní* a zarátaní pohybu robota,



Obr. 1.3: MCL, prevzaté z [16]

čo vedie k novej množine častíc s rovnocennou dôležitosťou. Tentoraz si však môžeme už všimnúť zvýšený počtet častíc v okolí troch pravdepodobných miest výskytu robota. Nové meranie priradí neuniformnú dôležitosť množine častíc, čo môžeme vidieť na obrázku 1.3d. V tomto bode je väčšina pravdepodobnostných častíc sústredená pri druhých dverách, čo je zároveň najpravdepodobnejšou polohou robota. Následný pohyb vedie opäť k prevzorkovaniu a k zahrnutiu pohybu, čo vedie opäť k novej množine častíc (obrázok 1.3e).

Ako by malo byť zrejmé z obrázku, množina častíc aproximuje k správnej polohe robota.

Algorithm 2 Monte Carlo lokalizácia

1: **function** MCL(χ_{t-1}, u_t, z_t, m)
2: $\bar{\chi}_t = \chi_t = \emptyset$
3: **for** $m = 1$ to M **do**
4: $x_t^{[m]} =$ **motion_model**$(u_t, x_{t-1}^{[m]})$
5: $w_t^{[m]} =$ **measurement_model**$(z_t, x_t^{[m]}, m)$
6: $\bar{\chi}_t = \bar{\chi}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
7: **end for**
8: **for** $m = 1$ to M **do**
9: nakresli i s pravdepodobnosťou $\propto w_t^{[m]}$
10: pridaj $x_t^{[i]}$ do χ_t
11: **end for**
12: **return** χ_t
13: **end function**

1.3.2 MCL algoritmus

Algoritmus 2 ukazuje základný MCL algoritmus, ktorý dostaneme úpravou diskrétneho Bayesovského filtra príslušnými pravdepodobnostnými pohybovými modelmi a modelmi merania senzorov. Základný MCL algoritmus reprezentuje vieru $bel(x_t)$ ako množinu M častíc $\chi_t = \{x_t^{[1]}, x_t^{[2]}, \ldots, x_t^{[M]}\}$.

Na riadku 4 nášho algoritmu vzorkujeme častice podľa pohybového modelu, využívajúc distribúciu častíc z aktuálnej viery. Model merania senzorov je potom aplikovaný v riadku 5, aby určil váhu dôležitosti jednotlivých častíc.

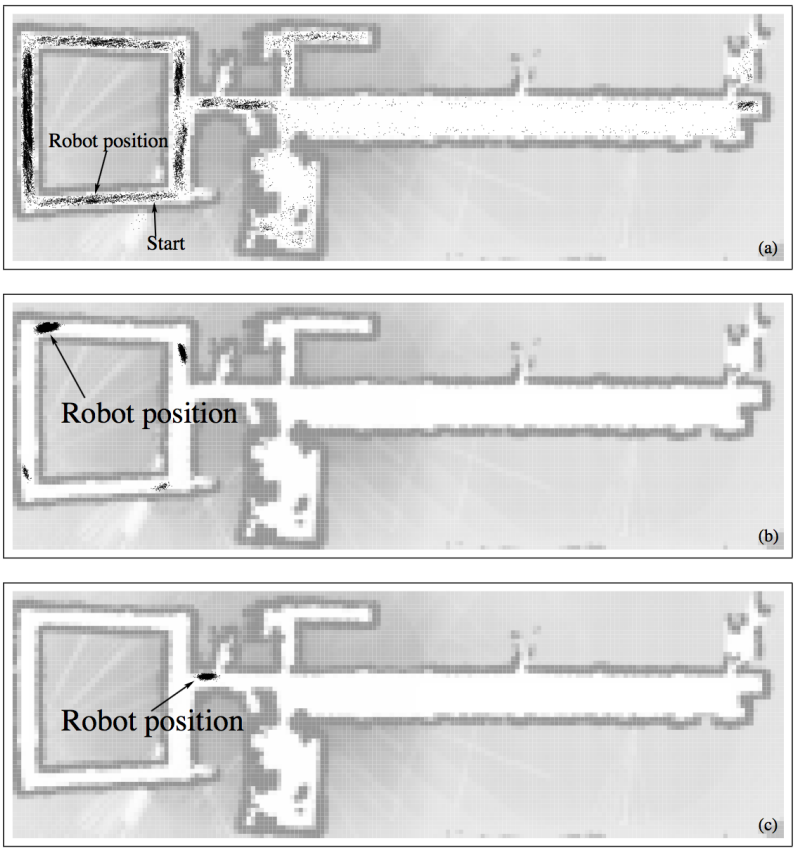
Počiatočnú vieru bel(x0) získame náhodným generovaním M častíc z prvotnej distribúcie $p(x_0)$ a priradením uniformnej dôležitosti každej častici.

Funkcie **motion_model** a **measurement_model** môžu byť implementované rôznymi modelmi.

1.3.3 Vlastnoti MCL

Presnosť aproximácie MCL vieme ľahko určiť veľkosťou množiny častíc M. Zväčšením tejto množiny dostaneme presnejšiu aproximáciu. Týmto parametrom vieme nastaviť správny pomer medzi presnosťou a výpočtovým zaťažením potrebným na beh MCL.

Bežnou stratégiou je nechať bežať vzorkovanie dokým nezískame ďalšiu dvojicu dát u_t, z_t. Takáto implementácia je prispôsobiteľná vzhľadom k výpočtovým zdrojom. Čím rýchlejší máme procesor, tým rýchlejšie bude bežať aj MCL. Takáto prispôsobivosť je však ťažko dosiahnuteľná pri použití Gaussovských techník.



Obr. 1.4: Príklad MCL, prevzaté z [16]

1.3.4 Rozšírený MCL

Vyššie spomenutý algoritmus MCL síce vie riešiť globálnu lokalizáciu, ale nedokáže sa zotaviť z chýb alebo iných lokalizačných problémov ako je napríklad manuálne premiestnenie robota, tzv. `robot kiddnapping`, čo doslova znamená únos robota. Ak je vidieť na obrázku 1.4, po zistení pozície robota, všetky častice aproximovali k tejto polohe a na ostatných miestach prostredia takmer vymizli. V takejto situácii prežijú ďalší krok algoritmu len častice v blízkosti jednej pozície a algoritmus nie je schopný sa zotaviť, ak táto pozícia nebola správna, čo je zásadný problém. Hlavne ak máme malú množinu častíc, alebo pri globálnej lokalizácii, keď sú častice rozprestreté po veľkej ploche.

Našťastie vieme tento problém vyriešiť veľmi jednoducho. Idea je taká, že pri prevzorkovaní množiny častíc pridáme do množiny náhodne umiestnené častice, čo môže vyjadrovať, že s malou pravdepodobnosťou sa mohlo stať, že robot „uhádol“ nesprávnu pozíciu, alebo bol premiestnený.

Algorithm 3 Rozšírená Monte Carlo lokalizácia

function MCL_AUGMENTED(χ_{t-1}, u_t, z_t, m)
  static w_{slow}, w_{fast}
  $\bar{\chi}_t = \chi_t = \emptyset$
  for $m = 1$ to M **do**
    $x_t^{[m]} =$ **motion_model**$(u_t, x_{t-1}^{[m]})$
    $w_t^{[m]} =$ **measurement_model**$(z_t, x_t^{[m]}, m)$
    $\bar{\chi}_t = \bar{\chi}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
    $w_{avg} = w_{avg} + \frac{1}{M}\ w_t^{[m]}$
  end for
  $w_{slow} = w_{slow} + \alpha_{slow}(w_{avg} - w_{slow})$
  $w_{fast} = w_{fast} + \alpha_{fast}(w_{avg} - w_{fast})$
  for $m = 1$ to M **do**
    s pravdepodobnosťou $max(0,0; 1,0 - w_{fast}/w_{slow})$
      pridaj náhodnú pozíciu do χ_t
    inak
      nakresli $i \in \{1, \ldots, N\}$ s pravdepodobnosťou $\propto w_t^{[m]}$
      pridaj $x_t^{[i]}$ do χ_t
  end for
  return χ_t
end function

1.4 Forward Kinematics for Differential Drive Robots

Náš robot používa mechanizmus známy ako `differential drive` [6]. Takýto robot má dve kolesá na jednej osi a každé koleso sa môže nezávisle točiť dopredu alebo dozadu. Tretie koleso je na otočnej základni a slúži len ako podpora robota.

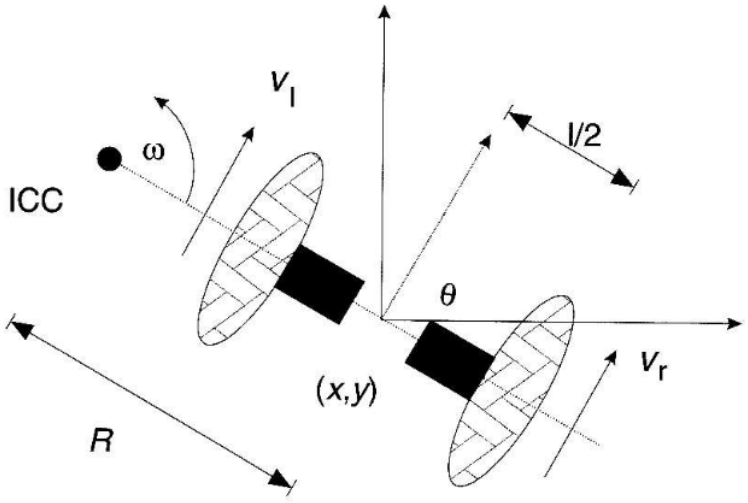
Rýchlosť kolies vieme ľubovoľne regulovať, ale bod, okolo ktorého sa bude robot otáčať pri pohybe, bude vždy ležať na priamke určenej osobou kolies. Tento bod sa nazýva *okamžité centrum zakrivenia (ICC)* (angl. Instantaneous Center of Curvature).

Pomocou zmeny rýchlosti kolies vieme meniť trajektóriu, ktorú robot prejde. Pretože uhlová rýchlosť ω okolo ICC musí byť rovná pre obe kolesá, vieme zapísať nasledujúce rovnice:

$$\omega\ r_r = v_r \tag{1.18}$$

$$\omega\ r_l = v_l \tag{1.19}$$

kde $r_r = r + \frac{l}{2}$ a $r_l = r - \frac{l}{2}$, l je vzdialenosť medzi stredmi kolies, v_r a v_l sú rýchlosti



Obr. 1.5: Differential drive - okamžitý bod zakrivenia [6]

kolies a r je polomer otáčania, čiže vzdialenosť medzi ICC a stredom osi kolies.

Hocikedy vieme vypočítať r a ω:

$$r = \frac{l}{2}\frac{v_l + v_r}{v_r - v_l} \tag{1.20}$$

$$\omega = \frac{v_r - v_l}{l} \tag{1.21}$$

Vieme rozlíšiť tri druhy pohybu:

1. Ak $v_l = v_r$, potom sa robot hýbe lineárne po rovnej čiare. Polomer r bude v nekonečne a neexistuje žiadna rotácia, tz. $\omega = 0$.

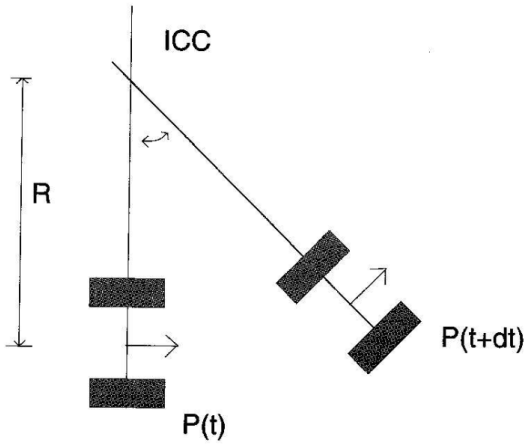
2. Ak $v_l = -v_r$, potom $R = 0$, robot sa teda točí okolo stredu osi kolies, tz. točí sa na mieste.

3. Ak $v_l = 0$ (resp. $v_r = 0$), potom sa robot točí okolo ľavého (resp. pravého) kolesa. V tomto prípade $r = \frac{l}{2}$.

Predpokladajme teraz, že robot na obrázku 1.5 je na súradniciach x, y a je otočený v uhle φ s x-ovou osou. Predpokladáme tiež, že stred robota je v strede osi kolies. Ak vieme v_l a v_r, teda rýchlosti, akými sa točia kolesá, tak vieme vypočítať súradnice ICC:

$$ICC_x = x - r\sin(\varphi);\ ICC_y = y + r\cos(\varphi) \tag{1.22}$$

a v čase $t + \delta t$ bude poloha robota nasledovná:



Obr. 1.6: Different ial drive - forward kinamatics [6]

$$\begin{pmatrix} x' \\ y' \\ \varphi' \end{pmatrix} = \begin{pmatrix} \cos(\omega\delta t) & -\sin(\omega\delta t) & 0 \\ \sin(\omega\delta t) & \cos(\omega\delta t) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x - ICC_x \\ y - ICC_y \\ \varphi \end{pmatrix} + \begin{pmatrix} ICC_x \\ ICC_y \\ \omega\delta t \end{pmatrix}$$

Táto rovnica vyjadruje pohyb robota rotujúceho v polomere r okolo ICC s uhlovou rýchlosťou ω. Iný pohľad na tento problém je, že pohyb robota je ekvivalentný týmto trom krokom (obrázok 1.6): 1) presunutie ICC do stredu sústavy 2) otočenie kolo stredu sústavy o uhol $\omega\delta t$ a 3) presunutie ICC späť na jeho pôvodné súradnice.

1.5 Mapy

Mapa prostredia je zoznam objektov a ich umiestnenia v prostredí. Formálne zapísane, mapa m je zoznam objektov v prostredí spolu s ich vlastnosťami:

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

kde N je počet objektov v prostredí a každé m_n, kde $1 \leq n \leq N$, špecifikuje vlastnosti daného objektu n. Interná reprezentácia mapy môže byť *metrická* alebo *topologická*.

Kapitola 2

Technológie

V tejto kapitole popíšeme konštrukciu robota ako aj jednotlivé jeho hlavné komponenty. Zameriame sa na hardvérové komponenty ale predstavíme ich aj zo softvérovej strany.

2.1 Robot Mikeš - konštrukcia a architektúra

Základná konštrukcia Robota Mikeša je ručne vyrobená z hliníkových profilov a preglejkových dosiek. Ako vidno na obrázku 2.1, pozostáva z dvoch úrovní, na ktoré sú zvrchu aj zospodu namontované rozličné komponenty: senzory, motory, vypínače, switch, elektroinštalácia...

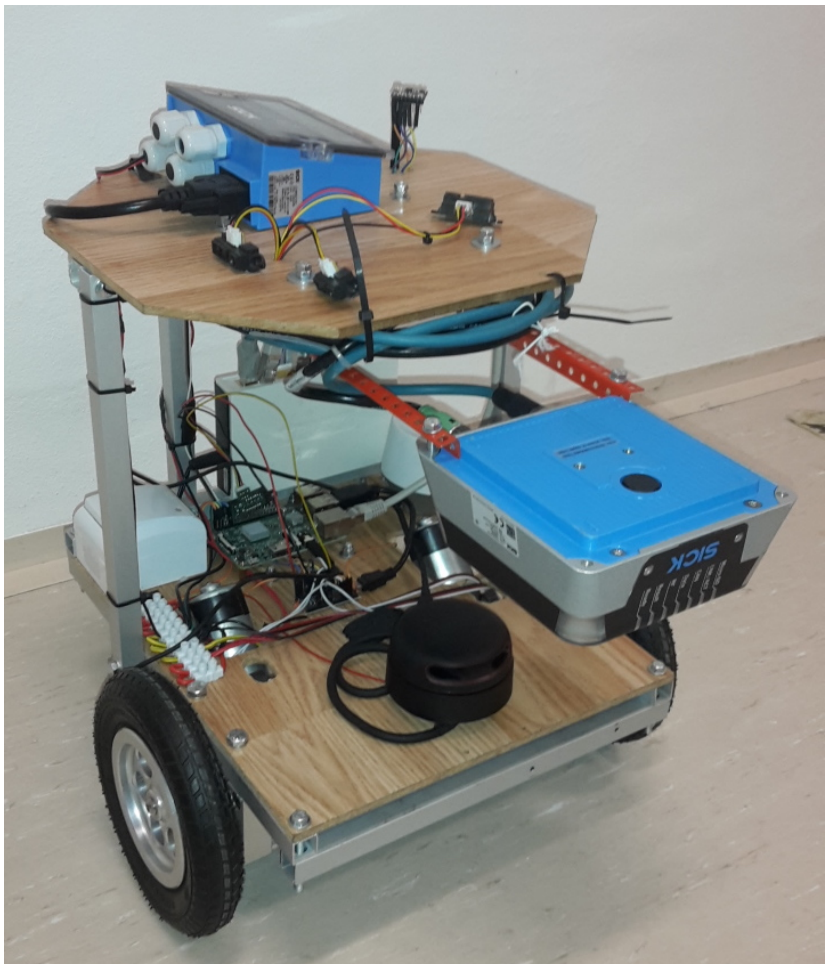
Na spodnej úrovni sa nachádza hlavný komponent našej práce - lidar. Okrem neho tu nájdeme minipočítač Raspberry PI, mikrokontrolér Arduino, vypínač elektrického zdroja, motory kolies, zadné koleso s otáčavou základňou, vypínač pre motory, sieťový switch, pomocou ktorého sa môžeme eternetovým káblom pripojiť na robota.

Na vyššej úrovni sú namontované komponenty ako RFID senzor, gyroskop a IR senzory (tieto všetky senzory sa využívajú v inej študentskej práci).

2.2 Raspbery PI

Raspberry Pi je jednodoskový minipočítač vyvinutý britskou spoločnosťou Raspberry Pi Foundation pôvodne za účelom propagovania výuky základov informatiky na školách a v rozvojových krajinách. Avšak stal sa oveľa populárnejším ako sa čakalo vďaka jeho využitiu v robotike.

Model Rapsbery Pi 3 je vybudovaný na 64-bitovej ARMv8 architektúre so štvorjadrovým CPU a 1GB operačnej pamäte. Ponúka integrované WiFi 802.11n a Bluetooth 4.1 pripojenie. Disponuje 4 USB portmi, 40 GPIO pinmi, Full HDMI portom, Ethernetom a kombinovaným 3,5mm jackom. Raspberry Pi nemá žiadny pevný disk. Na bootovanie operačného systému a vnútorné úložisko sa používa Misco SD slot.



Obr. 2.1: Robot Mikeš

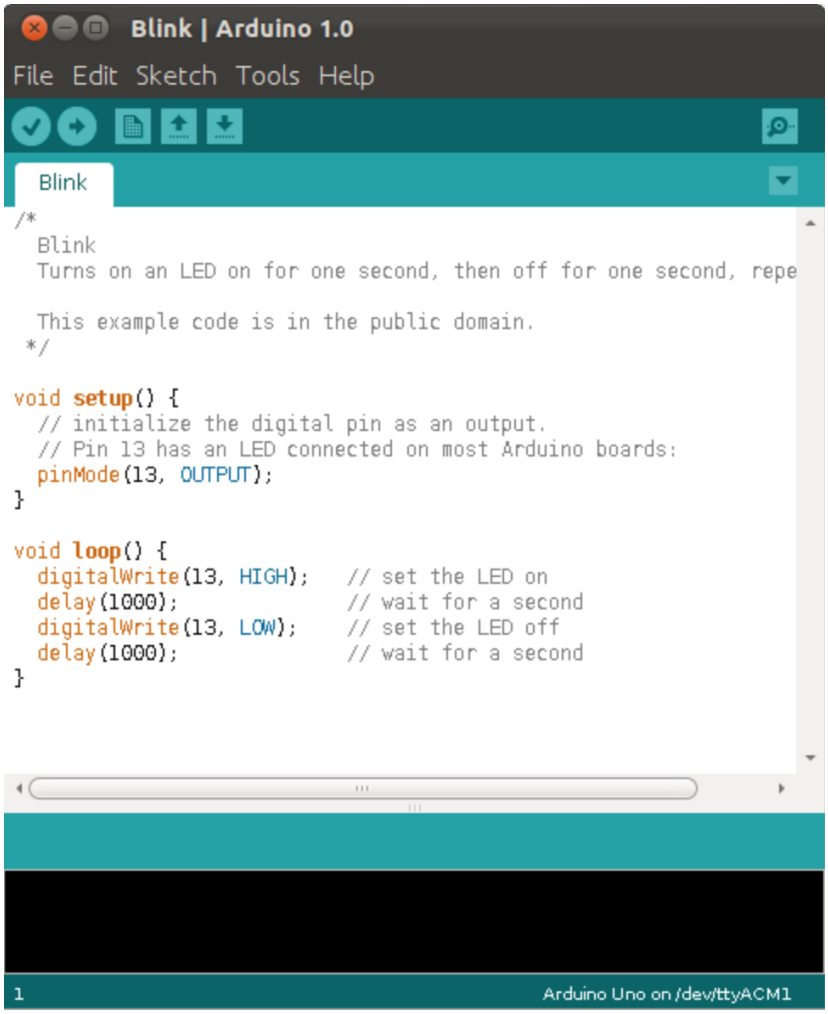
Najpoužívanejším operačným systémom je *Raspbian* - Linuxový operačný systém založený na Debiane. Samozrejme existujú aj iné operačné systémy tretích strán ako Pidora, Ubuntu MATE, OpenELEC alebo aj nie-linuxové systémy ako Windows 10 IoT Core a mnoho ďalších.

2.3 Arduino

Arduino je open-source platforma založená na ľahko ovládateľnom hardvéri a softvéri. Arduino dosky vedia čítať vstupy - svetlo na senzore, stlačenie tlačidla, online správa - a pretvoriť ich na výstupy - aktivovanie motorov, zažatie LED, online príspevok. Doske vieme povedať čo má robiť zaslaním sady inštrukcií mikrokontroléru na doske.

Program pre Arduino môže byť napísaný v rôznych jazykoch. Arduino projekt však poskytuje Arduino integrované vývojové prostredie (IDE), ktoré je založené na prostredí *Processing* a je prenositeľné. Poskytuje okrem klasických funkcií, ktoré pokytujú bežné IDE, aj tlačidlo na priame nahratie kódu na Arduino dosku. Program napísaný v tomto IDE sa nazýva *skeč* (angl. sketch) a ukadajú sa ako text vo formáte *.ino* (staršie

verzie používali formát *.pde*).



Obr. 2.2: Arduino integrované vývojové prostredie (IDE)

Arduino IDE podporuje jazyky C a C++ so špeciálnymi pravidlami štruktúry kódu. Tiež poskytuje *Arduino programovací jazyk* založený na jazyku *Wiring*, ktorý zahŕňa množstvo vstupných a výstupných procedúr. Kód vyžaduje iba dve základné funkcie - na naštartovanie skeču a hlavný programový cyklus. Na obrázku 2.2 môžeme vidieť Arduino IDE a jednoduchý skeč napísaný v jazyku C s dvomi základnými funkciami spomenutými vyššie:

setup() – Táto funkcia sa volá, keď sa skeč štartuje alebo reštartuje. Inicializujú sa v nej premenné, vstupné a výstupné piny a iné knižnice potrebné k behu skeču.

loop() – Potom ako bola zavolaná funkcia *setup()*, funkcia *loop()* je vykonávaná opakovane v hlavnom programe. Ovláda dosku, kým dosku nevypneme alebo nereštartujeme.

Väčšina Arduino dosiek má svetelnú diódu (LED) a rezistor pripojený na pin 13, čo

je pohodlne využiteľné pri množstve programových testov a funkcií. Tento program je typickým program Arduino začiatočníka, v ktorom opakovane bliká LED.

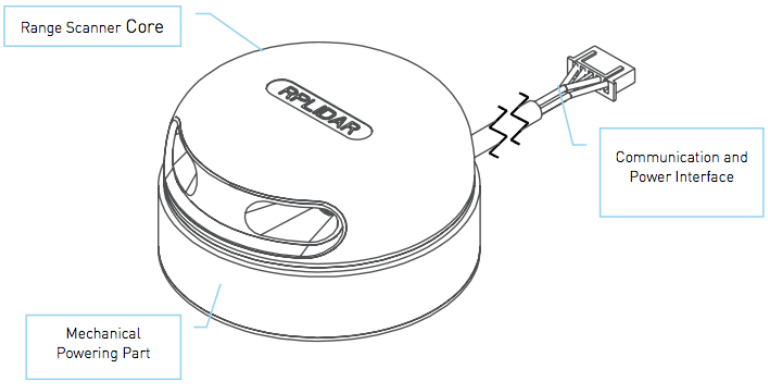
Vďaka tomu, že aj Arduino hardvér aj softvér sú voľne šíriteľné, umožňujú tak budovať a hardvéer či softvér úplne nezávisle podľa potrieb toho-ktorého užívateľa.

2.4 RPLidar A2

Slovo **lidar** označuje metódu diaľkového merania vzdialeností na základe merania doby šírenia impulzu laserového lúča odrazeného od snímaného objektu. Rozdiely v namaraných časoch a vlnových dĺžkach sa potom použijú na vytvorenie digitálnej reprezantácie objektu. Názov *lidar* sa niekedy uvádza ako skratka z anglického *Light Detection And Ranging*, čo môžeme voľne preložiť ako „detekcia a meranie svetla“, ale pôvodne pochádza zo spojenia slov *light* (angl. svetlo) a *radar*, teda „svetelný radar“.

RPLidar A2 Tento senzor [4, 2]od firmy SLAMTEC je 360-stupňový laserový 2D senzor využívajúci vyššie spomenutú metódu (ďalej označený tiež ako *lidar*). Vo vysokej rýchlosti otáčania je schopný zanamenať až 4000 vzoriek za sekundu do vzdialenosti 8m. Dáta zo senzora môžu byť použité na mapovanie, lokalizáciu a modelovanie prostredia/objektov.

Senzor pozostáva zo snímacieho jadra a mechanickej hnacej časti, ktorá roztáča jadro do vysokej rýchlosti (obrázok 2.3).

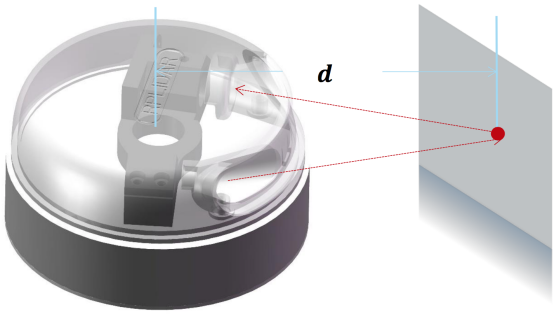


Obr. 2.3: RPLidar A2 - kompozícia systému [4]

2.4.1 Mechanizmus

Bežná pracovná frekvencia lidaru je 10hz, kedy uhlový rozdiel lúčov je 0,9 stupňa. Senzor je založený na laserovom triangulačnom princípe. Počas každého merania senzor

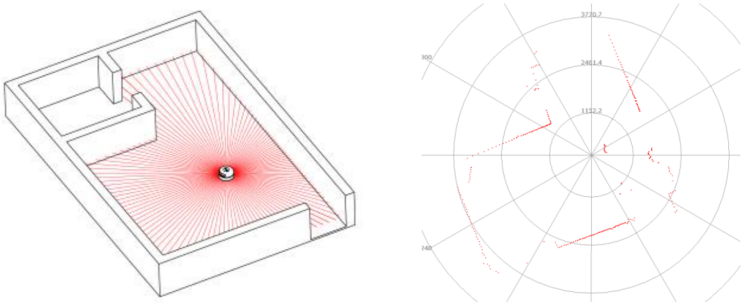
vysiela modulovaný infračervený laserový signál, ktorý sa následne odráža od objektov a je zaznamenaný senzorom (obrázok 2.4). Vizuálny zberný systém integrovaný v senzore začne spracúvať dáta a na výstup posiela vzdialenosť a uhly jednotlivých lúčov pomocou komunikačného rozhrania (obrázok 2.3 vpravo).



Obr. 2.4: RPLidar A2 mechanizmus: lúč sa odráža od objektu a je zachytený senzorom [4]

Adaptívny systém nastaví uhlové rozpätie automaticky podľa aktuálnej rýchlosti otáčania. Ak užívateľ potrebuje zistiť aktuálnu rýchlosť točenia, môže si ju od systému vypýtať pomocou komunikačného rozhrania.

Na obrázku 2.5 môžeme vidieť príklad nasnímaného prostredia pomocou lidaru.



Obr. 2.5: Sken prostredia získaný senzorom RPLidar A2 [4]

2.4.2 Dátové výstupy

Počas toho, ako je lidar spustený, vysiela získané dáta pomocou komunikačného rozhrania. Každý segment výstupných dát obsahuje informácie, ktoré sú uvedené v tabuľke 2.1.

Tabuľka 2.1: Doba výpočtu a operačná pamäť potrebná na spracovanie vstupu XYZ. V tomto popise môžeme vysvetliť detaily potrebné pre pochopenie údajov v tabuľke.

Typ dát	Jednotka	Popis
Vzdialenosť	mm	Aktuálne zmeraná vzdialenosť
Uhol	uhly	Aktuálny uhol daného merania (lúča)
Začiatočná značka	(bit)	Značka začiatku nového skenu
Kontrolný súčet	–	Kontrolný súčet výstupných dát

2.4.3 SDK

RPLidar A2 poskytuje užívateľom SDK [3] na integrovanie základných funkcií do ich vlastných projektov.

Software development kit (SDK) je súbor nástrojov pre vývoj softvéru, ktorý umožňuje vytváranie aplikácií pre určitú softvérovú platformu, hardvérovu platformu, herné konzoly, operačný systém, alebo podobné platformy. SDK tiež často obsahuje ukážkový kód a podporné technické poznámky alebo iné podklady.

Štruktúra SDK

SDK je napísaný v jazyku C++ a definuje základnú hlavičku rplidar.h, ktorá je zvyčajne tou jedinou, ktorú užívateľ potrebuje vložiť do svojho projektu, aby mohol využiť všetky funkcie SDK. Ďalej môžeme nájsť hlavičku rplidar_driver.h, ktorá definuje základnú triedu RPLidarDriver. Hlavičky rplidar_protocol.h a rplidar_cmd.h zase definujú protokol nízkoúrovňovej komunikácie a štruktúry používané na túto komunikáciu. Hlavička rptypes.h definuje hlavné štruktúry a konštany používané užívateľom, nezávisle na platforme.

Inicializácia SDK

Užívateľ vo svojom programe na komunikovanie s lidarom musí najskôr pomocou SDK vytvoriť inštanciu triedy RPlidarDriver, čo môže urobiť pomocou funkcie

```
RPlidarDriver *RPlidarDriver::CreateDriver (_u32 drivertype)
```

Každá RPlidarDriver inštancia sa môže viazať iba na jeden lidar, ale používateľ môže vo svojom programe voľne alokovať ľubovoľný počet inštancií na komunikáciu s viacerými lidarmi súčasne.

Keď program skončí, všetky RPLidarDriver inštancie by mali byť explicitne uvoľnené použitím nasledujúcej funkcie, aby sa uvoľnila sysémová pamäť:

```
RPlidarDriver::DisposeDriver(RPlidarDriver * drv).
```

Pripojenie na senzor

Po vytvorení RPlidarDriver inštancie by sme mali z programu zavolať funkciu `connect()`, ktorá otvorí sériový port na komunikáciu s lidarom. Všetky operácie so senzorom vyžadujú najskôr zavolanie tejto funkcie.

```
u_result RPlidarDriver::connect(const char * port_path, _u32 baudrate,
_u32 flag = 0)
```

Funkcia vráti konštantu RESULT_OK, ak pripojenie prebehlo úspešne.

Pri volaní funkcie `connect()` sa predvolene volá aj funkcia `stopMotor()`, ktorá zastavý rotujúce jadro. SDK volá funkciu `startMotor()`, ktorá naopak začne rotovať jadro, vždy pred začatím merania.

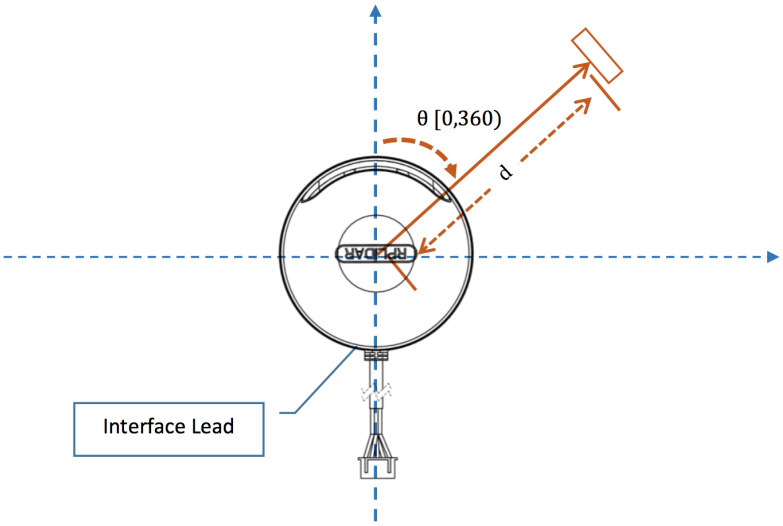
Po ukončení všetkých operácií v programe môže užívateľ zavolať funkciu `disconnect()`, ktorá uzavrie spojenie s lidarom a uvoľní sériový port.

Meranie a získanie dát

Funkcia `startScan()` spúšťa pracovné vlákno v pozadí, ktoré získava sken merania posielaný asynchrónne z lidaru. Prijatá dátová sekvencia je ukladaná v internej medzipamäti, z ktorej ju vyzdvihuje funkcia `grabScanData()`. Túto funkciu môžeme používať v našom programe na získanie nameraných dát zo senzora. Funkcia vždy vracia posledné a úplné 360-stupňové meranie. Po každom volaní funkcie `grabScanData()` sa vnútorná medzipamäť prečisťuje, aby bolo zaručené, že nedostaneme duplicitné dáta. V prípade ak ešte nie je kompletný 360-stupňový sken hotový, funkcia počká na dokončenie skenu alebo kým nevyprší čas. Túto časovú hranicu môžeme upravovať, aby sme dosiahli v našom programe podmienky, ktoré potrebujeme.

Po získaní najnovších dát je možné dáta usporiadať pomocou funkcie `ascendData()`, ktorá zoradí jednotlivé merania (lúče) podľa uhla. Snímacie jadro sa točí v smere hodinových ručičiek a získa tak 360-stupňový sken prostredia, čo je naznačené na obrázku 2.6.

Okrem dát merania, vieme z lidaru získať aj informácie funkciami ako `getHealth()`, ktorá vráti „zdravie“ lidaru, alebo `getFrequency()`, ktorá zase vráti frekvenciu točenia snímacieho jadra.



Obr. 2.6: Smer znímania dát a ich uporiadania pre RPLidar A2 [4]

Kapitola 3

Podobné systémy

Okrem robota Mikeša disponuje FT laboratórium aj robotom Smelý Zajko, ktorý je využívaný napríklad v súťaži RoboTour - Robotika.cz Outdoor Delivery Contest [13]. V tejto práci bola lokalizácia realizovaná len pomocou mapy, GPS a kompasu. Smelý Zajko bol pred nedávnom vybavený laserovým senzorom na meranie vzdialenost Hokuyo a v súčasnosti bola dokončená práca, kde sa tento senzor využíva na vyhýbanie sa prekážkam [7]. Signály z lidaru však opäť neboli využité na lokalizáciu. Tej sa venovali viaceré študentské práce: ešte v roku 2011 v diplomovej práci Lukáš Riško [14]. využil ultrazvukové senzory na pravdepodobnostnú lokalizáciu robota Robotnačka vo virtuálnom robotickom laboratóriu. V jednom z cvičení na neho nadviazal na neho Lukáš Slovák [15] využívajúc rozšírenú robotickú stavebnicu LEGO MINDSTORMS EV3. Výsledky však vzhľadom na povahu senzorov neboli veľmi presvedčivé. Zaujímavým príspevkom k lokalizácii bola diplomová práca Maríny Madovej [9], ktorá na lokalizáciu využívala rozpoznávanie význačných bodov v obraze metódou SURF. Niektoré výstupy z tejto práce (mapa pavilónu I vo formáte SVG, webové rozhranie pre doručovacieho robota) sú relevantné aj pre našu prácu.

Kapitola 4

Špecifikácia cieľov práce

Úlohou je vytvoriť spoľahlivý a využiteľný doručovací mobilný robotický systém do pavilónu, ktorý bude využívať dáta zo všetkých dostupných senzorov: gyroskopu, akcelerometra, kompasu, laserového range senzora, ultrazvukových a infračervených senzorov, otáčkových senzorov a kamery.

Základným cieľom je zabezpečiť správnu lokalizáciu a navigáciu robota v interiéri budovy. Ďalším cieľom je zabezpečiť komunikáciu s užívateľom.

Prvou požiadavkou je navrhnúť model architektúry, v ktorom budú popísané jednotlivé moduly, ich úlohy a interakcie medzi nimi.

Na lokalizáciu je nevyhnuté zaviesť súradnicový systém, v ktorom budeme ukladať aktuálnu pozíciu robota. Súradnicový systém sa bude odvíjať od mapy budovy, ktorú máme dostupnú zo staršej študentskej práce [9]. Ďalšou požiadavkou je vedieť spracovať mapu do internej reprezentácie v podobe zoznamu úsečiek a v podobe grid mapy s označenou priechodnosťou políčok.

Požiadavkou pre primárnu lokalizáciu bude zahrnúť údaje z otáčkových senzorov a pomocou nich určovať aktuálnu polohu robota.

Keďže sa počas lokalizácie len pomocou odometrie rýchlo zvyšuje chyba aktuálnej pozície, bude treba vybrať a implementovať pomocný lokalizačný algoritmus. Rozhodli sme sa pre lokalizáciu Monte Carlo. Pre tento typ lokalizácie je potrebné navrhnúť model pohybu a model počítania váhy jednotlivých hypotéz, v ktorom pokusmi určíme potrebné koeficienty.

Ďalšou požiadavkou je vytvoriť plánovač trasy podľa aktuálnej pozície a zadaného cieľa. Plánovač bude využívať grid mapu a prehľadávací algoritmus A*. Cieľ by mal robot dostávať z webového rozhrania.

Následne je treba vytvoriť modul, ktorý podľa naplánovanej trasy bude ovládať motory kolies a navigovať robota tak správnym smerom. Tento modul bude zároveň zabezpečovať obchádzanie prekážok.

Konečnou požiadavkou je zabezpečiť všetkým modulom potrebné vstupné dáta a

zosynchronizovať komunikáciu medzi jednotlivými modulmi.

Komunikácia s užívateľom bude zabezpečená pomocou webového rozhrania vytvoreného v študentskej práci [9].

Kapitola 5

Návrh a implementácia

V tejto kapitole popíšeme návrh architektúry projektu a podrobne opíšeme všetky jeho časti a ich interakciu.

5.1 Model architektúry

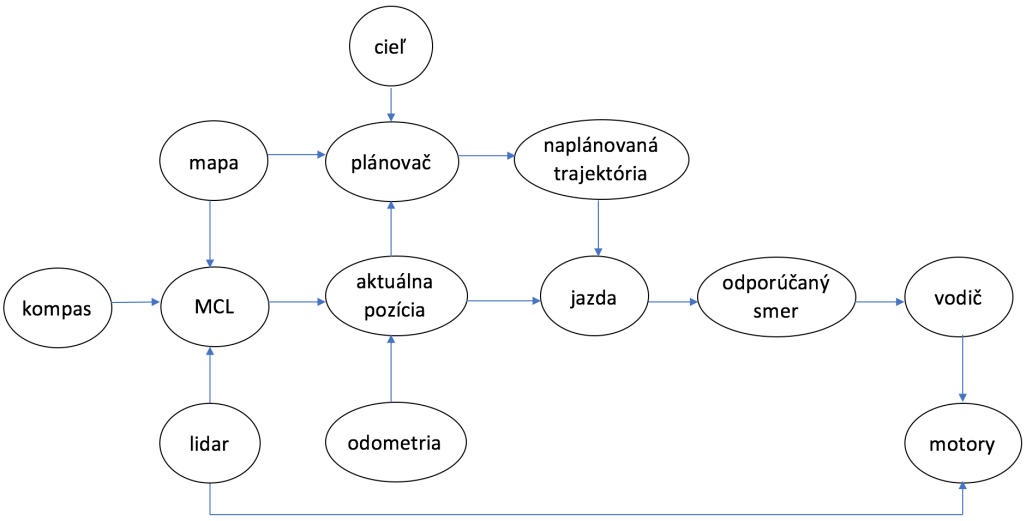
Graf architektúry na obrázku 5.1 vyjadruje, aké všetky moduly sa zapájajú a ako spolu interagujú.

5.1.1 Mapa

Keďže cieľom práce je spoľahlivý robotický doručovací systém v pavilóne, potrebujeme mať dostatočne presnú *mapu budovy*. V modele rátame s mapou, ktorá pozostáva zo segmentov (úsečiek), ktoré určujú steny a prekážky. V prípade pavilónu I (ukážka 5.1) mapa pozostáva zo segmentov stien vytvárajúcich tri polygóny: vonkajší obvod chodby, obvod počítačových miestností (H3 a H6) a obvod átria a kancelácií. Túto mapu v programe načítame a vykreslíme do samostatného grafického okna. Túto mapu využívame pri lokalizácii robota pomocou MCL.

Okrem tohto jednoduchého načítania a vykreslenia vytvoríme aj *grid mapu* pavilónu. Mapu budovy rozdelíme na štvorce. Dĺžku strany štvorca predpokladáme 25cm. Grid mapa bude slúžiť modulu *plánovač* na vypočítanie *trajektórie* pohybu k zadanému *cieľu*.

Túto grid mapu vytvoríme pomocou rátania priesečníkov: ak sa ľubovoľná strana štvorca pretína s ľubovoľným segmentom mapy, tak sa celý štvorec pretína s nejakým segmentom, a teda ho označíme ako nepriechodný. Navyše keďže sme si učili dĺžku strany štvorca menšiu ako je šírka robota, každý štvorec grid mapy, ktorý sa nachádza hneď pri nepriechodnom štvorci, označíme tiež za nepriechodný. Inými slovami, steny pavilónu akoby napuchnú o jeden šírku jedného štvorca.



Obr. 5.1: Návrh architektúry

Takto nám zostanú priechodné štvorce, ktoré budú určovať, kade sa môže robot hýbať. Musíme si dať pozor, aby zostali priechodné všetky časti budove, ktoré priechodné sú. V úzkych miestach sa môže stať, že po „napuchnutí“ stien sa tieto miesta znepriechodnia. Problém môžeme riešiť zmenou dĺžky strany štvorca grid mapy.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<svg xmlns="http://www.w3.org/2000/svg" height="4303" width="4578">
<line x1="0" y1="0" x2="0" y2="265" style="stroke:rgb(0,0,0);stroke-
    width:2" lineId="0" />
<line x1="0" y1="265" x2="0" y2="352" style="stroke:rgb(0,0,0);stroke-
    width:4" lineId="1" doors="I1"/>
<line x1="0" y1="352" x2="0" y2="465" style="stroke:rgb(0,0,0);stroke-
    width:2" lineId="2" />
<line x1="0" y1="465" x2="0" y2="552" style="stroke:rgb(0,0,0);stroke-
    width:4" lineId="3" doors="I2"/>
<line x1="0" y1="552" x2="0" y2="966" style="stroke:rgb(0,0,0);stroke-
    width:2" lineId="4" />
...
</svg>
```

Ukážka kódu 5.1: Ukážka SVG súboru mapy pavilónu I

5.1.2 Plánovač

Modul *plánovač* dostane na začiatku grid mapu pavilónu. Počas behu programu dostáva *aktuálnu pozíciu* určenú súradnicami na mape a *cieľ* cesty. Zo súradníc aktuálnej pozície vieme vypočítať, v ktorom štvorci grid mapy sa robot nachádza jednoducho: súradnice vydelíme šírkou štvorca a zaokrúhlime nahor. Súradnicový systém polohy robota má

počiatok v ľavom dolnom rohu mapy.

Na vypočítanie trajektórie pohybu, teda postupnosti štvorcov z aktuálne pozície do cieľa, použijeme prehľadávací algoritmus A*. Tento algoritmus je grafový prehľadávací algoritmus podobný prehľadávaniu do šírky s tým rozdielom, že vrcholy vo fronte sú utriedené podľa váhy a vždy vyberáme z fronty vrchol s najväčšou váhou. Váhy vrcholov sú určene nejakou funkciou. V našej práci použijeme tzv. *Manhatanskú vzdialenosť*. Test plánovača popisujeme v časti **??**.

Plánovač asynchrónne dostáva aktuálnu pozíciu robota, poprípade cieľ, a podľa toho prepočítava naplánovanú trajektóriu.

5.1.3 Cieľ

Cieľom cesty robota je miestnosť, resp. dvere do miestnosti, ktoré robot dostane pomocou webového rozhrania spomenutého v sekcii **??**. Na to, aby robot zastavil pred dverami do zadanej miestnosti, musíme vypočítať štvorec, do ktorého má prísť. Ako referenčný bod vezmeme stred segmentu, ktorý reprezentuje zadané dvere a opäť vydelíme jeho súradnice šírkou štvorca grid mapy a dostaneme tak cieľový štvorec, ktorý posielame plánovača.

5.1.4 Odometria

Odometriu využívame zistenie prejdenej vzdialenosti v milimetroch z počtu tikov nameraných an kolesách.

5.1.5 Aktuálna pozícia

Aktuálna pozícia je vyjadrená tromi veličinami: súradnicami x a y a otočením. Aktuálnu pozíciu aktualizujeme podľa odometrie. Okrem toho ju aktualizuje aj MCL vždy, keď nejaká hypotéza získa výrazne vyššiu váhu ako ostatné. Pomocou aktuálnej pozície vieme vykresľovať do mapy prejdenú trajektóriu.

5.1.6 MCL

Monte Carlo lokalizácia pomáha robotovi presnejšie sa orientovať v priestore budovy a zisťovať aktuálnu pozíciu. Vždy, keď nejaká hypotéza získa výrazne vyššiu váhu ako ostatné a aktuálna pozícia sa líši od tejto hypotézy, tak MCL nastaví aktuaálnu pozíciu na túto hypotézu.

Inicializácia

Pri inicializácii MCL sa generuje počiatočná množina polôh robota. Vygenerujeme náhodné súradnice a uhol. Treba však zistiť, či dané súradnice sú vygenerované vnútri budovy, tz. na chodbe.

Bod v polygóne

Na to sme použili tzv. *algoritmus počítania lúčov* (anlg. ray casting algorithm). Základnou ideou tohto algoritmu je, že pri ceste z bodu počítame, koľkokrát sme prešli cez nejakú hranu polygónu. Ak počet prechodov je nepárny, máme istotu, že bod je vnútri polygónu. V opačnom prípade sa bod nenachádza v polygóne.

V prípade pavilónu I, zisťujeme, či sa zadaný bod nachádza v polygóne opisujúcom vonkajší obvod chodby, ale zároveň sa nenachádza ani v polygóne opisujúcom átrium ani počítačové miestnosti (H3, H6).

MCL aktualizácia

Procedúra aktualizovania pravdepodobnosti hypotéz je volaná pravidelne a pozostáva z viacerých krokov. Predstavuje vlastne jednu iteráciu algoritmu MCL.

Procedúra dostane na vstup pohyb (koľko robot prešiel a akým smerom) od poslednej aktualizácie a aktuálne dáta namerané lidarom. Každú hypotézu posunieme najskôr o vzdialenosť, ktorú robot prešiel a otočíme o prejdený uhol. Hneď skontrolujeme, či hypotéza nevyšla týmto posunom z chodby (opäť algoritmus Bod v polygóne) a ak áno, jej pravdepodobnosť klesne na nulu.

V nasledujúcom kroku aktualizujeme váhu každej hypotézy, ktorá reprezentuje pravdepodobnosť toho, že sa robot nachádza na danej pozícii. Zoberieme každý lúč nameraných dát a vypočítame, aký dlhý lúč by sme mali v danom uhle namerať. Pomocou modelu merania senzora dostaneme pravdepodobnosť nameranej dĺžky pri vypočítanej dĺžke pomocou normálneho rozdelenia so stredom vo vypočítanej dĺžke.

Novú váhu normalizujeme koeficientom η, ktorý sme určili na hodnotu $0,2$.

Váhu hypotézy potom z malej časti upravíme na nový váhu. A malej časti preto, lebo nová váha predstavuje len jeden z lúčov lidaru.

Do úvahy berieme len tie lúče, ktoré majú nameranú nejakú dĺžku (nie sú nulové) a smerujú v požadovanom rozsahu v smere robota.

Ďalším krokom je vygenerovanie nasledujúcej množiny častíc. Každú hypotézu vyberieme do novej množiny hypotéz s takou pravdepodobnosťou, aká je jej váha. Vieme si to predstaviť ako tzv Roulette view - kruh s obvodom $\sum_x w_x$, kde w_x je váha každej hypotézy, a teda každá hypotéza zaberá kruhový oblúk s dĺžkou w_x. Potom generujeme hypotézy náhodne z tohto kruhu, teda z intervalu $\langle 0, \sum_x w_x)$.

Každú hypotézu z novej množiny „zašumíme“. To znamená, že jej súradnice o náhodný kúsok posunieme a tiež jej zmeníme uhol.

Nakoniec nahradíme v novej množine malé percento hypotéz novými, úplne náhodnými hypotézami, ktoré zvyšujú robustnosť algoritmu a umožňujú mu spamätať sa napríklad z manuálneho presunutia robota alebo ak nejaká nesprávna hypotéza získa príliš veľkú váhu.

5.1.7 Kompas

Kompas pomáha určiť pravdepodobnosť hypotéz v prvotnej množine MCL. Hypotézy otočené v smere robota dostanú vyššiu pravdepodobnosť ako hypotézy v zlom smere.

5.1.8 Lidar

Dáta z lidaru vstupujú do MCL algoritmu. Pomocou nich sa prepočítava pravdepodobnosť hypotéz.

S rovnakou dôležitosťou sa využívajú tieto dáta aj na nižšej úrovni. Vieme z nich zistiť, či sa robot blíži ku stene alebo či sa pred ním nachádza prekážka a podľa toho nastaviť rýchlosť motorov za účelom obídenia prekážky alebo odklonenia od steny.

5.1.9 Jazda

Modul *jazda* dostáva na vstup aktuálnu pozíciu a naplánovanú trajektóriu. Modul má dve úlohy: zistiť, na ktorom políčku naplánovanej trajektórie sa momentálne robot nachádza a vypočítať vektor z aktuálnej pozície do niektorého nasledujúceho políčka trajektórie. Predpokladáme, že najviac vyhovujúce bude tretie políčko od robota. Z vypočítaného vektora a aktuálnej pozície modul určí odporúčaný smer jazdy.

5.1.10 Odporúčaný smer jazdy

Tento smer reprezentujeme želaným uhlom, do ktorého sa robot chce časom dostať.

5.1.11 Vodič

Vodič je modul, ktorý neustále sleduje dáta z lidaru, prepočítava ich a zisťuje pritom, či sa robot nepriblíži ku stene alebo sa pred ním nenachádza iná prekážka. Po zistení prekážky prikáže robotovi vyhnúť sa jej: v prípade prekážky pred robotom sa robot zastaví a na mieste sa otočí doľava alebo doprava. Pre zistenie, do ktorej strany sa má robot otočiť, počítame priemerný ľavý lúč a priemerný pravý lúč. Prípade prekážky na ľavej (pravej) strane robota spomalíme pravé (ľavé) koliesko a tým sa robot vyhne prekážke alebo sa odkloní od steny.

Ak v má robot voľnú cestu, modul zisťuje odporúčaný smer jazdy a podľa neho nastavuje rýchlosti motorov. Robot takto nemusí zastaviť a na mieste sa otočiť ale vie sa hýbať po kruhovom oblúku podľa naplánovanej trajektórie. Kruhový oblú vykonávame len v prípade, ak sa odporúčaný smer jazdy líši najviac o 90 stupňov. V inom prípade volíme stratégiu zastavenia a otočenia sa na mieste o časť tohto uhla.

V prípade, že odporúčaný smer jazdy káže robotovi ísť doľava (doprava), ľavému (pravému) koliesku nastavíme nižšiu rýchlosť úmerne k uhlu odporúčaného smeru.

5.1.12 Motory

Motory reagujú na príkazy z vyššie spomenutého modulu vodič.

5.2 Implementácia

Program je napísaný prevažne v jazyku C, ktorý sa javí byť vhodným jazykom pre jednoduché robotické systémy najmä vďaka jeho rýchlosti.

Program je viacvláknový a pozostáva z niekoľkých modulov, ktoré reprezentujú moduly popísané v predchádzajúcej časti.

mikes.c Tento hlavný modul spúšťa funkciu `main()`, ktorá načíta konfiguračné nastavenia zo súboru, inicializuje všetky potrebné vlákna a premenné. Funkcia potom vyčkáva na signál a na ukončenie ostatných vlákien.

mikes_logs.c Pomocou tohto modulu zapisujeme log súbory.

base_module.c Tento modul poskytuje nízkoúrovňovú komunikáciu s robotom. V jeho hlavičke `base_module.h` definujeme základnú štruktúru `base_data_type` nasledovne:

```
typedef struct astruct {
  long counterA, counterB;     // 144 ticks per revolution
                               //  A=left, B=right
  short velocityA, velocityB; // current angular velocity in deg/s
  short dist1, dist2, dist3;  // IR distance sensors
  short cube;                 // cube presence IR sensor
  short heading;              // last compass reading 0-360
  short ax, ay, az;           // normalized accelerometer (ms^-2)
  short gx, gy, gz;           // normalized gyroscope
} base_data_type;
```

Túto štruktúru používame na ukladanie mechanických dát z robota. Premenné `counterA` a `counterB` vyjadrujú počet tikov od ich posledného vynulovania. Motory spravia 144 tikov za otáčku. Pomocou týchto počítadiel teda vieme vyjadriť prejdenú vzdialenosť v dĺžkových jednotkách (napr. mm). V premenných `velocityA` a `velocityB` ukladáme aktuálnu uhlovú rýchlosť motorov v stupňoch za sekundu. Nameranú vzdialenosť z IR snímačov vyjadrujú premenné `dist1, dist2, dist3` a `cube` (tento senzor sa využíva v inej študentskej práci). Nakoniec sa v štruktúre `base_data_type` nachádza aktuálny kurz `heading`, akcelerometer `ax, ay, az` a gyroskop `gx, gy, gz`.

lidar.cpp Jediný modul, ktorý nieje v jazyku C ale v jazyku C++. Dôvodom je kompatibilita s SDK, ktoré nám uľahčuje komunikáciu a prácu s lidarom. V hlavičke `lidar.h` definujeme nasledovný štruktúru `lidar_data_type`, do ktorého si v programe ukladáme dáta z lidaru – počet lúčov, vzdialenosť, uhol a kvalitu jednotlivých lúčov. Počet lúčov v jednotlivých meraniach sa môže líšiť v závislosti od nastavenej frekvencie točenia jadra, alebo aj od toho, či je lidar už zahriaty, či iných okolností.

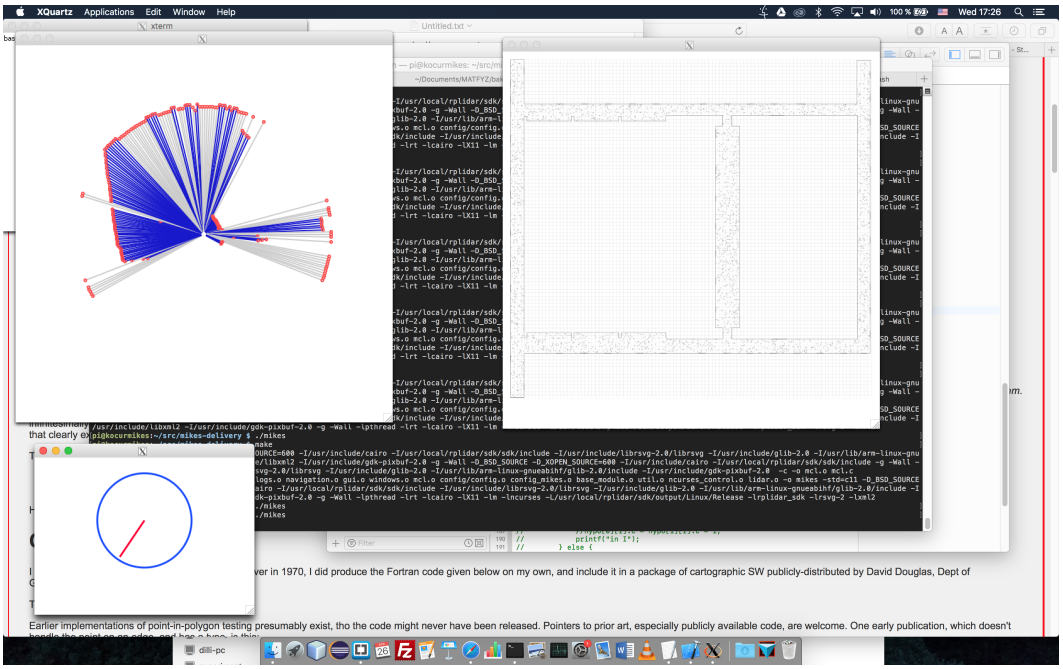
```
#define MAX_LIDAR_DATA_COUNT 720

typedef struct lidarstruct {
  uint16_t count;
  uint8_t quality[MAX_LIDAR_DATA_COUNT];
  uint16_t distance[MAX_LIDAR_DATA_COUNT];
  uint16_t angle[MAX_LIDAR_DATA_COUNT];
} lidar_data_type;
```

Pri inicializácii sa alokuje pamäť pre premenné na ukladanie nasnímaných dát a spustí sa metóda `connect_lidar()`. Vtedy sa pomocou SDK funkcií vytvorí nová inštancia triedy `RPlidarDriver` a skontroluje sa „zdravotný stav“ lidaru. Vytvorí sa nové vlákno, ktoré neustále spúšťa snímanie, zbiera dáta (`grabScanData()`), usporadúva ich (`ascendData()`) a tieto najnovšie dáta pod zámkom kopíruje do premennej `lidar_data`, z ktorej získavajú nasnímané dáta ostatné moduly.

Pri ukončovaní programu sa vo vlákne ukončí snímanie, zastaví sa motor lidaru, uvoľní sa inštancia ovládača a premenné.

gui.c a windows.c Moduly, ktoré pomocou Cairo [1] knižnice vykresľujú grafické rozhranie. Jednotlivé okná sú zobrazené na obrázku 5.2. Vidíme na ňom kompas, aktuálne lúče, ktoré lidar nasnímal a mapu pavilónu I.

config_mikes.c Pomocou tohto jednoduchého modulu vieme ovládať rôzne nastavenia: či sa majú zobrazovať grafické okná, či sa má robot spustiť automaticky po



Obr. 5.2: Grafické okná

zapnutí, či vypisujeme všetky logy do konzoly alebo či robota ovládame ručne pomocou klávesnice.

Kapitola 6

Popis experimentov

V tejto kapitole si predstavíme, aké experimenty sme počas práce uskutočnili. Poukážeme na to, čo bolo treba k jednotlivým experimentom pripraviť, popíšem konkrétne kroky experimentov a povieme si, čo od jednotlivých experimentov očakávame. Pri experimentoch predpokladáme, že máme spracovanú mapu budovy do zoznamu segmentov aj ako grid mapu a že máme možnosť vykresľovať výstupy experimentov do grafického okna mapy.

6.1 Lokalizácia

6.1.1 Simulovaný pohyb

Tento jednoduchý experiment simuluje pohyb robota a na výstup vypisuje súradnice po vykonaní jednotlivých krokov.

Očakávania

Vypisované súradnice budú zodpovedať vypočítaným súradniciam pri priamom pohybe, otočení na mieste aj pri pohybe po kruhovom oblúku.

Príprava

Určíme si pohyby, ktoré chceme, aby robot prešiel. Prepočítame vzdialenosti trajektórií kolies na tiky. Pohyb budeme reprezentovať ako dvojicu (L, R), kde L (resp. R) je počet tikov ľavého (resp. pravého) kolesa a vytvoríme postupnosť takýchto dvojíc, podľa želanej trajektórie. Vypočítame súradnice po každom pohybe.

Postupnosť krokov

Simulujeme trajektóriu robota pomocou postupnosti pohybov. Po každom pohybe na konzolu vypíšeme aktuálnu pozíciu a porovnáme s vypočítanou pozíciou.

6.1.2 Pohyb bez MCL

V tomto experimente testujeme lokalizáciu len na základe odometrie. Robotovi zadáme štartovnú pozíciu a necháme ho ísť vpred. Trajektóriu robota chceme zaznamenať do mapy.

Očakávania

Robot sa má hýbať vpred pokým nenarazí na prekážku, alebo sa neblíži k stenám. Modul *vodič* má riadiť motory kolies tak, aby obišiel prekážku alebo sa odklonil od steny a následne pokračoval rovno. Prejdená trajektória zaznačená do mapy a pozícia robota na mape po zastavení motorov by mali zodpovedať skutočnej trajektórii a pozícii robota.

Príprava

Potrebujeme pripraviť hlavný modul *vodič* a modul *jazda*. Taktiež treba pripraviť vykresľovanie trajektórie do mapy. Na mape nájdeme bod, z ktorého chceme aby robot štartoval a vypočítame uhol, ktorým bude robot otočený.

Postupnosť krokov

Robota umiestnime na vybrané miesto a správne ho nasmerujeme. Nastavíme mu túto polohu a spustíme modul vodič. Ten sa bude vyhýbať prekážkam, odkláňať sa od stien a poháňať vpred robota. Porovnávame prejdenú trajektóriu s trajektóriou vykreslenou v mape. Po dostatočne veľkej prejdenej vzdialenosti zastavíme robota a porovnáme výslednú reálnu polohu robota s tou na mape. Pri opakovaní experimentu necháme robota prejsť vždy väčšiu vzdialenosť.

6.1.3 MCL bez pohybu

Tento experiment testuje algoritmus MCL pri statickej polohe. Motory pri tomto experimente teda nezapájame.

Očakávania

Chceme zistiť, či MCL algoritmus správne prepočítava váhy jednotlivých hypotéz.

Príprava

MCl musí vedieť generovať primárnu množinu hypotéz a určiť im primárnu váhu. Aktualizácia MCL potom musí vedieť prepočítavať váhu jednotlivých hypotéz využitím nameraných dát z lidaru.

Postupnosť krokov

Robota umiestnime na vybrané miesto a správne ho nasmerujeme. Necháme iterovať MCL algoritmus a sledujeme ako sa menia váhy hypotéz. Po dostatočnom počte iterácií porovnáme výsledné najvýraznejšie hypotézy s očakávanými miestami. Experiment budeme opakovať na rôznych miestach budovy.

6.1.4 Pohyb a MCL

Predošlé experimenty spojíme do jedného experimentu, v ktorom sa aktuálna pozícia bude aktualizovať nielen pomocou otáčkových senzorov, ale aj vďaka MCL.

Očakávania

Experiment začne rovnako ako v 6.1.2. Vždy po prejdení určitej vzdialenosti sa spustí iterácia MCL, ktorej výstupom by mala byť najpravdepodobnejšia poloha robota. Ak táto poloha bude jedinečná, tz. všetky ostatné hypotézy budú mať oveľa nižšiu váhu, tak nastavíme aktuálnu pozíciu robota na túto hypotézu. MCL by tak mala opravovať chyby spôsobené odometriou a aktualizovať polohu robota vždy, keď je chyba už príliš veľká.

Príprava

K predošlým experimentom musíme pridať ich prepojenie. Chceme aby sa MCL nespúšťala nepretržite, ale vždy len po prejdení určitej vzdialenosti.

Postupnosť krokov

Postupnosť krokov je rovnaká ako v experimente 6.1.2, pričom na mapu vykresľujeme aj jednotlivé hypotézy a sledujeme ich váhu ako v 6.1.3. Všímame si výrazné hypotézy a či sa robotova pozícia aktualizuje vždy, keď dostaneme jedinečnú hypotézu.

6.2 Navigácia

6.2.1 Plánovanie 1

Očakávania

Argoritmus A* nájde najkratšiu cestu medzi zadaným štartom a cieľom simulovanej grid mapy.

Príprava

Pripravíme malú simulovanú grid mapu. Implementujeme algoritmus A*, ktorý potrebuje implementovať aj nejakú formu prioritnej fronty.

Postupnosť krokov

Algoritmu pošleme mapu a štartovnú a cieľovú pozíciu. Do konzoly vypíšeme nájdenú cestu, reprezentovanú ako postupnosť súradníc štvorčekov grid mapy.

6.2.2 Plánovanie 2

Očakávania

Argoritmus A* nájde najkratšiu cestu medzi aktuálnou pozíciou a zadaným cieľom na grid mape pavilónu I.

Postupnosť krokov

Algoritmu pošleme mapu pavilónu I, aktuálnu pozíciu robota a cieľové súradnice. Do mapy vykreslíme nájdenú cestu, reprezentovanú ako postupnosť štvorčekov grid mapy.

6.2.3 Plánovanie a pohyb - navigácia podľa trajektórie

V tomto experimente testujeme lokalizáciu a navigáciu na základe odometrie. Robotovi nastavíme štartovnú pozíciu a určíme cieľové súradnice. Algoritmus A* vypočíta naplánovanú trajektóriu ako postupnosť štvorčekov grid mapy. Modul jazda bude prepočítavať aktuálnu pozíciu na trajektórii a želaný smer podľa trajektórie. Podľa toho bude modul vodič riadiť rýchlosti motorov. Zároveň sa bude vyhýbať prekážkam a odkláňať sa od stien. Trajektóriu robota chceme zaznamenať do mapy.

Očakávania

Robot sa má otáčať v kruhovom oblúku do želaného smeru pokým nie je správne natočený. Prejdená trajektória zaznačená do mapy by sa mala držať v okolí naplánovanej trajektórie. Pozícia robota po zastavení motorov by mala zodpovedať poslednej pozícii na mape a cieľovým súradniciam.

Postupnosť krokov

Robota umiestnime na vybrané miesto a správne ho nasmerujeme. Nastavíme mu túto polohu a cieľové súradnice. Algoritmus A* nájde naplánovanú trajektóriu. Vodič bude riadiť rýchlosti motorov podľa želaného smeru z naplánovanej trajektórie, vyhýbať sa

prekážkam a odkláňať sa od stien. Trajektóriu vykresľujeme do mapy. Porovnávame prejdenú trajektóriu s trajektóriou vykreslenou v mape, sledujeme pritom, či sa trajektória robota nevzďaľuje príliš od naplánovanej trajektórie.

Kapitola 7

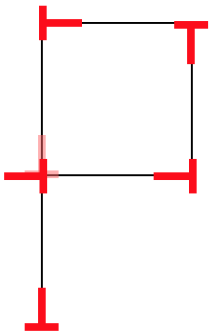
Výsledky

Kapitola opisuje prevedenie experimentov. Niektoré naplánované experimenty sa nepodarilo realizovať. Dôvodom je náročné nastavenie parametrov modulov vodič a MCL.

7.1 Lokalizácia

7.1.1 Simulovaný pohyb

V prvom kroku testuje priamy pohyb a otočenie na mieste zatiaľ čo v druhom kroku testujeme pohyb po kruhovom oblúku. Pri 30-tich tikoch prejde koleso dráhu približne 10cm. Na obrázku 7.1 vidíme trajektóriu, ktorú sme si určili a vypočítali jej pohyby a súradnice. Robot začína na spodnej pozícii. Pri každom pohybe vpred má robot prejsť približne 10cm. Uhol rátame od y-ovej súradnice v smere hodinových ručičiek.



Obr. 7.1: Simulovaný pohyb – postupnosť pohybov

```
0. pose x y heading:   0.00 102.71   0.00
1. pose x y heading:   0.00 205.42   0.00
2. pose x y heading:   0.00 205.42  89.33
3. pose x y heading: 102.70 206.62  89.33
4. pose x y heading: 102.70 206.62 179.86
5. pose x y heading: 102.95 103.91 179.86
```

```
7   6. pose x y heading: 102.95 103.91 269.19
8   7. pose x y heading:   0.25 102.46 269.19
```

Ukážka kódu 7.1: Výstup experimentu Simulovaný pohyb

V tabuľke 7.1 vidíme, že vypísané súradnice približne zodpovedajú tým vypočítaným. Nepresnosti vznikli zaokrúhľovaním pri konverzii milimetrov na tiky.

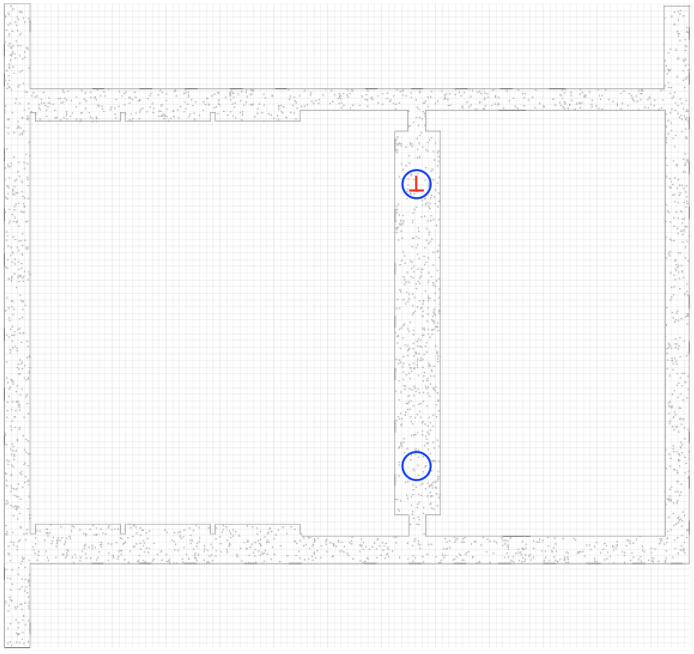
7.1.2 Pohyb bez MCL

Tento experiment sme realizovali spolu z naplánovanou trajektóriou ako experiment 7.2.3.

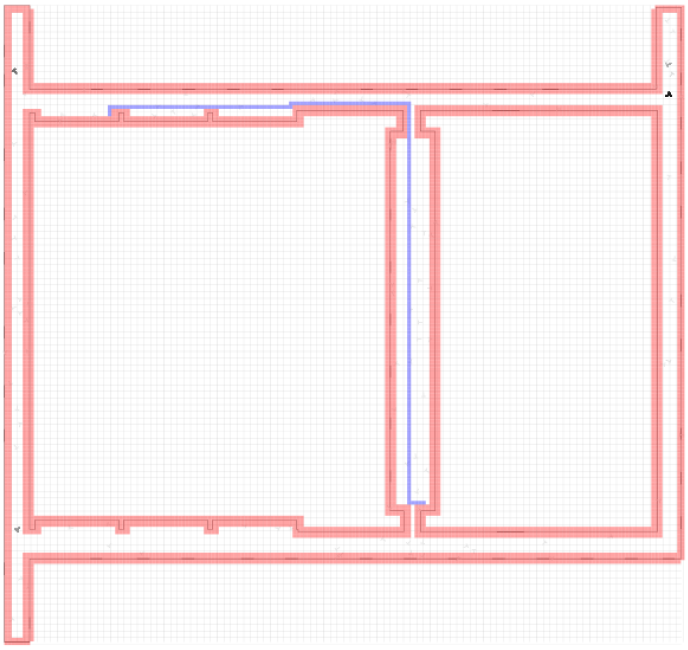
7.1.3 MCL bez pohybu

Obr. 7.2: Prvé testovanie MCL bez pohybu

Robota sme umiestnili do význačného miesta. Na obrázku 7.2 vidíme reálne umiestnenie robota v budove. Na mape na obrázku 7.3 je poloha robota vyznačená červeným obráteným T. Modré kružnice určujú miesta, kde by sa mala váha hypotéz výrazne zvýšiť. V tomto prípade máme dve pravdeodobné miesta, lebo tieto sú takmer totožné a robot ich nemá šancu bez ďalšieho pohybu odlíšiť.



Obr. 7.3: Experiment MCL bez pohybu - očakávaný výstup



Obr. 7.4: Prvé testovanie MCL bez pohybu – hypotézy

Môžeme si všimnúť, že MCL algoritmus priradil vysokú váhu nesprávnym hypotézam (obrázok 7.4 vpraho hore, vľavo hore a vľavo dole, modrá naplánovaná trajektória s týmto experimentom nesúvisí). Problémom je zdĺhavé nastavovanie parametrov algoritmu.

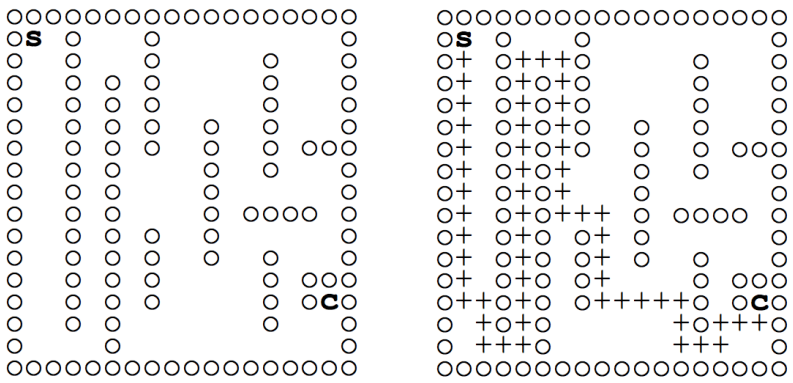
7.1.4 Pohyb a MCL

Tento veľký experiment sa nám nepodarilo zrealizovať kvôli náročnosti nastavenia parametrov v predchádzajúcich experimentoch.

7.2 Navigácia

7.2.1 Plánovanie 1

Pripravili sme malú simulovanú grid mapu (obrázok 7.5 vľavo). Symbol `s` označuje začiatočnú pozíciu a symbol `c` označuje cieľ.



Obr. 7.5: Testovanie algoritmu A* – simulácia grid mapy

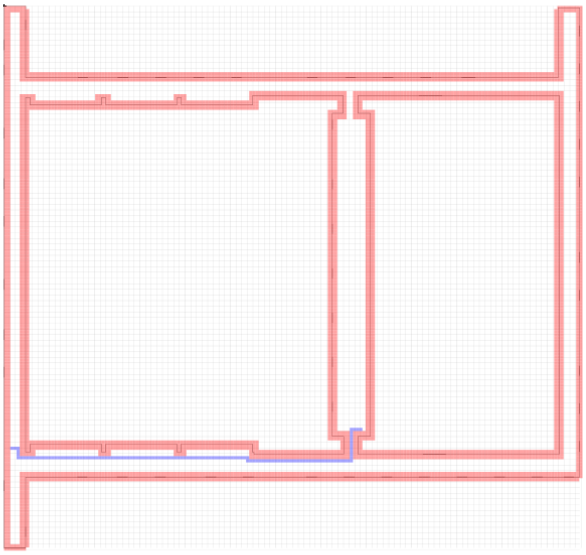
Výsledný výpis bol takmer správny: (1, 1), (2, 1), (3, 1), ... (15, 13), (15, 14), (14, 14), (14, 15), (14, 16). Celková dĺžka trajektórie bola 57 štvorčekov, avšak algoritmus do trajektórie nezahrnul cieľovú pozíciu. Trajektóriu, ktorú algoritmus naplánoval vidíme na obrázku 7.5 vpravo vyznačenú symbolmi +.

7.2.2 Plánovanie 2

Testom sme overili, že algoritmus A* funguje bez problémov aj na grid mape pavilónu (obrázok 7.6).

7.2.3 Plánovanie a pohyb - navigácia podľa trajektórie

Robot sa počas testu úspešne odkláňal od stien a vyhýbal prekážkam, ale želaný smer nevedel dokonale dodržať. Niekedy reagoval na prekážku prehnaným otočením. Výsledkom testu je, že je potrebné lepšie odladiť potrebné koeficienty.



Obr. 7.6: Testovanie algoritmu A* – grid mapa pavilónu I, naplánovaná trajektória je vyznačená modrou

Záver

V priebehu bakalárskej práce bol robot Mikeš vybavený novým senzorom RPLidar, ktorý sa nám podarilo úspešne integrovať do frameworku robota, preskúmať a odhaliť jeho silné aj slabé stránky, čím sme ušetrili prácu ďalším, ktorí budú s platfromou pracovať.

Vytvorili sme samostatný modul na prácu so senzorom, ktorý je ľahko znovupoužiteľný v ďalších projektoch.

Samostatný modul plánovania cesty na mriežkovej aproximácii vektorovej mapy informatického pavilónu, ktorý sme vyvinuli je funkčný a hľadá optimálnu trajektóriu.

Analyzovali a naprogramovali sme samostatný modul pre lokalizáciu pomocou odometrie robota. Tento modul je úspešne otestovaný a znovupoužiteľný.

Rozšírili sme framework robota o grafickú vizualizáciu mapy, priebehu lokalizačných algoritmov.

Navrhli a úspešne otestovali sme flexibilný modul na nízkoúrovňovú navigáciu robota podľa naplánovanej trajektórie využívajúc otáčkové senzory a lidar.

Z doterajších pokusov o pravdepodobnostnú lokalizáciu v záverečných prácach skupiny sa nám podarilo dosiahnuť najpokročilejšie riešenie, ktoré berie do úvahy aj rozličné apriori pravdepodobnosti jednotlivých hypotéz množiny na základe aktuálnych hodnôt hypotéz, čo môžeme považovať za rozšírenie existujúcej metódy. V čiastkových praktických testoch sme overili funkčnosť metódy.

Pripravili sme praktický robotický systém, ktorý môže slúžiť pri rozličných popularizačných podujatiach skupiny a fakulty či pri výskume a výuke (robot s našou implementáciou modulov už bol použitý na jednom cvičení k predmetu Algoritmy umelej inteligencie v robotike).

Na našu prácu je možné nadviazať viacerými spôsobmi. Sklenené dvere kancelárií v informatickom pavilóne sa ukázali byť neviditeľné pre lidar, čo sa v aktuálnej verzii interpretuje ako chyba merania a znižuje to pravdepodobnosť správne lokalizovaných hypotéz. Model mapy by bolo vhodné rozšíriť tak, aby tento jav bral v úvahu a naopak, príslušné relevantné hypotézy boli posilnené. Jednoduchou transformáciou v štýle Houghovej transformácie, ktorá hľadá parametricky popísateľné geometrické útvary by bolo vo výhľade senzora možné hľadať priamky zodpovedajúce stenám. Tieto detekované priamky môžu dramaticky zlepšiť presnosť lokalizácie, ktorá by stále zostávala

na pravdepodobnostnej báze. Plánovací algoritmus by bolo možné rozšíriť o trajektórie s viacerými požadovanými zastávkami po ceste zo štartu do cieľa. Mapa by mohla byť dynamická a v prípade, že robot na niektorom mieste zaznamená väčšie prekážky, mohol by si ich dočasne pridať do mapy, aby zlepšil plánovanie v nasledujúcich doručovacích zadaniach. A napokon, je potrebných viacero dôkladnejších testov celého integrovaného robotického systému.

Literatúra

[1] Cairo tutorial. https://www.cairographics.org/tutorial/, 2012. [citované 13.5.2017].

[2] RPLIDAR - Interface Protocol and Application Notes. http://bucket.download.slamtec.com/d25d26d45180b88f3913796817e5db92e81cb823/LD208_SLAMTEC_rplidar_datasheet_A2M8_v1.0_en.pdf, 2017. [citované 1.5.2017].

[3] RPLIDAR - Introduction to Standard SDK. http://bucket.download.slamtec.com/351a5409ddfba077ad11ec5071e97ba5bf2c5d0a/LR002_SLAMTEC_rplidar_sdk_v1.0_en.pdf, 2017. [citované 1.5.2017].

[4] RPLIDAR A2 - Introduction and Datasheet. http://bucket.download.slamtec.com/d25d26d45180b88f3913796817e5db92e81cb823/LD208_SLAMTEC_rplidar_datasheet_A2M8_v1.0_en.pdf, 2017. [citované 1.5.2017].

[5] F. Duchoň. *Lokalizácia a navigácia Mobilných robotv do vnútorného prostredia*. Slovenská technická univerzita v Bratislave, 2012.

[6] G. Dudek and M. Jenkin. *Computational Principles of Mobile Robotics*. Cambridge University Press, 2000.

[7] J. Dúc. Robotour with Laser Range Sensor. Aktuálne obhajovaná diplomová práca, Univerzita Komenského v Bratislave - Fakulta matematiky, fyziky a informatiky, 2017.

[8] W. Randolph Franklin. PNPOLY - Point Inclusion in Polygon Test. https://wrf.ecse.rpi.edu//Research/Short_Notes/pnpoly.html, 2006. [citované 15.4.2017].

[9] M. Madová. Robot poštár. Diplomová práca, Univerzita Komenského v Bratislave - Fakulta matematiky, fyziky a informatiky, 2014.

[10] T. Matula. Techniky umělé inteligence pro filtraci nevyžádané pošty. Diplomová práca, Vysoké učení technické v Brne - Fakulta informačních technologií, 2014.

[11] M. Moravčík. Autonómny mobilný robot pre súťaž Robotour. Diplomová práca, Univerzita Komenského v Bratislave - Fakulta matematiky, fyziky a informatiky, 2015.

[12] R. Murphy. *An Intorduction to AI Robotics.* A Brandfird Book, 2000.

[13] P. Petrovič and M. Nadhájsky. Robotour solution as a learned behavior based on Artificial Neural Networks, RIE 2010, Bratislava. `http://dai.fmph.uniba.sk/~petrovic/pub/RIE_RoboTourFMFI_final.pdf`, 2010. [citované 15.5.2017].

[14] L. Riško. Pravdepodobnostná robotika v lokalizácii, mapovaní a plánovaní. Diplomová práca, Univerzita Komenského v Bratislave - Fakulta matematiky, fyziky a informatiky, 2017.

[15] O. Slovák. Pokročilé experimenty s robotickou stavebnicou EV3. Diplomová práca, Univerzita Komenského v Bratislave - Fakulta matematiky, fyziky a informatiky, 2015.

[16] S. Thrun, W. Bugard, and D. Fox. *Probabilistic Robotics.* MIT Press, 2005.